

MTAT.07.017
Applied Cryptography

Introduction, Randomness, PRNG,
One-Time Pad, Stream Ciphers

University of Tartu

Spring 2020

Who am I?

Arnis Paršovs

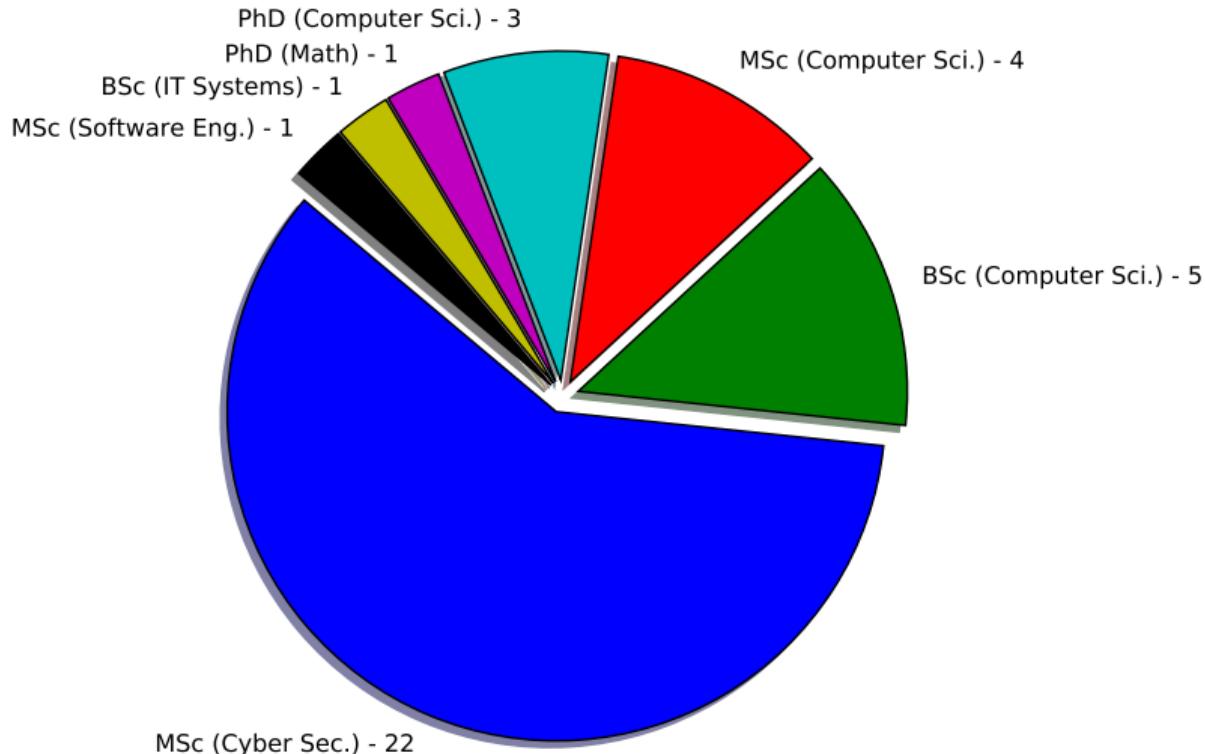
MSc in Cyber Security
TalTech, 2012

Computer Science PhD student at UT

Applied Cyber Security Group: <https://acs.cs.ut.ee/>



Who are you?



This course

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



Course timeline

16 weeks

- Lecture: will be published by every Monday 23:59
- Practice: Thursdays 14:15–16:00 (Narva mnt 18-1019)

6 ECTS – 10 hours weekly

- 2 hours for lectures
- 8 hours on homework (may vary)

Grading

- Homework every week
- Homeworks give maximum 70% of the final grade
- Deadlines are strict!
 - Homework deadline – beginning of the next lecture
 - Late submissions get 50% of the grade
 - Homeworks submitted later than 1 week after the deadline are not accepted!
- Exam gives another 30% of the final grade
 - Should be easy if you follow the lectures

Homework submission

- Homeworks must be implemented in Python 3
 - Test environment: Ubuntu 19.10, Python 3.6.x
 - Python packages from Ubuntu package repository (not pip)
- Create a private Bitbucket repository and grant me 'read' privileges:
<https://bitbucket.org/appcrypto/2020/src/master/setup/>
- Add your repository to the course grading page at
<https://cybersec.ee/appcrypto2020/>
- Homework templates will be published at course repository:
<https://bitbucket.org/appcrypto/2020/>
- Feedback will be given using code comment feature
- Teaching assistance over e-mail not available
- Do not look on homework solutions of others!
 - Plagiarism cases will be handled in accordance with UT Plagiarism Policy

Academic fraud

- It is an academic fraud to collaborate with other people on work that is required to be completed and submitted individually.
- The homeworks in Applied Cryptography course are required to be completed and submitted individually!
- You can help your peers to learn by explaining concepts, but don't provide them with answers or your own work!
 - If you don't see the borders – work alone.
- Copying code samples from internet resources (e.g., stackoverflow.com) may be considered plagiarism:
 - the most basic building blocks may be OK
 - combination (composition) of building blocks is NOT OK
 - If you don't see the borders – limit yourself to Python API documentation.

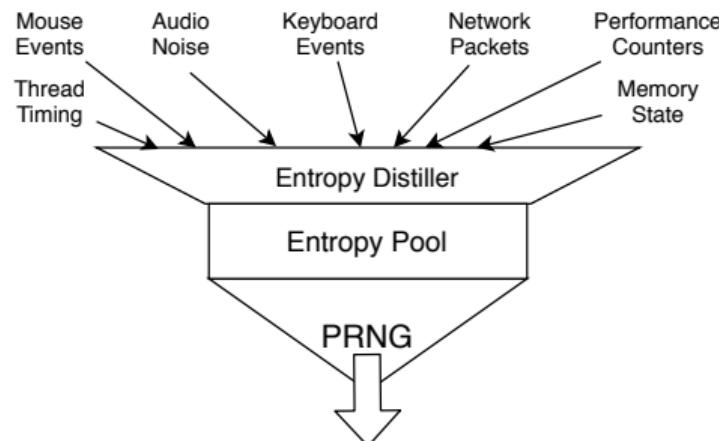
Randomness

- What is a random sequence?
 - Sequence of numbers that does not follow any deterministic pattern
 - None of the numbers can be predicted based on the previous numbers
 - Has no description shorter than itself
 - Sequence of bits that cannot be compressed
- Where do we need randomness in the real life?
- Why do we need randomness in crypto?
 - For keys, passwords, nonces, etc.
- 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. The output is determined by the *seed* value.

Linux /dev/urandom implementation:



- Knowing some part of the input does not allow to predict anything about the output
- PRNG is used when true-RNG is not available
- Can be used to “extend” randomness
- Entropy of the output depends on the entropy of the input

Randomness

- Can we tell whether some sequence is random?

...41592653589...

3.141592653589793...

...000000.....

- Statistical randomness tests
 - Able to “prove” non-randomness

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

ASCII Table

0	<NUL>	32	<SPC>	64	@	96	'	128	Ä	160	†	192	¿	224	‡
1	<SOH>	33	!	65	A	97	a	129	Å	161	º	193	i	225	.
2	<STX>	34	"	66	B	98	b	130	Ç	162	¢	194	¬	226	,
3	<ETX>	35	#	67	C	99	c	131	É	163	£	195	✓	227	"
4	<EOT>	36	\$	68	D	100	d	132	Ñ	164	§	196	f	228	%
5	<ENQ>	37	%	69	E	101	e	133	Ö	165	•	197	≈	229	Â
6	<ACK>	38	&	70	F	102	f	134	Ü	166	¶	198	Δ	230	Ê
7	<BEL>	39	'	71	G	103	g	135	á	167	ß	199	«	231	Á
8	<BS>	40	(72	H	104	h	136	à	168	®	200	»	232	È
9	<TAB>	41)	73	I	105	i	137	â	169	©	201	...	233	É
10	<LF>	42	*	74	J	106	j	138	ã	170	™	202		234	Í
11	<VT>	43	+	75	K	107	k	139	ã	171	‘	203	À	235	Í
12	<FF>	44	,	76	L	108	l	140	å	172	”	204	Ã	236	Í
13	<CR>	45	-	77	M	109	m	141	ç	173	≠	205	Ö	237	Í
14	<SO>	46	.	78	N	110	n	142	é	174	Æ	206	Œ	238	Ó
15	<SI>	47	/	79	O	111	o	143	è	175	Ø	207	œ	239	Ô
16	<DLE>	48	0	80	P	112	p	144	ê	176	∞	208	-	240	apple
17	<DC1>	49	1	81	Q	113	q	145	ë	177	±	209	-	241	Ò
18	<DC2>	50	2	82	R	114	r	146	í	178	≤	210	"	242	Ú
19	<DC3>	51	3	83	S	115	s	147	ì	179	≥	211	"	243	Ú
20	<DC4>	52	4	84	T	116	t	148	î	180	¥	212	`	244	Ú
21	<NAK>	53	5	85	U	117	u	149	ï	181	µ	213	‘	245	ı
22	<SYN>	54	6	86	V	118	v	150	ñ	182	ø	214	÷	246	,
23	<ETB>	55	7	87	W	119	w	151	ó	183	Σ	215	◊	247	-
24	<CAN>	56	8	88	X	120	x	152	ò	184	∏	216	ÿ	248	
25		57	9	89	Y	121	y	153	ô	185	ñ	217	Ý	249	ˇ
26	<SUB>	58	:	90	Z	122	z	154	ö	186	ƒ	218	/	250	.
27	<ESC>	59	;	91	[123	{	155	õ	187	a	219	€	251	°
28	<FS>	60	<	92	\	124		156	ú	188	ø	220	<	252	,
29	<GS>	61	=	93]	125	}	157	ù	189	Ω	221	>	253	"
30	<RS>	62	>	94	^	126	~	158	û	190	æ	222	fi	254	,
31	<US>	63	?	95	_	127		159	ü	191	ø	223	fl	255	ˇ

Hexadecimal (Base16) encoding

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

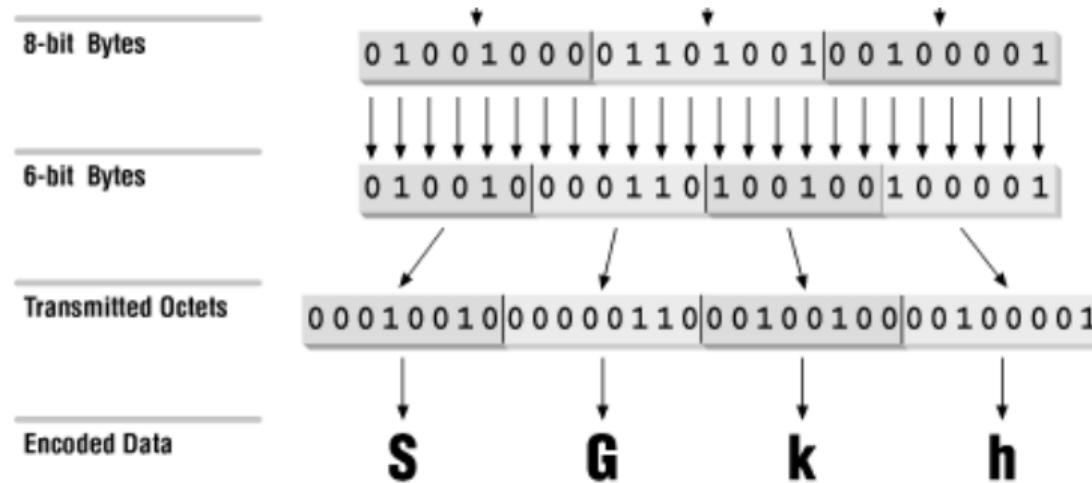
- One hex symbol represents 4 bits
- Two hex symbols needed to represent a byte

2E = 0010 1110

Base64 encoding

bn+ITbj/TRwcSAwT8CZnFZN0me5/AGdFIGNLBPPo7Nc07T6XTpsTw0Q
xnM++9xJXKkEEcaEn2Vo9MiAVPVUR5PsFGKZbL7coPRdHD058RokCF4
aizWv6+Dqg0lsXsmXliWusn0Q==

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



Bitwise operations

AND:

- extract part of bit string

0 0 1 1 1 1 0 0		
0 0 0 0 0 1 1 0	(bit mask)	>>> 60 & 6
-----		4
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
-----		62
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
-----		58
0 0 1 1 1 0 1 0	(XOR)	

Shift:

- shift and pad with 0

0 0 1 1 1 1 0 0		>>> 60 >> 2
-----		15
0 0 0 0 1 1 1 1	(right shift by two)	

Bitwise operation: AND

- Extract bits we are interested in

Example:

0	0	1	1	1	1	0	0
0	0	0	0	0	1	1	0

0	0	0	0	0	1	0	0

(bit mask)

(AND)

Python:

```
>>> 60 & 6  
4
```

Bitwise operation: 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:

```
>>> 60 | 6  
62
```

Bitwise operation: 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 operation: Shift

- Shift (right or left) and pad with zeros

Example:

0 0 1 1 1 0 0

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

Python:

```
>>> 60 >> 2
```

```
15
```

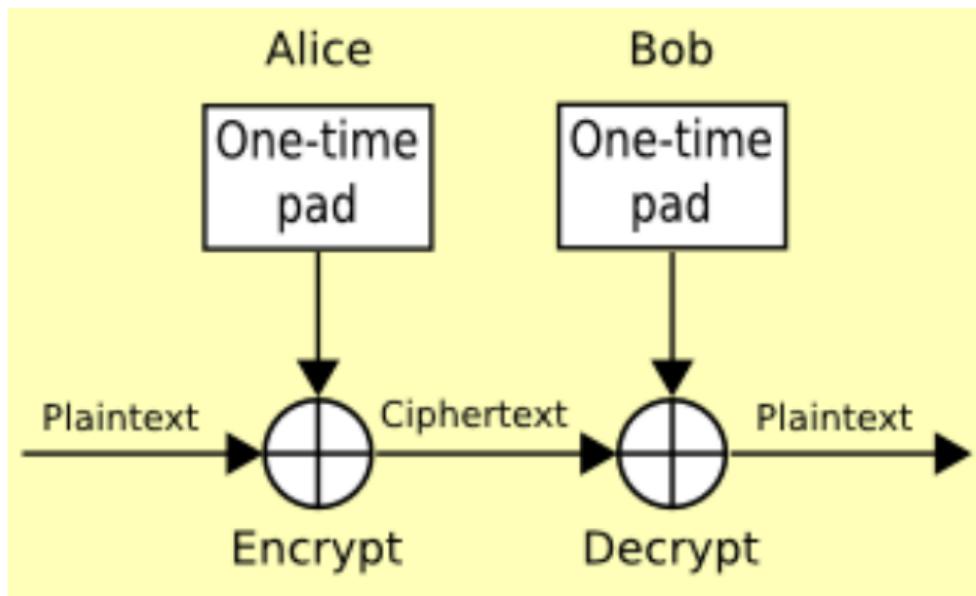
```
>>> 15 << 1
```

```
30
```

- Fast multiplication and division by 2

One-Time Pad (OTP)

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



One-Time Pad (OTP)

Information-theoretically secure (unbreakable), if:

- Key (one-time pad) is truly random
- Key is never reused

$$\text{plaintext1} \oplus \text{key} = \text{ciphertext1}$$

$$\text{plaintext2} \oplus \text{key} = \text{ciphertext2} \oplus \text{plaintext2} = \text{key}$$

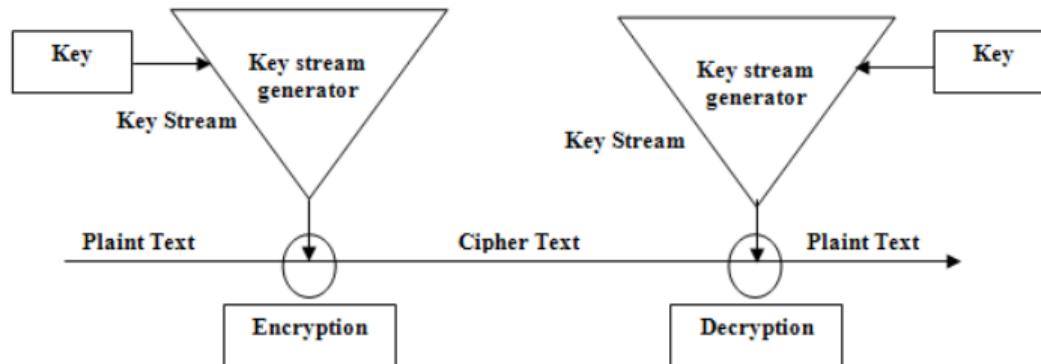
$$\text{key} \oplus \text{ciphertext1} = \text{plaintext1}$$



- Not used in practice

Stream cipher

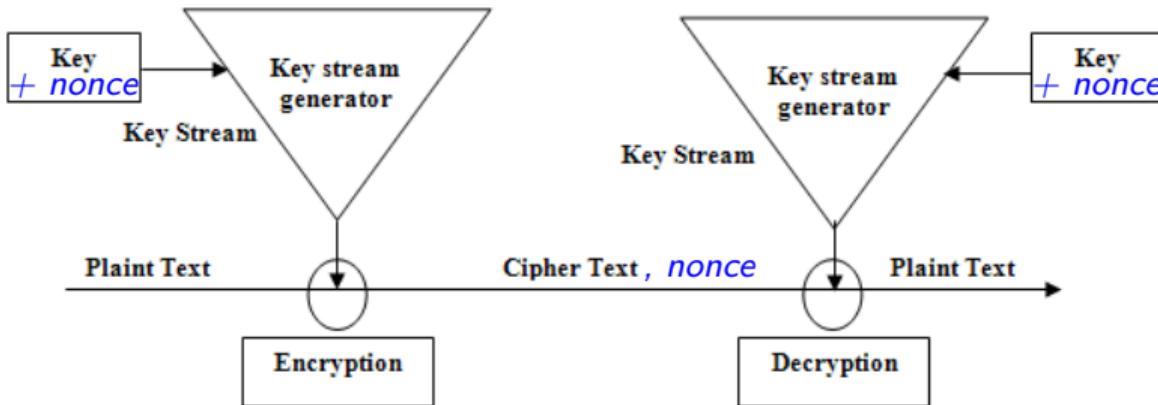
- Key generation: a small key “seeds” the PRNG
- Encryption operation: XOR (\oplus) the plaintext with the key
- Decryption operation: XOR (\oplus) the ciphertext with the key



- Stream ciphers differ by the PRNG used
- Why is it less secure than one-time pad?
- Encryption on its own does not provide integrity!
- **The same keystream must never be reused!**

Stream cipher

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



- The same *nonce* must never be reused!
- How to generate *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) – 3p

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 “01/otp.py” to your repository:

```
$ git add 01/otp.py  
$ git commit -m "homework 01 solution" 01/otp.py  
$ git push
```

Task: Template

```
#!/usr/bin/env python3
import os, sys      # do not use any other imports/libraries
# took x.y hours (please specify here how much time your solution required)

def bn(b):
    # b - bytes to encode as integer
    # your implementation here
    return i

def nb(i, length):
    # i - integer to encode as bytes
    # length - specifies in how many bytes the number should be encoded
    # your implementation here
    b = b'
    return b

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()
```

Python 3 str and bytes data objects

str object stores Unicode characters:

```
>>> s = 'Főő'  
>>> type(s), len(s)  
(<class 'str'>, 3)  
>>> s[0], s[1], s[2]  
('F', 'ő', 'ő')
```

bytes object stores bytes:

```
>>> bytes([97,98])  
b'ab'  
>>> b = b'F\xc5\x8d\xc5\x8d'  
>>> b = s.encode()  
>>> type(b), len(b)  
(<class 'bytes'>, 5)  
>>> b[0], b[1], b[2], b[3], b[4]  
(70, 197, 141, 197, 141)  
>>> b.decode()  
'Főő'
```

```
>>> import codecs  
>>> codecs.encode(b, 'hex')  
b'46c58dc58d'  
>>> codecs.encode(b, 'base64')  
b'RsWNxY0=\n'  
>>> codecs.encode(b, 'base64').decode()  
'RsWNxY0=\n'
```

Python: bytes to integer

```
>>> b = b'abC'  
>>> i = b[0]  
>>> i  
97  
>>> bin(i)  
'0b1100001'  
>>> i = i << 8  
>>> bin(i)  
'0b110000100000000'  
>>> i = i | b[1]  
>>> bin(i)  
'0b110000101100010'  
>>> i = i << 8  
>>> bin(i)  
'0b110000101100010000000000'  
>>> i = i | b[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
- ...

Task: One-Time Pad (OTP)

- Encrypter:
 - Read the plaintext file contents into bytes object (e.g., `b = open('file.txt', 'rb').read()`)
 - Convert plaintext bytes to one big integer
 - Obtain random key the same length as plaintext (use `os.urandom()`)
 - Convert key bytes to one big integer
 - XOR plaintext and key integers (**please, use this approach**)
 - Save the key (one-time pad) and XOR'ed result (ciphertext) to file:
 - Convert ciphertext integer to bytes object
 - Once more: use bitwise operations!
 - Banned: functions: `to_bytes()`, `from_bytes()` and operator `**!`
- Decrypter:
 - Perform the operations in reverse order

Task: Test Case

```
$ echo -n -e "\x85\xce\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/enc file.key fileorig/plain
$ hexdump -C fileorig/plain
00000000  00 00 61 62 43 00      |..abC.|
```

Note that when you convert bytes to integer, you lose the most significant zero bytes.

Please!

- Include information of how much time the tasks took (as a comment at the top of your source code)
- Give feedback about the parts that were hard to grasp or you have an idea for improvement
- Do not waste your time on input validation
- Do not use imports/libraries that are not explicitly allowed
- The output of your solution must byte-by-byte match the format of example output shown on the slides
 - Remove any nonrequired debugging output before committing
 - Unless required, the solution must not create/delete any files
- Commit the (finished) solution to the main branch of your repository with the filename required

Thank you!

MTAT.07.017

Applied Cryptography

Abstract Syntax Notation One (ASN.1)

University of Tartu

Spring 2020

Abstract Syntax Notation One

“ASN.1 is a standard interface description language for defining data structures that can be serialized and deserialized in a cross-platform way. It is broadly used in telecommunications and computer networking, and especially in cryptography.”

Notation to describe *abstract* types and values
Describes *information* – not representation

Similar to XML schema, however:

- ASN.1 is rich with built-in data types
- ASN.1 is not tied to particular encoding mechanism

ASN.1 example

```
-- ASN.1 module
MyQAProtocol DEFINITIONS ::= BEGIN
    MyQuestion ::= SEQUENCE {
        id INTEGER (0..999),
        text UTF8String
    }

    MyAnswer ::= SEQUENCE {
        id INTEGER (0..999),
        text UTF8String
    }
    -- new type defined
END
```

ASN.1 simple types

NULL -- only possible value is Null
BOOLEAN -- True or False
INTEGER -- whole numbers -infinity..+infinity
REAL -- mantissa, base, exponent
OCTET STRING -- values 0x00..0xFF
BIT STRING -- 0-s and 1-s
UTF8String -- UTF-8 characters
NumericString -- [space]0123456789
PrintableString -- printable ASCII chars
IA5String -- ASCII chars 0x00..0x7F
BMPString -- UNICODE BMP code points
UTCTime -- time in form "YYMMDDhhmmssZ"

There are more...

ASN.1 structured types

```
YearInfo ::= SEQUENCE {
    year          INTEGER (0..9999),
    isLeapYear   BOOLEAN
}
```

```
Person ::= SET {
    name        IA5String,
    age         INTEGER,
    female     BOOLEAN
}
```

```
Prize ::= CHOICE {
    car        IA5String,
    cash      INTEGER,
    nothing    NULL
}
```

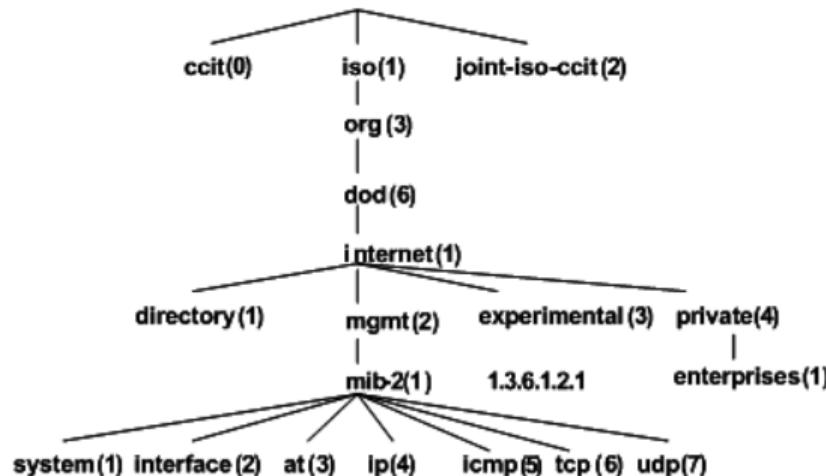
ASN.1 OBJECT IDENTIFIER

Algorithm ::= OBJECT IDENTIFIER

rsa Algorithm ::= {1.2.840.113549.1.1.1}

iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-1(1) rsaEncryption(1)

OID tree:



ASN.1 encodings

```
-- ASN.1 type definition
Question ::= SEQUENCE {
    id INTEGER,
    questionText UTF8String
}
```

How do we encode this structure for transmission?

The standard ASN.1 encoding rules:

- Basic Encoding Rules (BER)
- Distinguished Encoding Rules (DER)
- Packed Encoding Rules (PER)
- XML Encoding Rules (XER)
- JSON Encoding Rules (JER)

XML Encoding Rules (XER)

```
-- ASN.1 type definition
Question ::= SEQUENCE {
    id INTEGER,
    questionText UTF8String
}

<!-- XER-encoded object -->
<Question>
    <id>42</id>
    <questionText>Why is it so?</questionText>
</Question>
```

- Human readable
- Inefficient encoding
- Canonicalization needed

Distinguished Encoding Rules (DER)

- Efficient encoding
- A value can be encoded only in a single way
- Data is encoded as type-length-value (TLV) element:

```
message UTF8String :: = "Hello"
```

Type: UTF8String

Length: 5 bytes

Value: "Hello"

DER encoded:

```
[0x0c] [0x05] [0x48 0x65 0x6c 0x6c 0x6f] ...
```

```
$ echo -e -n "\x0c\x05Hello" > hello.der
$ sudo apt install dumpasn1
$ dumpasn1 hello.der
0    5: UTF8String 'Hello'
```



Task: ASN.1 DER encoder – 10p

Implement ASN.1 DER encoder that can encode subset of ASN.1 types by implementing these functions:

```
def asn1_boolean(bool):  
def asn1_integer(i):  
def asn1_bitstring(bitstr):  
def asn1_octetstring(octets):  
def asn1_null():  
def asn1_objectidentifier(oid):  
def asn1_sequence(der):  
def asn1_set(der):  
def asn1_printablestring(string):  
def asn1_utctime(time):  
def asn1_tag_explicit(der, tag):  
def asn1_len(content): <-- helper function
```

Task: ASN.1 DER encoder

And encodes this artificial ASN.1 structure (test case):

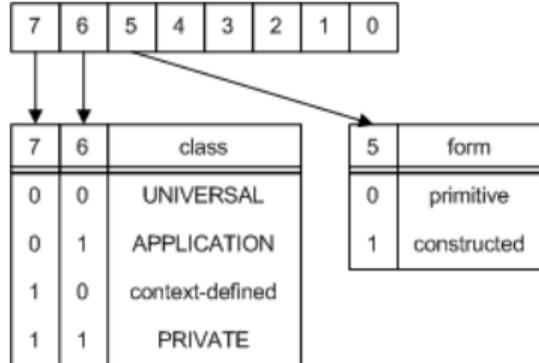
```
$ dumpasn1 asn1.der
0 114: [0] { explicit tags
2 112:   SEQUENCE {
4 16:     SET {
6 1:       INTEGER 5
9 4:       [2] {
11 2:         INTEGER 200
:
15 5:       [11] {
17 3:         INTEGER 65407
:
15 5:       }
22 1:     BOOLEAN TRUE
25 2:     BIT STRING 5 unused bits
:
: '011'B
:
: Error: Spurious zero bits in bitstring.
29 51:   OCTET STRING
:
: 00 01 02 02 02 02 02 02 02 02 02 02 02 02
:
: 02 02 02 02 02 02 02 02 02 02 02 02 02 02
:
: 02 02 02 02 02 02 02 02 02 02 02 02 02 02
:
: 02 02 02
82 0:   NULL
84 7:   OBJECT IDENTIFIER '1 2 840 113549 1'
93 6:   PrintableString 'hello.'
101 13:   UTCTime 23/02/2015 01:09:00 GMT
:
: }
```

NB! dumpasn1 fails to decode negative integers and outputs bitstrings in reverse order.

```
0 warnings, 1 error.
asn1_tag_explicit(asn1_sequence(asn1_set(...)+asn1_boolean(true)+...), 0)

$ ./asn1_encoder.py asn1.der
```

Type-Length-Value: Type



Universal tags (Bits 4,3,2,1,0):

00001 (1) - BOOLEAN
00010 (2) - INTEGER
00011 (3) - BIT STRING
00100 (4) - OCTET STRING
00101 (5) - NULL
00110 (6) - OBJECT IDENTIFIER
01010 (10) - ENUMERATED
01100 (12) - UTF8String
10000 (16) - SEQUENCE
10001 (17) - SET
10011 (19) - PrintableString
10111 (23) - UTCTime
...

0x0c – 00 0 01100 (universal, primitive, UTF8String)

A Layman's Guide to a Subset of ASN.1, BER, and DER:

<http://luca.ntop.org/Teaching/Appunti/asn1.html>

ASN.1 encoding rules: Specification of BER, CER and DER:

<http://www.itu.int/ITU-T/studygroups/com17/languages/X.690-0207.pdf>

Type-Length-Value: Length

- `asn1_len(value_bytes)`: – 1.5p
 - If the number of value bytes < 128 then length byte encodes the number of bytes in the value
 - Else the most significant bit of the first length byte is set to 1 and the remaining 7 bits encode the number of length bytes that follow
 - The following length bytes encode the number of value bytes (use `nb()` without the `length` parameter)

Example:

Length 126: 01111110

Length 127: 01111111

Length 128: 10000001 10000000

Length 1027: 10000010 00000100 00000011

$$(4 \ll 8) \mid 3$$

$$= 1027$$

ASN.1 DER encoding

- `asn1_boolean(bool): - 0p`
 - Encodes boolean value
 - Universal, primitive, tag 1 (00 0 00001)
 - Value byte contains 0x00 for FALSE and 0xff for TRUE
- `asn1_integer(int): - 1p`
 - Encodes integer (only positive integers must be supported)
 - Universal, primitive, tag 2
 - Two's complement integer encoding:
 - Convert integer to bytestring using `nb()` without the `length` parameter
 - Integer value 0 is encoded as zero byte (not empty bytestring)
 - If the most significant bit of MSB for a positive integer is 1 then prepend zero (0x00) byte

INTEGER: 140

Type	Length	Padding	Integer
00000010	00000010	00000000	10001100

250,000 Estonian ID cards could be faulty

<https://news.err.ee/116849/250-000-estonian-id-cards-could-be-faulty>



A **coding mistake by the Certification Center**, the company behind the software of ID cards, means 250,000 ID cards could cause problems for users in the future.

The problem concerns Estonian ID cards issued between September 2014 and September 2015, and if not fixed, will mean users are unable to use ID cards with the new version of the Google Chrome browser.

"We let a fault slip through our software development process," Certification Center head Kalev Pihl told Postimees. The problem surfaced when Google worked out a **new version of Chrome, which has more detailed checks**.

ASN.1 DER encoding

- `asn1_bitstring(str_of_bits)`: – 2p
 - Encodes an arbitrary bitstring value (e.g. '010101')
 - Universal, primitive, tag 3
 - Bitstring is right-padded with zero bits to form full byte string
 - First byte of value bytes encodes number of padding bits

BIT STRING: 010101

Type	Length	Padding-length	Padded-bitstring
DER:	00000011	00000010	00000010 01010100

- `asn1_octetstring(bytes)`: – 0.2p
 - Encodes an arbitrary string of octets
 - Universal, primitive, tag 4
- `asn1_null()`: – 0.2p
 - Denotes a null value
 - Universal, primitive, tag 5
 - No value bytes

ASN.1 DER encoding

- `asn1_objectidentifier(list_of_oid_components)`: – 3p
 - An object identifier, which is a sequence of integer components
 - Universal, primitive, tag 6
 - The first value byte has value: $40 * \text{comp1} + \text{comp2}$
 - The following value bytes encode comp3, comp4, ...
 - Each component is encoded using 7 rightmost bits of the bytes
 - Each byte's leftmost bit (except for the last) is 1

Example:

OBJECT IDENTIFIER: 1.2.840 (US (ANSII))					
0000	0110	0000	0011	0010	1010
Type	Length	$40*1+2$			
		6			
		72			
		$(6 << 7) 72 = 840$			

- `asn1_sequence(der_bytes)`: – 0.2p
 - Encodes ordered collection of one or more types
 - Universal, **constructed**, tag 16
 - Value bytes contain DER encoded data

ASN.1 DER encoding

- `asn1_set(der_bytes)`: – 0.2p
 - Encodes unordered collection of one or more types
 - Universal, **constructed**, tag 17
 - Value bytes contain DER encoded data
- `asn1_printablestring(bytes)`: – 0.2p
 - Encodes an arbitrary string of printable characters [a-zA-Z0-9' ()+,.-/:=?]
 - Universal, primitive, tag 19
 - Value bytes contain printable string characters
- `asn1_utctime(date_str)`: – 0.2p
 - Encodes "coordinated universal time" (GMT – Greenwich Mean Time)
 - Universal, primitive, tag 23
 - Value bytes contain string representation of time in form "YYMMDDhhmmssZ"

ASN.1 Tagging

ASN.1 notation may be ambiguous:

```
Ambiguous ::= SEQUENCE {  
    val1 INTEGER OPTIONAL,  
    val2 INTEGER OPTIONAL  
}
```

Unable to decode if encoded structure contains only one value!

Fix is to tag the values:

```
unambiguous ::= SEQUENCE {  
    val1 [1] IMPLICIT INTEGER OPTIONAL,  
    val2 [2] EXPLICIT INTEGER OPTIONAL  
}
```

- IMPLICIT overwrites the existing type byte of TLV
- EXPLICIT prepends type and length bytes (encapsulates original TLV)

ASN.1 DER encoding

- `asn1_tag_explicit(der, tag):`
 - Tags/encapsulates any data type
 - **Context-defined, constructed**, tag n (5 rightmost bits)
 - No need to implement support for tag > 30
 - Value bytes contain DER-encoded data

```
>>> asn1 = asn1_tag_explicit(asn1_sequence(asn1_null()), 5)
>>> open('asn1', 'wb').write(asn1)
```

```
$ dumpasn1 asn1
0  4: [5] {
2  2:   SEQUENCE {
4  0:     NULL
      :
      }
    :
```

Banned functions

Your solution should **not** use:

- functions: `bin()`, `int()`, `hex()`, `str()`, `bytarray()`, `divmod()`
- exponentiation: `**`, `pow()`
- division and modulus: `/`, `%` (unless needed for computing bitstring padding size)

Use bitwise operations as much as possible!

For example, to convert `str` containing bit representation to `int`:

```
i = 0
for bit in '010001':
    i<<=1
    if bit=='1':
        i |= 1
```

As a general rule for all homeworks: encoding values to “bin” and “hex” representation is allowed only for printing out non-printable binary data.

MTAT.07.017

Applied Cryptography

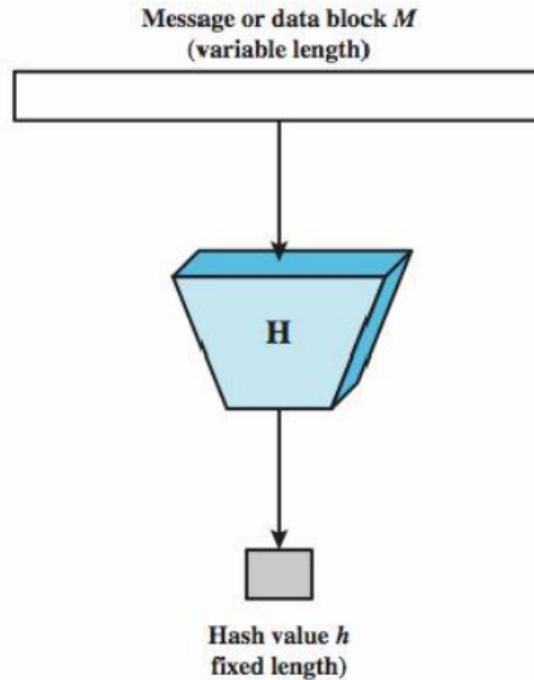
Hash functions and HMAC

University of Tartu

Spring 2020

Hash function

A hash function is a function that takes an arbitrary block of data and returns a fixed-size unique bit string representation.



Related terms: hash, message digest, fingerprint, checksum

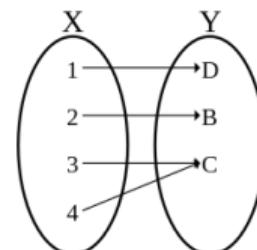
Cryptographic hash function

Properties:

- Easy to compute hash value (fast)
- Hard to restore message from hash (one-way)
- Hard to find messages with the same hash (collision resistant)
- Similar messages have a very different hashes (avalanche effect)

Attacks:

- Collision attack:
 $\text{hash}(x) = \text{hash}(y)$ | find any x and y such that $x \neq y$
- (First) Preimage attack:
 $\text{hash}(x) = h$ | given h , find any x
- Second preimage attack:
 $\text{hash}(x) = \text{hash}(y)$ | given x and $\text{hash}(x)$, find $y \neq x$



Security Level

“In cryptography, a **brute-force attack**, or exhaustive key search [...] might be used when it is not possible to take advantage of other weaknesses in an encryption system. It consists of systematically checking all possible keys or passwords until the correct one is found.”

https://en.wikipedia.org/wiki/Brute-force_attack

E.g.: If a cryptosystem which has 56-bit key can be brute-forced using 2^{56} operations¹ then the cryptosystem has security level of 56 bits.

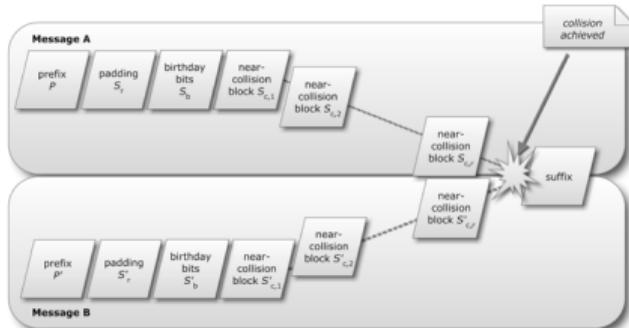
- 2^{128} operations infeasible
- 2^{80} become feasible

Note, 2^{81} operations take twice the time of 2^{80} (2^{128} vs 2^{256}).

¹The term “operation” is not defined.

Cryptographic hash functions

- MD5 – 128-bit output
 - collision attack in $2^{24.1}$ (brute-force 2^{64})
 - chosen-prefix collision attack in 2^{39}



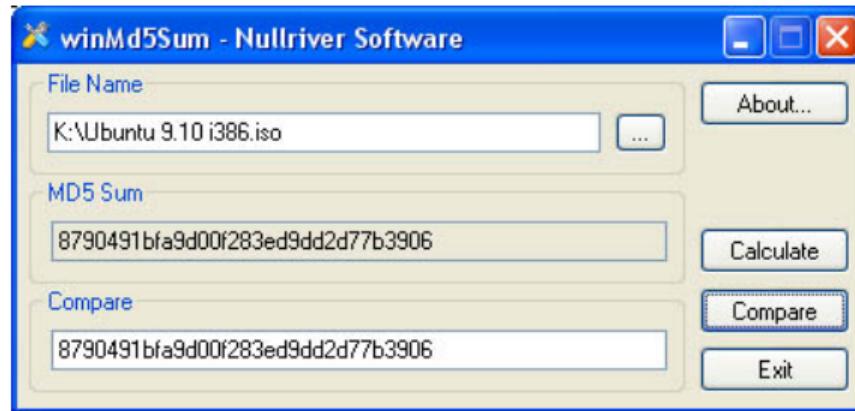
```
C:\TEMP> md5sum hello.exe
cdc47d670159eef60916ca03a9d4a007
C:\TEMP> .\hello.exe
Hello, world!
C:\TEMP> md5sum erase.exe
cdc47d670159eef60916ca03a9d4a007
C:\TEMP> .\erase.exe
This program is evil!!!
Erasing hard drive...1Gb...2Gb... just kidding!
Nothing was erased.

(press enter to quit)
C:\TEMP>
```

- preimage and second preimage attack in $2^{123.4}$ (brute-force 2^{128})
- SHA-1 – 160-bit output
 - theoretical practical collision attack in $2^{63.1}$ (brute-force 2^{80})
 - chosen-prefix collision attack in $2^{63.4}$
- SHA-256 – 256-bit output
- SHA-512 – 512-bit output
- SHA-3 – 224/256/384/512-bit output

Data identification and integrity verification

- Integrity and authenticity of distributed files



- Distribution of MS Windows updates
- Disk imaging in digital forensics
- Remote file comparison (rsync)

Server-side password storage

LAW & DISORDER / CIVILIZATION & DISCONTENTS

Sony hacked yet again, plaintext passwords, e-mails, DOB posted

The hackers of Lulz Security have broken into yet more Sony websites, this ...

by Peter Bright - June 3 2011, 4:06am EEST



```
Hello , Iam Idahc a Lebanese Hacker  
I was Bored and I play the game of the year : "hacker vs Sony"  
I hacked little database of 120 user with an sql injection.....  
site:http://apps.pro.sony.eu/  
  
username password mobile office email website  
[REDACTED] +34 [REDACTED] 18 go [REDACTED] es csie.unav  
[REDACTED] iscul +44 7818 04 [REDACTED] dam@3d [REDACTED] ik  
[REDACTED] .-JafarAbd ourfar [REDACTED] 145899 [REDACTED] our@pendarfi  
[REDACTED] sterChamp manter [REDACTED] 1 1522 [REDACTED] ster@ingenio  
[REDACTED] arth +44 7887 83 [REDACTED] 44 788 [REDACTED] techniche.com  
[REDACTED] o alro +44 075927 [REDACTED] 44 (0) [REDACTED] he.com  
[REDACTED] bern +34 630 163 [REDACTED] 91 63 [REDACTED] yahoo.es  
[REDACTED] marie c68763c0c7 [REDACTED] 465cf [REDACTED] jr www.argie.  
[REDACTED] the Argthe +30694 [REDACTED] +3069 [REDACTED] :lde@scarlet.  
[REDACTED] van vanbar +32 47 [REDACTED] 5 +32 [REDACTED] .iisi@virgin  
[REDACTED] iloc locbon +44 (0 [REDACTED] 66545 [REDACTED] iit.com  
[REDACTED] mal malbru +33 6 [REDACTED] 8 +33 [REDACTED]  
[REDACTED] oJohansen senato [REDACTED] 91355 [REDACTED]
```

- Solution – store password hashes in database
 - Compare received plaintext password with hash from db

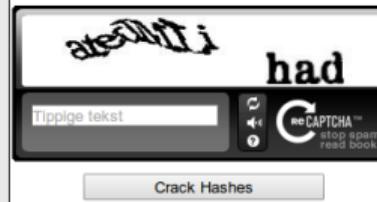
Server-side password storage

```
mysql> SELECT username, password FROM users;
+-----+-----+
| username | password          |
+-----+-----+
| jeff     | b1b3773a05c0ed0176787a4f1574ff0075f7521e |
| katrin   | 2730f2c29354932611d328cfffc9f01e10328ec |
| mike     | e72e941812b920c908bba17798d5e27ebf627912 |
+-----+-----+
```

Free Password Hash Cracker

Enter up to 10 hashes:

```
b1b3773a05c0ed0176787a4f1574ff0075f7521e  
2730f2c29354932611d328cfffc9f01e10328ec  
e72e941812b920c908bba17798d5e27ebf627912
```



Supports: LM, NTLM, md2, md4, md5, md5(md5), md5-half, sha1, sha1(sh1_bin()), sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+

Hash	Type	Result
b1b3773a05c0ed0176787a4f1574ff0075f7521e	sha1	qwerty
2730f2c29354932611d328cfffc9f01e10328ec	sha1	hillary999
e72e941812b920c908bba17798d5e27ebf627912	Unknown	Not Found

hillary99

- Solution – add user-specific salt to the password

```
db_salt = os.urandom(8).encode('hex')
db_password = hashlib.sha1(password + db_salt).hexdigest()
```

Server-side password storage

```
mysql> SELECT username, password, salt FROM users;
```

username	password	salt
jeff	0771580376c18f7faeae9de565ff663eef8c5cc	7d3a5ccd7fc28aa9
katrin	9c70ccbf02e5b8be46ebcd149326d5d375895187	df9372246bfcfd8d0
mike	622cd81265db68c3b2616400f312c2a7096f5848	9a73764e2bf40db8

Free Password Hash Cracker

Enter up to 10 hashes:

```
0771580376c18f7faeae9de565ff663eef8c5cc  
9c70ccbf02e5b8be46ebcd149326d5d375895187  
622cd81265db68c3b2616400f312c2a7096f5848
```



Supports: LM, NTLM, md2, md4, md5, md5(md5), md5-half, sha1, sha1(sh1_bin()), sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+

Hash	Type	Result
0771580376c18f7faeae9de565ff663eef8c5cc	Unknown	Not Found
9c70ccbf02e5b8be46ebcd149326d5d375895187	Unknown	Not Found
622cd81265db68c3b2616400f312c2a7096f5848	Unknown	Not Found

- No benefit in building password specific lookup table
- Even the same passwords will have different hash

Server-side password storage

- Brute-force cracking still possible:

```
>>> hashlib.sha1('qwerty'+'7d3a5ccd7fc28aa9').hexdigest()  
'0771580376c18f7faeae9de565ff663eff8c5cc'
```

MD5	23,070.7 M/s
SHA-1	7,973.8 M/s
SHA-256	3,110.2 M/s
SHA-512	267.1 M/s
NTLM	44,035.3 M/s
DES	185.1 M/s
WPA/WPA2	348.0 k/s

Table: GPU speed

- Slow down brute-force by arbitrary factor using iterated hash: $h(h(h(h(x))))$
 - Trade-off: performance vs cost of brute-forcing
 - Goal is to increase asymmetry
- Recommended to use: bcrypt, scrypt (RAM-intensive)

Server-side password storage

Task: Calculate the security level of password hashes stored in a company X database. Company's X password security policy requires password to be exactly 7 lowercase letters.

- Number of letters in English alphabet: 26
- Hash operations required:
 - to brute-force 1-letter password: 26
 - to brute-force 2-letter password: 26×26
 - to brute-force 3-letter password: $26 \times 26 \times 26$
 - to brute-force 7-letter password: 26^7

$$26^7 = 8031810176 \approx 2^{32}$$

If using 2000 iterations:

$$2^{32} \times 2000 = 2^{32} \times 2^{11} \approx 2^{43}$$

Password storage has 43-bit security level.²

²Assuming user chooses letters randomly.

Commitment scheme

Here is the proof of my clairvoyant powers – next U.S. president will be
 $\text{SHA256}(x) = \text{d99d0b129d5864e1813438a885034452...}$

- Binding – due to collision resistance of SHA256
- Hiding – due to one-wayness of SHA256

```
>>> for candidate in [b'Joe Biden', b'Bernie Sanders', b'Donald Trump']:  
...     print(hashlib.sha256(candidate).hexdigest(), candidate)  
...  
d99d0b129d5864e1813438a885034452... Joe Biden  
d181f00b6eb6ae128e950e8a2bc39a51... Bernie Sanders  
e4f2e1f0e2ae4d3ce7018cf3b4f3577c... Donald Trump
```

- Improve hiding property by adding randomness

```
>>> prediction = b'Joe Biden' + os.urandom(16)  
>>> hashlib.sha256(prediction).hexdigest()  
'9b8ca016ea530921e2b7bd9046424441...'  
>>> prediction  
b'Joe Biden|\xdeg\x14\xb0\xb1h\x88\x83<%u\xa6x\xfe\xc5'
```

- Function must be secure against chosen-prefix collision attack

Coin flipping over phone

How to generate random number from 0 to 99 over phone:

1. Alice: "my commitment value ($\text{SHA256}(x)$) is
108c995b953c8a35561103e2014cf828..."
2. Bob: "my value is 84"
3. Alice: "my value was 65"
4. Bob checks if $\text{SHA256}("65") = "108c995b953c8a35561103e2014cf828..."$

Random number generated:

$$65 + 84 = 149 \bmod 100 = 49$$

Hash-based PRNG

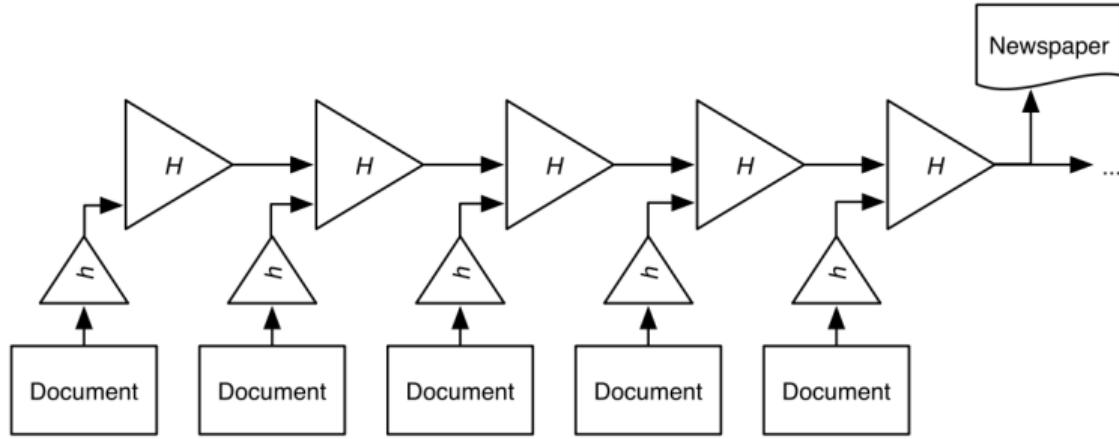
sha1(seed)	sha1(seed + "0")
sha1(sha1(seed))	sha1(seed + "1")
sha1(sha1(sha1(seed))))	sha1(seed + "2")
sha1(sha1(sha1(sha1(seed)))))	sha1(seed + "3")
...	...

```
import hashlib
def hash_prng(seed):
    i = 0
    while True:
        print(hashlib.sha1(seed + str(i).encode()).hexdigest())
        i += 1

hash_prng(b'fookey')
```

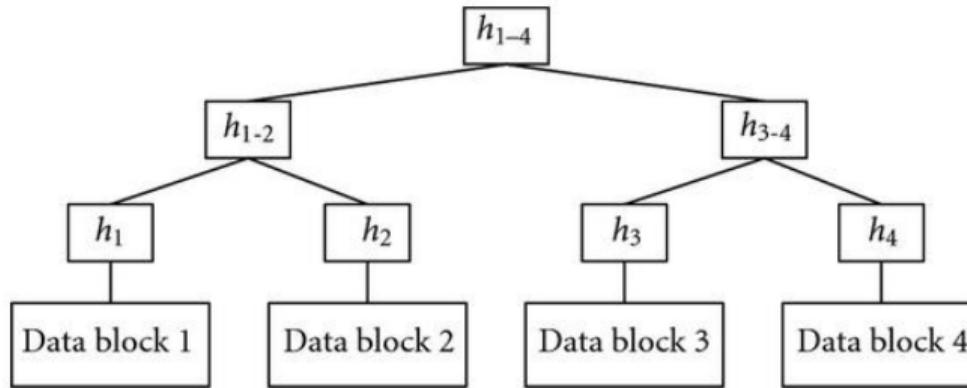
- Standardized construction – Hash-DRBG (NIST SP 800-90)
- If we have a PRNG we can build a stream cipher

Hash chain



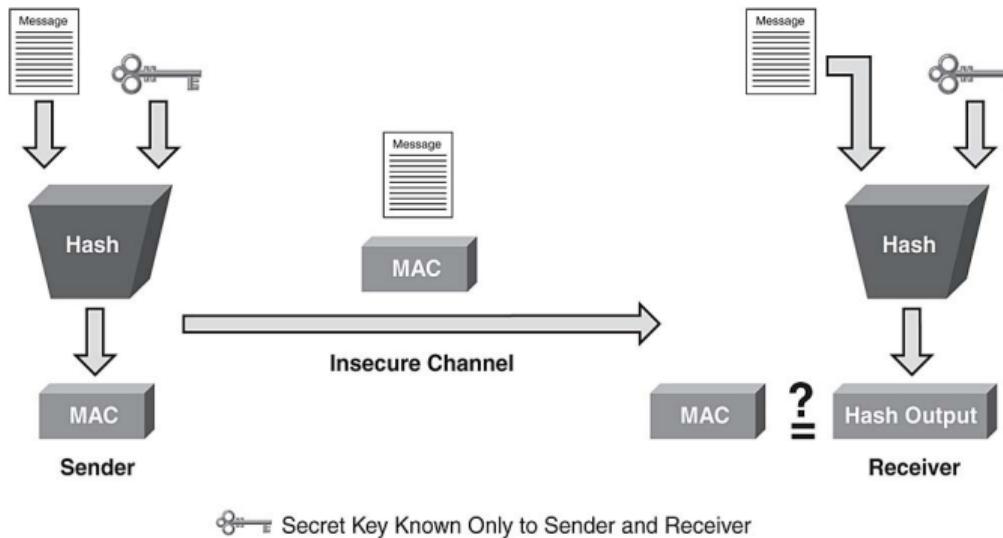
- Linking-based time-stamping
- Also known as a blockchain

Hash tree (Merkle tree)



- Easy to prove that a node belongs to the tree
- To prove that “Data block 3” is part of the tree h_{1-4} :
 - “Data block 3”
 - “ h_4 ”
 - “ h_{1-2} ”
- Used in BitTorrent to identify downloads

HMAC: Hash-based Message Authentication Code



- A valid MAC cannot be produced without knowing the key
- Naive implementation `hash(key + message)` is vulnerable!
 - Safe to use HMAC construction (RFC 2104)
- MAC does not guarantee freshness of the message
- Can we use MAC as a digital signature?

Questions

- What are the properties of hash function?
- What attacks cryptographic hash function must resist?
- What does the size of hash function output influence?
- What is a “security level” in cryptography?
- What is commitment scheme useful for?
- Why it is better to store hashed passwords in db?
- How can we increase the security level of password hashing?
- How can we create encryption scheme from hash function?
- What is HMAC useful for?
- Why using MD5/SHA1 for HMAC is not insecure?

Task: HMAC – 3p

Implement a tool that calculates and verifies the integrity of a file.

```
$ ./hmac.py
Usage:
-verify <filename>
-mac <filename>

$ ./hmac.py -mac somefile
[?] Enter key: secretkey
[+] Calculated HMAC-SHA256: f5e94378cae5a3d0836e145f28807bb7076d28cd22b2481d45f92a904be9d2e8
[+] Writing HMAC DigestInfo to somefile.hmac

$ ./hmac.py -verify somefile
[+] Reading HMAC DigestInfo from somefile.hmac
[+] HMAC-SHA256 digest: f5e94378cae5a3d0836e145f28807bb7076d28cd22b2481d45f92a904be9d2e8
[?] Enter key: secretkey
[+] Calculated HMAC-SHA256: f5e94378cae5a3d0836e145f28807bb7076d28cd22b2481d45f92a904be9d2e8
[+] HMAC verification successful!

$ dumpasn1 somefile.hmac
 0 49: SEQUENCE {
 2 13:   SEQUENCE {
 4  9:     OBJECT IDENTIFIER sha-256 (2 16 840 1 101 3 4 2 1)
15  0:     NULL
      :
 17 32:   OCTET STRING
      :
      : F5 E9 43 78 CA E5 A3 D0 83 6E 14 5F 28 80 7B B7
      :
      : 07 6D 28 CD 22 B2 48 1D 45 F9 2A 90 4B E9 D2 E8
      :
  }
```

DigestInfo

```
DigestInfo ::= SEQUENCE {
    digestAlgorithm    AlgorithmIdentifier,
    digest      OCTET STRING
}

AlgorithmIdentifier ::= SEQUENCE {
    algorithm    OBJECT IDENTIFIER,
    parameters   ANY DEFINED BY algorithm OPTIONAL
}
```

```
$ dumpasn1 hashobject
0 33: SEQUENCE {
2  9:  SEQUENCE {
4  5:    OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
11 0:    NULL
     }
13 20:   OCTET STRING DA 39 A3 EE 5E 6B 4B 0D 32 55 BF ...
     }
```

- Standard structure to store algorithm and calculated digest
- Defined in PKCS#1 v1.5 signature creation (RFC 2313)
- Our hash functions have no parameters (ASN.1 NULL)

Task: HMAC

- Use python's `hashlib` and `hmac` library
e.g., `hmac.new(b'somekey', None, hashlib.md5)`
- Must support hashing of huge files
 - Read file by 512 byte chunks
 - Feed chunks sequentially to `hash.update()`
 - Finally call `hash.digest()`
- HMAC digest must be written to “.hmac” file using `DigestInfo` ASN.1 structure
 - Use your own ASN.1 encoder
 - Please embed your encoder in your solution
 - For decoding use `pyasn1`
- MAC'er must use HMAC-SHA256
- Verifier must support HMAC-MD5, HMAC-SHA1 and HMAC-SHA256 (algorithm must be read from `DigestInfo`)
 - OID for MD5: 1.2.840.113549.2.5
 - OID for SHA1: 1.3.14.3.2.26
 - OID for SHA256: 2.16.840.1.101.3.4.2.1

Task: Test Cases

```
$ echo -e -n "\x01" > file_sha256
$ python hmac.py -mac file_sha256
[?] Enter key: testkey
[+] Calculated HMAC-SHA256: a8be648dd48738b964391a00d4522fe988d10e3d5b2dbf8629a3dcfc0ce93ffd
[+] Writing HMAC DigestInfo to file_sha256.hmac
$ python hmac.py -verify file_sha256
[+] Reading HMAC DigestInfo from file_sha256.hmac
[+] HMAC-SHA256 digest: a8be648dd48738b964391a00d4522fe988d10e3d5b2dbf8629a3dcfc0ce93ffd
[?] Enter key: testkey
[+] Calculated HMAC-SHA256: a8be648dd48738b964391a00d4522fe988d10e3d5b2dbf8629a3dcfc0ce93ffd
[+] HMAC verification successful!

$ wget https://bitbucket.org/appcrypto/2020/raw/master/03/hmac_testcases.tgz
$ tar -zxvf hmac_testcases.tgz

$ python hmac.py -verify file_md5
[+] Reading HMAC DigestInfo from file_md5.hmac
[+] HMAC-MD5 digest: 9e8031ab9d85a5fa0753344bc8c31a2f
[?] Enter key: secretkey
[+] Calculated HMAC-MD5: 9e8031ab9d85a5fa0753344bc8c31a2f
[+] HMAC verification successful!

$ python hmac.py -verify file_sha1
[+] Reading HMAC DigestInfo from file_sha1.hmac
[+] HMAC-SHA1 digest: ebfb4fc1a84d5f9fcbd1b7c8d5d625ac9f5b4c81
[?] Enter key: secretkey
[+] Calculated HMAC-SHA1: ebfb4fc1a84d5f9fcbd1b7c8d5d625ac9f5b4c81
[+] HMAC verification successful!

$ python hmac.py -verify file_sha256
[+] Reading HMAC DigestInfo from file_sha256.hmac
[+] HMAC-SHA256 digest: c40932474350a3f29a9f800e68b6429c64b7526800f8701ae9b4e73db8a3b700
[?] Enter key: secretkey
[+] Calculated HMAC-SHA256: 737f438db779461e6163aa236797099f08b154de6f5741843a549866ae57a5fd
[-] Wrong key or message has been manipulated!
```

pyasn1 library – decoding DER

```
$ sudo apt install python3-pyasn1
$ python3
>>> from pyasn1.codec.der import decoder
>>> der = open('asn1.der', 'rb').read()
>>> decoder.decode(der)
(<Sequence value object at 0x7f2a0cc3cf10 componentType=<NamedTypes object at 0x7f2a0cc128d0
    types > tagSet=<TagSet object at 0x7f2a0cc3cf0 tags 0:32:16-128:32:0>
    subtypeSpec=<ConstraintsIntersection object at 0x7f2a0cc12850>
    sizeSpec=<ConstraintsIntersection object at 0x7f2a0cc12890> payload [<Set value object
    at 0x7f2a0cc3c890 componentType=[...] <UTCTime value object at 0x7f2a0cc3ca50 tagSet
    <TagSet object at 0x7f2a0cc3c590 tags 0:0:23> encoding us-ascii payload
    [150223010900Z]>], '')
>>> decoder.decode(der)[0][0][2]
<Integer value object at 0x7f2a0cc3ced0 tagSet <TagSet object at 0x7f2a0cc3ca90 tags
    0:0:2-128:32:11> payload [65407]>
>>> int(decoder.decode(der)[0][0][2])
65407
>>> decoder.decode(der)[0][0][2].__class__.__name__
'Integer'
• Can't handle large (>3MB) DER encoded structures
• Can't handle DER structures with implicit tagging
```

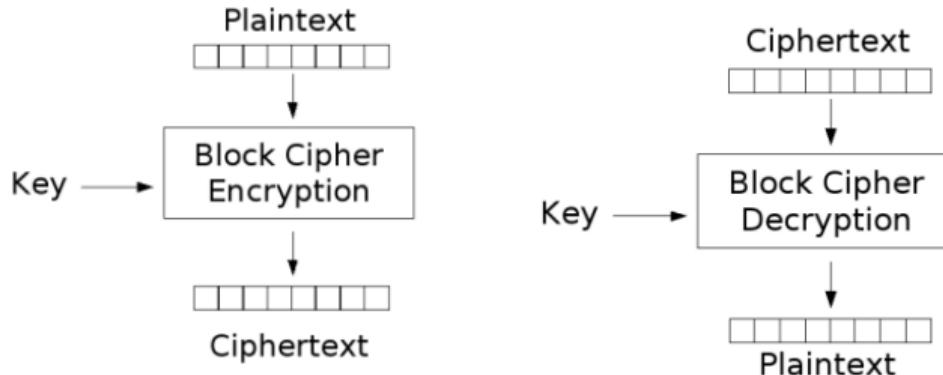
MTAT.07.017
Applied Cryptography

Block Ciphers (AES)

University of Tartu

Spring 2020

Block Ciphers



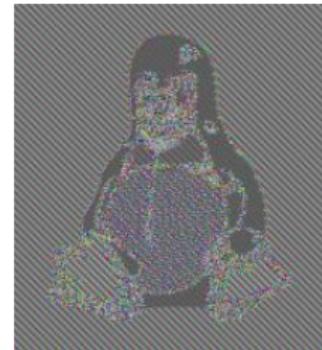
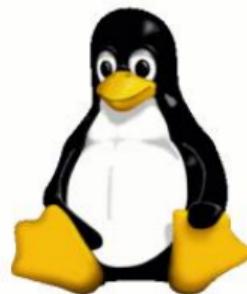
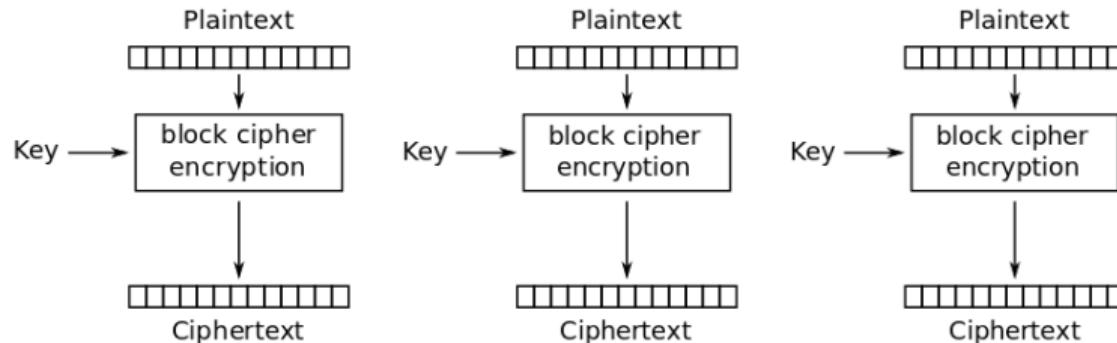
Properties:

- Deterministic
- Without the key plaintext cannot be found
- Valid plaintext-ciphertext pairs do not leak the key
- Diffusion & Confusion

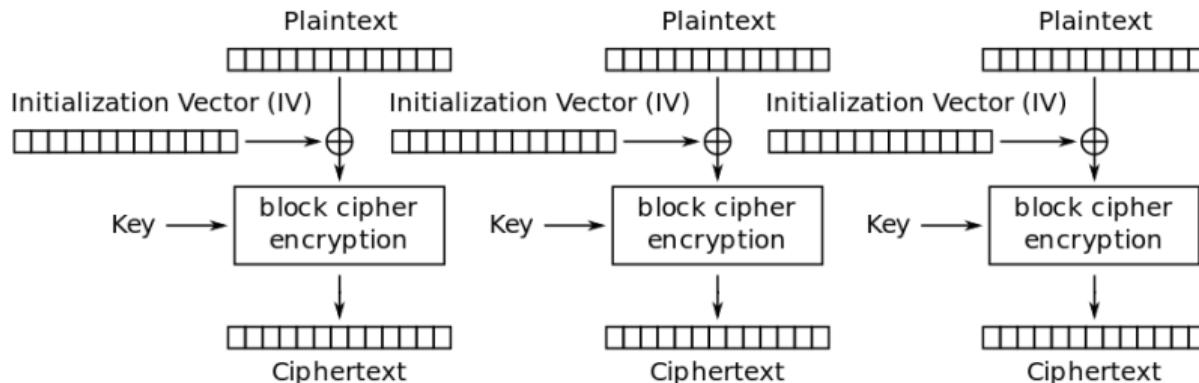
AES – Advanced Encryption Standard (NIST 2001)

- 16 byte (128 bit) block size
- key sizes – 128/192/256 bits

Electronic Codebook (ECB) mode



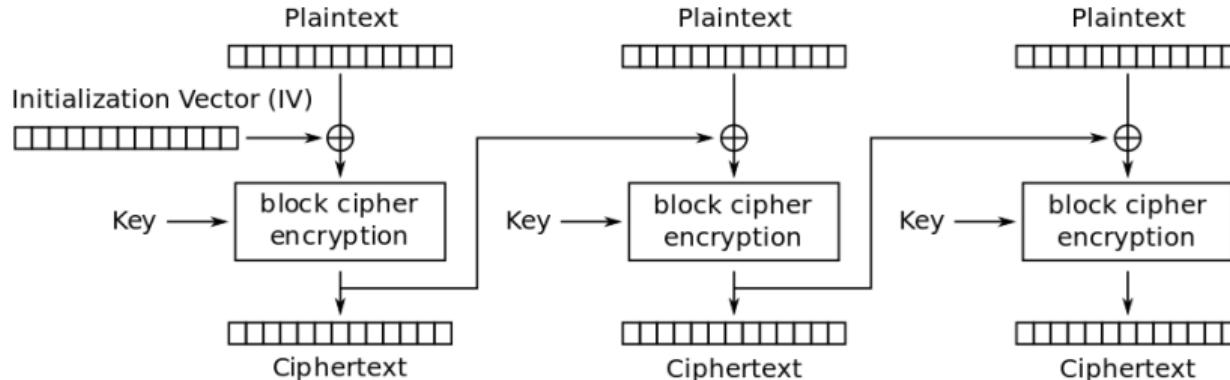
Initialization Vector (IV)



- On encryption XOR plaintext with random IV
- IV must be sent along with the ciphertext
- IV does not have to be secret
- On decryption XOR plaintext with IV

The problem? The ciphertext is two times larger than the plaintext

Cipher Block Chaining (CBC) mode



- Serial encryption
- Parallel decryption
- What about integrity (malleability)?

Plaintext padding

- Random padding:

1a **XX YY ZZ**

- Zero padding:

1a **00 00 00**

- ANSI X.923:

1a **00 00 03**

- ISO/IEC 10126:

1a **F1 A6 03**

- ISO/IEC 7816-4:

1a **80 00 00**

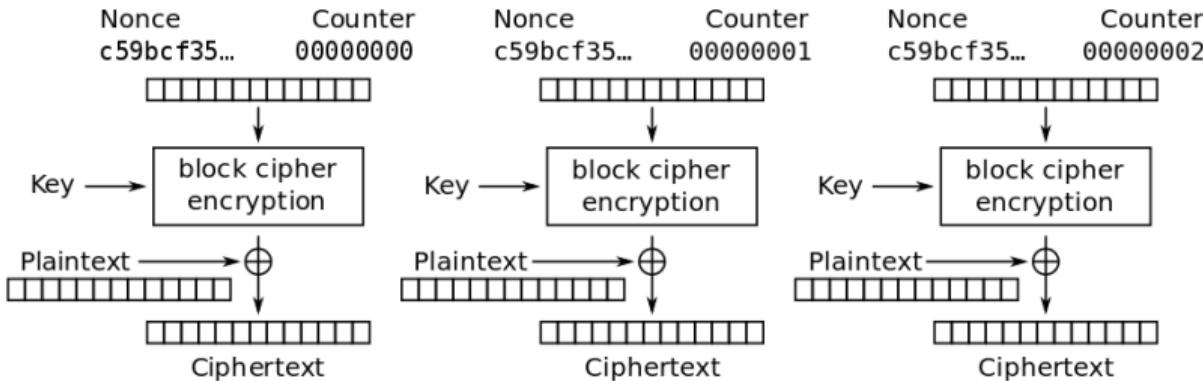
- PKCS#5/PKCS#7:

1a **03 03 03**

1a **05 05 05 05**

Padding is added even if the plaintext occupies a full block.

Counter (CTR) mode



- Block cipher-based PRNG
- Turns block cipher into a stream cipher
- Nonce must never be reused!
- Block ciphers vs stream ciphers

Disk Encryption

- Encrypt the whole disk using CBC?

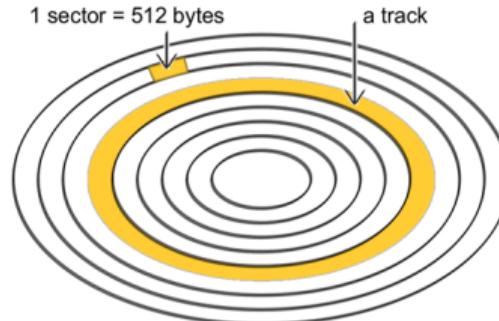
- Each sector is encrypted separately
 - Use sector number as IV

- How to provide integrity?

- MAC is not an option
 - XTS mode

- Password change without disk reencryption

- Master password is used to encrypt data
 - Master password is stored encrypted in disk header
 - User's password decrypts master password
 - Enterprise solutions have several encryptions of master key



Password-based encryption

Deriving strong (128-bit) keys from short, low entropy passwords?

- Use hash of the password
 - Distributes entropy over all 128 bits
- Use salt as an addition to password
 - Prevents multi-target attacks
- Use iterated hash to slow down brute-force
 - Adds arbitrary number of operations to brute-force

Password-Based Key Derivation Function 2 (PBKDF2)

```
key = PBKDF2(PRF, Password, Salt, iter, kLen)
```

- PRF – pseudorandom function (e.g., HMAC-MD5, HMAC-SHA1)
 - Password – password entered by the user
 - Salt – random cryptographic salt
 - Recommended at least 64 bits
 - iter – number of iterations desired
 - Recommended at least 1'000 iterations (increases the security level by 10 bits)
 - NIST recommends 10'000'000 for critical keys (increases the security level by 23 bits)
 - kLen – desired length of the derived key
-
- WPA2 uses `key = PBKDF2(HMAC-SHA1, passphrase, ssid, 4096, 256)`
 - Truecrypt uses PBKDF2 with 2000 iterations

New deployments should use scrypt

Task: Password-based file encryption – 6p

Implement utility that encrypts and decrypts files using a password:

```
$ ./aes.py
```

Usage:

```
-encrypt <plaintextfile> <ciphertextfile>
```

```
-decrypt <ciphertextfile> <plaintextfile>
```

```
$ ./aes.py -encrypt plain.txt plain.txt.enc
```

```
[+] Benchmark: 721709 PBKDF2 iterations in 1 second
```

```
[?] Enter password: asd
```

```
$ ./aes.py -decrypt plain.txt.enc plain.txt
```

```
[?] Enter password: asd
```

- The integrity of the ciphertext is provided using HMAC
- Parameters are stored ASN.1 DER-encoded as a header of the ciphertext file

Task: Password-based file encryption

```
EncInfo ::= SEQUENCE {
    kdfInfo pbkdf2params,
    cipherInfo aesInfo,
    hmacInfo DigestInfo
}
pbkdf2params ::= SEQUENCE {
    salt OCTET STRING,
    iterationCount INTEGER (1..MAX),
    keyLength INTEGER (1..MAX)
}
aesInfo ::= SEQUENCE {
    algorithm OBJECT IDENTIFIER,
    iv OCTET STRING OPTIONAL,
}

$ dumpasn1 plain.txt.enc
0 86: SEQUENCE {
2 18:  SEQUENCE {
4 8:   OCTET STRING 7D 64 F8 30 70 5B AE 73
14 3:   INTEGER 34856
19 1:   INTEGER 36
:
22 29:  SEQUENCE {
24 9:    OBJECT IDENTIFIER aes128-CBC (2 16 840 1 101 3 4 1 2)
35 16:    OCTET STRING 03 C3 62 AC 25 C2 25 48 E4 B8 11 38 25 D5 2C 25
:
53 33:  SEQUENCE {
55 9:   SEQUENCE {
57 5:     OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
64 0:     NULL
:
66 20:   OCTET STRING 85 DF 81 4E C2 32 2A CC C8 BC 4B 36 C5 30 46 2C 4A F1 29 15
:
:
}

Warning: Further data follows ASN.1 data at position 88.
```

Task: Test cases

```
$ echo -n "hello world" > plain
$ ./aes.py -encrypt plain plain.enc
[+] Benchmark: 721709 PBKDF2 iterations in 1 second
[?] Enter password: asd
$ ./aes.py -decrypt plain.enc plain.new
[?] Enter password: asd
$ hexdump -C plain.new
00000000  68 65 6c 6c 6f 20 77 6f  72 6c 64          |hello world|
0000000b

$ echo -e -n "hello world \x01\x01\x02\x02" > plain
$ ./aes.py -encrypt plain plain.enc
[+] Benchmark: 721856 PBKDF2 iterations in 1 second
[?] Enter password: asd
$ ./aes.py -decrypt plain.enc plain.new
[?] Enter password: asd
$ hexdump -C plain.new
00000000  68 65 6c 6c 6f 20 77 6f  72 6c 64 20 01 01 02 02  |hello world ....|
00000010

$ ./aes.py -decrypt plain.enc plain.new
[?] Enter password: asdd
[-] HMAC verification failure: wrong password or modified ciphertext!

$ wget https://bitbucket.org/appcrypto/2020/raw/master/04/big.enc
$ ./aes.py -decrypt big.enc big
[?] Enter password: bigfilepassword (big.enc is using a huge salt on purpose)
    |   |__ key AES: bc3b5f2729e8fdb43bbb394a455cbdb8
    |   |__ key HMAC: bc64eb1bbfcdbeb6e563b68151cf3c8ea478f33a

$ openssl dgst -sha1 big
SHA1(big)= 34edb7d89a791969d710283c7464a80fe2e39249
```

Task: Password-based file encryption

- Slow down password brute-force attacks to 1 try/second
 - Benchmark the time required for 10 000 iterations
 - Extrapolate the iteration count to 1 second

```
start = datetime.datetime.now()  
...  
stop = datetime.datetime.now()  
time = (stop-start).total_seconds()
```

- Use PBKDF2 with HMAC-SHA1 to obtain 36 bytes:

```
pbkdf2_hmac('sha1', password, salt, iter, 36)
```

- Use the first 16 bytes as AES-128 key
- The next 20 bytes as HMAC-SHA1 key
- Generate IV (16 bytes) and salt (8 bytes) randomly
- Implement CBC mode using *pure* AES-128

```
cipher = AES.new(key_aes)  
cipher.encrypt(strxor(plaintext_block, iv_current))  
strxor(cipher.decrypt(ciphertext_block), iv_current)
```

- Use PKCS#5 padding

Task: Password-based file encryption

- Do not buffer the whole plaintext/ciphertext in memory
 - Process plaintext/ciphertext in chunks of max 512 bytes
- On the encryption:
 - Write ciphertext into a temporary file (filename+.tmp)
 - Calculate HMAC and write DER header into the ciphertext file
 - Append to the ciphertext file the ciphertext from the temporary file
 - Remove the temporary file (os.unlink())
- On the decryption:
 - Read the first 10 bytes from the ciphertext file
 - Calculate the length of the DER header by parsing the length byte(s) of header's outer ASN.1 SEQUENCE (all possible sizes of DER header must be handled)
 - Read and decode the DER header
 - Ask for the password and derive the keys
 - Calculate and verify HMAC of the ciphertext
 - In the second pass decrypt the ciphertext (f.seek(offset))

Side channel attack

```
def authorize_admin(submitted_password):  
    hardcoded_password = 'qwerty'  
    if submitted_password == hardcoded_password:  
        return 1 # access granted  
    return 0 # access denied
```

- Function vulnerable to timing attack (comparison stops on first incorrect byte):
 - password 'aaaaaa' – 1ms
 - password 'baaaaa' – 1ms
 - password 'qaaaaa' – 2ms
 - password 'qwaaaa' – 3ms
 - password 'qweaaa' – 4ms
 - password 'qweraa' – 5ms
- Exploitable over low-latency networks
- Using `sleep(random())` before `return` will not help
- Constant-time string comparison function needed

```
def is_equal(a, b):  
    result = 0  
    for x, y in zip(a, b):  
        result |= ord(x) ^ ord(y)  
    return result == 0
```

Your `hmac.py` solution (HW 03) is vulnerable to timing attack

Questions

- How block cipher works (takes as an input, returns)?
- What happens to ciphertext if single plaintext or key bit is changed?
- Why encrypting every block of the file independently is not secure?
- Why do we apply initialization vector (IV) to plaintext block?
- How to provide integrity of ciphertext?
- When should we use stream cipher and when block cipher?
- How to convert short password to 128-bit encryption key?
- What is side-channel vulnerability?

MTAT.07.017
Applied Cryptography

Public Key Cryptography
(Asymmetric Cryptography)

University of Tartu

Spring 2020

Public Key Cryptography

The benefits of asymmetric cryptography:

- Secret key can be negotiated over non-confidential channel
 - Channel has to provide authenticity for the exchanged messages
- Provides possibility for digital signatures
 - Data origin authentication

RSA



Adi Shamir, Ron Rivest, Leonard Adleman (1977)

- The most popular public-key cryptosystem

RSA algorithm

- Key generation:

1. Choose two distinct prime numbers p and q (usually 1024 bits)
2. Compute $n = pq$ (2048 bits)
3. Compute $\varphi(n) = (p - 1)(q - 1)$
4. Choose an integer e such that e and $\varphi(n)$ are coprime
5. Find an integer d such that $de \equiv 1 \pmod{\varphi(n)}$

n - modulus

e - public exponent (encryption exponent)

d - private exponent (decryption exponent)

Public key: (n, e)

Private key: (d)

- Encryption:

- $c \equiv m^e \pmod{n}$

Naive approach:

`>>> m**e % n`

- Decryption:

- $m \equiv c^d \pmod{n}$

Much faster:

`>>> pow(m, e, n)`

- Integer factorization problem

RSA encryption

The basis:

What is encrypted with one key can be decrypted only with the another and vice versa.

- What is encryption for?
 - Only the recipient could decrypt
- How do you encrypt?
 - Using public key of the recipient
- How does recipient decrypt?
 - Using the private key

Hybrid encryption

How to encrypt plaintexts larger than the modulus size?

- Increase modulus size
 - 2x modulus size increase – 8x slowdown
- Split into blocks and encrypt separately
 - Asymmetric encryption much slower than symmetric
- Use RSA to encrypt symmetric data encryption key:



RSA signing

The concept of signing:

Encryption with private key, decryption with public key

- What is signing for?
 - Everyone could authenticate the origin
- How do you sign?
 - Encrypting with your private key
- How do others verify?
 - Decrypting with your public key

In practice message digest is encrypted (signed)

Exponentiation

Which operation is harder to calculate:

$$x^{16384} \quad \text{or} \quad x^{8191} \quad ?$$

Number of multiplications: number of bits + number of “1” bits

```
>>> bin(16384)  
'0b1000000000000000'  
>>> bin(8191)  
'0b111111111111'
```

Example:

$$x^8 = \underbrace{x \cdot x}_{y} \cdot \underbrace{y \cdot z}_{y} \quad x^7 = \underbrace{x \cdot x}_{y} \cdot y \cdot y \cdot x$$

RSA exponents

Key generation:

1. Choose two distinct prime numbers p and q
 2. Compute $n = pq$
 3. Compute $\varphi(n) = (p - 1)(q - 1)$
 4. Choose an integer e such that e and $\varphi(n)$ are coprime
 5. Find an integer d such that $de \equiv 1 \pmod{\varphi(n)}$
- Can we choose e that will provide faster encryption?
 - Yes! $e = 2^{16} + 1 = 65537$ (0b1000000000000001)
 - $e = 3$ (0b11) may also be used (not recommended)
 - Some implementations use random public exponent
 - Can we instead choose d that will provide faster decryption?
 - No! This will leak information about secret d
 - In fact, constant time exponentiation must be used for d

RSA private key file format

```
$ openssl genrsa -out priv.pem 2048
Generating RSA private key, 2048 bit long modulus
.....+++++
.....+++++
e is 65537 (0x10001)

$ cat priv.pem
-----BEGIN RSA PRIVATE KEY-----
MIICXQIBAAKBgQDAdIy9YfBPIk9Imlq0luDERItfj9FINQn/Gv+q+cHk06RgpphX
6c+sIvVk/bEHmWGfVwCcxMF5tzIeC/ns8sHu1b1IsUXSJwi0ArzuxPMBfay5TUu1
6oP/K01BxYaWa3xMSa+FmaAQsugBcWCYTMOiv+H7YkZdpDRZ++IbfWyX1wIDAQAB
AoGAQLdcZ1JYVaktYa3Qh0VXSu2feGzrq/+CeZ+u9CDPbxG/1Z4k7Y5npnAo0IVT
Z5Pp1sF4ffjp9FXwM/SKUsbL6n/TR7U253KjzxzuBPMayjTMqqHTVDwbcJ0zhG
emF1s3aZtRmA8nvrooAQqhr5pfNcL/Oi0mjf2+E4St3IxECQQD9rhVTm4NV1prr
f2813zebpqbhzUPCbUK9/FmfZcx1Tg7EX6RP1Jf0aTXQgCL9nGrZhKVraSSdwAZP
4q/obaOfAkEA3HP/hsUoi1m4VGXLW0cD+c5UYNcfJkkmHJi7AAbzE06FFrHyJhLC
9CsB40VayKbhxmAtN6Djhudltav/oFTgyQJAeysm1610GONTfLwm9vUmPscVjjrn
tRbTtRta7/wqTdL2iNECQQCU9ufnB5Yxy1u0RScYQ6ij4vXV5tD8buPgQsRhQ5xa
qfzQvWQap0hR3F40jq7GcI1orvQcEgYOLFp7VyüeqH56
-----END RSA PRIVATE KEY-----      PEM format (Base64-encoded ASN.1 DER)
```

```
$ openssl rsa -in priv.pem -outform der -out priv.der
writing RSA key
```

RSA private key file format

```
$ dumpasn1 priv.der
0 605: SEQUENCE {
4   1:   INTEGER 0
7 129:   INTEGER
:     00 DA 74 8C BD 61 F0 4F 22 4F 48 9A 5A B4 96 E0
:       D7 ...
139  3:   INTEGER 65537
144 128:   INTEGER
:     40 B7 5C 66 52 58 55 A9 2D 61 AD D0 87 45 57 4A
:       97 ...
275  65:   INTEGER
:     00 FD AE 15 53 9B 83 55 D6 9A EB 7F 6F 35 DF 37
:       9F ...
342  65:   INTEGER
:     00 DC 73 FF 86 C5 28 8B 59 B8 54 65 CB 5B 47 03
:       C9 ...
409  64:   INTEGER
:     7B 2B 26 D7 AD 4E 1B 43 53 7C BC 26 F6 F5 26 3E
:       DD ...
475  65:   INTEGER
:     00 87 56 C7 66 CB 9F 6A 7D 78 46 87 FF E2 57 A4
:       D1 ...
542  65:   INTEGER
:     00 94 F6 E7 E7 07 96 31 CA 5B 8E 45 27 18 43 A8
:       7A ...
: }
```

RSA private key file format

```
--  
-- Representation of RSA private key with information for  
-- the CRT algorithm.  
  
RSAPrivateKey ::= SEQUENCE {  
    version            Version,  
    modulus             INTEGER,   -- n  
    publicExponent     INTEGER,   -- e  
    privateExponent    INTEGER,   -- d  
    prime1              INTEGER,   -- p  
    prime2              INTEGER,   -- q  
    exponent1           INTEGER,   -- d mod (p-1)  
    exponent2           INTEGER,   -- d mod (q-1)  
    coefficient         INTEGER,   -- (inverse of q) mod p  
    otherPrimeInfos     OtherPrimeInfos OPTIONAL  
}
```

<http://tools.ietf.org/html/rfc3447>

RSA public key file format

```
$ openssl rsa -inform der -in priv.der -pubout -outform der -out pub.der  
writing RSA key
```

```
$ openssl rsa -inform der -in pub.der -pubin -out pub.pem  
writing RSA key
```

```
$ dumpasn1 pub.der  
0 159: SEQUENCE {  
3 13:   SEQUENCE {  
5 9:     OBJECT IDENTIFIER rsaEncryption (1 2 840 113549 1 1 1)  
16 0:     NULL  
:  
18 141:   BIT STRING, encapsulates {  
22 137:     SEQUENCE {  
25 129:       INTEGER  
:  
:       00 DA 74 8C BD 61 F0 4F 22 4F 48 9A 5A B4 96 E0  
:  
:       C4 44 8B 5F 8F D1 48 35 09 FF 1A FF AA F9 C1 E4  
:  
:       D3 A4 60 A6 98 57 E9 CF AC 22 F5 64 FD B1 07 99  
:  
:       61 9F 57 00 9C C4 C1 79 B7 32 1E 0B F9 EC F2 C1  
:  
:       EE D5 BD 48 B1 45 D2 27 08 8E 02 BC EE C4 F3 01  
:  
:       7D AC B9 4D 4B B5 EA 83 FF 2B 49 41 C5 86 96 6B  
:  
:       7C 4C 49 AF 85 99 A0 10 B2 E8 01 71 60 98 4C C3  
:  
:       A2 BF E1 FB 62 46 5D A4 34 59 FB E2 1B 7D 6C 97  
:  
:       D7  
157 3:     INTEGER 65537  
:  
:   }  
:  
: }
```

RSA public key file format

```
SubjectPublicKeyInfo ::= SEQUENCE {
    algorithm AlgorithmIdentifier,
    subjectPublicKey     BIT STRING ::= RSAPublicKey
}

RSAPublicKey ::= SEQUENCE {
    modulus             INTEGER,   -- n
    publicExponent     INTEGER   -- e
}

AlgorithmIdentifier ::= SEQUENCE {
    algorithm OBJECT IDENTIFIER ::= rsaEncryption,
    parameters ANY DEFINED BY algorithm OPTIONAL ::= NULL
}

rsaEncryption OBJECT IDENTIFIER ::= { pkcs-1 1 }
```

<https://tools.ietf.org/html/rfc3280>

<https://tools.ietf.org/html/rfc3447>

Task: RSA utility – 5p

Implement RSA encryption and signing utility.

```
$ ./rsa.py
```

Usage:

```
encrypt <public key file> <plaintext file> <output ciphertext file>
decrypt <private key file> <ciphertext file> <output plaintext file>
sign <private key file> <file to sign> <signature output file>
verify <public key file> <signature file> <file to verify>
```

- Must support private and public keys in PEM and DER format (detect by content; use `codecs.decode(val, 'base64')`)
- Halt if requested to encrypt plaintext larger than possible
- Must sign SHA256 hash (`DigestInfo`) of the file to be signed
- Verification must output “Verified OK” / “Verification failure”
- Encryption and signing according to PKCS#1 v1.5
- Use your own ASN.1 DER encoder and `pyasn1` for decoding

RSA PKCS#1 v1.5

Encryption process:

1. Pad plaintext: $0x00 \parallel 0x02 \parallel PS \parallel 0x00 \parallel D$
 - D – plaintext to encrypt
 - PS (padding string) – at least 8 random bytes (except 0x00)
 - Plaintext must be padded to the size of modulus n
2. Convert padded byte string to integer
3. Calculate ciphertext: $c \equiv m^e \pmod{n}$
 - in python: `c = pow(m, e, n)`
4. Convert ciphertext integer to byte string

Decryption process:

1. Convert ciphertext to integer
2. Calculate decryption: $m \equiv c^d \pmod{n}$
3. Convert decrypted integer to byte string
4. Remove padding

<https://tools.ietf.org/html/rfc2313>

RSA PKCS#1 v1.5

Signing process:

1. Construct plaintext (DER DigestInfo of the file to sign)
2. Pad plaintext: $0x00 \parallel 0x01 \parallel PS \parallel 0x00 \parallel D$
 - D – plaintext to sign
 - PS (padding string) – zero or more $0xFF$ bytes
 - Plaintext must be padded to size of modulus n
3. Convert padded byte string to integer
4. Calculate signature: $s \equiv m^d \pmod{n}$
5. Convert signature integer to byte string (**length of n**)

Verification process:

1. Convert signature byte string to integer
2. Calculate decryption: $m \equiv s^e \pmod{n}$
3. Convert decrypted integer to byte string
4. Remove padding to obtain DigestInfo DER structure
5. Compare DigestInfo with DigestInfo of the signed file

<https://tools.ietf.org/html/rfc2313>

Task: Test case

```
#!/bin/bash
echo "[+] Generating RSA key pair..."
openssl genrsa -out priv.pem 2041
openssl rsa -in priv.pem -pubout -out pub.pem

echo "[+] Testing encryption..."
echo "hello" > plain.txt
./rsa.py encrypt pub.pem plain.txt enc.txt
openssl rsautl -decrypt -inkey priv.pem -in enc.txt -out dec.txt
diff -u plain.txt dec.txt

echo "[+] Testing decryption..."
openssl rsautl -encrypt -pubin -inkey pub.pem -in plain.txt -out enc.txt
./rsa.py decrypt priv.pem enc.txt dec.txt
diff -u plain.txt dec.txt

echo "[+] Testing signing..."
dd if=/dev/urandom of=filetosign bs=1M count=1
./rsa.py sign priv.pem filetosign signature
openssl dgst -sha256 -verify pub.pem -signature signature filetosign

echo "[+] Testing successful verification..."
openssl dgst -sha256 -sign priv.pem -out signature filetosign
./rsa.py verify pub.pem signature filetosign
Verified OK

echo "[+] Testing failed verification..."
openssl dgst -md5 -sign priv.pem -out signature filetosign
./rsa.py verify pub.pem signature filetosign
Verification failure
```

Diffie-Hellman Key Exchange

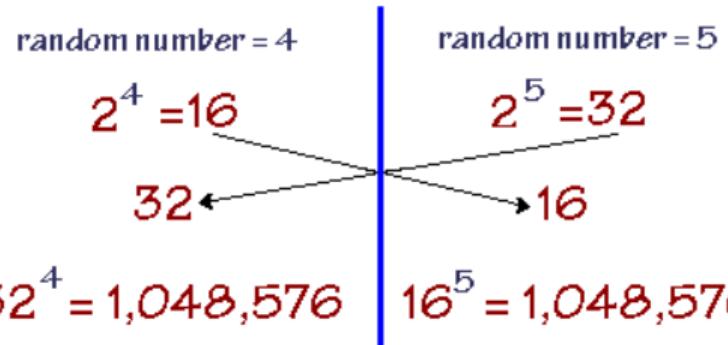


Ralph Merkle, Martin Hellman, Whitfield Diffie (1976)

- The first public-key algorithm

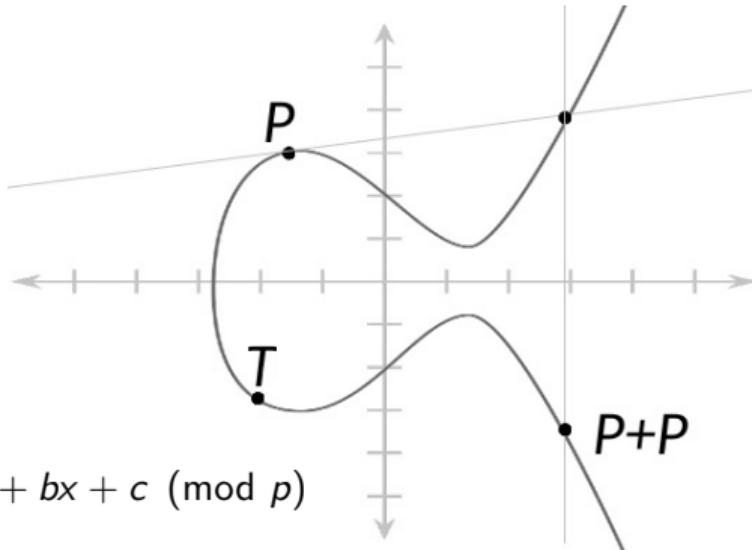
Diffie-Hellman (DH) Key Exchange

common number = 2



- $(2^5)^4 = 2^{5 \cdot 4} = (2^4)^5$
- In practice multiplicative group of integers modulo p is used
- Discrete logarithm problem:
 - hard to find x, given $2^x = 32 \pmod{p}$
- ElGamal and DSA based on DL problem
- Secure against passive eavesdropping

Elliptic Curve Cryptography



$$y^2 \pmod{p} = x^3 + ax^2 + bx + c \pmod{p}$$

- Formulas for point addition, doubling
- DL problem: find x , given $a^x \equiv b \pmod{p}$
- EC DL problem: find x , given $\underbrace{P + P + \dots + P}_{x \text{ times}} = xP = T$
- Security: 256-bit ECC \approx 3072-bit RSA

Key length recommendations (NIST)

Date	Minimum of Strength	Symmetric Algorithms	Asymmetric	Discrete Logarithm Key Group	Elliptique Curve	Hash (A)	Hash (B)
2010 (Legacy)	80	2TDEA*	1024	160 1024	160	SHA-1** SHA-224 SHA-256 SHA-384 SHA-512	SHA-1 SHA-224 SHA-256 SHA-384 SHA-512
2011 - 2030	112	3TDEA	2048	224 2048	224	SHA-224 SHA-256 SHA-384 SHA-512	SHA-1 SHA-224 SHA-256 SHA-384 SHA-512
> 2030	128	AES-128	3072	256 3072	256	SHA-256 SHA-384 SHA-512	SHA-1 SHA-224 SHA-256 SHA-384 SHA-512
>> 2030	192	AES-192	7680	384 7680	384	SHA-384 SHA-512	SHA-224 SHA-256 SHA-384 SHA-512
>>> 2030	256	AES-256	15360	512 15360	512	SHA-512	SHA-256 SHA-384 SHA-512

Adversary (threat) model

- What are the capabilities of the adversary?

Passive attacks (eavesdropping):



Active attacks (man-in-the-middle):



- Which attack is harder to execute?
- Without a threat model the word “secure” tells nothing

Questions

- What does public key cryptography give us?
- How will you create RSA encrypted message to me?
- How will you verify my RSA signed message?
- What is hybrid encryption useful for?
- How are passive attacks different from active attacks?
- Why active attacks are harder to execute?
- What is threat model useful for?
- Why 2048-bit RSA does not have security level of 2048 bits?
- What will happen to cryptography the day quantum computers are invented?

MTAT.07.017

Applied Cryptography

Public Key Infrastructure (PKI)
Public Key Certificates (X.509)

University of Tartu

Spring 2020

Key management

- The hardest problem
- How to obtain the key of the other party?
 - Symmetric key?
 - Confidential and authentic channel needed
 - Asymmetric key?
 - Authentic channel needed
- Trust models:
 - Trust on first use (e.g., SSH)
 - Decentralized model - web of trust (e.g., PGP)
 - Centralized model - trusted third party (e.g., TLS)

Trust on first use (TOFU)

- Used by SSH (encrypted telnet)
- For the first time:

```
$ ssh user@math.ut.ee
The authenticity of host 'math.ut.ee (193.40.37.95)' can't be established.
RSA key fingerprint is SHA256:2x7va2E9JDr1xwWRemr5gYQrguFjBGikei9bXDi6K44.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'math.ut.ee,193.40.37.95' (RSA) to the list of known hosts.
user@math.ut.ee's password:

$ cat ~/.ssh/known_hosts
cs.ut.ee,193.40.36.81 ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAQEA2HotF0bR9U8MgTE67bGJrLFy4DQ/uVN6ZWNucwtlUJ+
math.ut.ee,193.40.37.95 ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAQEA+EuCKTMZU9LYhNqBLfz8KGgqvLv90wiadUOAHv2LT
```

- In the future:

```
$ ssh user@cs.ut.ee
user@math.ut.ee's password:
```

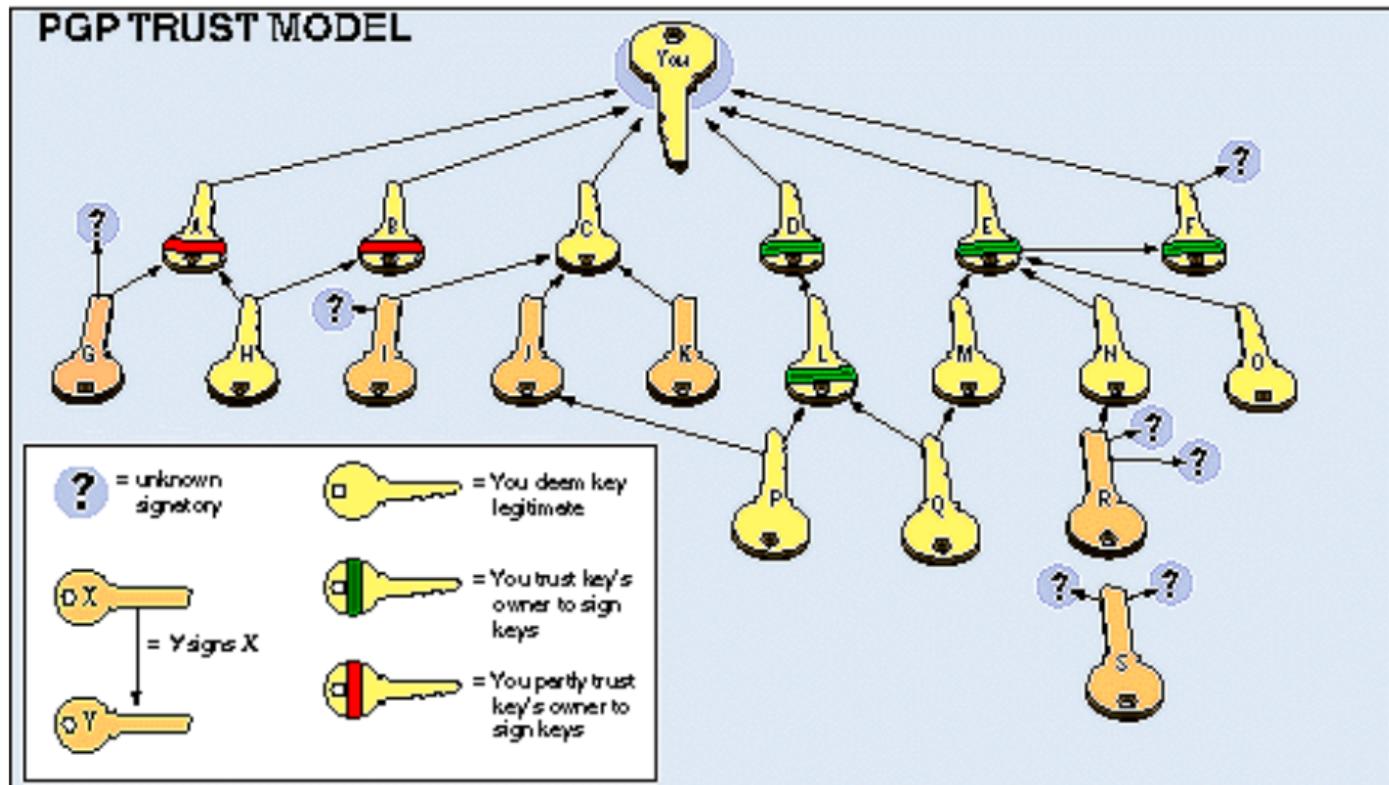
Trust on first use (TOFU)

- If the key has changed:

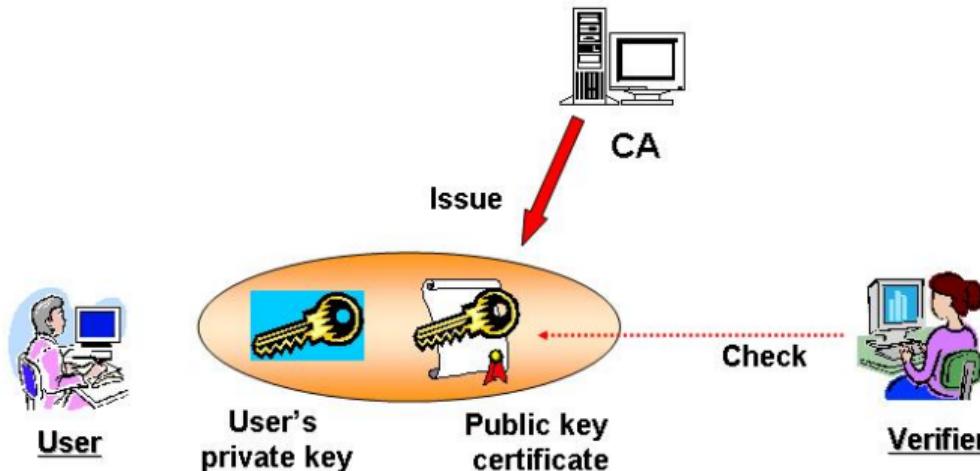
```
$ ssh user@math.ut.ee
oooooooooooooooooooooooooooooooooooooooooooo
@      WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!      @
oooooooooooooooooooooooooooooooooooooooooooo
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
SHA256:4JlsTx1vbxHYaF6ALHD/dTkbX5N6ViZZQtNItAKd04k.
Please contact your system administrator.
```

- Threat model?
- How to make it more secure?

Web of trust (WOT)



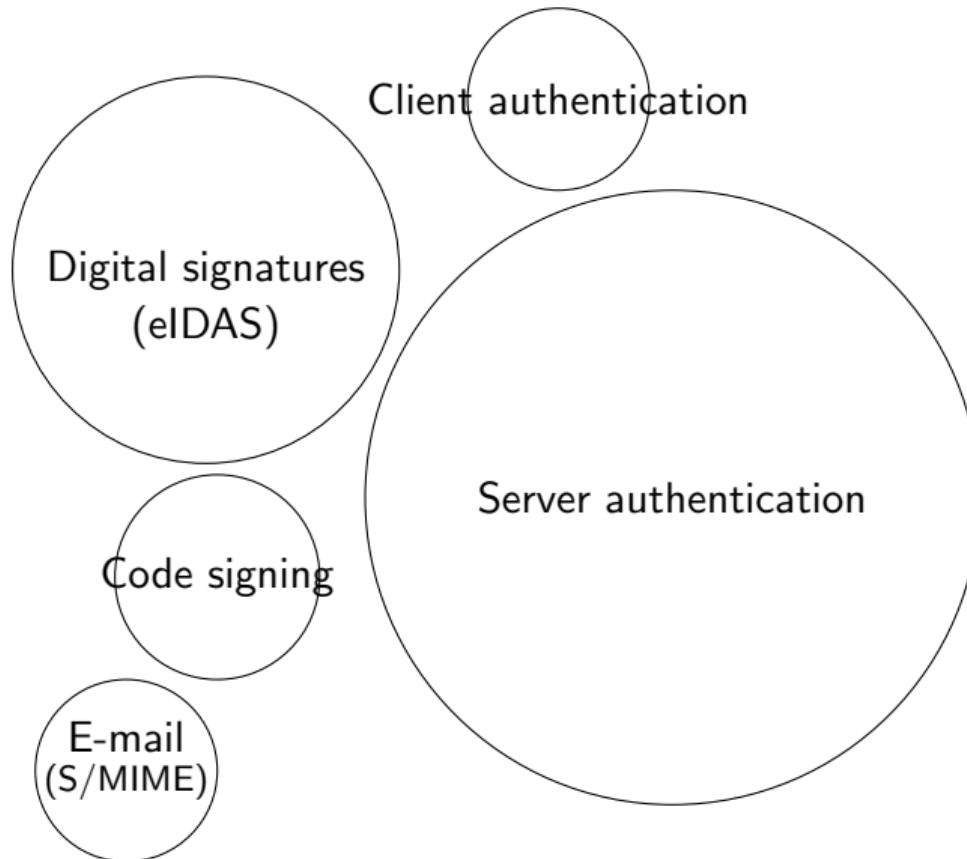
Public key infrastructure (PKI)



Public key certificate binds a public key with an identity

- Main actors:
 - User (subject, subscriber, end-entity)
 - Certificate authority (trusted third party, issuer)
 - Verifier (relying party)
- The passport analogy

PKI use cases



Certificate Authority (CA)

Certificate Authority – **Trusted** Third Party

Where the trust comes from?

- Software vendors decided on your behalf (Mozilla, Google)
- EU Regulation 910/2014 (eIDAS)



- Root CA – self-signed certificate (trust anchor)
- CA can delegate trust to subordinate/intermediate CAs
- Useful for risk limitation

How to become a CA?

- The goal: profit (Mark Shuttleworth, Thawte (VeriSign), \$575 million)
- Get your root CA trusted:
 - Compliance audit (WebTrust, ETSI TS)
 - Ernst & Young or KPMG (15k EUR/year)
 - Liability insurance (required by eIDAS)
 - Insurance industry reluctant (3k EUR/year)
 - Use of Hardware Security Modules (HSM)



<http://www.mozilla.org/projects/security/certs/policy/InclusionPolicy.html>

http://bugzilla.mozilla.org/show_bug.cgi?id=414520

X.509 certificate

Subject: “John Smith”

Issuer: Verisign, Inc. Root CA

Public key: BC F7 C6 74 F5 32 D0 34 ...

Serial #: 11:21:56:2D:2E

Valid: from 2018.01.01 15:00 to 2020.01.01 15:00

Other data: ...

Signed: CA's Signature



X.509 certificate

```
Certificate ::= SEQUENCE {
    tbsCertificate      TBSCertificate,
    signatureAlgorithm   AlgorithmIdentifier,
    signatureValue       BIT STRING  }

TBSCertificate ::= SEQUENCE {
    version            [0] EXPLICIT Version DEFAULT v1(0),
    serialNumber        INTEGER,
    signature           AlgorithmIdentifier,
    issuer              Name,
    validity            Validity,
    subject              Name,
    subjectPublicKeyInfo SubjectPublicKeyInfo,
    extensions          [3] EXPLICIT Extensions OPTIONAL -- v3(2) only }

Validity ::= SEQUENCE {
    notBefore           UTCTime,
    notAfter            UTCTime }

Extensions ::= SEQUENCE SIZE (1..MAX) OF Extension
Extension ::= SEQUENCE {
    extnID              OBJECT IDENTIFIER,
    critical            BOOLEAN DEFAULT FALSE,
    extnValue           OCTET STRING }
```

<http://tools.ietf.org/html/rfc5280>

X.509 certificate

- tbsCertificate – DER structure to be signed by CA
- version – X.509v1 or X.509v3 used
 - X.509 v3 introduces certificate extensions
- serialNumber – unique for every certificate issued by CA
- signature – AlgorithmIdentifier from outer Certificate sequence
- issuer – identity of CA who signed the certificate
- validity – period in which certificate should be assumed valid
- subject – identity of a subject whose public key in certificate
- subjectPublicKeyInfo – subject's public key
- extensions – optional extensions providing more information

Distinguished Name (DN) in X.509 Certificate

The issuer and subject field is defined as the X.501 type Name:

```
Name ::= RDNSequence
```

```
RDNSequence ::= SEQUENCE OF RelativeDistinguishedName
```

```
RelativeDistinguishedName ::= SET OF AttributeTypeAndValue
```

```
AttributeTypeAndValue ::= SEQUENCE {
```

```
    type      OBJECT IDENTIFIER,
```

```
    value     ANY -- DEFINED BY type }
```

- Yet another notation for unique identifiers
- Used in LDAP and related protocols
- Example: CN=John Doe, OU=Helpdesk, O=Burgers Inc., C=US

Distinguished Name (DN) in X.509 Certificate

```
2 74: SEQUENCE {
4 11:   SET {
6 9:     SEQUENCE {
8 3:       OBJECT IDENTIFIER countryName (2 5 4 6)
13 2:       PrintableString 'US'
      :
      }
    :
  17 18:   SET {
  19 16:     SEQUENCE {
  21 3:       OBJECT IDENTIFIER organizationName (2 5 4 10)
  26 9:       UTF8String 'Burgers Inc.'
      :
      }
    :
  37 20:   SET {
  39 18:     SEQUENCE {
  41 3:       OBJECT IDENTIFIER organizationalUnitName (2 5 4 11)
  46 11:       UTF8String 'Helpdesk'
      :
      }
    :
  59 17:   SET {
  61 15:     SEQUENCE {
  63 3:       OBJECT IDENTIFIER commonName (2 5 4 3)
  68 8:       UTF8String 'John Doe'
      :
      }
    :
  65 14: }
```

(2 5 4 4)	: surname (SN)
(2 5 4 42)	: givenName (GN)
(2 5 4 5)	: serialNumber
(2 5 4 7)	: localityName (L)
(2 5 4 8)	: stateOrProvinceName (ST)
(1 2 840 113549 1 9 1)	: emailAddress

Certificate extensions (X.509v3 only)

```
Extensions ::= SEQUENCE SIZE (1..MAX) OF Extension
Extension ::= SEQUENCE {
    extnID      OBJECT IDENTIFIER,
    critical     BOOLEAN DEFAULT FALSE,
    extnValue   OCTET STRING }
```

- Every extension has its OID
- RFC 5280 defines several standard extensions
- Certificate (path) validation algorithm must handle those

"Each extension in a certificate is designated as either critical or non-critical. A certificate-using system MUST reject the certificate if it encounters a critical extension it does not recognize or a critical extension that contains information that it cannot process. A non-critical extension MAY be ignored if it is not recognized, but MUST be processed if it is recognized."

<http://tools.ietf.org/html/rfc5280>

Certificate extensions (X.509v3 only)

- Key usage – limits the purpose of the key contained in the certificate

```
KeyUsage ::= BIT STRING {  
    digitalSignature      (0),  
    nonRepudiation        (1), -- contentCommitment  
    keyEncipherment       (2),  
    dataEncipherment      (3),  
    keyAgreement          (4),  
    keyCertSign           (5),  
    cRLSign               (6),  
    encipherOnly          (7),  
    decipherOnly          (8) }
```

- If extension is not present the key may be used for all purposes
- Extended key usage – indicates a more specific purpose of the key

```
ExtKeyUsageSyntax ::= SEQUENCE SIZE (1..MAX) OF KeyPurposeId  
KeyPurposeId ::= OBJECT IDENTIFIER  
id-kp-serverAuth   OBJECT IDENTIFIER ::= { 1 3 6 1 5 5 7 3 1 }  
id-kp-clientAuth   OBJECT IDENTIFIER ::= { 1 3 6 1 5 5 7 3 2 }  
id-kp-codeSigning   OBJECT IDENTIFIER ::= { 1 3 6 1 5 5 7 3 3 }  
id-kp-emailProtection OBJECT IDENTIFIER ::= { 1 3 6 1 5 5 7 3 4 }
```

- Usage must be consistent with the key usage extension

Certificate extensions (X.509v3 only)

- Basic constraints – identifies whether subject is CA
 - For CA certificate identifies maximum subordinate CAs it may have

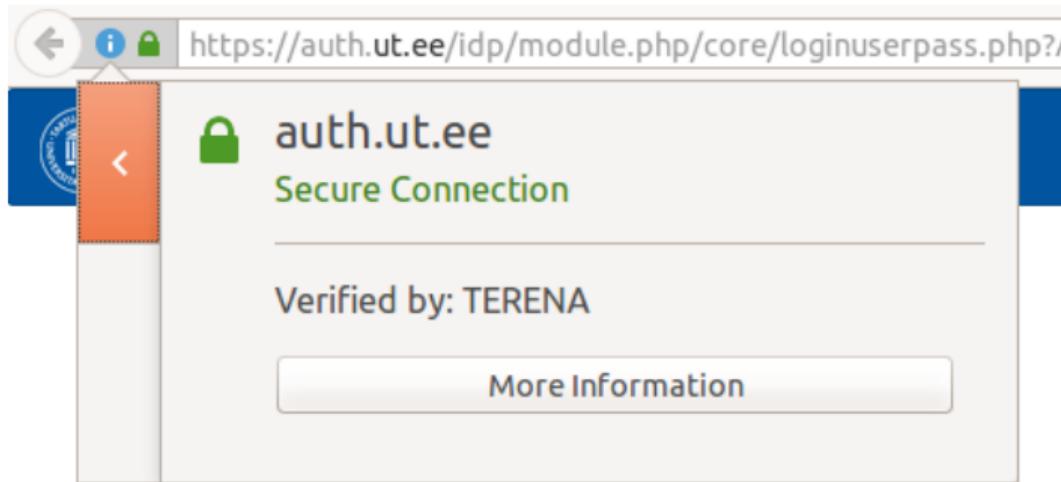
```
id-ce-basicConstraints OBJECT IDENTIFIER ::= { id-ce 19 }
BasicConstraints ::= SEQUENCE {
    cA             BOOLEAN DEFAULT FALSE,
    pathLenConstraint   INTEGER (0..MAX) OPTIONAL }
```
 - If cA is TRUE, the key usage extension must be absent or must have keyCertSign bit set
- Certificate policies – contains pointer to policy information
 - URL to certificate practice statement (CPS)
 - OID of the CPS document version
 - Explicit notice text

Certificate extensions (X.509v3 only)

- Subject alternative name
 - Identifies subject alternatively to the subject name
 - Include email, DNS name, IP addresses, URI, etc.
 - New standards promote use of this extension
- Authority key identifier and subject key identifier
 - Uniquely identifies subject and issuer
- CRL distribution points
 - Includes URI where CRL is available (HTTP or LDAP)
- Authority information access
 - Indicates how to access information about CA services
- Subject information access
 - Indicates how to access information about subject

Extensions may include a picture of the subject, attributes, roles, etc.

Use in HTTPS (TLS)



- TLS server certificates – the most popular use case
- What does the browser verify before the connection is considered secure?
 - Certificate signed by a trusted CA
 - Host name in the address bar matches the CN in the certificate
 - Validity date, extensions, etc.

Server certificate

General Details

This certificate has been verified for the following uses:

SSL Client Certificate

SSL Server Certificate

Issued To

Common Name (CN) auth.ut.ee
Organization (O) Tartu Ülikool
Organizational Unit (OU) <Not Part Of Certificate>
Serial Number 0E:0F:48:19:D1:F0:27:06:43:27:39:F6:B3:17:8A:9F

Issued By

Common Name (CN) TERENA SSL CA 3
Organization (O) TERENA
Organizational Unit (OU) <Not Part Of Certificate>

Period of Validity

Begins On September 27, 2016
Expires On October 2, 2019

Fingerprints

SHA-256 Fingerprint EC:3A:B9:87:D0:F8:A6:B9:DE:6F:96:3D:F7:E2:4C:08:
BA:7F:96:DB:49:D1:9A:D4:82:53:B5:E9:51:C8:51:02

SHA1 Fingerprint F1:F8:AA:AA:11:A5:92:5F:D9:7B:CE:57:B0:12:29:3B:0A:39:F3:37

Server certificate

General Details

Certificate Hierarchy

- DigiCert Assured ID Root CA
 - TERENA SSL CA 3
 - auth.ut.ee

Certificate Fields

- auth.ut.ee
 - Certificate
 - Version
 - Serial Number
 - Certificate Signature Algorithm
 - Issuer
 - Validity
 - Not Before
 - Not After

Field Value

CN = TERENA SSL CA 3
O = TERENA
L = Amsterdam
ST = Noord-Holland
C = NL

Export...

Server certificate

```
$ openssl x509 -in authutee.crt -text
Version: 3 (0x2)
Serial Number: 0e:0f:48:19:d1:f0:27:06:43:27:39:f6:b3:17:8a:9f
Signature Algorithm: sha256WithRSAEncryption
Issuer: C=NL, ST=Noord-Holland, L=Amsterdam, O=TERENA, CN=TERENA SSL CA 3
Validity
    Not Before: Sep 27 00:00:00 2016 GMT
    Not After : Oct  2 12:00:00 2019 GMT
Subject: C=EE, ST=Tartumaa, L=Tartu, O=Tartu ulikool, CN=auth.ut.ee
Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
        Public-Key: (2048 bit)
            Modulus:
                00:e2:46:77:e9:c6:88:cf:c9:d5:27:0d:d5:7d:f5:
            Exponent: 65537 (0x10001)
X509v3 extensions:
    X509v3 Basic Constraints: critical
        CA:FALSE
    X509v3 Key Usage: critical
        Digital Signature, Key Encipherment
    X509v3 Extended Key Usage:
        TLS Web Server Authentication, TLS Web Client Authentication
    X509v3 Subject Alternative Name:
        DNS:auth.ut.ee, DNS:parool.ut.ee, DNS:passwd.ut.ee
    X509v3 CRL Distribution Points:
        Full Name: URI:http://crl3.digicert.com/TERENASSLCA3.crl
Signature Algorithm: sha256WithRSAEncryption
    0c:89:c4:3d:2f:c0:76:d9:92:f2:eb:2f:c0:e8:7f:fd:b0:56:
        64:d2:fc:56:c3:86:5c:70:50:4c:10:85:f5:68:07:b2:4b:cc:
```

Certificate authorities

Certificate Manager X

Your Certificates People Servers Authorities Authorities Others

You have certificates on file that identify these certificate authorities

Certificate Name	Security Device
Amazon Root CA 1	Builtin Object Token
Amazon Root CA 2	Builtin Object Token
Amazon Root CA 3	Builtin Object Token
Amazon Root CA 4	Builtin Object Token
Amazon	Software Security Device
AS Sertifitseerimiskeskus	
EE Certification Centre Root CA	Builtin Object Token
KLASS3-SK 2016	Software Security Device
KLASS3-SK 2010	Software Security Device
KLASS3-SK 2010	Software Security Device
Atos	
Atos TrustedRoot 2011	Builtin Object Token
Autoridad de Certificacion Firmaprofesional CIF A62634068	
Autoridad de Certificacion Firmaprofesional CIF A62634068	Builtin Object Token
Baltimore	

[View...](#) [Edit Trust...](#) [Import...](#) [Export...](#) [Delete or Distrust...](#)

OK

Identity verification

- Domain Validation (DV): \$20/year \$0/year
- Checks whether you control the domain

  <https://www.pilet.ee/cgi-bin/splususer/splususer.cgi>

- Organization Validation (OV): \$200/year
- Checks whether you operate the organization

  <https://www.eesti.ee/et/index.html>

- Extended Validation (EV): \$500/year
- Checks whether you operate the organization x2

  **Swedbank AS (EE)** | <https://www.swedbank.ee/private>

Domain Validation (DV) vs Organization Validation (OV)

General	Details
This certificate has been verified for the following uses:	
SSL Client Certificate	
SSL Server Certificate	
Issued To	
Common Name (CN)	*.pilet.ee
Organization (O)	<Not Part Of Certificate>
Organizational Unit (OU)	Domain Control Validated
Serial Number	40:D3:7F:5E:19:23:E3:CF
Issued By	
Common Name (CN)	Go Daddy Secure Certificate Authority - G2
Organization (O)	GoDaddy.com, Inc.
Organizational Unit (OU)	http://certs.godaddy.com/repository/
Period of Validity	
Begins On	January 19, 2018
Expires On	January 20, 2019
Fingerprints	
SHA-256 Fingerprint	9A:75:13:90:DA:B9:DA:2B:81:EC:34 04:65:9B:9E:A2:3F:F7:9A:44:29:7C
SHA1 Fingerprint	3A:BE:B9:27:AB:EE:CB:CE:A0:63:14:1E:82:E
This certificate has been verified for the following uses:	
SSL Client Certificate	
SSL Server Certificate	
Issued To	
Common Name (CN)	*.eesti.ee
Organization (O)	Estonian Information System Authority
Organizational Unit (OU)	<Not Part Of Certificate>
Serial Number	0F:65:52:BB:9B:6B:B7:30:FA:86:C3:4D:FC:44:E9:11
Issued By	
Common Name (CN)	DigiCert SHA2 High Assurance Server CA
Organization (O)	DigiCert Inc
Organizational Unit (OU)	www.digicert.com
Period of Validity	
Begins On	June 29, 2016
Expires On	August 29, 2019
Fingerprints	
SHA-256 Fingerprint	DC:9F:7C:F3:BC:1C:BF:5A:BD:90:C2:6F:BA:C AE:90:1E:73:0F:C4:C0:AE:49:9B:DE:4F:F8:E
SHA1 Fingerprint	F5:2E:57:B8:CD:86:52:E3:F0:CB:EB:7D:E8:C4:93:B3:39

Certificate signing request (CSR)

```
$ openssl genrsa -out priv.pem 2048
```

```
$ openssl req -new -key priv.pem -out auth.ut.ee.csr
```

You are about to be asked to enter information that will be incorporated into your certificate request.

What you are about to enter is what is called a Distinguished Name or a DN.

There are quite a few fields but you can leave some blank

For some fields there will be a default value,

If you enter '.', the field will be left blank.

Country Name (2 letter code) [AU]:EE

State or Province Name (full name) [Some-State]:.

Locality Name (eg, city) []:.

Organization Name (eg, company) [Internet Widgits Pty Ltd]:.

Organizational Unit Name (eg, section) []:.

Common Name (e.g. server FQDN or YOUR name) []:auth.ut.ee

Email Address []:.

Please enter the following 'extra' attributes

to be sent with your certificate request

A challenge password []:asdasd

An optional company name []:.

```
$ cat auth.ut.ee.csr
```

-----BEGIN CERTIFICATE REQUEST-----

```
MIIIBZzCBOQIBADAOHQswCQYDVQQGEwJFRTEZMBcGA1UEAwQd3d3LmFwcGNyeXB0
```

Certificate signing request (CSR)

```
$ openssl req -in auth.ut.ee.csr -text
Certificate Request:
Data:
    Version: 0 (0x0)
    Subject: C=EE, CN=auth.ut.ee
    Subject Public Key Info:
        Public Key Algorithm: rsaEncryption
            Public-Key: (2048 bit)
                Modulus:
                    00:94:94:da:30:50:8d:ed:4d:b9:1f:3c:74:58:c3:
                    bc:1c:32:cd:82:11:55:c3:0f:af:09:a7:b2:91:de:
                    82:56:7c:a1:e8:0d:b5:cf:b5:ee:48:5d:d2:d1:9c:
                    cd:f6:98:d9:ad:f7:84:fa:09:7a:a9:6f:55:d0:25:
                    2f:63:bc:8b:3d:bc:29:e7:3d
                Exponent: 65537 (0x10001)
    Attributes:
        challengePassword :unable to print attribute
Signature Algorithm: sha256WithRSAEncryption
    6c:f7:be:7f:02:42:8f:4b:53:0d:52:4e:9f:e8:5c:dc:b0:7d:
    ba:b8:01:7d:00:8b:ff:68:78:29:6e:55:5d:18:a1:fe:01:a7:
    e8:cf:26:64:17:7d:b1:e4:a9:00:22:c2:12:72:c1:b3:34:f9:
    00:cd
```

Certificate signing request (CSR)

```
$ openssl req -in auth.ut.ee.csr -outform der -out auth.ut.ee.csr.der
$ dumpasn1 auth.ut.ee.csr.der
0 376: SEQUENCE {
4 226:   SEQUENCE {
7 1:     INTEGER 0
10 34:    SEQUENCE {
12 11:      SET {
14 9:        SEQUENCE {
16 3:          OBJECT IDENTIFIER countryName (2 5 4 6)
21 2:            PrintableString 'EE'
:
25 19:      SET {
27 17:        SEQUENCE {
29 3:          OBJECT IDENTIFIER commonName (2 5 4 3)
34 10:            UTF8String 'auth.ut.ee'
:
25 19:      SET {
27 17:        SEQUENCE {
29 3:          OBJECT IDENTIFIER commonName (2 5 4 3)
34 10:            UTF8String 'auth.ut.ee'
:
25 19:      SET {
27 17:        SEQUENCE {
29 3:          OBJECT IDENTIFIER rsaEncryption (1 2 840 113549 1 1 1)
51 9:            NULL
:
64 141:          BIT STRING, encapsulates {
68 137:            SEQUENCE {
71 129:              INTEGER
:
203 3:                00 94 94 DA 30 50 8D ED 4D B9 1F 3C 74 58 C3 BC
203 3:                INTEGER 65537
:
208 23:              [0] {
210 21:                SEQUENCE {
212 9:                  OBJECT IDENTIFIER challengePassword (1 2 840 113549 1 9 7)
223 8:                    SET {
225 6:                      UTF8String 'asdasd'
:
233 13:          SEQUENCE {
235 9:            OBJECT IDENTIFIER sha256WithRSAEncryption (1 2 840 113549 1 1 11)
246 0:            NULL
:
248 129:          BIT STRING
:
248 129:            6C F7 BE 7F 02 42 8F 4B 53 0D 52 4E 9F E8 5C DC
:
248 129:            B0 7D BA B8 01 7D 00 8B FF 68 78 29 6E 55 5D 18
:
248 129:            A1 FF 01 47 E8 CF 26 64 17 7D B1 F4 A9 00 22 C2
```

Certificate signing request (CSR)

"A certification request consists of a distinguished name, a public key, and optionally a set of attributes, collectively signed by the entity requesting certification. Certification requests are sent to a certification authority, which transforms the request into an X.509 public-key certificate."

```
CertificationRequest ::= SEQUENCE {
    certificationRequestInfo CertificationRequestInfo,
    signatureAlgorithm AlgorithmIdentifier,
    signature          BIT STRING
}
```

```
CertificationRequestInfo ::= SEQUENCE {
    version        INTEGER v1(0),
    subject        Name,
    subjectPKInfo SubjectPublicKeyInfo,
    attributes     [0] IMPLICIT Attributes
}
```

PKCS#10: <https://tools.ietf.org/html/rfc2986>

Enrollment: Step 1



Symantec Corporation (US)

<https://website-security.geotrust.com/5938?toc=w88710382>

...

Note: * = required.

Business Email*

Country *

Estonia

Do you agree to receive communications from DigiCert about available offers and services?

Yes please, I'd like to hear about offers and services.

First Name*

Last Name*

Company*

Province

Phone Number*

What are you primarily interested in?*

Securing transactions & protecting sensitive info online



How many domains are you looking to secure?*

1-10



Do you have a shopping cart on your website?*

No



What is your timeframe for implementation?*

Immediate



Need a quote?*

No



Comments

Continue

By clicking Continue you agree to DigiCert, Inc. or its affiliates processing your data in accordance with DigiCert's [Privacy Policy](#).

Enrollment: Step 2

GeoTrust, Inc. (US)

https://products.geotrust.com/orders/enrollmer

SSL Trial

GeoTrust SSL Trial Enrollment

Enter CSR

After generating your server's Certificate Signing Request as described in [Generate CSR](#), paste the CSR in the form below. Please make sure that it contains the complete header and footer "BEGIN" and "END" lines exactly as in the example below.

Certificate Signing Request *

```
-----BEGIN CERTIFICATE REQUEST-----  
MIIBeDCB4gIBADAIoM0swCQYDVQQGEwJFRTEtMBEGA1UEAwwKYXV0aC51dC5lZTCB  
nzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEAlJTaMFCN7U25Hzx0WM08HDLNghFV  
ww+vCaeykd6CVnyh6A21z7XuSF3S0ZzN9pjZrfeE+gl6qW9V0CVYRJgfOyPcsHGd  
stcY6nKHGdykotKVgsGM348hPvn00xF9718N0kdPqc7B1n3E4M0eKYeHD3TRFo+p  
nRsvY7yLPbwp5z0CAwEAAsAXMBUGCSqGS1b3DQEJBzEIDAZhC2RhC20wDQYJKoZI  
hvcNAQELBQADgYEAbPe+fwJCj0tTDVJ0n+hc3LB9urgBfQCL/2h4KW5VXRih/gGn  
6M8mZBd9seSpACLCEnLBszT5D3jA0sLWP/J3vTUR938sdqmuV3y/KJP+aM0o9az+  
VSLECMi+B5qeX0Bu2YQLafQEpou8QhNe/arjX+MJvdhK1WwbVTxMKcibAM0=
```

-----END CERTIFICATE REQUEST-----

Back

Continue

Enrollment: Step 3

GeoTrust, Inc. (US) | <https://products.geotrust.com/orders/enrollment/>

SSL Trial

GeoTrust SSL Trial Enrollment

Verify Server URL

The CSR you generated is designed to work with the following URL:

<https://auth.ut.ee>

If this is not the correct URL (computed from the Common name in the CSR), or if any of the CSR Information below is incorrect, then please generate a new CSR and click the Replace CSR button.

[Replace CSR](#)

CSR Information

Note: The value for the Common Name must exactly match the name of the server you plan to secure.

Common Name: auth.ut.ee
Organization:
Org. Unit:
Locality:
State:
Country: EE

[Continue](#)

Enrollment: Step 4

GeoTrust, Inc. (US)

<https://products.geotrust.com/orders/enrollment/ContactInfo.do>

GeoTrust SSL Trial Enrollment

Approval of your certificate request

The GeoTrust SSL Trial service relies upon the Subscriber or the subscriber authorized administrator to approve all certificate requests for all hosts in the domain. We'll send an email to the domain contact in the domain's WHOIS record of each domain listed in your certificate information. To validate control of the domain, the owner of domain or an authorized representative must approve the request.

Select your authorized approvers

We send the approval email to the WHOIS contacts for each domain. In case the domain doesn't have any WHOIS contacts, we send the approval email to the pre-approved email addresses. You can customize the email recipients for each of your base domains if required.

ut.ee

No WHOIS contacts found for this domain. [View subdomains in your order](#)

No of Subdomains: 1

List of authorized approvers

You can select a different email address for this base domain, if required.

Pre-approved email addresses

- admin@ut.ee
- administrator@ut.ee
- hostmaster@ut.ee
- webmaster@ut.ee
- postmaster@ut.ee

Back

Continue

Task: Certificate issuer – 6p

Implement a utility that issues TLS server certificate based on a certificate signing request:

```
$ ./issue_cert.py  
usage: issue_cert.py private_key CA_cert csr output_cert  
  
$ ./issue_cert.py UT_priv.pem UT_rootCA.pem appcrypto.ee.csr issued.pem  
[+] Issuing certificate for "www.appcrypto.ee"  
  
$ openssl verify -CAfile UT_rootCA.pem -purpose sslserver issued.pem  
issued.pem: OK  
  
$ openssl verify -CAfile UT_rootCA.pem -purpose smimesign issued.pem  
CN = www.appcrypto.ee  
error 26 at 0 depth lookup: unsupported certificate purpose  
error issued.pem: verification failed
```

- Must support PEM/DER inputs, PEM output
- Sign subject's certificate using CA private key
- Use sha256WithRSAEncryption (1.2.840.113549.1.1.11)
- Use CN from subject's CSR DN (other fields must be ignored!)

Task: Certificate issuer

- Specify any serial you want
- Certificate must be valid for at least \pm 3 months (may hardcode)
- Fetch subject's public key from CSR (subjectPublicKeyInfo)
- Fetch issuer's distinguished name from CA certificate
- Critical extensions:
 - basic constraints CA:FALSE
 - key usage: digitalSignature
 - extended key usage: id-kp-serverAuth
- Use your own DER encoder and pyasn1 for decoding

Task: Certificate issuer

```
$ openssl x509 -in issued.pem -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 908108597 (0x3620a335)
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = EE, O = University of Tartu, OU = IT dep, CN = UT Root CA
        Validity
            Not Before: Jan 20 12:59:31 2020 GMT
            Not After : Jul 20 12:59:31 2020 GMT
        Subject: CN = www.appcrypto.ee
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            Public-Key: (2048 bit)
                Modulus:
                    01:4c:8b:43:f6:7f:12:fe:67:97:0d:10:3f:a3:26:
                    5c:b7:20:1d:fa:81:95:8d:5a:b0:9e:3e:10:f3:ef:
                    f4:fc:e3:a2:20:52:11:7d
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints: critical
                CA:FALSE
            X509v3 Key Usage: critical
                Digital Signature
            X509v3 Extended Key Usage: critical
                TLS Web Server Authentication
        Signature Algorithm: sha256WithRSAEncryption
            03:b0:61:34:de:82:8b:9a:3f:00:b0:73:95:27:00:09:11:6a:
            03:1d:b6:0b:51:00:c0:80:cb:1a:b0:07:21:fc:b1:f6:0c:cc:
```

Task: Hints

- pyasn1 will fail to decode CSR if it contains no attributes (challenge password) since it expects implicit tagging:
 - Make sure your CSR contains challenge password
- pyasn1 can easily encode decoded substructures:

```
encoder.encode(decoder.decode(der)[0][0][5])
```
- You may want to implement `asn1_bitstring_der()` which takes byte string as input (padding always 0x00)
- Read ASN.1 definitions or `dumpasn1` example certificates to find out DER encoding of certificate and its extensions

```
openssl x509 -inform pem -in cert.pem -outform der -out cert.der
```
- For debugging use two windows to compare your `dumpasn1` output with reference output

Questions

- What does PKI and X.509 certificates solve?
- Which are the two most important fields in X.509 certificate?
- Who defines trusted CAs for digital signature certificates?
- What is Hardware Security Module useful for?
- What browser checks in a certificate received from the server?
- Who defines trusted CAs for web server certificates?
- How are DV certificates different from OV certificates?
- How does CA verify whether the buyer owns the domain?

MTAT.07.017
Applied Cryptography

Certificate Revocation List (CRL)
Online Certificate Status Protocol (OCSP)

University of Tartu

Spring 2020

Certificate validity

It may be required to invalidate (revoke) a certificate before its expiration.

Examples:

- Private key compromised or lost
- Misissued certificate
- Data has changed
- Contract ended

Solution – Certificate Revocation List (CRL):

List of unexpired certificates that have been revoked by CA

- How can the relying party find the CRL?
- How is the integrity of CRL data assured?
- How frequently the CA should issue CRL?
- How frequently the relying party should refresh CRL?
- How can the relying party know that CRL is fresh?

CRL Distribution Points

General Details

Certificate Hierarchy

- ▼UTN-USERFirst-Hardware
 - ▼TERENA SSL CA
 - auth.ut.ee

Certificate Fields

- Subject's Public Key
- ▼Extensions
 - Certificate Authority Key Identifier
 - Certificate Subject Key ID
 - Certificate Key Usage
 - Certificate Basic Constraints
 - Extended Key Usage
 - Certificate Policies
 - CRL Distribution Points**
 - Authority Information Access
 - Certificate Subject Alt Name

Field Value

Not Critical
URI: http://crl.tcs.terena.org/TERENASSLCA.crl

Certificate Revocation List (CRL)

```
CertificateList ::= SEQUENCE {
    tbsCertList          TBSCertList,
    signatureAlgorithm   AlgorithmIdentifier,
    signatureValue       BIT STRING }
```



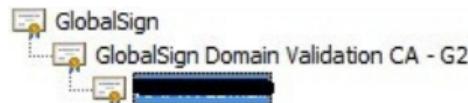
```
TBSCertList ::= SEQUENCE {
    version              Version OPTIONAL, -- if present, MUST be v2(1)
    signature            AlgorithmIdentifier,
    issuer               Name,
    thisUpdate           UTCTime,
    nextUpdate           UTCTime OPTIONAL,
    revokedCertificates SEQUENCE OF SEQUENCE {
        userCertificate     CertificateSerialNumber,
        revocationDate      UTCTime,
        crlEntryExtensions  Extensions OPTIONAL -- in v2 } OPTIONAL,
    crlExtensions        [0] EXPLICIT Extensions OPTIONAL -- in v2 }
```

<http://tools.ietf.org/html/rfc5280>

Certificate Revocation List (CRL)

- tbsCertList – DER structure to be signed by CRL issuer
- version – for v1 absent, for v2 contains 1
 - v2 introduces CRL and CRL Entry extensions
- signature – AlgorithmIdentifier from tbsCertList sequence
- issuer – identity of issuer who issued (signed) the CRL
 - CRL issued not by CA itself – indirect CRL
- thisUpdate – date when this CRL was issued
- nextUpdate – date when next CRL will be issued
- revokedCertificates – list of revoked certificates
 - userCertificate – serial number of revoked certificate
 - revocationDate – time when CA processed revocation request
 - crlEntryExtensions – provides additional revocation information
- crlExtensions – provides more information about CRL

Certificate chain



- How to validate a certificate chain?
- Where to look whether subject's certificate is not revoked?
 - In CRL issued by intermediate CA (usually every 12h)
 - Grace period
- Where to look whether intermediate CA is not revoked?
 - In CRL issued by root CA (usually every 3 month)
 - Grace period?!
- Where to look whether the root CA is not revoked?
 - In CRL issued by root CA itself (flawed)

Who is liable for actions made after the root CA private key has been compromised?

Liability analysis

Let's assume that subject's private key has been compromised.

Who (subject, CA or relying party) is liable for actions made with the key:

- in the time period after revocation information has appeared in CRL?
- in the time period after CRL has been issued but not available to relying parties (e.g., CA server downtime)?
- in the time period before next CRL has been issued?
- in the time period before CA has marked the certificate revoked in their internal database?
- in the time period before CA has been informed about the key compromise?

Questions

- How can the relying party find the CRL?
- How is the integrity of CRL data assured?
- How frequently the CA should issue CRL?
- How frequently the relying party should refresh CRL?
- How can the relying party know that CRL is fresh?
- How to verify if root CA certificate has not been revoked?
- Is the subject liable for transactions made after certificate is revoked?
- Is the subject liable for transactions made in certificate validity period?

Online Certificate Status Protocol

CRL shortcomings:

- Size of CRLs
- Client-side complexity
- Outdated status information

“The Online Certificate Status Protocol (OCSP) enables applications to determine the (revocation) state of an identified certificate.”

- Where can the relying party find the OCSP responder?
- How is the certificate identified in the OCSP request?
- How is the integrity of OCSP response assured?
- How to ensure the freshness of OCSP response?
- How frequently certificate status should be checked?

Authority Information Access

Certificate Hierarchy

- ▼DigiCert High Assurance EV Root CA
 - ▼DigiCert SHA2 High Assurance Server CA
 - *.eesti.ee

Certificate Fields

	Certificate Subject Key ID
	Certificate Subject Alt Name
	Certificate Key Usage
	Extended Key Usage
	CRL Distribution Points
	Certificate Policies
	Authority Information Access
	Certificate Basic Constraints
	Certificate Signature Algorithm
	Certificate Signature Value

Field Value

Not Critical
OCSP: URI: <http://ocsp.digicert.com>
CA Issuers: URI: <http://cacerts.digicert.com/DigiCertSHA2HighAssuranceServerCA.crt>

OCSP over HTTP

Follow TCP Stream

Stream Content

```
POST / HTTP/1.0
Content-Type: application/ocsp-request
Content-Length: 120

0v0t0M0K0I0...+.....1....6..2\ch....a.I.....4E@=.00..>.....
...9.7w.+.....#0!0...+....0.....K..4".Z..T. ]HTTP/1.0 200 Ok
last-modified: Wed, 07 Mar 2012 18:19:19 GMT
expires: Wed, 14 Mar 2012 18:19:19 GMT
content-type: application/ocsp-response
content-transfer-encoding: binary
content-length: 1165
cache-control: max-age=514527, public, no-transform, must-revalidate
date: Thu, 08 Mar 2012 19:23:52 GMT
connection: close

0...
.....0..~..+.....0.....00..k0.....J0H1.0...U....US1.0...U.
..Thawte, Inc.1"0 ..U....Thawte SSL OCSP Responder..20120307181919Z0s0q0I0...+.....1....6..2
\ch....a.I.....4E@=.00..>.....
...9.7w.+.....20120307181919Z...20120314181919Z0
...*H..
.....R..)c>csh.4....t..j}WS.....
ct...a.]'IU.~Y..v[..\...).D...%T...].o.(?
....@aY....~D..Q\..U.j...>....)....u....415....-9.....0....0....0.....9....E.....0
...*H..
.....0<1.0....U....US1.0....U.
..Thawte, Inc.1.0....U...
```

Find Save As Print Entire conversation (1694 bytes) ASCII EBCDIC Hex Dump C Arrays Raw

[Help](#) [Filter Out This Stream](#) [Close](#)

Request syntax

```
OCSPRequest ::= SEQUENCE {  
    tbsRequest TBSRequest,  
    optionalSignature [0] Signature OPTIONAL }  
  
Signature ::= SEQUENCE {  
    signatureAlgorithm AlgorithmIdentifier,  
    signature          BIT STRING,  
    certs             [0] SEQUENCE OF Certificate OPTIONAL }  
  
TBSRequest ::= SEQUENCE {  
    version          [0] Version DEFAULT v1(0),  
    requestorName   [1] GeneralName OPTIONAL,  
    requestList     SEQUENCE OF SEQUENCE {  
        reqCert         CertID,  
        singleRequestExtensions [0] Extensions OPTIONAL }  
    requestExtensions [2] Extensions OPTIONAL }  
  
CertID ::= SEQUENCE {  
    hashAlgorithm    AlgorithmIdentifier,  
    issuerNameHash  OCTET STRING, -- Hash of Issuer's DN  
    issuerKeyHash   OCTET STRING, -- Hash of Issuer's public key  
                      (i.e., hash of subjectPublicKey BIT STRING content)  
    serialNumber     CertificateSerialNumber }
```

Response syntax

```
OCSPResponse ::= SEQUENCE {
    responseStatus      OCSPResponseStatus,
    responseBytes       [0] EXPLICIT ResponseBytes OPTIONAL }

OCSPResponseStatus ::= ENUMERATED {
    successful          (0), --Response has valid confirmations
    malformedRequest    (1), --Illegal confirmation request
    internalError       (2), --Internal error in issuer
    tryLater            (3), --Try again later
                        --(4) is not used
    sigRequired         (5), --Must sign the request
    unauthorized        (6)  --Request unauthorized
}

ResponseBytes ::=     SEQUENCE {
    responseType   OBJECT IDENTIFIER, --id-pkix-ocsp-basic
    response        OCTET STRING }
```

- `responseBytes` provided only if `responseStatus` is “successful”

Response syntax

```
response ::= SEQUENCE {
    tbsResponseData      ResponseData,
    signatureAlgorithm   AlgorithmIdentifier,
    signature            BIT STRING,
    certs                [0] EXPLICIT SEQUENCE OF Certificate OPTIONAL }

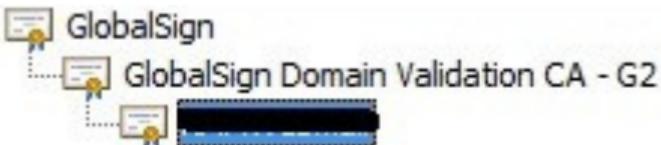
ResponseData ::= SEQUENCE {
    version              [0] EXPLICIT Version DEFAULT v1,
    responderID         [1] Name,
    producedAt          GeneralizedTime,
    responses            SEQUENCE OF SEQUENCE {
        certID             CertID,
        certStatus         CertStatus,
        thisUpdate         GeneralizedTime,
        nextUpdate         [0] EXPLICIT GeneralizedTime OPTIONAL,
        singleExtensions  [1] EXPLICIT Extensions OPTIONAL }
    responseExtensions  [1] EXPLICIT Extensions OPTIONAL }

CertStatus ::= CHOICE {
    good                [0] IMPLICIT NULL,
    revoked             [1] IMPLICIT SEQUENCE {
        revocationTime   GeneralizedTime,
        revocationReason [0] EXPLICIT CRLReason OPTIONAL }
    unknown             [2] IMPLICIT NULL }
```

How to check the freshness of response?

- Check the signed timestamp (producedAt and thisUpdate)
 - What should be the allowed time difference?
 - Replay attacks
 - Reliance on the correctness of system clock
- Include nonce in the OCSP request and check it in the response
 - OCSP nonce extension (optional)
 - Prevents replay attacks
 - Vulnerable to downgrade attacks

Who signs OCSP response?

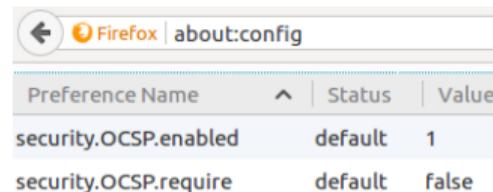


The key used to sign the response MUST belong to one of the following:

- CA who issued the certificate in question
- CA Authorized Responder who holds a specially marked certificate issued directly by the CA, indicating that the responder may issue OCSP responses for that CA
 - OCSP signing delegation SHALL be designated by the inclusion of `id-kp-OCSPSigning` flag in an `extendedKeyUsage` extension of the responder's certificate
 - How to check the revocation status of this certificate?
- Trusted Responder whose public key is trusted by the requester
 - Trust must be established by some out-of-band means

Revocation checking in browsers

- CRLs are not supported
- Problems with OCSP:
 - Privacy leakage
 - Slower initial page loading
 - Chrome uses OCSP only to check EV certificates (uses CRLSets)
 - Firefox is not brave enough to fail-safe:



Preference Name	Status	Value
security.OCSP.enabled	default	1
security.OCSP.require	default	false

- Solution is OCSP stapling (web server provides OCSP response to the browser)
- Shorter certificate validity period may help
- How frequently the OCSP status should be queried?

Questions

- Where can the relying party find the OCSP responder?
- How is the certificate identified in the OCSP request?
- How is the integrity of OCSP response assured?
- How to ensure the freshness of OCSP response?
- How frequently the validity status should be checked?
- What problem does the OCSP nonce extension solve?
- What is a downgrade attack?

Hypertext Transfer Protocol (HTTP)

- Application layer client-server, request-response protocol
- Runs over TCP (Transmission Control Protocol) port 80

Client request (<http://example.com/hello>):

GET /hello HTTP/1.1	POST /hello HTTP/1.1
Host: example.com	Host: example.com
Connection: close	Content-Length: 24
	Connection: close

Server response:

sending_this_binary_blob

HTTP/1.1 200 OK
Date: Wed, 24 Mar 2020 19:35:42 GMT
Server: Apache
Content-Length: 7033
Content-Type: text/html

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Tran...>

- Header lines must all end with <CR><LF> (b"\r\n")
- Header lines are separated from the body by an empty line
- POST requests have a non-empty request body

Sockets in Python

```
>>> import socket  
>>> s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  
>>> s.connect(("example.com", 80))  
>>> s.send(b'GET / HTTP/1.1\r\nHost: example.com\r\n\r\n')  
37  
>>> print(s.recv(20))  
b'HTTP/1.1 200 OK\r\nAge'
```

- `recv()` returns bytes that are available in the read buffer
- `recv()` will wait if the read buffer empty (blocking by default)
- `recv()` will return 0 bytes if the connection is closed
- You must know how many bytes you must get
- Correct way to read HTTP response:
 - Read byte-by-byte until the full response header is received
 - Extract body size from Content-Length header
 - Read byte-by-byte until the full response body is received
 - Avoid endless loops by checking the return value of `recv()`

Task: OCSP checker – 5p

Implement a utility that queries OCSP for certificate validity:

```
$ ./ocsp_check.py valid.pem
[+] URL of OCSP responder: http://aia.sk.ee/esteid2018
[+] Downloading issuer certificate from: http://c.sk.ee/esteid2018.der.crt
[+] Querying OCSP for serial: 132457411991227041950906933396399710966
[+] Connecting to aia.sk.ee...
[+] OCSP producedAt: 2020-03-23 16:36:40
[+] OCSP thisUpdate: 2020-03-23 16:36:40
[+] OCSP status: good

$ ./ocsp_check.py revoked.pem
[+] URL of OCSP responder: http://aia.sk.ee/esteid2015
[+] Downloading issuer certificate from: http://c.sk.ee/ESTEID-SK_2015.der.crt
[+] Querying OCSP for serial: 165400448864000643393593611773932020928
[+] Connecting to aia.sk.ee...
[+] OCSP producedAt: 2020-03-23 16:36:44
[+] OCSP thisUpdate: 2020-03-23 16:36:44
[+] OCSP status: revoked
```

Task: OCSP checker

- Extract OCSP responder's URL and CA certificate URL from certificate's Authority Information Access (AIA) extension
- Send HTTP requests using Python sockets (the correct way!)
- Use urlparse for easy URL parsing:

```
>>> from urllib.parse import urlparse  
>> urlparse("http://example.com/abc")  
ParseResult(scheme='http', netloc='example.com', path='/abc', params='', query='', fragment='')  
>>> urlparse.urlparse("http://example.com/abc").netloc  
'example.com'
```

- Use regular expression to get length of HTTP response body:

```
>>> import re  
>>> re.search('content-length:\s*(\d+)\s', header.decode(), re.S+re.I).group(1)
```

- Construct OCSP request using your ASN.1 DER encoder
- OCSP response parsing code is in the template
- Signature verification checks can be skipped

Task: OCSP checker

- OCSP request must include “Content-Type: application/ocsp-request”
- aia.sk.ee returns “revoked” for unrecognized CertIDs
- dumpasn1 fails to decode OCSP request
 - use openssl asn1parse
- OCSP request for valid.pem:

```
$ openssl asn1parse -inform der -in valid.pem_ocsp_req
 0:d=0  hl=2 l=  81 cons: SEQUENCE
 2:d=1  hl=2 l=  79 cons: SEQUENCE
 4:d=2  hl=2 l=  77 cons: SEQUENCE
 6:d=3  hl=2 l=  75 cons: SEQUENCE
 8:d=4  hl=2 l=  73 cons: SEQUENCE
10:d=5  hl=2 l=   9 cons: SEQUENCE
12:d=6  hl=2 l=   5 prim: OBJECT            :sha1
19:d=6  hl=2 l=   0 prim: NULL
21:d=5  hl=2 l=  20 prim: OCTET STRING      [HEX DUMP]:455DBEF01E1E2B1058EF1F969918A80708A62182
43:d=5  hl=2 l=  20 prim: OCTET STRING      [HEX DUMP]:D9AC70DB5F7EBE94F8AOE4BE47A2D034AD9A2A12
65:d=5  hl=2 l=  16 prim: INTEGER           :63A65E9ED37BF0115C2C8928DF0FE2F6
```

Comments

Wrong way to download HTTP response body:

- Reading the response in one go (**wrong!**):

```
body = s.recv(content_length)
```

"The receive calls normally return any data available, up to the requested amount, rather than waiting for receipt of the full amount requested." (man page recv section 2)

- Reading until the socket is closed (**wrong!**):

```
body = b''
buf = s.recv(1024)
while len(buf):
    buf = s.recv(1024)
    body+= buf
```

After sending a response, an HTTP/1.1 server will wait for more request/response exchanges, unless header "Connection: close" was specified by the client. Therefore `s.recv()` will hang until the timeout configured by the server is reached.

MTAT.07.017
Applied Cryptography

Digital Signatures (XAdES)

University of Tartu

Spring 2020

eIDAS Regulation¹

Article 3:

- (10) 'electronic signature (ES)' means data in electronic form which is attached to or logically associated with other data in electronic form and which is used by the signatory to sign;
- (11) 'advanced electronic signature (AdES)' means an electronic signature which meets the requirements set out in Article 26 (*uniquely linked to the signatory*);
- (12) 'qualified electronic signature (QES)' means an advanced electronic signature that is created by a qualified electronic signature creation device, and which is based on a qualified certificate for electronic signatures;

Article 25:

2. A qualified electronic signature shall have the equivalent legal effect of a handwritten signature.

¹ Regulation (EU) No 910/2014 of the European Parliament and of the Council of 23 July 2014 on electronic identification and trust services for electronic transactions in the internal market and repealing Directive 1999/93/EC

Estonian Digital Signature – QES

Küberturvalisuse seadus.bdoc

No card readers found

Help Settings

Container: /tmp/mozilla_user0/Küberturvalisuse seadus.bdoc

SIGNATURE

CRYPTO

My eID

Ver. 4.2.2.51

Container files

VP_14052018_o252.rtf

Container signatures

KERSTI KALJULAID - Signature is valid
46912302711 - Signed on 14. May 2018 at 12:04

START ENCRYPT SAVE AS SEND WITH E-MAIL SIGN WITH MOBIIL-ID

Legal effect of Qualified Electronic Signature

The same as of a handwritten signature

- Can be used to sign contracts
 - Most of the contracts do not have to be in writing
- Must be used to sign legal acts
- Can be used as an evidence in court
 - Can unsigned e-mails be used as a proof?

Code of Civil Procedure:

§ 277. Contestation of authenticity of documents

(3) Authenticity of an electronic document bearing a digital signature may be contested only by substantiating the circumstances which give reason to presume that the document has not been prepared by the holder of the digital signature.

- What could be these circumstances?

Qualified Trust Service Provider (QTSP) – CA

- Only a QTSP may issue qualified certificate for electronic signature
- Qualified status granted by supervisory body
 - Estonian Information System Authority (RIA)

Article 22

1. Each Member State shall establish, maintain and publish trusted lists, including information related to the qualified trust service providers for which it is responsible, together with information related to the qualified trust services provided by them.

- Estonian Trusted List (signed by RIA):
<https://sr.riik.ee/tsl/estonian-tsl.xml>
- EU List of Trusted Lists (signed by Maarten Joris Ottoy): https://ec.europa.eu/information_society/policy/esignature/trusted-list/tl-mp.xml
- EU-level PKI

Qualified Electronic Signature Creation Device (QSCD)

ANNEX II

1. Qualified electronic signature creation devices shall ensure, by appropriate technical and procedural means, that at least:
 - (a) the confidentiality of the electronic signature creation data used for electronic signature creation is reasonably assured;
 - (b) the electronic signature creation data used for electronic signature creation can practically occur only once;
 - (c) the electronic signature creation data used for electronic signature creation cannot, with reasonable assurance, be derived and the electronic signature is reliably protected against forgery using currently available technology;
 - (d) the electronic signature creation data used for electronic signature creation can be reliably protected by the legitimate signatory against use by others.

Security certification according to Common Criteria EAL4+

- EN 419 211 - Protection profiles for secure signature creation device²

²https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=uriserv:OJ.L_.2016.109.01.0040.01.ENG

Validation of Qualified Electronic Signature

Article 32

1. The process for the validation of a qualified electronic signature shall confirm the validity of a qualified electronic signature provided that:
 - (a) the certificate that supports the signature was, at the time of signing, a qualified certificate for electronic signature complying with Annex I;
 - (b) the qualified certificate was issued by a qualified trust service provider and was valid at the time of signing;
 - (c) the signature validation data corresponds to the data provided to the relying party;
 - (d) the unique set of data representing the signatory in the certificate is correctly provided to the relying party;
 - (e) the use of any pseudonym is clearly indicated to the relying party if a pseudonym was used at the time of signing;
 - (f) the electronic signature was created by a qualified electronic signature creation device;
 - (g) the integrity of the signed data has not been compromised;
 - (h) the requirements provided for in Article 26 were met at the time of signing.

Trusted Timestamping

Article 3:

(33) 'electronic time stamp' means data in electronic form which binds other data in electronic form to a particular time establishing evidence that the latter data existed at that time;

<http://tools.ietf.org/html/rfc3161>

Signed statement of timestamping authority (TSA):

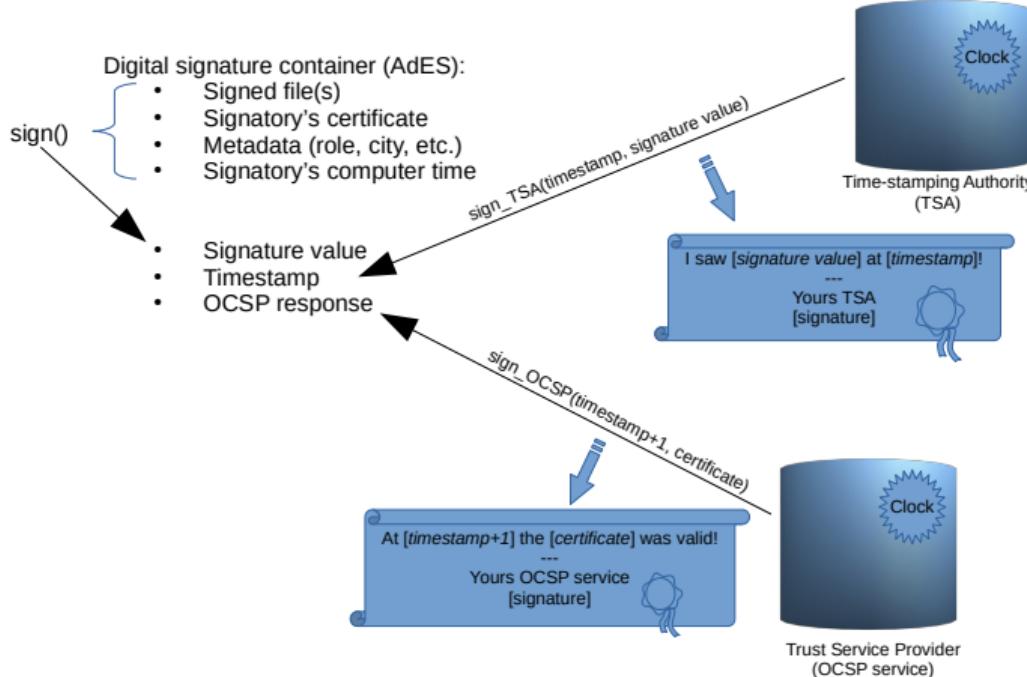
```
> This data [data] was presented to me at this time: [time]
> --
> TSA [signature]
```

- TSA must use accurate time source
- TSA must log issued timestamps

Article 41:

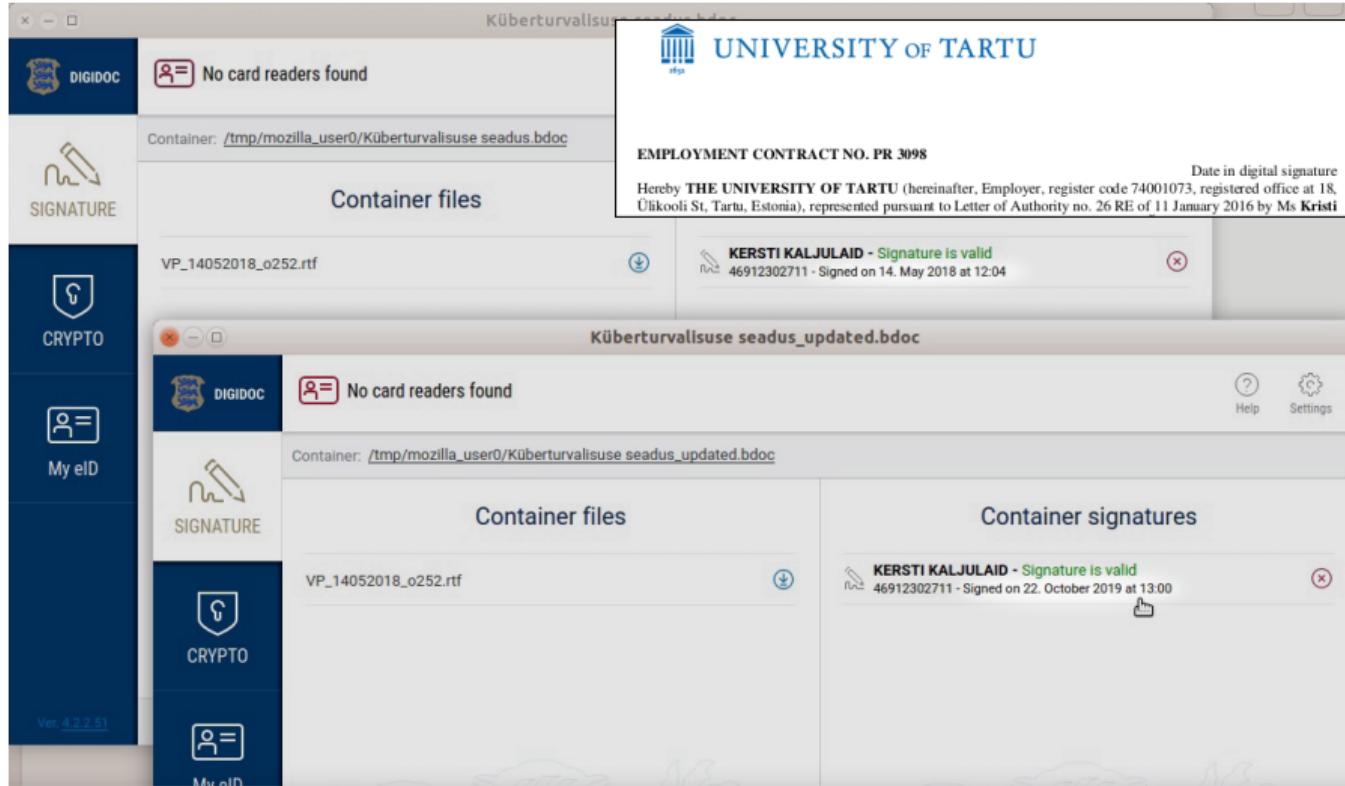
A qualified electronic time stamp shall enjoy the presumption of the accuracy of the date and the time it indicates and the integrity of the data to which the date and time are bound.

Digital signature container



- OCSP request must be made *after* the timestamp (<https://www.youtube.com/watch?v=eYG17IGOCi0>)
- Unable to verify whether the signature was not made while the certificate was suspended

Time of signing



- Modifying the time of signing (<https://www.youtube.com/watch?v=ysYouhl1Yx4>)

Digital Signature File Formats

Advanced electronic signature (AdES) file format specifications to be recognised by public sector bodies³:

- XML – XAdES Baseline Profile (ETSI TS 103171)
 - Used in Estonia (.asice/.bdoc/.ddoc formats)
- CMS – CAdES Baseline Profile (ETSI TS 103173)
- PDF – PAdES Baseline Profile (ETSI TS 103172)

Implemented using Associated Signature Container (ASiC) Baseline Profile (ETSI TS 103174)

```
asic-container.asice: Zip ("application/vnd.etsi.asic-e+zip")
+ mimetype
+ document.docx
+ META-INF/manifest.xml
+ META-INF/signatures0.xml
+ META-INF/signatures1.xml
```

³ https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=OJ:JOL_2015_235_R_0006

XML Signature (ASICE)

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<asic:XAdESSignatures xmlns:asic="http://uri.etsi.org/02918/v1.2.1#" [...]>

<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#" Id="S1">

    <ds:SignedInfo Id="S1-SignedInfo">
        <ds:SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256"/>
        <ds:Reference Id="S1-ref-1" URI="document.docx">
            <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
            <ds:DigestValue>SJ07h/iCeb9jDLXMQ6qEx8nYkhNR+MWBLge6YfyU7+U=</ds:DigestValue>
        </ds:Reference>
        <ds:Reference Id="S1-ref-SignedProperties" Type="http://uri.etsi.org/01903#SignedProperties" URI="#S1-SignedProperties">
            <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
            <ds:DigestValue>qRlc2fxIYkqe3/1sHpZuk+eBKMZ7rIsqBZbYhigV5g=</ds:DigestValue>
        </ds:Reference>
    </ds:SignedInfo>

    <ds:SignatureValue>MtKIGLOB...3D62QA==</ds:SignatureValue> - signature of <SignedInfo>
    <ds:X509Certificate>MIIiyDCCB7...SVU=</ds:X509Certificate>

    <xades:SignedProperties Id="S1-SignedProperties">
        <xades:SigningTime>2018-06-05T15:01:11Z</xades:SigningTime>
        <xades:SigningCertificate>
            <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
            <ds:DigestValue>cuPIt8LpJIs+eFUgUwIrNsUiKaH/NTezVgkRXixABBo=</ds:DigestValue>
        </xades:SigningCertificate>
        ...
        <xades:DataObjectFormat ObjectReference="#S1-ref-1">
            <xades:MimeType>application/octet-stream</xades:MimeType>
        </xades:DataObjectFormat>
        ...
    </xades:SignedProperties>

    <xades:SignatureTimeStamp>...
    <xades:CertificateValues>...
    <xades:OCSPValues>...
    ...
</ds:Signature>
...
</asic:XAdESSignatures>
```

XML Signature (BDOC and DDOC)

BDOC (2015 – today):

- OCSP response serves also as a timestamp
- Hash of signature included in OCSP nonce extension
- This hack is not recognized by eIDAS standards

DDOC (2002 – 2017):

- Single self-contained XML file
- Signed files base64-encoded in Datafile element
- Supports only SHA-1

```
<?xml version="1.0" encoding="UTF-8"?>
<SignedDoc format="DIGIDOC-XML" version="1.3" xmlns="http://www.sk.ee/DigiDoc/v1.3.0#">
    <DataFile Filename="document.doc" Id="D0">UEsDBBQABgA...AS1EAAAAA</DataFile>
    <Signature Id="S0">
```

```
        <SignedInfo>
            <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
            <Reference URI="#D0">
                <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                <DigestValue>Q43ti5R/wg18q0oHsygLFTXEOqU=</DigestValue>
            </Reference>
            <Reference URI="#S0-SignedProperties">
                <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                <DigestValue>GOHmQqHCqMxULzfWS0Nl2i0mIU=</DigestValue>
            </Reference>
        </SignedInfo>
```

```
        <SignatureValue Id="S0-SIG">kgsCQ6...M4rkcj8=</SignatureValue>
        <X509Certificate>IID4z....V8APa</X509Certificate>
```

Task: ASiC-E XAdES verifier – 5p

Implement a utility that verifies ASICE XAdES digital signature:

```
$ ./asice_verify.py good.asice
[+] Signatory: PARŠOVS, ARNIS, 38608050013
[+] Signed file: hello.txt
[+] Timestamped: 2019-04-03 14:04:52
[+] Signature verification successful!

$ ./asice_verify.py forgery1.asice
[+] Signatory: PARŠOVS, ARNIS, 38608050013
[+] Signed file: hello.txt
[-] a wrong certificate hash included under the signature!

$ ./asice_verify.py forgery(2|3|4|5|6|7|8).asice
[...]
```

- The error messages have to be meaningful
- Must support:
 - a single signed file and a single signature
 - only the algorithms used in the testcase file
- Not required to verify certificates and signature on:
 - certificates, timestamp, OCSP response

Task: ASiC-E XAdES verifier

- Hash is calculated on canonicalized XML elements
 - Hash of canonicalized <SignatureValue> is timestamped
- Code for parsing timestamp and OCSP response is in the template
- Use zipfile for reading ZIP container:

```
import zipfile
archive = zipfile.ZipFile(filename, 'r')
xml = archive.read('META-INF/signatures0.xml')
```

- Use BeautifulSoup for XML traversal:

```
from bs4 import BeautifulSoup
x = BeautifulSoup(xml, features="xml")
x.XAdESSignatures.KeyInfo.X509Data.X509Certificate.encode_contents()
x.XAdESSignatures.Signature.SignedInfo.Reference['URI']
x.XAdESSignatures.Signature.SignedInfo.find('Reference',
                                              attrs={'URI': '#S0-SignedProperties'})
```

Questions

- The main requirements for signature to have QES status?
- The benefits of QES compared to electronic signature?
- Can the authenticity of QES be contested?
- Can unsigned e-mail be used as a proof in the court?
- How can the TSP become a QTSP?
- What is required for a product to be recognized as a QSCD?
- Why MIME type and certificate are included under the signature?
- How to prove that the certificate was valid at the time of signing?
- Will it be possible to verify ASICE signature after TSA/OCSP certificates expire?

MTAT.07.017

Applied Cryptography

Smart Cards (EstEID)

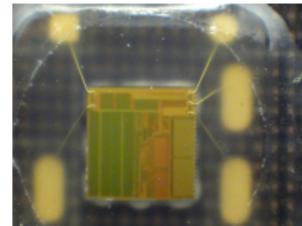
University of Tartu

Spring 2020

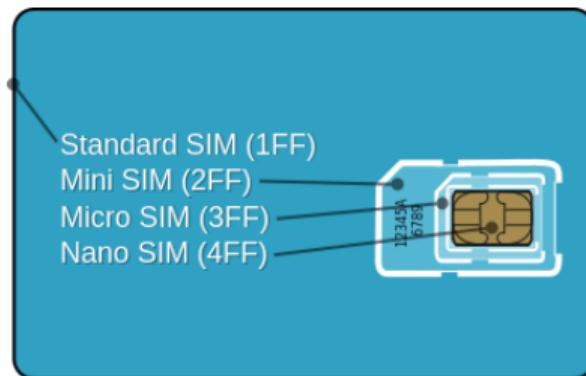
Smart card

Chip card or integrated circuit card (ICC)

Contains protected non-volatile memory and microprocessor



ISO/IEC 7816 defines dimensions and location of the contacts, electrical interface, transmission protocols, etc.



- Contact smart cards
- Contactless smart cards
- Dual interface cards

Smart card communication

APDU: Application Protocol Data Unit

terminal → card: command

terminal ← card: response

- Command APDU:

[CLA] [INS] [P1] [P2] [L_c] [C_{data}] ... [L_e]

Header (5 bytes) + data (0 ... 255 bytes)

Case 1: 00 a4 00 0c [00]

Case 2: 00 b2 01 0c ff

Case 3: 00 a4 01 0c 02 ee ee

Case 4: 00 a4 01 00 02 ee ee ff

- Response APDU:

[R_{data}] ... [SW1] [SW2]

Data (0 ... 256 bytes) + status word (2 bytes)

62 00

45 53 54 90 00

Standard commands and responses

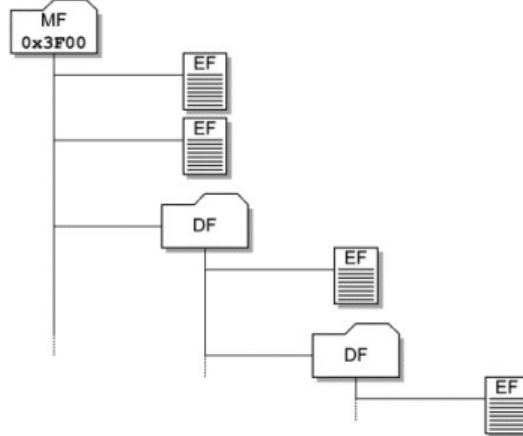
#	ClaIns P1 P2	Lc	Send Data	Le	Recv Data	Specification	Description
A0 04 00 00 00				3GPP TS 11.11		INVALIDATE	
84 16 00 00 xx MAC				VSDC		CARD BLOCK	
A0 20 00 xx 08 CHV Value				3GPP TS 11.11		VERIFY	
00 82 00 xx 06 Manual				GEMPLUS MPCOS-EMV		EXTERNAL AUTHENTICATE	
00 84 xx xx		08	Rnd Num	GEMPLUS MPCOS-EMV		GET CHALLENGE	
00 88 XX xx 0A Manual				GEMPLUS MPCOS-EMV		INTERNAL AUTHENTICATE	
A0 88 00 00 10 RAND : Rnd num		xx	SRES(4B)	3GPP TS 11.11		RUN GSM ALGORITHM	
A0 A2 00 xx xx Pattern				3GPP TS 11.11		SEEK	
00 A4 04 00 xx AID		00		GlobalPlatform		SELECT	
A0 A4 00 00 02 File ID				3GPP TS 11.11		SELECT	
A0 B0 xx xx		xx		3GPP TS 11.11		READ BINARY	
00 B2 xx			00	VSDC		READ RECORD	
00 C0			1C Key Info	GlobalPlatform		GET RESPONSE	
80 CA xx xx xx				VSDC		GET DATA	
80 D0 xx xx xx	Data to be written in EEPROM			VSDC		LOAD STRUCTURE	
A0 D6 xx xx xx	Data to be written in EEPROM			3GPP TS 11.11		UPDATE BINARY	
00 DA xx xx xx	Data			VSDC		PUT DATA	
00 DC xx xx xx	Data (and MAC)			VSDC		UPDATE RECORD	
80 E0 xx xx xx	FCI length			3GPP TS 11.11		CREATE FILE	
00 E2 00 00 xx	Record			3GPP TS 11.11		APPEND RECORD	
A0 E4 00 00 02 xx xx				3GPP TS 11.11		DELETE FILE	
...					...		

#	SW1 SW2	Message
'6X XX'	Transmission protocol related codes	
'61 XX'	SW2 indicates the number of response bytes still available	
'62 00'	No information given	
'62 82'	The end of the file has been reached before the end of reading	
'62 83'	Invalid DF	
'62 84'	Selected file is not valid. File descriptor error	
'6A 00'	Bytes P1 and/or P2 are incorrect.	
'6A 82'	File not found	
'6A 83'	Record not found	
'9F XX'	Success, XX bytes of data available to be read via "Get_Response" task.	
...	...	

<http://web.archive.org/web/20090630004017/http://cheef.ru/docs/HowTo/APDU.info>

<http://web.archive.org/web/20090623030155/http://cheef.ru/docs/HowTo/SW1SW2.info>

Smart card file system



- Adressable objects:
 - MF – Master File (root directory)
 - DF – Dedicated File (directory)
 - EF – Elementary File (data file)
- 2-byte file identifier (FID)
- There is no ls/dir command!
- Legacy

Using SELECT FILE

To change pointer to Dedicated File EEEE:

[0x00, 0xA4, 0x01, 0x0C, 0x02, 0xEE, 0xEE]

- CLA - 0x00
- INS - 0xA4 (command - SELECT FILE)
- P1 - what type of object to select
 - 0x00 - Master File (root)
 - 0x01 - Dedicated File (directory)
 - 0x02 - Elementary File (data file)
 - 0x04 - Card Application (chip applet)
- P2 - type of response
 - 0x00 - Include object description FCI (FCP+FMD)
 - 0x04 - Include object description FCP (file control parameters)
 - 0x08 - Include object description FMD (file management data)
 - 0x0C - Do not respond with description
- Lc - length of file identifier
- Data - file identifier for EF, DF or AID for application

Answer To Reset (ATR)

“Bytes returned by a contact smart card on power up.
Conveys information about the parameters proposed by the card.”

```
$ pcsc_scan
ATR: 3B FA 18 00 00 80 31 FE 45 FE 65 49 44 20 2F 20 50 4B 49 03
+ TS = 3B --> Direct Convention
+ TO = FA, Y(1): 1111, K: 10 (historical bytes)
TA(1) = 18 --> Fi=372, Di=12, 31 cycles/ETU
    129032 bits/s at 4 MHz, fMax for Fi = 5 MHz => 161290 bits/s
TB(1) = 00 --> VPP is not electrically connected
TC(1) = 00 --> Extra guard time: 0
TD(1) = 80 --> Y(i+1) = 1000, Protocol T = 0
TD(2) = 31 --> Y(i+1) = 0011, Protocol T = 1
TA(3) = FE --> IFSC: 254
TB(3) = 45 --> Block Waiting Integer: 4 - Character Waiting Integer: 5
+ Historical bytes: FE 65 49 44 20 2F 20 50 4B 49
    Category indicator byte: FE (proprietary format)
+ TCK = 03 (correct checksum)
Possibly identified card (using /home/user/.cache-smartcard_list.txt):
3B FA 18 00 00 80 31 FE 45 FE 65 49 44 20 2F 20 50 4B 49 03
    Estonian Identity Card (EstEID v3.5 (10.2014) cold) (eID)
```

Historical bytes can be used to identify the card:

```
>>> "FE654944202F20504B49".decode('hex')
'\xfeeID / PKI'
```

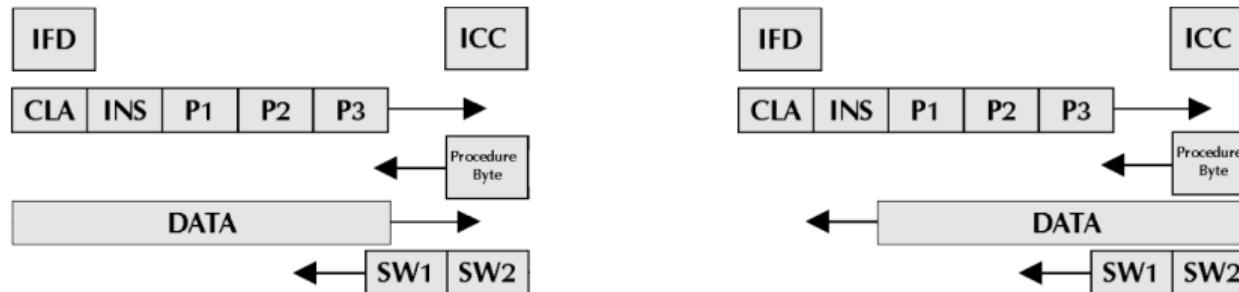
Some cards can return two different ATRs:

- Cold ATR – when power is supplied to the card
- Warm ATR – when reset signal is sent

Transmission protocols ($T=0/T=1$)

$T=0$ (byte-oriented protocol):

- APDU is sent over the wire as it is
- In one round data can be sent only in one direction:



- It must be known in which direction the data will be sent
- Data sent/received must be exactly P3 bytes in length

To return data if terminal also sent data:

1. Card responds with SW: 61 XX
2. Terminal sends GET RESPONSE: 00 C0 00 00 XX
3. Card returns XX bytes + SW

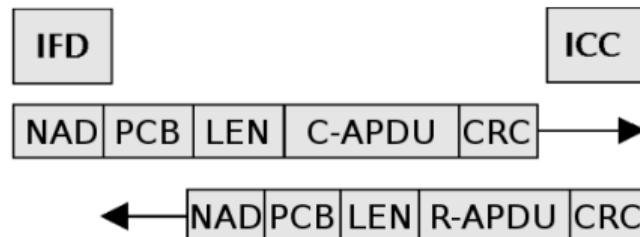
If APDU contains incorrect L_e value:

1. Card responds with SW: 6C XX
2. Terminal resends the command with L_e set to XX
3. Card returns XX bytes + SW

Transmission protocols ($T=0/T=1$)

$T=1$ (block-oriented protocol):

- APDUs are encapsulated in blocks forming Transport Protocol Data Units



- In one round data can be sent in both directions
- Supports extended APDU (max 65'535 bytes)
- More advanced error detection

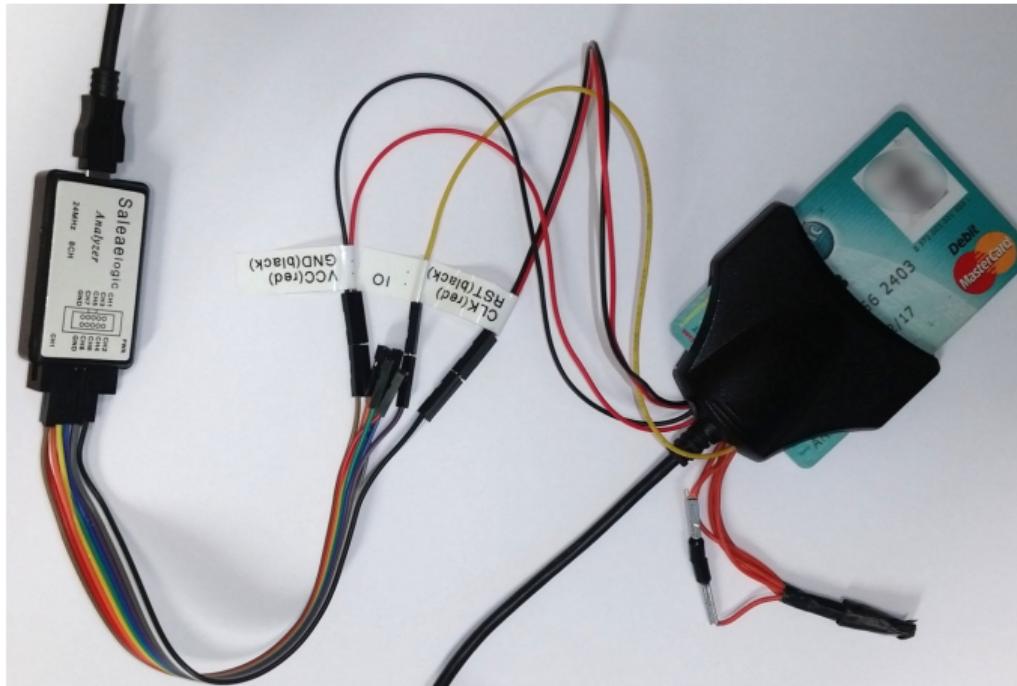
The same APDU can be sent over both $T=0$ and $T=1$ electrical protocols (unless using $T=1$ extended APDU feature).

Electrical communication



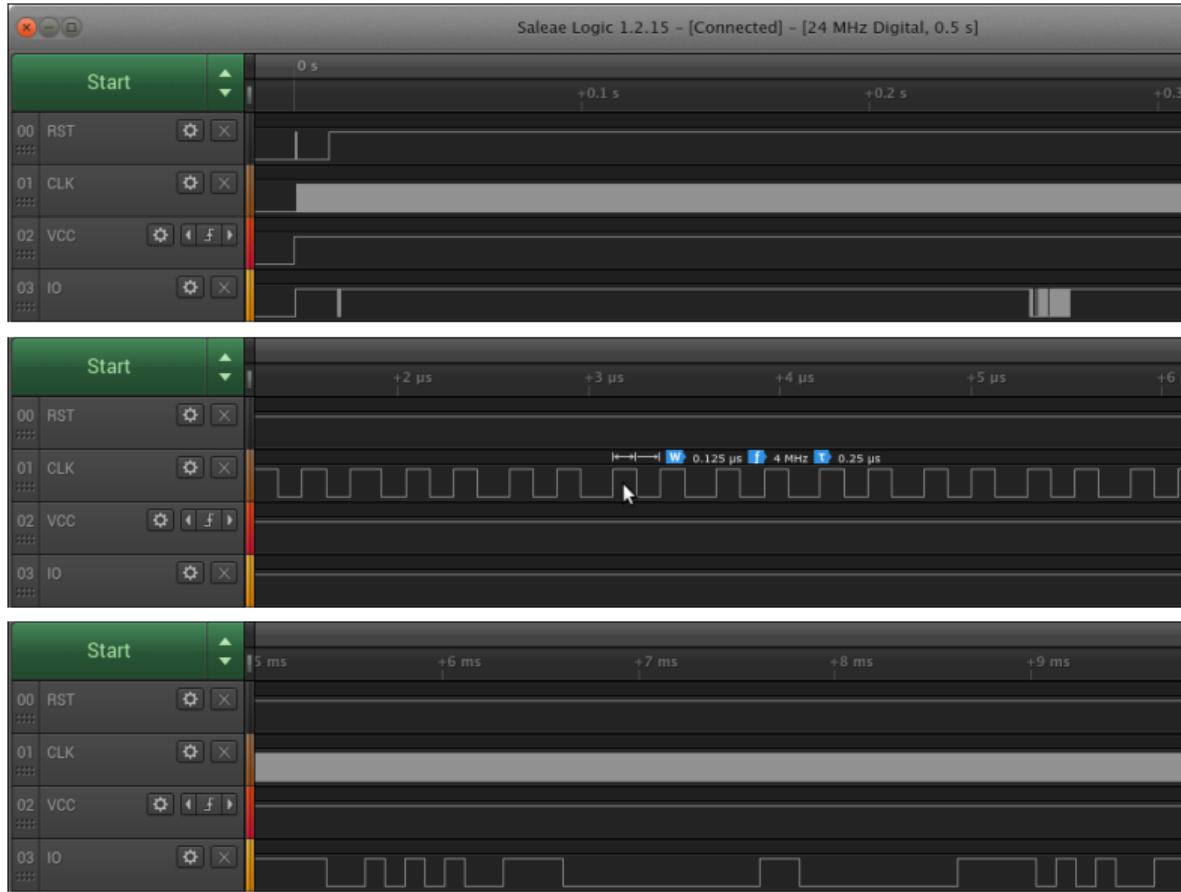
- VCC – Positive supply voltage (1.8V, 3.0V, 5.0V)
- GND – Ground
- RST – Reset signal to boot and restart the card
- I/O – Synchronous data transmission
- CLK – Clock signal for I/O sync (1 MHz to 5MHz)
 - Used by older cards to clock CPU
- C4, C8, C6 – not used

Sniffing electrical communication



- Saleae USB Logic Analyzer 8CH 24MHz – \$220
 - Chinese clone on eBay – EUR 6
- Wires soldered to the reader contacts (I/O, VCC, RST, CLK)

Sniffing electrical communication



Estonian Electronic Identity Card (EstEID)



Nationwide PKI:

- Subjects – Estonian (e-)residents
 - Certificate Authority (QTSP) – SK ID Solutions AS
 - Registration Authority – Police and Border Guard Board
 - Manufacturer – Gemalto / IDEMIA
 - In addition generates key pair for you

Electronic functionality of Estonian ID card

- Two RSA/ECC key pairs:
 - Authentication (and decryption) key
 - Digital signature key
- Corresponding certificates
- PIN1/PIN2/PUK
- PIN retry counters
- Personal data file (16 records)
- Key usage counters
- Card management operations
 - PIN reset
 - Certificate overwrite
 - Applet reinstall

ID card chip platforms



(a) MICARDO



(b) MULTOS



(c) jTOP SLE66/SLE78 JavaCard



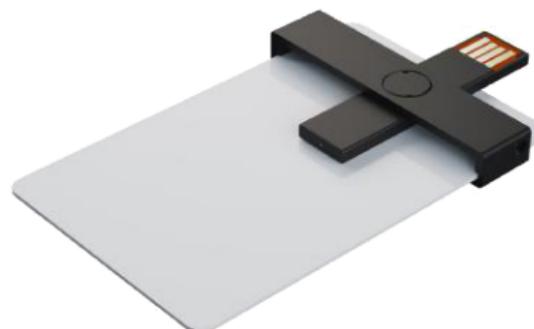
(d) Oberthur IAS-ECC JavaCard

EstEID specifications:

- Previous generations: <http://www.id.ee/public/TB-SPEC-EstEID-Chip-App-v3.4.pdf>
- Latest generation (2018): <https://installer.id.ee/media/id2019/TD-ID1-Chip-App.pdf>

Preparation: hardware

- Get a smart card reader
 - OMNIKEY CardMan 1021 – EUR 6 (Swedbank)
 - Gemalto IDBridge CT710 – EUR 17 (SEB)
 - Pluss ID (+iD) – EUR 13 (Klick)



- Built-in readers may have issues
- Plug the reader into the USB port

Preparation: hardware

- If using VirtualBox forward USB to guest Ubuntu
 - For USB 2.0/3.0 support install VirtualBox Extension Pack
 - Uninstall USBPcap (from Wireshark) from guest OS
 - Add USB filter if host OS fails to release the USB device
 - After filter is created, new USB devices attached will be automatically redirected to guest OS
- Check if the smart card reader is detected by Ubuntu:

```
$ dmesg
[ 1599.744116] usb 4-2: new full-speed USB device number 3 using uhci_hcd
[ 1599.921740] usb 4-2: New USB device found, idVendor=08e6, idProduct=3437
[ 1599.921751] usb 4-2: New USB device strings: Mfr=1, Product=2, SerialNumber=0
[ 1599.921760] usb 4-2: Product: USB SmartCard Reader
[ 1599.921767] usb 4-2: Manufacturer: Gemplus

$ lsusb
Bus 004 Device 003: ID 08e6:3437 Gemplus GemPC Twin SmartCard Reader <-- external USB
Bus 005 Device 002: ID 03f0:0324 Hewlett-Packard SK-2885 keyboard
Bus 002 Device 004: ID 0b97:7762 02 Micro Inc. Oz776 SmartCard Reader <-- DELL's built-in
```

Preperation: software

- Install pcscd:

```
$ sudo apt install pcscd pcsc-tools
$ pcsc_scan -n
Scanning present readers...
0: 02 Micro Oz776 00 00
1: Gemalto PC Twin Reader 01 00
Reader 0: 02 Micro Oz776 00 00
    Card state: Card removed,
Reader 1: Gemalto PC Twin Reader 01 00
    Card state: Card inserted,
    ATR: 3B DE 18 FF C0 80 B1 FE 45 1F 03 45 73 74 45 49 44 20 76 65 72 20 31 2E 30 2B
Possibly identified card (using /usr/share/pcsc/smartcard_list.txt):
    Estonian Identity Card (EstEID v1.0 2006 cold)
```

- Install pyscard:

```
$ sudo apt install python3-pyscard
$ python3
>>> import smartcard
>>> smartcard.System.readers()
['02 Micro Oz776 00 00', 'Gemalto PC Twin Reader 01 00']
>>> connection = smartcard.System.readers()[1].createConnection()
>>> connection.connect()
>>> connection.getATR()
[59, 222, 24, 255, 192, 128, 177, 254, 69, 31, 3, 69, 115, 116, 69, 73, 68, 32, 118, 101, 114, 32, 49, 46, 48]
>>> connection.transmit([0x0a, 0xa4, 0x00, 0x00, 0x02])
([], 110, 0)
```

Python: establishing connection

```
import sys
from smartcard.CardType import AnyCardType
from smartcard.CardRequest import CardRequest
from smartcard.CardConnection import CardConnection
from smartcard.util import toHexString

# this will wait until card inserted in any reader
channel = CardRequest(timeout=10, cardType=AnyCardType()).waitforcard().connection

# using T=0 for compatibility and simplicity
channel.connect(CardConnection.T0_protocol)

print("[+] Selected reader:", channel.getReader())

# detect and print EstEID card type (EstEID spec page 15)
atr = channel.getATR()
if atr == [0x3B,0xFE,0x18,0x00,0x00,0x80,0x31,0xFE,0x45,0x45,0x73,...]:
    print("[+] EstEID v3.x on JavaCard")
elif atr == [0x3B,0xFA,0x18,0x00,0x00,0x80,0x31,0xFE,0x45,0xFE,0x65,...]:
    print("[+] EstEID v3.5 (10.2014) cold (eID)")
elif atr == [0x3B,0xDB,0x96,0x00,0x80,0xB1,0xFE,0x45,0x1F,0x83,0x00,...]:
    print("[+] Estonian ID card (2018)")
else:
    print("[-] Unknown card:", toHexString(atr))
    sys.exit(1)
```

Python: transmitting APDUs

```
from smartcard.util import toHexString

def send(apdu):
    data, sw1, sw2 = channel.transmit(apdu)

    # success
    if [sw1,sw2] == [0x90,0x00]:
        return data
    # (T=0) card signals how many bytes to read
    elif sw1 == 0x61:
        return send([0x00, 0xC0, 0x00, 0x00, sw2]) # GET RESPONSE of sw2 bytes
    # (T=0) card signals incorrect Le
    elif sw1 == 0x6C:
        return send(apdu[0:4] + [sw2]) # resend APDU with Le = sw2
    # probably error condition
    else:
        print("Error: %02x %02x, sending APDU: %s" % (sw1, sw2, toHexString(apdu)))
        sys.exit(1)
```

- APDU commands and responses are lists containing integers (e.g., [0,50,199,255])
- For pretty-printing, a list of integers can be converted to hex string with spaces (e.g., toHexString([0,50,199,255])=="00 32 C7 FF")
- To convert list of integers to bytes, use: bytes([97,98,67])==b"abC".

Task 1: EstEID info – 2p

Implement a utility that reads personal data file and PIN retry counters from the ID card:

```
$ ./esteid_info.py
[+] Selected reader: Gemalto PC Twin Reader 00 00
[+] EstEID v3.5 (10.2014) cold (eID)
[+] Personal data file:
    [1] Surname: PARŠOVS
    [2] First name line 1: ARNIS
    [3] First name line 2:
    [4] Sex: M
    [5] Nationality: LVA
    [6] Birth date: 05.08.1986
    [7] Personal ID code: 38608050013
    [8] Document number: EA0043798
    [9] Expiry date: 27.08.2020
    [10] Place of birth: LÄTI / LVA
    [11] Date of issuance: 27.08.2015
    [12] Type of residence permit:
        [13] Notes line 1: EL KODANIK / EU CITIZEN
        [14] Notes line 2: ALALINE ELAMISÖIGUS
        [15] Notes line 3: PERMANENT RIGHT OF RESIDENCE
        [16] Notes line 4: LUBATUD TÖÖTADA
[+] PIN retry counters:
    PIN1: 3 left
    PIN2: 3 left
    PUK: 3 left
```

```
$ ./esteid_info.py
[+] Selected reader: Gemalto PC Twin Reader 00 00
[+] Estonian ID card (2018)
[+] Personal data file:
    [1] Surname: PARŠOVS
    [2] First name: ARNIS
    [3] Sex:
    [4] Citizenship:
    [5] Date & place of birth:
    [6] Personal ID code: 38608050013
    [7] Document number: NA0004369
    [8] Expiry date: 02 01 2024
    [9] Date & place of issuance: 02 01 2019
    [10] Type of residence permit:
        [11] Notes line 1:
        [12] Notes line 2:
        [13] Notes line 3:
        [14] Notes line 4:
        [15] Notes line 5:
[+] PIN retry counters:
    PIN1: 3 left
    PIN2: 3 left
    PUK: 3 left
```

Task 1: Personal data file

EstEID spec page 24:

- Select MF/EEEE/5044
- With READ RECORD read all 16 records from the file
- Decode them to unicode using CP1252 codepage

```
send([0x00, 0xA4, 0x00, 0x0C]) # SELECT FILE (MF)
send([0x00, 0xA4, 0x01, 0x0C]+[0x02, 0xEE, 0xEE]) # MF/EEEE
send([0x00, 0xA4, 0x02, 0x0C, 0x02, 0x50, 0x44]) # MF/EEEE/5044
r = send([0x00, 0xB2, 0x07, 0x04]) # READ RECORD 7
print("Personal ID code:", bytes(r).decode("cp1252"))
```

EstEID₂₀₁₈ spec page 30:

- Select AID: A000000077010800070000FE00000100
- Select MF/5000/50XX (XX – personal data file record 1 to 15)
- With READ BINARY read the contents of the file

```
send([0x00, 0xA4, 0x04, 0x00, 0x10, 0xA0, 0x00, 0x00, 0x00, 0x77, 0x01, ...]
send([0x00, 0xA4, 0x00, 0x0C]) # SELECT FILE (MF)
send([0x00, 0xA4, 0x01, 0x0C]+[0x02, 0x50, 0x00]) # MF/5000
send([0x00, 0xA4, 0x02, 0x0C]+[0x02, 0x50, 0x06]) # MF/5000/5006
r = send([0x00, 0xB0, 0x00, 0x00, 0x00]) # READ BINARY
print("Personal ID code:", bytes(r).decode("utf8"))
```

Task 1: PIN retry counters

EstEID spec page 28:

- Select MF/0016
- With READ RECORD read records 1, 2 and 3 (for PIN1, PIN2 and PUK, respectively)
- Record's 6th byte will contain integer value of how many tries are left

EstEID₂₀₁₈ spec page 25:

- Select MF for PIN1 and PUK
- Select MF/ADF2 for PIN2
- Send an empty VERIFY in P2 specifying 1, 2 or 133 (for PIN1, PUK and PIN2, respectively)
- VERIFY will return status word 63CX (X - remaining tries)

Task 2: EstEID getcert – 2p

Implement a utility that downloads the certificates stored on the ID card:

```
$ ./esteid_getcert.py --cert auth --out auth.pem
[+] Selected reader: Gemalto PC Twin Reader 00 00
[+] EstEID v3.5 (10.2014) cold (eID)
[=] Retrieving auth certificate...
[+] Certificate size: 1505 bytes
[+] Certificate stored in auth.pem
$ openssl x509 -in auth.pem -text | grep 'X509v3 Key Usage' -A 1
    X509v3 Key Usage: critical
        Digital Signature, Key Agreement

$ ./esteid_getcert.py --cert sign --out sign.pem
[+] Selected reader: Gemalto PC Twin Reader 00 00
[+] Estonian ID card (2018)
[=] Retrieving sign certificate...
[+] Certificate size: 973 bytes
[+] Certificate stored in sign.pem
$ openssl x509 -in sign.pem -text | grep 'X509v3 Key Usage' -A 1
    X509v3 Key Usage: critical
        Non Repudiation

$ wget https://sk.ee/upload/files/ESTEID-SK_2015.pem.crt
$ wget https://c.sk.ee/esteid2018.pem.crt
$ openssl verify -partial_chain -CAfile ESTEID-SK_2015.pem.crt sign.pem
error sign.pem: verification failed
$ openssl verify -partial_chain -CAfile esteid2018.pem.crt sign.pem
sign.pem: OK
```

Task 2: certificate reading¹

- Select AID: A000000077010800070000FE00000100₂₀₁₈
- Select MF/EEEE/AACE or MF/ADF1/3401₂₀₁₈ (authentication)
- Select MF/EEEE/DDCE or MF/ADF2/341F₂₀₁₈ (digital signature)
- Certificate is stored in a DER form in a file of fixed size
 - With READ BINARY read the first 10 bytes of certificate
 - Calculate the length of certificate by parsing the length byte(s) of certificate's outer ASN.1 SEQUENCE
 - All possible DER length values must be correctly handled
- Read the whole certificate (in a loop) using READ BINARY
 - With one READ BINARY maximum 231 bytes can be read
 - The offset is two byte integer. The most significant byte must be specified in P1, the least significant byte in P2.
 - Two byte integer i can be split into [MSB, LSB] using [$i \gg 8$, $i \& 0xFF$]
 - Make sure that only the minimum number of bytes are sent to/from the card (!)

¹EstEID spec page 35 / EstEID₂₀₁₈ spec page 37

Biometric passport (e-passport)



- Contactless smart card chip
- Stores information printed on the data page
 - Including facial image – 20KB JPG 480x640
- Data digitally signed by country signing CA
- Can be read only using the key encoded in MRZ
- Fingerprint reading requires terminal authentication
- Possibility for automated border clearance or “E-gates”

MTAT.07.017

Applied Cryptography

Smart Cards (JavaCard)

University of Tartu

Spring 2020

Smart card security model

Parties involved in a smart card-based system:

- Cardholder
- Data owner
- Terminal owner
- Card issuer
- Card (software)manufacturer

Smart card threat models:

- attacks by the terminal against the cardholder
- attacks by the cardholder against the terminal
- attacks by the cardholder against the data owner
- attacks by the cardholder against the issuer
- attacks by the terminal owner against the issuer
- attacks by the issuer against the cardholder
- attacks by the (software)manufacturer against the data owner

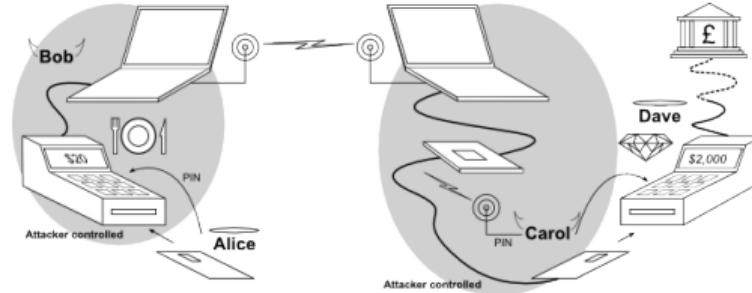
<http://www.schneier.com/paper-smart-card-threats.html>

Estonian ID card



- Used to:
 - Store RSA/ECC private keys
 - Perform on-card signing/decryption
 - Authorize cryptographic operations (using PIN)
- Cardholder / Data owner / Terminal / Card issuer / Card (software)manufacturer
- Attacks:
 - by the terminal against the cardholder
 - by the cardholder against the terminal owner
 - by the cardholder against the data owner
 - by the issuer against the cardholder
 - by the (software)manufacturer against the data owner

Payment cards (EMV)



- Used to:
 - Store symmetric MAC key
 - Authentication of transactions (using PIN)
- Attacks:
 - by the terminal against the cardholder
 - Relay attacks
 - by the terminal owner against the issuer
 - by the issuer against the cardholder

Mobile phone SIM cards



- Used to:
 - Store 128-bit symmetric subscriber authentication key
 - Perform RUN GSM ALGORITHM
 - Authorize operations (using PIN)
 - Store contacts and SMS messages
 - Store settings (operator information)
 - Mobile-ID
- Attacks:
 - by the cardholder against the data owner
 - by the terminal owner against the issuer
 - by the issuer against the cardholder

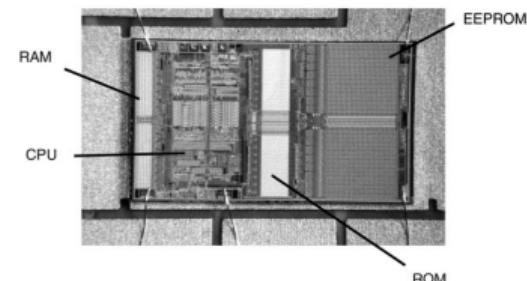
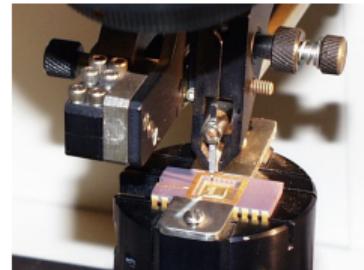
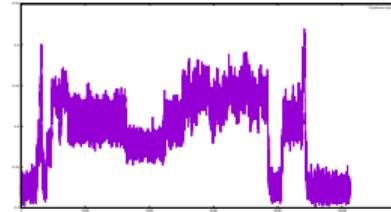
Pay TV



- Used to:
 - Decrypt TV signal
 - Store channel filters
- Attacks:
 - by the cardholder against the data owner/issuer
 - by the terminal owner against the issuer

Attacks against smart cards

- Side channel attacks:
 - Timing analysis
 - Power analysis
- Fault injection:
 - Voltage, clock rate, radiation
- Physical attacks:
 - Chemical etching
 - Chip re-wiring
 - Adding a track
 - Cutting a track
- Countermeasures:
 - Metal layers
 - Onboard sensors (temp, light, frequency)
 - ...

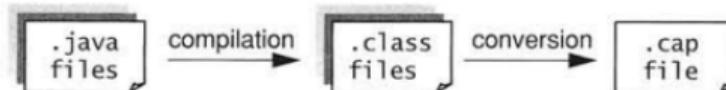


Common Criteria (CC) security certification

- Target of Evaluation (TOE) – a product that has to be evaluated
- Protection Profile (PP) – identifies security requirements for class of products
 - For example, PP for a Secure Signature Creation Device
- Security Target (ST) – identifies the security properties of TOE
- Evaluation Assurance Level (EAL) – level of verification (1 to 7 + augmentation)
- Evaluation facilities – certified IT security testing laboratories
- Certification Bodies – issue CC certificates (e.g., ANSSI and BSI)

JavaCard

- Card capable of running code written in Java
- Stripped-down version of Java
 - Data types: boolean, byte, short
 - Not supported: char, String, float, int
 - One dimensional arrays
 - No threads
- Rich cryptography API available
 - Employs cryptographic coprocessor
 - Algorithm support depends on card
(<https://www.fi.muni.cz/~xsvenda/jcalgtest/table.html>)
 - Side-channel protection guaranteed only for crypto API calls
- Java .class file has to be converted to .cap file



- Estonian ID cards issued since 2011 are JavaCards

JavaCard applet

```
$ cat TestApplet.java
package appcrypto;

import javacard.framework.*;
import javacard.security.*;
import javacardx.crypto.*;

public class TestApplet extends Applet {
    RandomData rnd;

    public static void install(byte[] ba, short ofs, byte len) {
        (new TestApplet()).register();
    }

    public void process(APDU apdu) {
        byte[] buf = apdu.getBuffer(); // contains first 5 APDU bytes

        switch (buf[ISO7816.OFFSET_INS]) {
            case (byte)0x00:
                if (buf[ISO7816.OFFSET_LC] != (byte)1) {
                    ISOException.throwIt(ISO7816.SW_DATA_INVALID);
                }
                apdu.setIncomingAndReceive(); // read APDU data bytes
                short len = (short)(buf[ISO7816.OFFSET_CDATA] & (short)0xff); // get rid of sign
                rnd = RandomData.getInstance(RandomData.ALG_SECURE_RANDOM);
                rnd.generateData(buf, (short)0, len);
                apdu.setOutgoingAndSend((short)0, len); // return response data
                return;
        }
        ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
    }
}
```

C-APDU: 00 00 00 00 01 03
R-APDU: F0 43 CA 90 00

Converting to CAP

```
$ sudo apt install opensc openjdk-8-jdk ant
$ wget https://github.com/martinpaljak/ant-javacard/releases/download/19.03.04/ant-javacard.jar
$ git clone https://github.com/martinpaljak/oracle_javacard_sdks

$ cat build.xml
<?xml version="1.0" encoding="UTF-8"?>
<project default="applet" basedir=".">>

<target name="jcpro">
    <taskdef name="javacard" classname="pro.javacard.ant.JavaCard" classpath="ant-javacard.jar"/>
</target>

<target name="applet" depends="jcpro">
    <javacard>
        <cap jckit="oracle_javacard_sdks/jc222_kit/" output="applet.cap"
            sources="/home/user/eclipse-workspace/appcrypto/src/"
            <applet class="appcrypto.TestApplet" aid="0102030405060708"/>
        </cap>
    </javacard>
</target>
</project>

$ ant
applet:
      [cap] INFO: using JavaCard 2.2.2 SDK in oracle_javacard_sdks/jc222_kit
      [cap] INFO: Setting package name to appcrypto
      [cap] Building CAP with 1 applet from package appcrypto (AID: 0102030405)
      [cap] appcrypto.TestApplet 0102030405060708
[compile] Compiling files from /home/user/eclipse-workspace/appcrypto/src
[compile] Compiling 1 source file to /tmp/jccpro2232880529567474390
[javacard] NB! Please use JavaCard SDK 3.0.5u3 or later for verifying!
      [verify] Verification passed
      [cap] CAP saved to /tmp/jc/applet.cap
BUILD SUCCESSFUL
Total time: 1 second
```

GlobalPlatform

- Standard for applet management on JavaCards
- Multiple applets can be installed
 - Applet is SELECT'ed using Application Identifier (AID)
 - Applet can be set as the default applet (selected by default)
 - Applets are isolated (with exceptions – Shareable Interface)
- Applet can be deleted (usually), but never downloaded
- Security Domain (SD)
 - Every applet belongs to a SD
 - Card Issuer Security Domain (ISD)
 - Supplementary Security Domains (SSDs)
 - Secure Channel Protocol for communication with SD

Installing CAP file

```
$ wget https://github.com/martinpaljak/GlobalPlatformPro/releases/download/v0.3.5/gp.jar
$ java -jar gp.jar --install applet.cap --default
CAP loaded

$ java -jar gp.jar --list
[...]
AID: 0102030405060708 (.......)
    App SELECTABLE: Default selected

AID: 0102030405 (....)
    ExM LOADED: (none)
    0102030405060708 (....)

$ opensc-tool -s 00:00:00:00:01:05:00
Received (SW1=0x90, SW2=0x00):
47 62 C6 A1 E3 Gb...

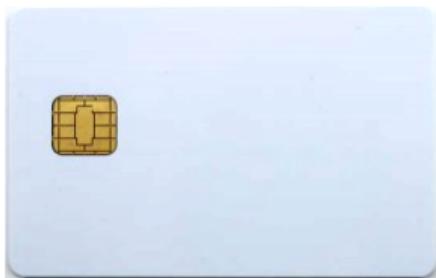
$ opensc-tool -s 00:00:00:00:01:a0:00
Received (SW1=0x90, SW2=0x00):
CD 12 03 41 BA 22 69 32 1D 43 1E 46 21 26 76 8C ...A."i2.C.F!&v.
1B D5 E1 F5 A6 6C 65 7E 87 68 B7 36 D9 6B 61 B0 .....le~.h.6.ka.
07 E5 CF E6 D0 CE E2 28 9A 53 F4 6C 3B CB 3C OF .....(.S.l;.<.
C7 E4 5D 9C EC FE 94 7D 07 6A 90 20 A1 F6 2E E4 ...].....}j. ....
26 D1 23 79 3F D2 F1 93 0E 1C 5E 11 8E 60 3E FB &.#y?.....^..>.
1D 23 C4 E7 04 CB E2 67 96 77 48 F0 AF E2 30 00 .#.g.wH...0.
F9 E9 C4 63 10 D5 C0 6E 6F A4 4C 3B C6 54 75 99 ...c...no.L;.Tu.
0B 42 6D C8 2F 40 D5 FD 2D CB 70 8C 1F 3C 30 4D .Bm./@..-.p..<0M
D2 88 5F 68 3B 43 1C 85 D9 AD C0 B7 EC 01 3C 0E ..._h;C.....<.
D4 EE 01 F0 B5 EB 5E 8D B2 5C F4 D1 2E 09 77 BC .....^..\\....w.

$ java -jar gp.jar --deletedeps --delete 0102030405
```

Blank JavaCard (Infineon)

- 40 cards available

- Chip: Infineon SLE78
- EEPROM: 150K
- RAM: ?
- JavaCard 3.0.4
- GlobalPlatform 2.2.1
- DES/3DES/AES256
- MD5/SHA1/SHA256/SHA512
- RSA-2048 (on-card generation)
- ECC-521 (on-card generation)
- CC EAL5+ high certification



Infineon jTOP SLE78
(SLJ52GCA150)

Warning: Infineon RSA key generation flaw!

Blank JavaCard (Feitian)

- 16 cards available



Feitian FT-Java/D11CR

- Chip: ST31
- EEPROM: 50K
- RAM: 5K
- JavaCard 2.2.2
- GlobalPlatform 2.1.1
- DES/3DES/AES128
- MD5/SHA1/SHA224/SHA256
- RSA-2048 (on-card generation)
- ECC-256 (on-card generation)
- Contactless Interface
- Garbage collector
- No security certifications

Warning: On-card RNG flawed!

Task: JavaCard applet – 6p

Write a JavaCard applet that performs on-card RSA 2048-bit key generation and decryption:

```
$ ./test_applet.py
[+] Selected reader: Gemalto PC Twin Reader (E0660B9A) 00 00
[+] Infineon jTOP SLE78 (SLJ52GCA150)
[+] Generating 2048-bit RSA key...
[+] Key generated in 15.01943 seconds!
[+] Retrieving public key...
[+] n=1907947716745697801984296668306272913349298245522576117840417272520071383935970391499
[+] e=65537
[?] Enter message to encrypt: secret message!
[+] Encrypted message: 912d7285385d51c5511d3c108bd3b2361f7895d8f5f1b32192e3194afed220022257
[+] Sending ciphertext to card...
[+] Message decrypted in 0.742395 seconds!
[+] Decrypted message (15 bytes): secret message!
```

Commit TestApplet.java to your repository.

Task: JavaCard applet

- Find out the communication protocol from `test_applet.py`
- JavaCard API calls to use:

```
keypair = new KeyPair(KeyPair.ALG_RSA, KeyBuilder.LENGTH_RSA_*);  
keypair.genKeyPair();  
  
pub = (RSAPublicKey) keypair.getPublic();  
pub.getExponent(byte[] buffer, short offset);  
pub.getModulus(byte[] buffer, short offset);  
  
rsa = Cipher.getInstance(Cipher.ALG_RSA_PKCS1, false);  
rsa.init(keypair.getPrivate(), Cipher.MODE_DECRYPT);  
rsa.doFinal(byte[] inBuff, short inOffset, short inLength,  
            byte[] outBuff, short outOffset);
```

Task: JavaCard applet

- Size limit for data APDU body is 255 bytes
 - The first two bytes of ciphertext are embeded in P1 and P2
 - Make the ciphertext continuous using:

```
Util.arrayCopyNonAtomic(byte[] src, short srcOff, byte[] dest,  
                      short destOff, short length);
```

- Avoid memory leaks – initialize objects only once!
- Make sure that keypair is generated only once
 - Ignore (without error) repeated keygen requests
- Java has signed types – cast byte to short using 0xff mask
- Debugging is possible only via the data or SW returned!

JavaCard: memory management

EEPROM (or flash):

- Slow writes, subject to wear
- Preserves data on power loss

Persistent Objects:

- Class-member variables
- Static variables
- Array data

RAM:

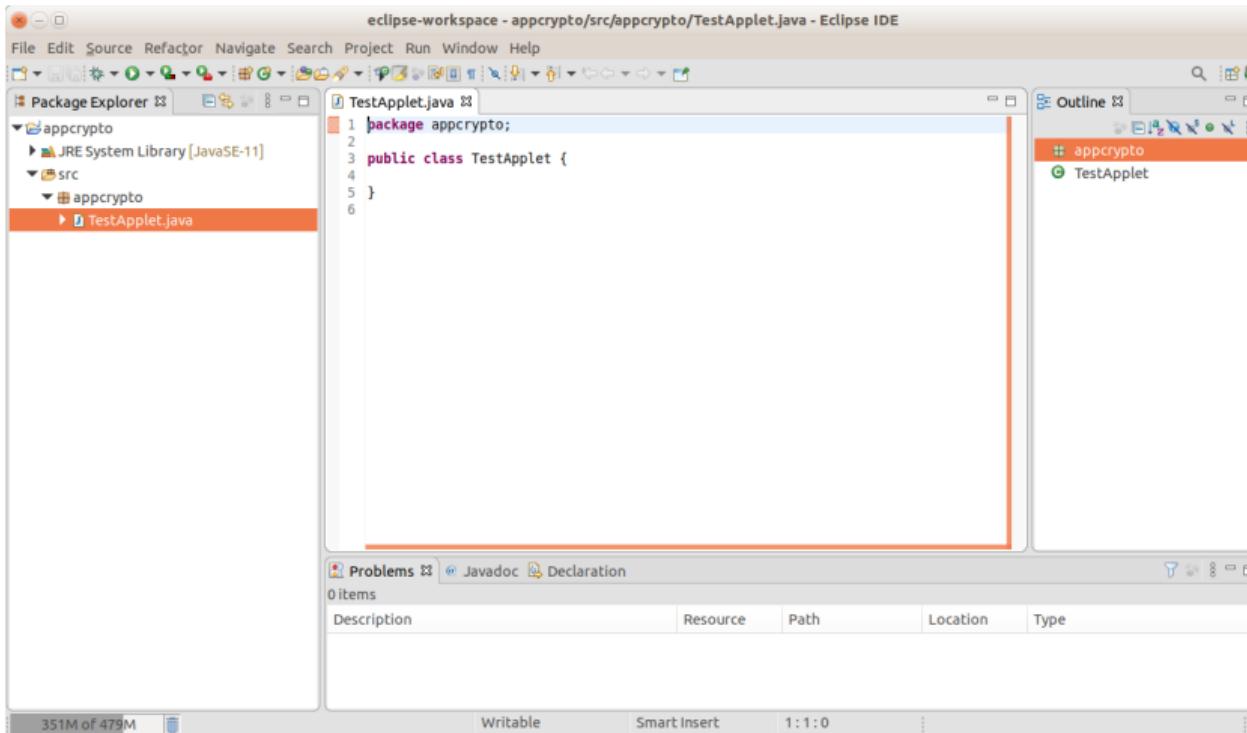
- Fast writes (1000x faster)
- Loses data on power loss
- Small storage space

Transient Objects:

- Local variables
- Method parameters
- Transient array data
`(makeTransientByteArray())`
- APDU buffer

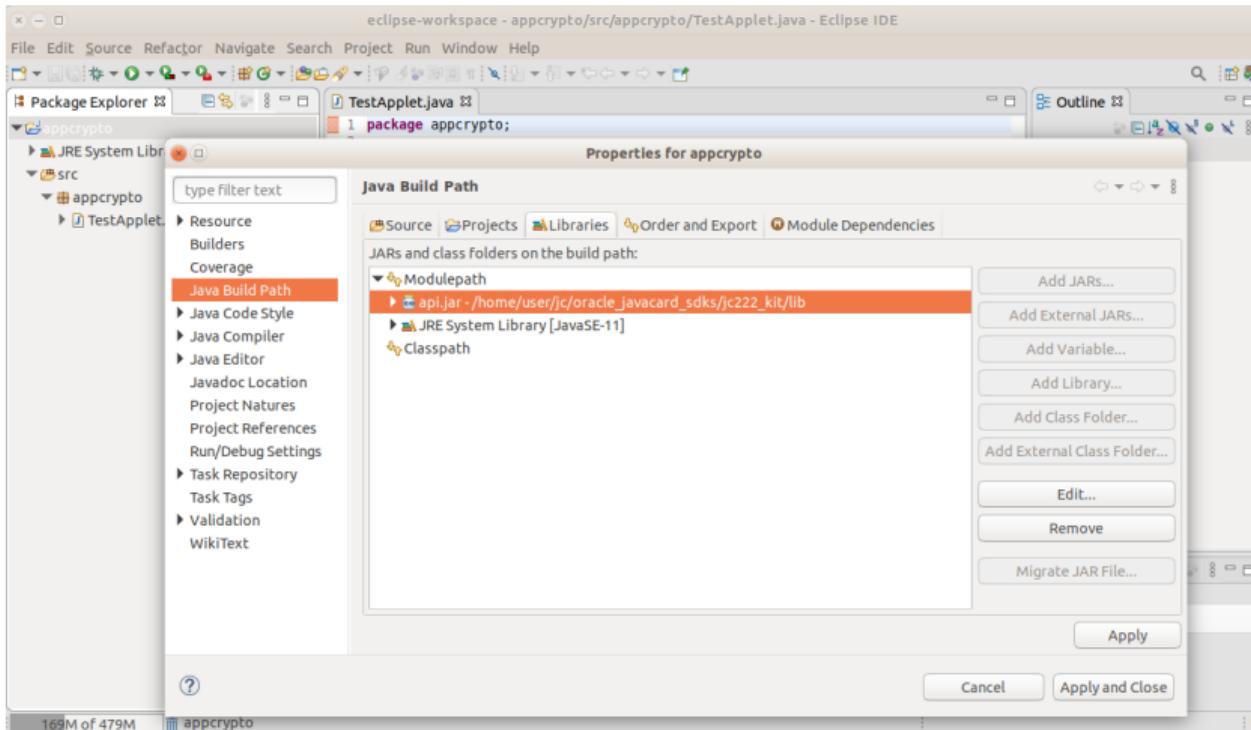
JCRE may not include a garbage collector
(space of unreferenced objects is not reclaimed).

JavaCard development under Eclipse



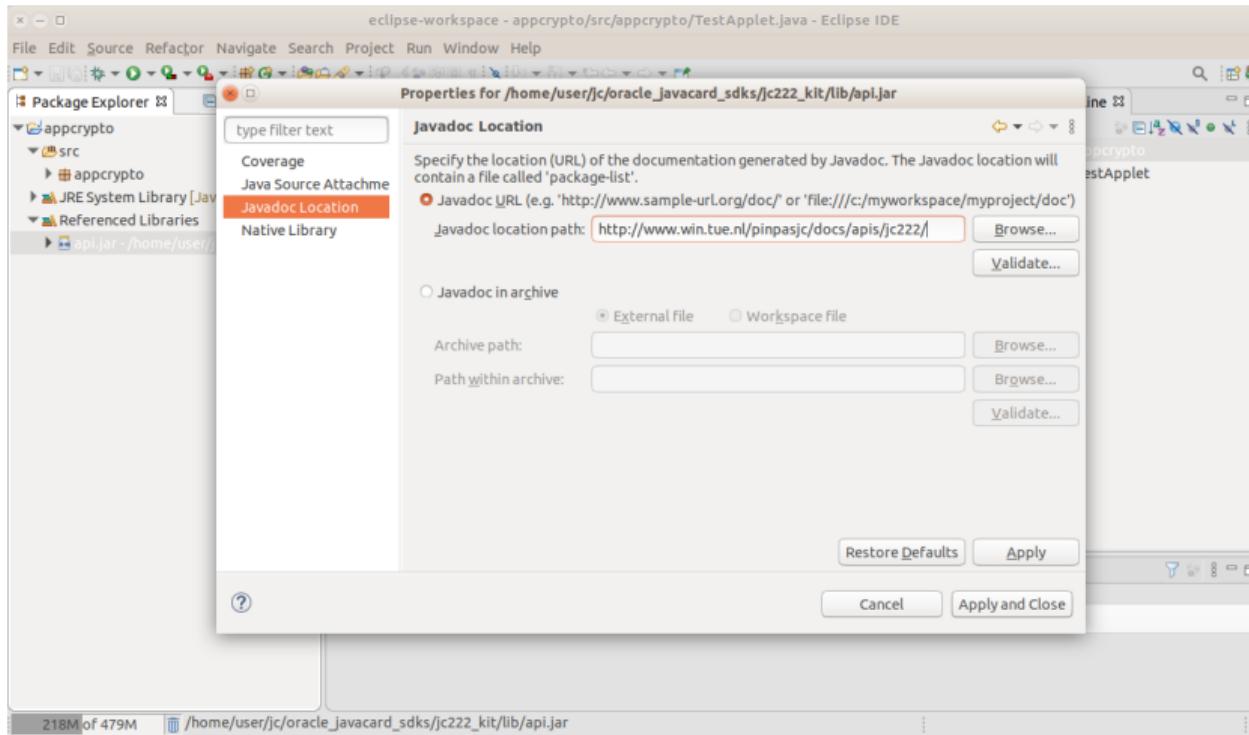
- `sudo snap install eclipse --classic`
- Create a project: “File – New – Java Project – Project name: appcrypto”.
- Right-click on the project “New – Class – Name: TestApplet, Package: appcrypto”.

JavaCard development under Eclipse



- Right-click on your project “Build Path – Configure Build Path... – Libraries – Add External JARs” and add `oracle_javacard_sdks/jc222_kit/lib/api.jar`. This will enable JavaCard code validation and completion.

JavaCard development under Eclipse



- Right-click on `api.jar` – “Properties – Javadoc Location – Javadoc location path:” and specify `http://www.win.tue.nl/pinpasjc/docs/apis/jc222/`. This will enable javadoc for JavaCard API calls.

MTAT.07.017

Applied Cryptography

Transport Layer Security (TLS)

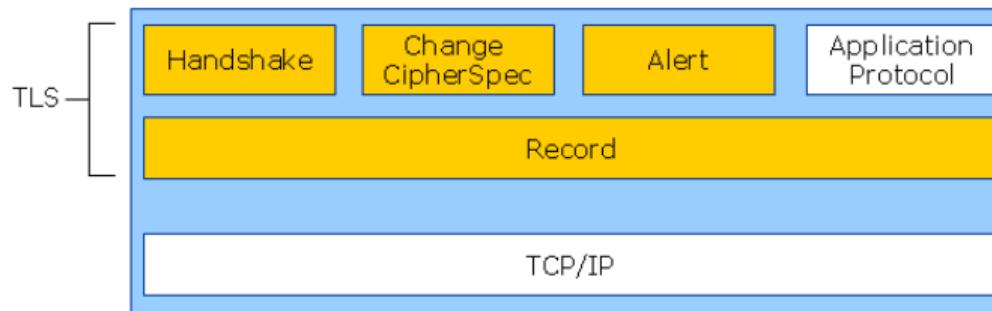
University of Tartu

Spring 2020

Transport Layer Security

“TLS is a cryptographic protocol that provides communication security over the Internet.”

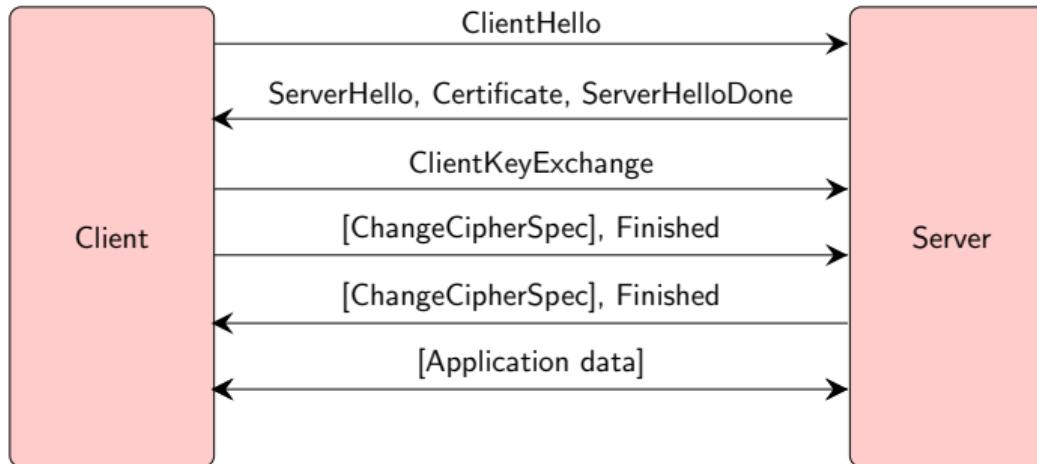
- Provides confidentiality, integrity and server authentication
- The most successful and widely used cryptographic protocol (!!!)
- Any application protocol can be encapsulated in TLS



TLS version history

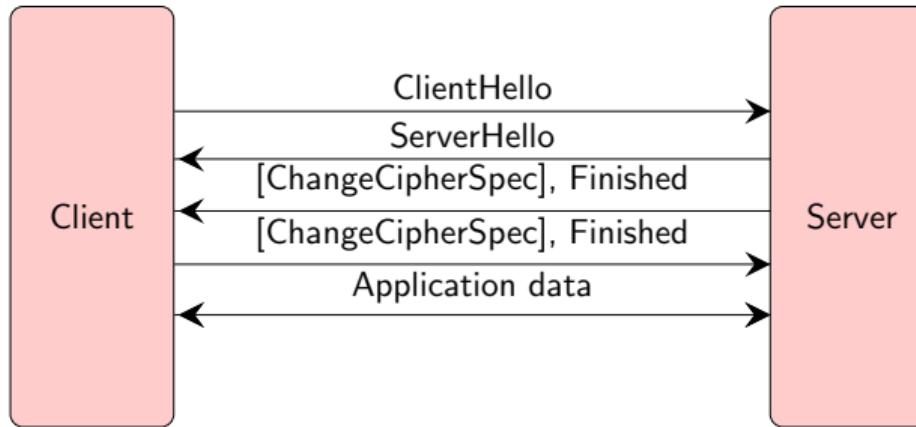
- SSL 1.0 – never publicly released
- SSL 2.0 – Netscape (1995)
- SSL 3.0 – Netscape (1996)
- TLS 1.0 (SSL 3.1) – RFC 2246 (1999)
- TLS 1.1 – RFC 4346 (2006)
- **TLS 1.2** – RFC 5246 (2008)
- TLS 1.3 – RFC 8446 (2018)

TLS handshake



- Client verifies server's X.509 certificate
- Client extracts from the certificate server's public key
- Client encrypts a random symmetric key using server's public key
- Only the server can decrypt the symmetric key
- Now the client and server share the same symmetric key
- Symmetric key is used for the actual data encryption/authentication

TLS session resumption

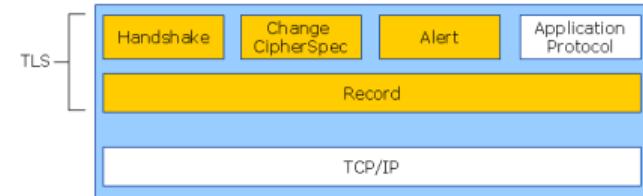


- Resumed TLS connections share the same “master secret”
- Several TLS *connections* can belong to the same TLS *session*
- If TLS connection fails, TLS session becomes non-resumable
- Abbreviated handshake improves performance, saving:
 - 1 round-trip time across the network
 - 1 asymmetric crypto operation

TLS Record Layer

[Type] [Version] [Length] [Data]

- Type: type of encapsulated data:
 - Handshake message (0x16)
 - Change Cipher Spec message (0x14)
 - Alert message (0x15)
 - Application data (0x17)
- Protocol version: 0x0303 (for TLS v1.2)
- Length: length of the data (2 bytes)
- Data: encapsulated data
 - Can contain several same type messages



Record Content Type (0x16 – Handshake Message)

SSL Version (3.1 – TLSv1)

Record Length (0x61)

16 03 01 00 61 ...

TLS record header is never encrypted!

Dissecting TLS with Wireshark

Capturing from wlp2s0 (port 443)

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/>

No. Time Source Destination Protocol Length Info

1	0.000000000	192.168.1.143	195.80.123.145	TCP	74 46876 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TStamp=4134474515 TSval=25
2	0.007436690	195.80.123.145	192.168.1.143	TCP	74 443 → 46876 [SYN, ACK] Seq=0 Ack=1 Win=4380 Len=0 MSS=1436 TStamp=4134474515 TSval=25
3	0.007464753	192.168.1.143	195.80.123.145	TCP	66 46876 → 443 [ACK] Seq=1 Ack=1 Win=64240 Len=0 TStamp=4134474515 TSec=1
4	0.007586961	192.168.1.143	195.80.123.145	TLSv1.2	120 Client Hello
5	0.018650085	195.80.123.145	192.168.1.143	TLSv1.2	3259 Server Hello, Certificate, Server Hello Done
6	0.018677790	192.168.1.143	195.80.123.145	TCP	66 46876 → 443 [ACK] Seq=55 Ack=3194 Win=61047 Len=0 TStamp=4134474527
7	0.026255408	192.168.1.143	195.80.123.145	TLSv1.2	73 Alert (Level: Fatal, Description: Certificate Unknown)
8	0.026292941	192.168.1.143	195.80.123.145	TCP	66 46876 → 443 [FIN, ACK] Seq=62 Ack=3194 Win=64080 Len=0 TStamp=413447
9	0.032758388	195.80.123.145	192.168.1.143	TCP	66 443 → 46876 [ACK] Seq=3194 Ack=62 Win=4369 Len=0 TStamp=2506765279 TSec=1
10	0.032760500	195.80.123.145	192.168.1.143	TCP	66 443 → 46876 [ACK] Seq=3194 Ack=62 Win=4369 Len=0 TStamp=2506765279 TSec=1

Frame 4: 120 bytes on wire (960 bits), 120 bytes captured (960 bits) on interface 0

► Ethernet II, Src: IntelCor_86:60:eb (e4:b3:18:86:60:eb), Dst: Tp-LinkT_91:35:b4 (c4:e9:84:91:35:b4)

► Internet Protocol Version 4, Src: 192.168.1.143, Dst: 195.80.123.145

► Transmission Control Protocol, Src Port: 46876, Dst Port: 443, Seq: 1, Ack: 1, Len: 54

▼ Transport Layer Security

 ▼ TLSv1.2 Record Layer: Handshake Protocol: Client Hello

 Content Type: Handshake (22)

 Version: TLS 1.2 (0x0303)

 Length: 49

 ▼ Handshake Protocol: Client Hello

 Handshake Type: Client Hello (1)

 Length: 45

 Version: TLS 1.2 (0x0303)

 ► Random: 5e9d08c1029221e89be5f8bc63e2e851054a267b7cb69b49...

 Session ID Length: 0

0000 c4 e9 84 91 35 b4 e4 b3 18 86 60 eb 08 00 45 00 E
0010 00 6a 4f d2 40 00 40 06 e9 a2 c0 a8 01 8f c3 50 j0 @ @ P
0020 7b 91 b7 1c 01 bb 42 22 86 ef 3d 5d fc 43 80 18 { B" =] C
0030 fa f0 30 f9 00 00 01 01 08 0a f6 6f 13 13 95 6a o . j
0040 33 c5 16 03 03 00 31 01 00 00 2d 03 03 5e 9d 08 3 A .
0050 c1 02 92 21 e8 9b e5 f8 bc 63 e2 e8 51 05 26 . . . ! c . Q . J8
0060 7b 7c b6 9b 49 c7 db 97 f1 0e 7b a9 82 00 00 06 { | . I {
0070 00 05 00 2f 00 35 01 00 . . . / - 5

Length of TLS record data (tls.record.length), 2 bytes

Packets: 259 · Displayed: 259 (100.0%) · Profile: Default

7 / 23

Alert message

Signals about TLS related issues to other party

[Level] [Description]

- Level (1 byte):
 - Warning (0x01)
 - Fatal (0x02)
- Description (1 byte):

```
close_notify(0),  
unexpected_message(10),  
bad_record_mac(20),  
decryption_failed(21),  
handshake_failure(40),  
bad_certificate(42),  
unsupported_certificate(43),  
certificate_revoked(44),  
certificate_expired(45),  
illegal_parameter(47),  
unknown_ca(48),  
access_denied(49),  
decrypt_error(51),  
user_canceled(90),  
...
```

```
▼ TLSv1 Record Layer: Alert (Level: Fatal, Description: Certificate Unknown)  
  Content Type: Alert (21)  
  Version: TLS 1.0 (0x0301)  
  Length: 2  
▼ Alert Message  
  Level: Fatal (2)  
  Description: Certificate Unknown (46)
```

Change Cipher Spec message

Signals to other party that from now on the negotiated cipher suite will be used to protect outgoing messages

[0x01]

▼ TLSv1 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
Content Type: Change Cipher Spec (20)
Version: TLS 1.0 (0x0301)
Length: 1
Change Cipher Spec Message

Application data

Contains (most likely encrypted) application data in a form as required by the application protocol (e.g., HTTP request/response etc.)

[Application Data]

▼ TLSv1.1 Record Layer: Application Data Protocol: http

Content Type: Application Data (23)

Version: TLS 1.1 (0x0302)

Length: 448

Encrypted Application Data: 3a37312ac35ea3809f392b2b76174849218d83f179d6d305...

Handshake message

Contains protocol handshake parameters

[Type] [Length] [Body]

- Type: message type:

```
hello_request(0), client_hello(1), server_hello(2),  
certificate(11), server_key_exchange (12),  
certificate_request(13), server_hello_done(14),  
certificate_verify(15), client_key_exchange(16),  
finished(20)
```

- Length: length of the body (3 bytes)
- Body: message body
 - Can be split over several TLS records

Handshake message: ClientHello

- The highest TLS version supported (2 bytes)
- Client randomness (32 bytes)
 - Timestamp in first 4 bytes
- Session ID length (1 byte) + session ID
- Cipher suites length (2 bytes)
- List of cipher suites supported:
 - 0x0005 – TLS_RSA_WITH_RC4_128_SHA
 - 0x002f – TLS_RSA_WITH_AES_128_CBC_SHA
 - 0x0035 – TLS_RSA_WITH_AES_256_CBC_SHA
 - 0x0039 – TLS_DHE_RSA_WITH_AES_256_CBC_SHA
- Compression methods length (1 byte)
- List of compression methods supported:
 - 0x00 – null (mandatory)
 - 0x01 – DEFLATE (gzip)
- Extensions (optional)

```
▼ TLSv1 Record Layer: Handshake Protocol: Client Hello
  Content Type: Handshake (22)
  Version: TLS 1.0 (0x0301)
  Length: 47
  ▼ Handshake Protocol: Client Hello
    Handshake Type: Client Hello (1)
    Length: 43
    Version: TLS 1.0 (0x0301)
    ▼ Random
      gmt_unix_time: May 3, 2013 17:55:01.000000000 EEST
      random_bytes: 7a8be086f64064e973ce6735a9db15aa7af154
      Session ID Length: 0
      Cipher Suites Length: 4
      ▼ Cipher Suites (2 suites)
        Cipher Suite: TLS_RSA_WITH_RC4_128_SHA (0x0005)
        Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
        Compression Methods Length: 1
      ▼ Compression Methods (1 method)
        Compression Method: null (0)
```

Handshake message: ServerHello

- TLS version selected (2 bytes)
- Server randomness (32 bytes)
 - Timestamp in first 4 bytes
- Session ID length (1 byte) + session ID
- Cipher suite selected (2 bytes)
- Compression method selected (1 byte)
- Extensions (optional)

```
▼ TLSv1 Record Layer: Handshake Protocol: Server Hello
  Content Type: Handshake (22)
  Version: TLS 1.0 (0x0301)
  Length: 74
▼ Handshake Protocol: Server Hello
  Handshake Type: Server Hello (2)
  Length: 70
  Version: TLS 1.0 (0x0301)
▼ Random
  gmt_unix_time: May  3, 2013 17:55:01.000000000 EEST
  random_bytes: 6dbfaec346d39439ac083b8c0df9f485b327c
  Session ID Length: 32
  Session ID: 6a9137403d3be6f28e95ca0b0053734f6edad2813
  Cipher Suite: TLS_RSA_WITH_RC4_128_SHA (0x0005)
  Compression Method: null (0)
```

Handshake message: Certificate

- Length of certificate list (3 bytes)
- List of certificates
 - Certificate length (3 bytes)
 - DER encoded certificate
- The first is server's certificate
- Other certificates are optional
 - Usually intermediate CA certificates

```
▼ TLSv1 Record Layer: Handshake Protocol: Certificate
  Content Type: Handshake (22)
  Version: TLS 1.0 (0x0301)
  Length: 2951
▼ Handshake Protocol: Certificate
  Handshake Type: Certificate (11)
  Length: 2947
  Certificates Length: 2944
▼ Certificates (2944 bytes)
  Certificate Length: 694
  ▶ Certificate (id-at-commonName=ubuntu)
  Certificate Length: 983
  ▶ Certificate (id-at-commonName=ESTEID-SK 2007,id-
  Certificate Length: 1258
  ▶ Certificate (id-at-commonName=Juur-SK,id-at-orgai
```

Handshake message: ServerHelloDone

- Empty message body
- Tells that there will be no more messages from the server in this protocol round

```
▼ TLSv1 Record Layer: Handshake Protocol: Server Hello Done
  Content Type: Handshake (22)
  Version: TLS 1.0 (0x0301)
  Length: 4
▼ Handshake Protocol: Server Hello Done
  Handshake Type: Server Hello Done (14)
  Length: 0
```

Handshake message: ClientKeyExchange

Contains (two-byte length-prefixed) encrypted 48-byte random “pre-master secret”

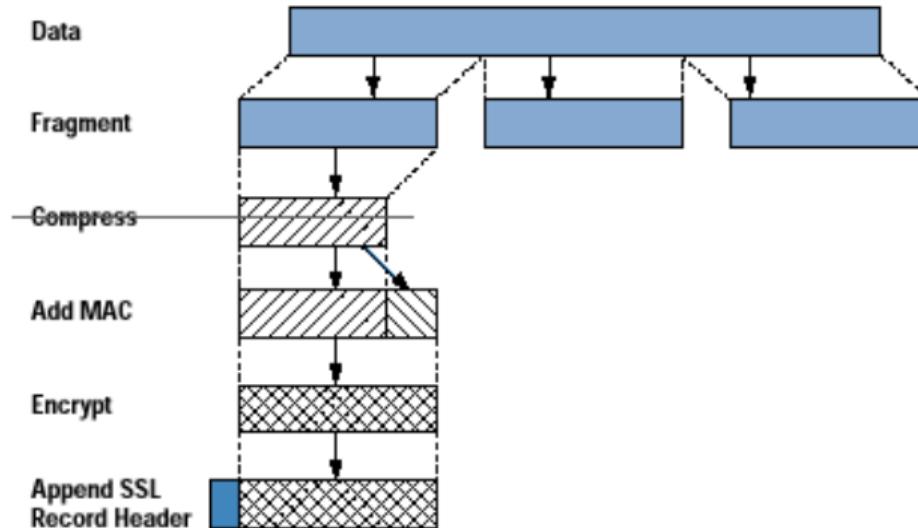
- Encrypted using the public key from the server's certificate
 - Encrypted according to PKCS#1 v1.5
 - The first two bytes in the pre-master secret contain the TLS version
 - Must be checked by the server
 - Prevents some attacks (?)
 - Next 46 bytes are truly random bytes
- ▼ TLSv1 Record Layer: Handshake Protocol: Client Key Exchange
Content Type: Handshake (22)
Version: TLS 1.0 (0x0301)
Length: 262
- ▼ Handshake Protocol: Client Key Exchange
Handshake Type: Client Key Exchange (16)
Length: 258
- ▼ RSA Encrypted PreMaster Secret
Encrypted PreMaster length: 256
Encrypted PreMaster: 34f527956711a0e5100b30571910486042!

Handshake message: Finished

- The first encrypted message
- Serves to verify if encryption works
- Contains hash of concatenation of all previous handshake messages (excluding the TLS record header)
 - Must be verified by other party to detect downgrade attacks

▼ TLSv1.1 Record Layer: Handshake Protocol: Encrypted Handshake Message
Content Type: Handshake (22)
Version: TLS 1.1 (0x0302)
Length: 64
[Handshake Protocol: Encrypted Handshake Message](#)

TLS encryption process



- How many symmetric keys are needed?
 - MAC & encrypt (+ IV for block ciphers)
 - Separate keys for each direction
- How to derive these keys from the 48-byte pre-master secret?

Key derivation

- TLS defines PRF() (pseudo-random function)
 - Uses SHA256
 - Produces infinitely long pseudo-random output
- From the 48-byte “pre-master secret” a 48-byte “master secret” is derived:
`PRF(premaster + 'master secret' + client_random + server_random, 48)`
- From the “master secret” is derived a key block in the size needed:
`PRF(master_secret + 'key expansion' + server_random + client_random, 136)`
- The key block is split into the keys needed:

```
client_mac_key = key_block[:20]
server_mac_key = key_block[20:40]
client_enc_key = key_block[40:56]
server_enc_key = key_block[56:72]
client_iv = ...
...
```

MAC calculation

```
HMAC_hash(key, seq + type + version + length + data)
```

- hash: hash algorithm from the negotiated cipher suite
- key: client/server MAC key
- seq: client/server sequence number (8 bytes)
 - Starts from 0
 - Incremented for every TLS record sent
- type: TLS record type
- version: TLS protocol version (2 bytes)
- length: length of the data (2 bytes)
- data: TLS record payload

Task: TLS getcert – 4p

Implement TLS v1.2 client that can retrieve server's certificate:

```
$ ./tls_getcert.py https://www.eesti.ee/ --certificate server.pem
--> ClientHello()
<--- Handshake()
    <--- ServerHello()
        [+] server randomness: 0F0F1A925B17BD55B165A30259AD7BFF4514788382741D6E521A16C7DAC83D76
        [+] server timestamp: 1978-01-03 07:01:22
        [+] TLS session ID: EF82F90493D73D79776B3A4F9B3E852130727353190FAF379428CC22AB0E1570
        [+] Cipher suite: TLS_RSA_WITH_AES_256_CBC_SHA
    <--- Handshake()
        <--- Certificate()
            [+] Server certificate length: 1882
            [+] Server certificate saved in: server.pem
    <--- Handshake()
        <--- ServerHelloDone()
--> Alert()
[+] Closing TCP connection!

$ openssl x509 -in server.pem -text | grep 'Subject:'
    Subject: C = EE, L = Tallinn, O = Estonian Information System Authority, CN = *.eesti.ee

3 0.0074.. 192.168.1.143      195.80.123.145      TCP      66 46876 → 443 [ACK] Seq=1 Ack=1 Win=64240 Len=0 TSval=41:
4 0.0075.. 192.168.1.143      195.80.123.145      TLSv1.2   120 Client Hello
5 0.0186.. 195.80.123.145      192.168.1.143      TLSv1.2   3259 Server Hello, Certificate, Server Hello Done
6 0.0186.. 192.168.1.143      195.80.123.145      TCP      66 46876 → 443 [ACK] Seq=55 Ack=3194 Win=61047 Len=0 TSva:
7 0.0262.. 192.168.1.143      195.80.123.145      TLSv1.2   73 Alert (Level: Fatal, Description: Certificate Unknown)
```

Task: TLS getcert

```
$ ./tls_getcert.py https://www.twitter.com/
--> ClientHello()
<--- Handshake()
<--- ServerHello()
[+] server randomness: 68167B6301F828E2D57C3A41D39DD5B67D9085959FAE7E72A4A5E11EFDF84E83
[+] server timestamp: 2025-05-03 23:24:03
[+] TLS session ID:
[+] Cipher suite: TLS_RSA_WITH_AES_128_CBC_SHA
<--- Handshake()
<--- Certificate()
[+] Server certificate length: 1652
<--- Handshake()
<--- ServerHelloDone()
--> Alert()
[+] Closing TCP connection!

$ ./tls_getcert.py https://www.live.com/
--> ClientHello()
<--- Handshake()
<--- ServerHello()
[+] server randomness: 5E9D052EFD33DC0F286721B02E731E9798F9E531693F3F6FB429B4561D8647CA
[+] server timestamp: 2020-04-20 05:13:02
[+] TLS session ID: 061600007D202EE4C11CAD43A98AE9F860C4FF4C41F9FDC48C5083FB5D9A35F7
[+] Cipher suite: TLS_RSA_WITH_AES_256_CBC_SHA
<--- Certificate()
[+] Server certificate length: 2015
<--- ServerHelloDone()
--> Alert()
[+] Closing TCP connection!
```

Task: TLS getcert

- Use Wireshark to see what bytes are actually sent out on the wire
 - Use capture filters 'host www.twitter.com and port 443'
- NB! One TLS record can contain several handshake messages
- Unix timestamp can be obtained using `int(time.time())`
- Unix timestamp can be printed using:
`datetime.datetime.fromtimestamp(int(time.time())).strftime('%Y-%m-%d %H:%M:%S')`
- [http://blog.fourthbit.com/2014/12/23/
traffic-analysis-of-an-ssl-slash-tls-session](http://blog.fourthbit.com/2014/12/23/traffic-analysis-of-an-ssl-slash-tls-session)

MTAT.07.017

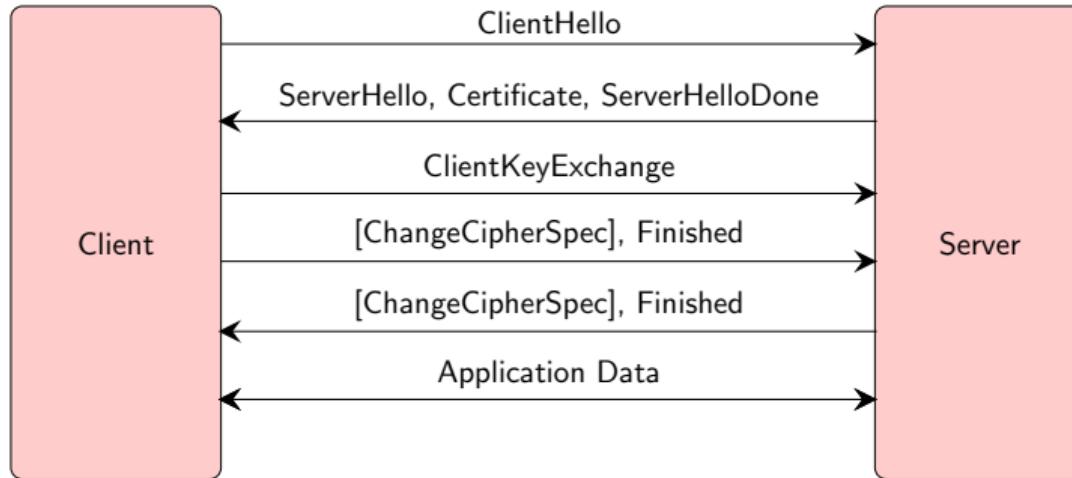
Applied Cryptography

Transport Layer Security (TLS)
Advanced Features

University of Tartu

Spring 2020

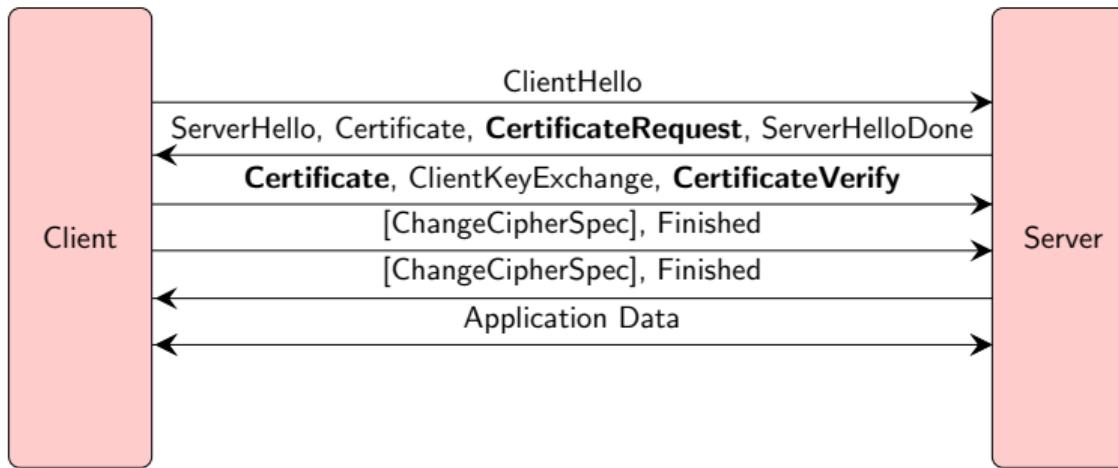
Server-authenticated TLS



Client usually is authenticated on the application level by some shared secret (e.g., password). This can go wrong:

- Server can be impersonated
- Server can be compromised
- Password can be reused in another service
- Password can be guessed
- Password can be phished

Client Certificate Authentication

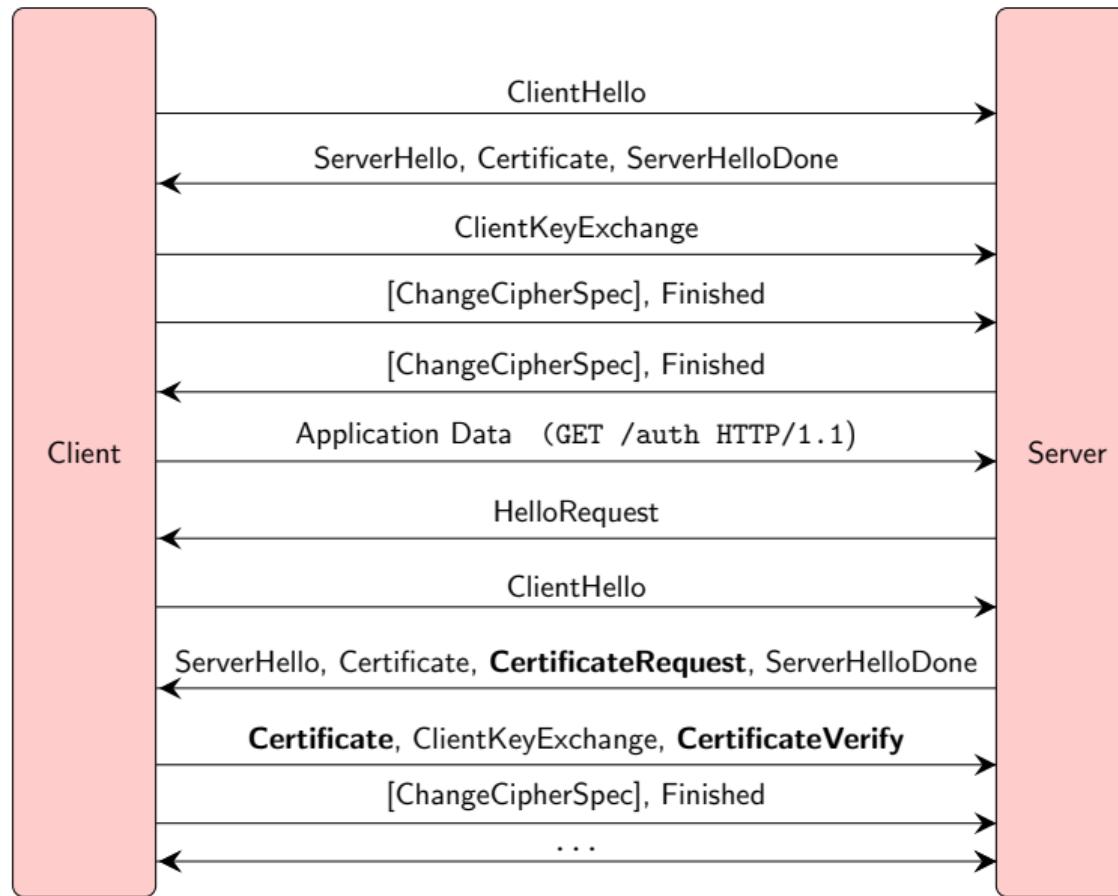


- CertificateVerify – signature over all handshake messages
 - Can CertificateVerify be reused in another handshake?
 - Why CertificateVerify is after ClientKeyExchange?
 - Client's Certificate is sent before ChangeCipherSpec
 - Client proves its identity by signing and not by decrypting
 - Solves most of the problems of password authentication

Renegotiation

- Any party can initiate negotiation of a new TLS session:
 - Client by sending ClientHello
 - Server by sending HelloRequest
- Handshake messages of the new TLS session are protected by the cipher suite negotiated in the previous TLS session
- Used by the server to renegotiate stronger cipher suite or to request a client certificate authentication if on the application level a client tries to access some resource that require such security measure
- Client-initiated renegotiation usually disabled on the server

Certificate request on renegotiation



TLS decryption

<https://www.eff.org/document/20141228-spiegel-scarlefever-program-attack-secure-sockets-layer-ssl-tls>

UNCLASSIFIED

RSA Keys (Stating the Obvious)

If the Key Exchange type is RSA:

- If we can get a hold of the server's RSA private key, we can decrypt the Client Key Exchange message and read the pre-master secret key. No other heavy work need be done.
- Valid for life of certificate

UNCLASSIFIED

Can we prevent it?

Perfect Forward Secrecy



PFS is achieved by using the server's long-term private key to authenticate a short-term/ephemeral asymmetric key that is used to encrypt the actual data.

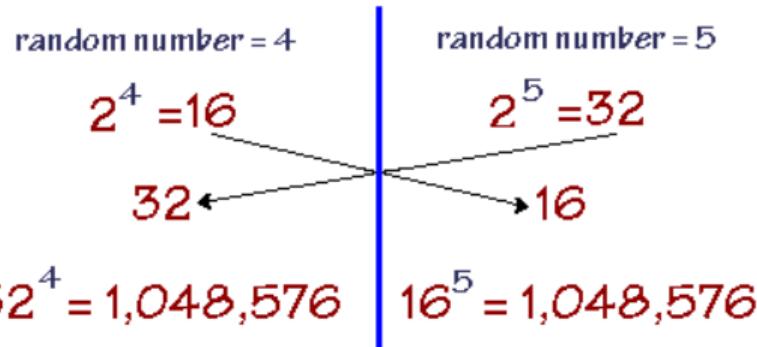
Benefits:

- Attacker who has compromised server's private key cannot decrypt network traffic
 - Attacker has to execute active MITM attacks
- Attacker has to crack x asymmetric keys to decrypt x sessions made to the server

Used in TLS cipher suites: TLS_(EC)DHE_RSA_WITH_*

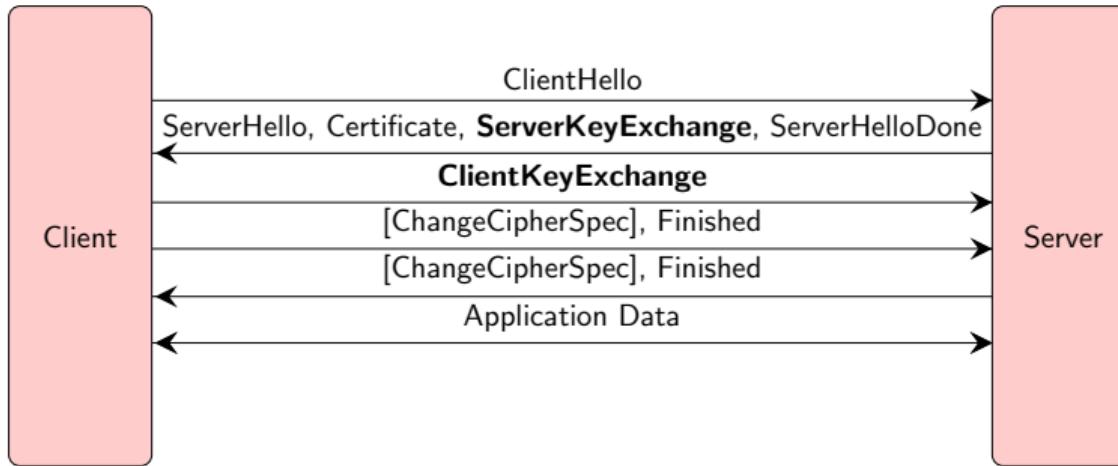
Diffie-Hellman (DH) Key Exchange

common number = 2



- $(2^5)^4 = 2^{5 \cdot 4} = (2^4)^5$
- In practice multiplicative group of integers modulo p is used
- Discrete logarithm problem:
 - hard to find x, given $2^x = 32 \pmod{p}$
- ElGamal and DSA based on DL problem
- Secure against passive eavesdropping

(EC)Diffie-Hellman Key Exchange



- ServerKeyExchange contains DH group, server's DH public key and server's RSA signature over DH public key, client randomness and server randomness
- ClientKeyExchange contains client's DH public key
- How is “pre-master secret” calculated?
- Handshake requires two public key operations (DH+RSA)
- Achieves perfect forward secrecy

TLS extensions

- ClientHello can contain length-prefixed extensions
- ServerHello will contain a response to client's extensions
- Most popular extensions:
 - Server Name Indication (SNI) extension (RFC 3546)

```
▼ Extension: server_name
  Type: server_name (0x0000)
  Length: 17
  ▼ Server Name Indication extension
    Server Name list length: 15
    Server Name Type: host_name (0)
    Server Name length: 12
    Server Name: www.eesti.ee
```

- TLS Session Tickets (RFC 5077)

```
▼ Extension: SessionTicket TLS
  Type: SessionTicket TLS (0x0023)
  Length: 180
  Data (180 bytes)
```

- Elliptic Curves (RFC 4492)

```
▼ Extension: elliptic_curves
  Type: elliptic_curves (0x000a)
  Length: 8
  Elliptic Curves Length: 6
  ▼ Elliptic curves (3 curves)
    Elliptic curve: secp256r1 (0x0017)
    Elliptic curve: secp384r1 (0x0018)
    Elliptic curve: secp521r1 (0x0019)
```

Task: TLS client – 6p

Implement TLS v1.2 client that can obtain HTTP GET response:

```
$ ./tls_client.py https://127.0.0.1:4433/
--> ClientHello()
<--- Handshake()
    <--- ServerHello()
        [+] server randomness: 5EA61ACB07EED9CFFE9312B60CB06CC62F779636AFD8CB28C0DC295FB7210AB9
        [+] server timestamp: 2020-04-27 02:35:39
        [+] TLS session ID:
        [+] Cipher suite: TLS_RSA_WITH_RC4_128_SHA
<--- Handshake()
    <--- Certificate()
        [+] Server certificate length: 554
<--- Handshake()
    <--- ServerHelloDone()
--> ClientKeyExchange()
--> ChangeCipherSpec()
--> Finished()
<--- ChangeCipherSpec()
<--- Handshake()
    <--- Finished()
--> Application_data()
GET / HTTP/1.0
<--- Application_data()
HTTP/1.0 200 OK
Content-Length: 6

Hello!
[+] Closing TCP connection!
```

Task: TLS client

- Client has to support `TLS_RSA_WITH_RC4_128_SHA` cipher suite
- Template contains fully implemented `PRF()`, `derive_master_secret()`, `derive_keys()`, `encrypt()`, `decrypt()` and client/server Finished hash calculation code
 - Make sure that the correct inputs are provided to these functions (!!!)
- TLS client should work on `www.facebook.com`
- Grading:
 - 2 points if the server accepts your `ClientKeyExchange` message
 - 2 points if the server accepts your `Finished` message
 - 2 points if your code can show HTTP response
- `tls_server` binary can be used for development (port 4433)
- Wireshark “Decode As” – “TCP Destination 4433” – “SSL”

Debugging

```
$ ./tls_server --port 4433
[+] Connection from 127.0.0.1:36098
<--- Handshake()
    <--- ClientHello()
        [+] version: 0303
        [+] client randomness: 5EA61ACBF1615730E2F11004E6F2588A74D35C2A6B879F7B57CDEEB049B3C9CB
        [+] client timestamp: 2020-04-27 02:35:39
        [+] TLS session ID:
        [+] Cipher suites:
            TLS_RSA_WITH_RC4_128_SHA
        [+] Compression methods:
            null
        [+] Extensions length: 0
--> ServerHello()
    [+] server randomness: 5EA61ACB07EED9CFFE9312B60CB06CC62F779636AFD8CB28C0DC295FB7210AB9
    [+] server timestamp: 2020-04-27 02:35:39
    [+] TLS session ID:
    [+] Cipher suite: TLS_RSA_WITH_RC4_128_SHA
--> Certificate()
    [+] Server certificate length: 554
--> ServerHelloDone()
<--- Handshake()
    <--- ClientKeyExchange()
        [+] PreMaster length: 128
        [+] PreMaster (encrypted): 6dcc4be09228d6cf824b472148d94f83f3e795c1bf5aad382fb9d7afcffbeb4c830a1a00db45c7eff58258de40f09389be3b213cc0a8d924ab47ab05400f78
        [+] PreMaster: 03033a509170bbec36bada0ac14fd371c571dabf6f75cd6988304ef482894c456450d82e7ab031deeeb4d5e66724d60f
<--- ChangeCipherSpec()
    [+] Applying cipher suite:
        [+] master_secret = PRF(03033a509170bbec36bada0ac14fd371c571dabf6f75cd6988304ef482894c456450d82e7ab031deeeb4d5e66724d60f, "master secret" + 5ea61
        [+] master_secret: 331ed80b70ac83297ae0e2bf574e1878fb77a81e9ef1fafdeab696db1764737ea71362d9b540dc7df553f7f5e95f67
        [+] client_mac_key: fed3adc88628915961391c9973c4471b41c123
        [+] server_mac_key: f6c7f9e459b860205737615bf6aef4828c4e824f
        [+] client_enc_key: dbbc87574da813fbaba39fc9156833b
        [+] server_enc_key: 07aa8106a4518574f1e3378ae4ee9700
<--- Handshake()
    <--- Finished()
    [+] client_verify (received): 2612c639ae9ac7eb5dc84881
    [+] client_verify (calculated): 2612c639ae9ac7eb5dc84881
--> ChangeCipherSpec()
--> Finished()
<--- Application_data()
GET / HTTP/1.0
```

RC4 (TLS_RSA_WITH_RC4_128_SHA)

```
$ ./tls_client.py https://www.swedbank.ee/
--> ClientHello()
<--- Alert()
[-] fatal: 40

$ ./tls_client.py https://www.eesti.ee/
--> ClientHello()
<--- Alert()
[-] fatal: 40

$ ./tls_client.py https://www.facebook.com/
--> ClientHello()
<--- Handshake()
    <--- ServerHello()
        [+]
server randomness: E2F4273E8D8F964BD8EA1290CF93
        [+]
server timestamp: 2090-08-29 04:00:46
        [+]
TLS session ID: AFBE4C85228EAE878D64B7DCCE510E6
        [+]
Cipher suite: TLS_RSA_WITH_RC4_128_SHA
<--- Handshake()
    <--- Certificate()
        [+]
Server certificate length: 1798
<--- Handshake()
    <--- ServerHelloDone()
--> ClientKeyExchange()
--> ChangeCipherSpec()
--> Finished()
<--- ChangeCipherSpec()
<--- Handshake()
    <--- Finished()
--> Application_data()
GET / HTTP/1.0
<--- Application_data()
HTTP/1.1 301 Moved Permanently
Vary: Accept-Encoding
Location: https://www.facebook.com/
Content-Type: text/html; charset="utf-8"
X-FB-Debug: 301zzVa6cieErTa/IN7GigNt7c8r2cKUkrv4ap3wifQI6xo
Date: Sun, 26 Apr 2020 22:26:34 GMT
Alt-Svc: h3-27e=:443"; ma=3600
Connection: close
Content-Length: 0
[+] Closing TCP connection!
```

SSL Report: www.facebook.com (157.240.18.35)

Assessed on: Sun, 26 Apr 2020 21:01:40 UTC | HIDDEN | [Clear cache](#)

Qualys. SSL Labs

Summary

Overall Rating: **B**



Overall Rating: B

Category	Score
Certificate	100
Protocol Support	70
Key Exchange	88
Cipher Strength	88

Visit our [documentation page](#) for more information, configuration guides, and books. Known issues are documented [here](#).

This server accepts RC4 cipher, but only with older protocols. Grade capped to B. [MORE INFO ↗](#)

This server supports TLS 1.0 and TLS 1.1. Grade capped to B. [MORE INFO ↗](#)

This server supports TLS 1.3.

Static Public Key Pinning observed for this server.

HTTP Strict Transport Security (HSTS) with long duration deployed on this server. [MORE INFO ↗](#)

DNS Certification Authority Authorization (CAA) Policy found for this domain. [MORE INFO ↗](#)

Most common pitfalls

- Server fails to verify MAC of client's Finished message
 - Make sure client's Finished message is encrypted using the correct keys. Compare keys – if they are different make sure key derivation receives the correct values of premaster secret, client and server randomness
- Server fails to verify hash in client's Finished message
 - Make sure all handshake messages sent and received are appended to the `handshake_messages` variable
- Client fails to verify hash in server's Finished message
 - Plaintext version of client's Finished message must be appended to the `handshake_messages`
- Server returns fatal Alert "decryption failed" after receiving client's Finished message
 - Make sure the server did not choose non-RC4 cipher suite

MTAT.07.017

Applied Cryptography

The Onion Router (Tor)

University of Tartu

Spring 2020

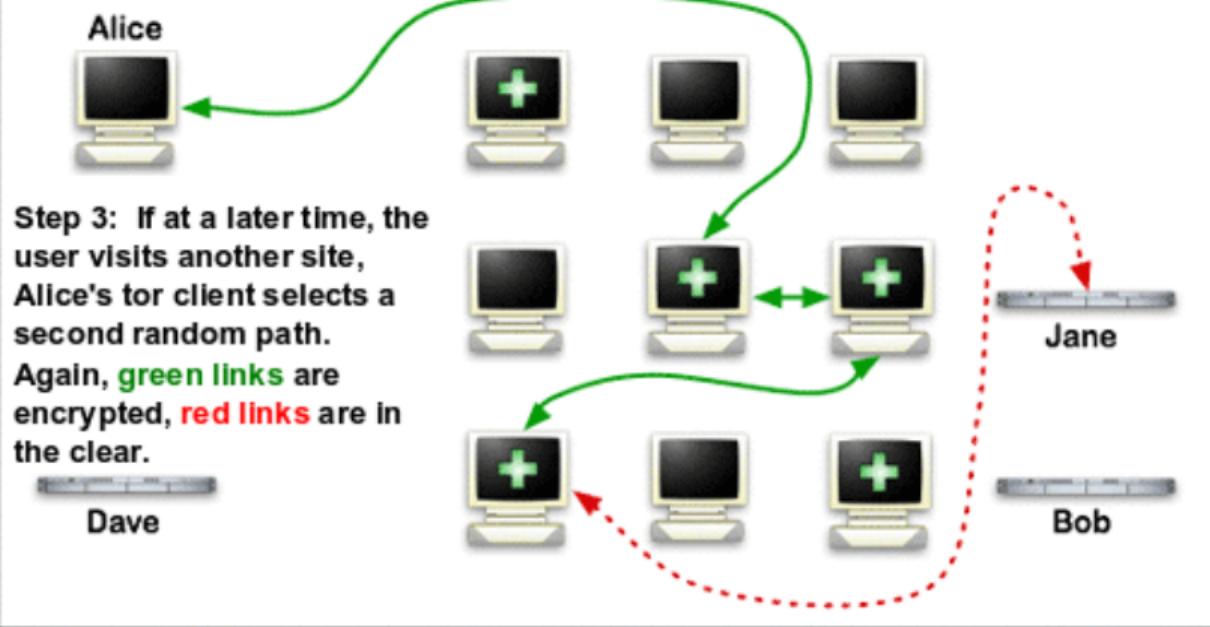
Tor

“Tor is a software for enabling online anonymity and censorship resistance. Tor directs Internet traffic through a free, worldwide, volunteer network consisting of more than seven thousand relays to conceal a user’s location or usage from anyone conducting network surveillance or traffic analysis.”

[https://en.wikipedia.org/wiki/Tor_\(anonymity_network\)](https://en.wikipedia.org/wiki/Tor_(anonymity_network))

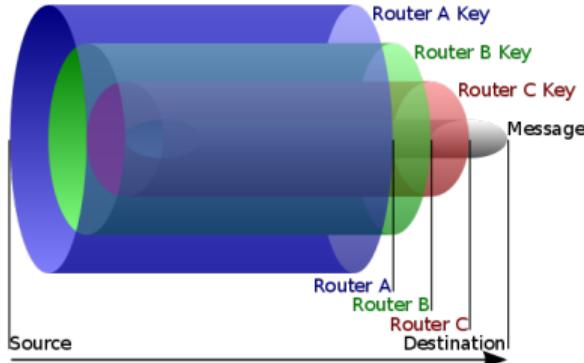
 How Tor Works: 3

 Tor node
---> unencrypted link
----> encrypted link

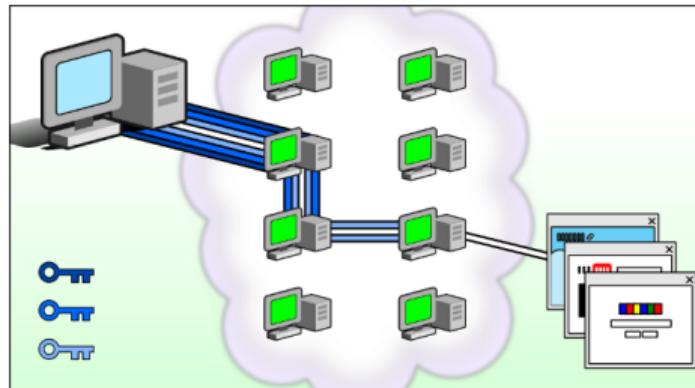


- No single node knows the entire path!

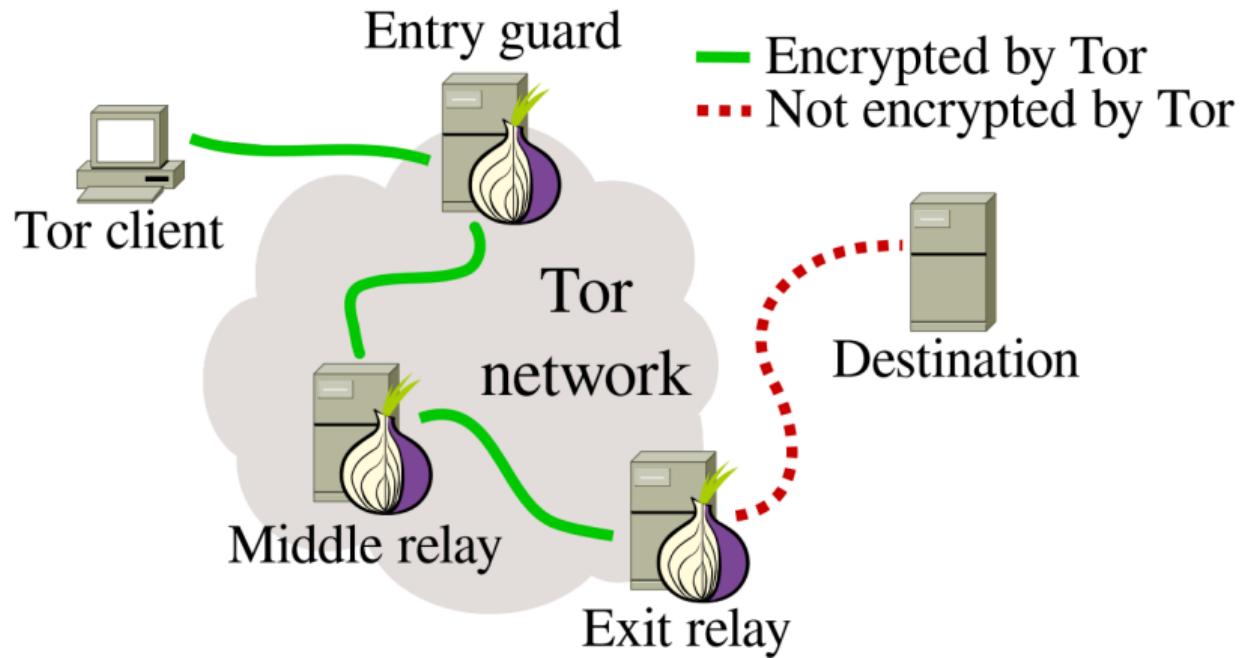
Onion Routing



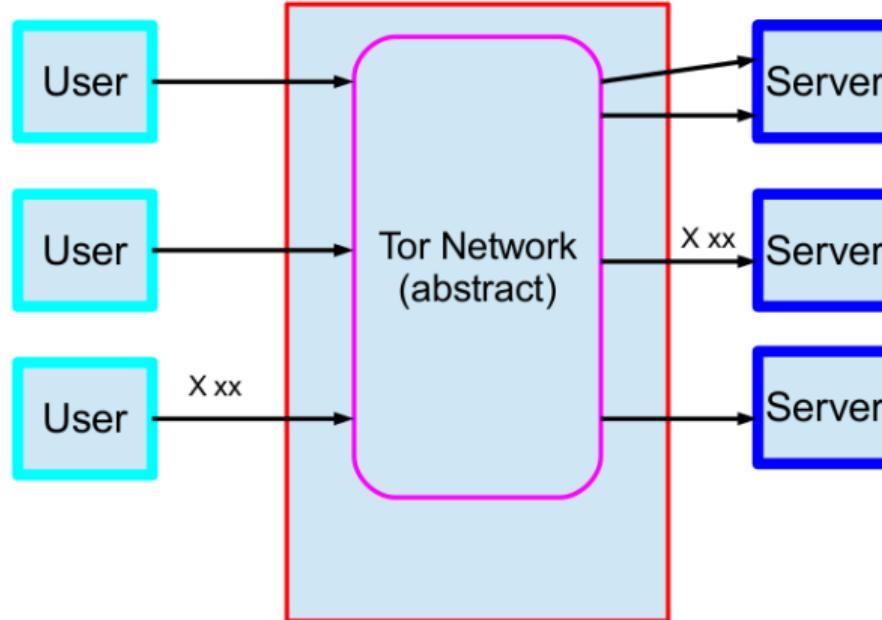
- Data received by router A:
 - $E_a(b, E_b(c, E_c(dest, m)))$
- Data received by router B:
 - $E_b(c, E_c(dest, m))$
- Data received by router C:
 - $E_c(dest, m)$



Tor Nodes

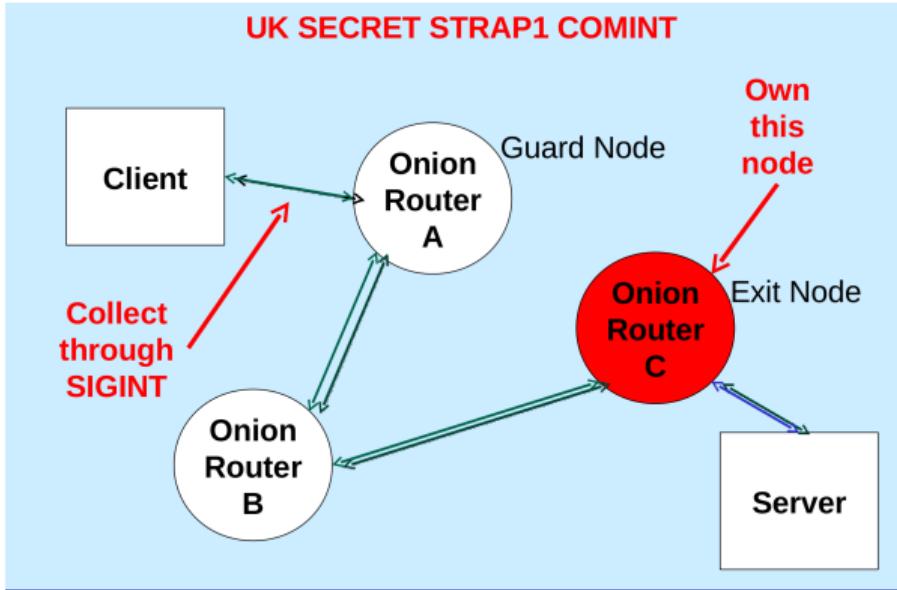


Traffic correlation attack



- End-to-end correlation
- No need to know the full path
- Entry guard should not be rotated frequently

GCHQ attacks on Tor



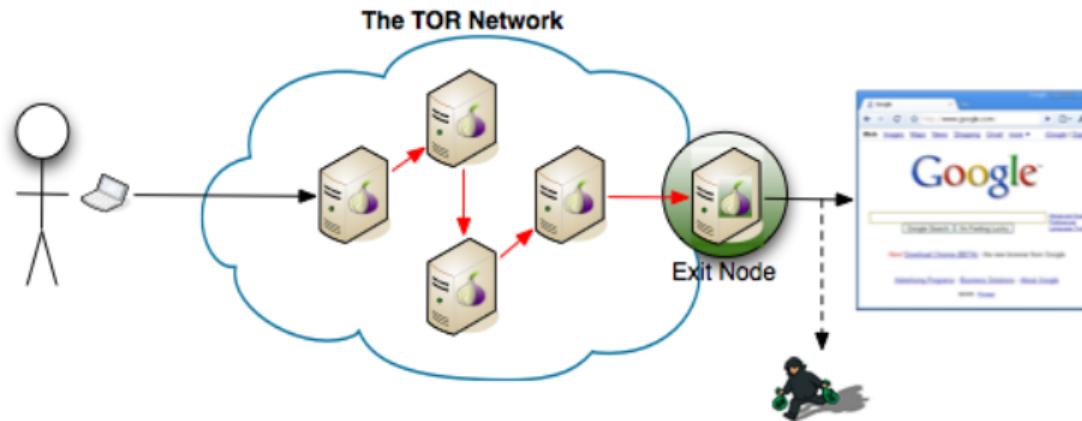
This information is exempt from disclosure under the Freedom of Information Act 2000 and may be subject to exemption under other UK information legislation. Refer disclosure requests to GCHQ on [REDACTED]

© Crown Copyright. All rights reserved.

https://archive.org/details/spiegel_-_media-35538

- How many Tor nodes are run by GCHQ/NSA?
- Tor nodes: <https://torstatus.rueckgr.at/>

Malicious exit node attacks



- Sniffing
 - WikiLeaks
- Man-in-the-middle attacks
 - “Spoiled Onions: Exposing Malicious Tor Exit Relays”

Malicious exit node attacks

Fingerprint	IP addresses	Country	Bandwidth	Attack	Sampling rate	First active	Discovery
F8FD29D0†	176.99.12.246	Russia	7.16 MB/s	HTTPS MitM	<i>unknown</i>	2013-06-24	2013-07-13
8F9121BF†	64.22.111.168/29	U.S.	7.16 MB/s	HTTPS MitM	<i>unknown</i>	2013-06-11	2013-07-13
93213A1F†	176.99.9.114	Russia	290 KB/s	HTTPS MitM	50%	2013-07-23	2013-09-19
05AD06E2†	92.63.102.68	Russia	5.55 MB/s	HTTPS MitM	33%	2013-08-01	2013-09-19
45C55E46†	46.254.19.140	Russia	1.54 MB/s	SSH & HTTPS MitM	12%	2013-08-09	2013-09-23
CA1BA219†	176.99.9.111	Russia	334 KB/s	HTTPS MitM	37.5%	2013-09-26	2013-10-01
1D70CDED†	46.38.50.54	Russia	929 KB/s	HTTPS MitM	50%	2013-09-27	2013-10-14
EE215500†	31.41.45.235	Russia	2.96 MB/s	HTTPS MitM	50%	2013-09-26	2013-10-15
12459837†	195.2.252.117	Russia	3.45 MB/s	HTTPS MitM	26.9%	2013-09-26	2013-10-16
B5906553†	83.172.8.4	Russia	850.9 KB/s	HTTPS MitM	68%	2013-08-12	2013-10-16
EFF1D805†	188.120.228.103	Russia	287.6 KB/s	HTTPS MitM	61.2%	2013-10-23	2013-10-23
229C3722	121.54.175.51	Hong Kong	106.4 KB/s	sslstrip	<i>unsampled</i>	2013-06-05	2013-10-31
4E8401D7†	176.99.11.182	Russia	1.54 MB/s	HTTPS MitM	79.6%	2013-11-08	2013-11-09
27FB6BB0†	195.2.253.159	Russia	721 KB/s	HTTPS MitM	43.8%	2013-11-08	2013-11-09
0ABB31BD†	195.88.208.137	Russia	2.3 MB/s	SSH & HTTPS MitM	85.7%	2013-10-31	2013-11-21
CADA00B9†	5.63.154.230	Russia	187.62 KB/s	HTTPS MitM	<i>unsampled</i>	2013-11-26	2013-11-26
C1C0EDAD†	93.170.130.194	Russia	838.54 KB/s	HTTPS MitM	<i>unsampled</i>	2013-11-26	2013-11-27
5A2A51D4	111.240.0.0/12	Taiwan	192.54 KB/s	HTML Injection	<i>unsampled</i>	2013-11-23	2013-11-27
EBF7172E†	37.143.11.220	Russia	4.34 MB/s	SSH MitM	<i>unsampled</i>	2013-11-15	2013-11-27
68E682DF†	46.17.46.108	Russia	60.21 KB/s	SSH & HTTPS MitM	<i>unsampled</i>	2013-12-02	2013-12-02
533FDE2F†	62.109.22.20	Russia	896.42 KB/s	SSH & HTTPS MitM	42.1%	2013-12-06	2013-12-08
E455A115	89.128.56.73	Spain	54.27 KB/s	sslstrip	<i>unsampled</i>	2013-12-17	2013-12-18
02013F48	117.18.118.136	Hong Kong	538.45 KB/s	DNS censorship	<i>unsampled</i>	2013-12-22	2014-01-01
2F5B07B2	178.211.39	Turkey	204.8 KB/s	DNS censorship	<i>unsampled</i>	2013-12-28	2014-01-06
4E2692FE	24.84.118.132	Canada	52.22 KB/s	OpenDNS	<i>unsampled</i>	2013-12-21	2014-01-06

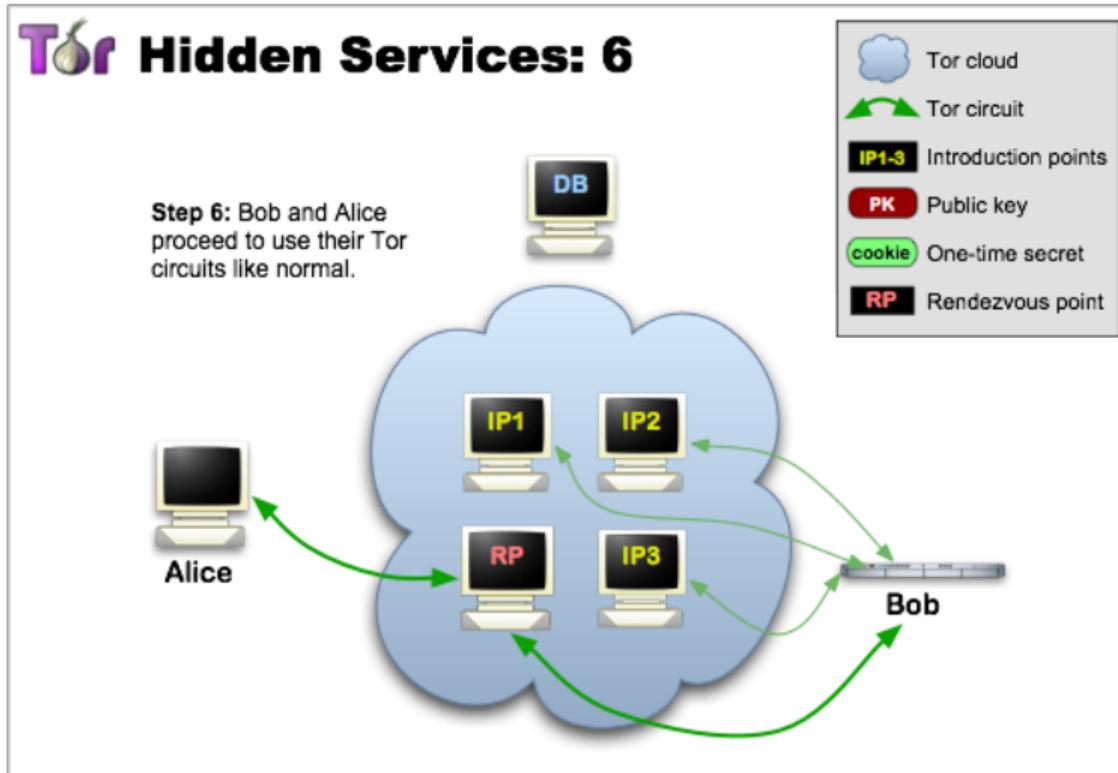
Deanonymization attacks

- Browser fingerprinting
- IP leakage through DNS requests
- Client-side exploits
- Tor Browser Bundle



Tor Onion Services

Reachable using .onion address (e.g., `duskgytldkxiuqc6.onion`)



Tor Onion Services (setup)

```
# apt install tor

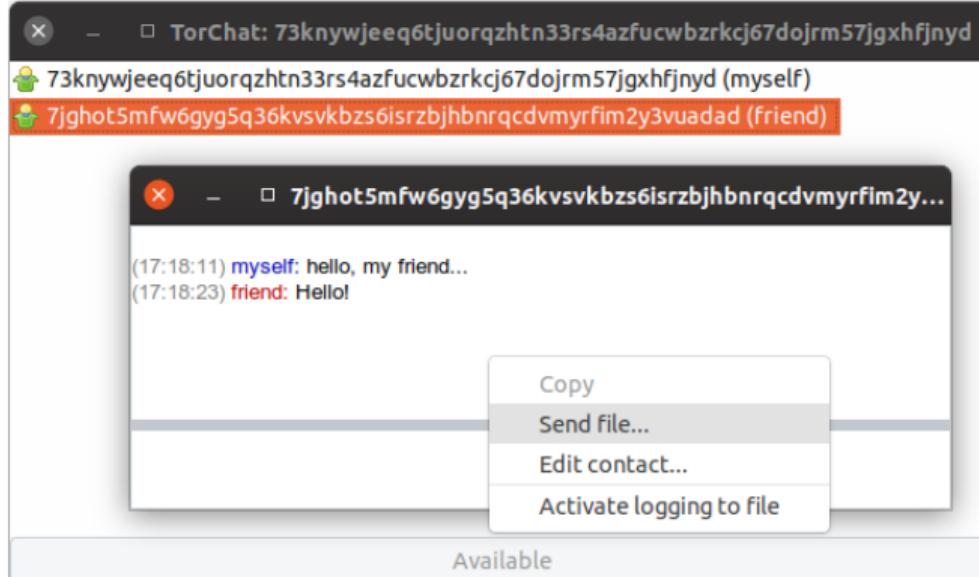
# cat /etc/tor/torrc | grep HiddenService
HiddenServiceDir /var/lib/tor/hidden_service/
HiddenServicePort 80 127.0.0.1:80

# /etc/init.d/tor restart

# hexdump -C /var/lib/tor/hidden_service/hs_ed25519_secret_key
00000000  3d 3d 20 65 64 32 35 35  31 39 76 31 2d 73 65 63  |== ed25519v1-sec|
00000010  72 65 74 3a 20 74 79 70  65 30 20 3d 3d 00 00 00  |ret: type0 ==...|
00000020  90 e5 5b 2b c4 77 9a 80  bf 2f 63 90 a7 1d e2 73  |..[+.w....c....s|
...
# cat /var/lib/tor/hidden_service/hostname
7jghot5mfw6gyg5q36kvsvkzs6isrzbjhbnrqcdvmyrfim2y3vuadad.onion
```

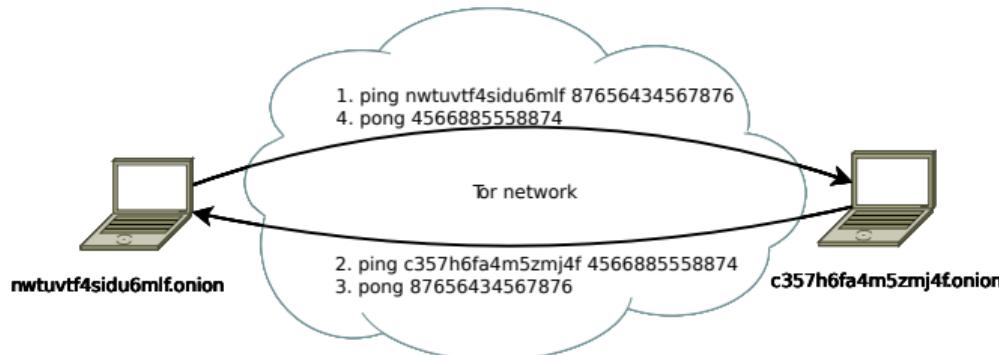
TorChat (Instant Messenger)

```
$ sudo dpkg -i torchat.deb  
$ torchat
```



http://kodu.ut.ee/~arnis/torchat_thesis.pdf

TorChat Protocol

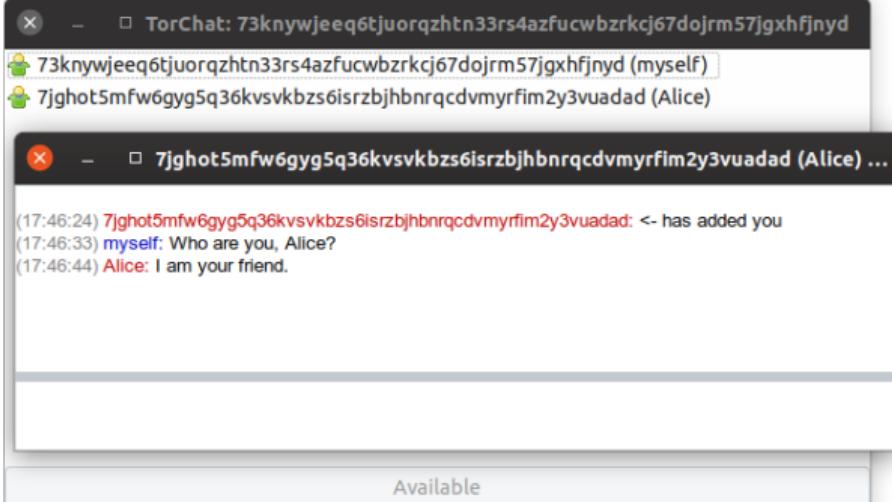


- Two connections:
 - Outgoing connection to send data
 - Incoming connection to receive data
- The handshake process authenticates the incoming connections
- Commands (command separator is the newline \n character):
 - message – UTF-8 encoded message
 - status – available, away or xa
 - add_me – request addition to buddy list
 - profile_name – name in the buddy list

Task: TorChat – 5p

Implement a TorChat client that is compatible with the official TorChat client:

```
$ ./torchat.py --myself 7jghot5mfw6gyg5q36kvszbek6isrzbjhbnrqcdvmyrfim2y3vuadad --peer 73knywjeeq6tjuorqzhtn33rs4azfucwbzrkcj67dojrm57jgxhfjnyd
[+] Connecting to peer 73knywjeeq6tjuorqzhtn33rs4azfucwbzrkcj67dojrm57jgxhfjnyd
[+] Sending: ping 7jghot5mfw6gyg5q36kvsbek6isrzbjhbnrqcdvmyrfim2y3vuadad 36251593687951013345829332738750401089
[+] Listening...
[+] Client 127.0.0.1:49900
[+] Received: ping 73knywjeeq6tjuorqzhtn33rs4azfucwbzrkcj67dojrm57jgxhfjnyd 114550899558407750467319526697617679065735261062805617012379414337324471905588
[+] Received: pong 36251593687951013345829332738750401089
[+] Incoming connection authenticated!
[+] Sending: pong 114550899558407750467319526697617679065735261062805617012379414337324471905588
[+] Received: client TorChat
[+] Received: version 0.9.9.553
[+] Received: profile_name Bob
[+] Received: status available
[+] Sending: add_me
[+] Sending: status available
[+] Sending: profile_name Alice
[+] Received: status available
[+] Received: message Who are you, Alice?
[?] Enter message: I am your friend.
[+] Sending: message I am your friend.
[+] Received: status available
[+] Received: status available
[+] Received: status available
```



The screenshot shows two windows of the official TorChat client. The top window is titled 'TorChat: 73knywjeeq6tjuorqzhtn33rs4azfucwbzrkcj67dojrm57jgxhfjnyd'. It contains a list of users: '73knywjeeq6tjuorqzhtn33rs4azfucwbzrkcj67dojrm57jgxhfjnyd (myself)' and '7jghot5mfw6gyg5q36kvsbek6isrzbjhbnrqcdvmyrfim2y3vuadad (Alice)'. The bottom window is titled '7jghot5mfw6gyg5q36kvsbek6isrzbjhbnrqcdvmyrfim2y3vuadad (Alice) ...'. It shows a message history: '(17:46:24) 7jghot5mfw6gyg5q36kvsbek6isrzbjhbnrqcdvmyrfim2y3vuadad: <- has added you', '(17:46:33) myself: Who are you, Alice?', and '(17:46:44) Alice: I am your friend.' Below the windows, a status bar says 'Available'.

Task: TorChat

- Setup a Tor onion service for port 11009 (redirected to 127.0.0.1:8888)
- Connect to peer's .onion address on port 11009 and send the ping command with a random cookie
 - The cookie must be a random decimal number with 128-bit entropy
- Listen on 127.0.0.1:8888 for peer's connect back
 - Verify that the TorChat ID in the peer's ping command is correct
 - Verify that the cookie in the peer's pong command matches the cookie you sent
- Send the pong command with peer's cookie over the outgoing connection (only after the pong has been verified)
- Send the add_me, status and profile_name commands
- After every message received, read the user input and send the response message (only after the incoming connection has been authenticated)

Server sockets in Python

```
import socket
sserv = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sserv.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
sserv.bind(('', 8888))
sserv.listen(0)

(s, address) = sserv.accept()
print("[+] Client %s:%s" % (address[0], address[1]))
```

- `bind(('', 8888))` and `listen()` listens for client connections on all IPs on all network interfaces
- `accept()` will wait until client connects and will return a tuple:
 - client socket (has `send()` and `recv()` methods)
 - address tuple – IP and port
- `SO_REUSEADDR` forces the kernel to reuse port even if it is in busy (`TIME_WAIT`) state (prevents error when rebinding)

<http://docs.python.org/3/howto/sockets.html>

Most common pitfalls

- Cannot connect to the peer
 - Check that Tor service is running
 - Check that the official TorChat is running and its status is online
 - Check that the “.onion” suffix in the hostname is not missing
- Connect-back from the peer not received
 - Check that your TorChat ID in ping command is correct
 - Check that your pong command ends with the command separator
 - To test whether your onion service is available over the Tor, use:
“torify telnet youronionaddress.onion 11009”
- The peer ignores your commands
 - Check that your commands are sent over the correct socket
 - Check that what you print is what you send

Questions

- What is the security objective Tor tries to achieve?
- Tor middle node sees only encrypted packets. How it is achieved?
- What could a malicious Tor exit node do?
- What could a malicious Tor middle node do?
- What could a malicious Tor entry guard node do?
- How to detect whether the user is using Tor network?
- Under what threat model Tor is secure?
- How are Tor Onion Services identified?
- What prevents someone from impersonating Tor Onion Service?

MTAT.07.017
Applied Cryptography

Bitcoin

University of Tartu

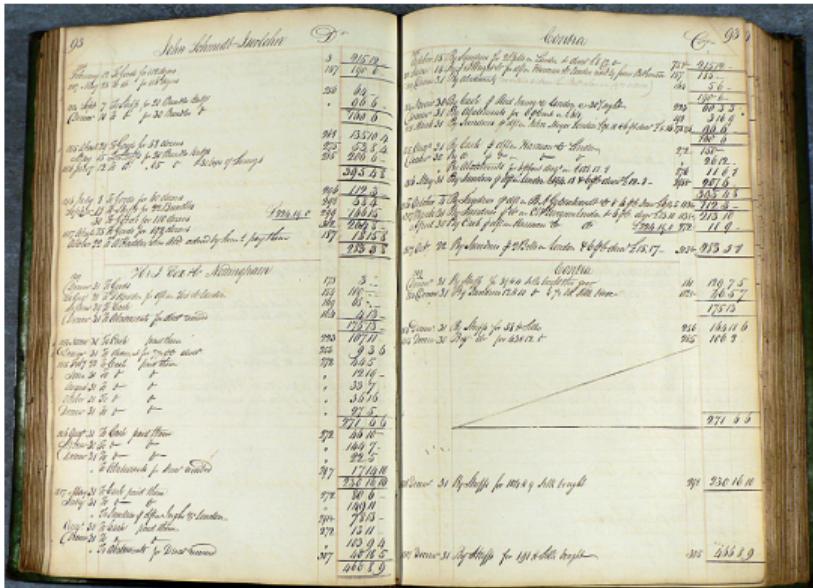
Spring 2020

Bitcoin

“Bitcoin is a cryptocurrency whereby the creation and transfer of bitcoins is facilitated by an open-source peer-to-peer cryptographic protocol that functions without the intermediation of any central authority.”

<http://en.wikipedia.org/wiki/Bitcoin>

Traditional banks



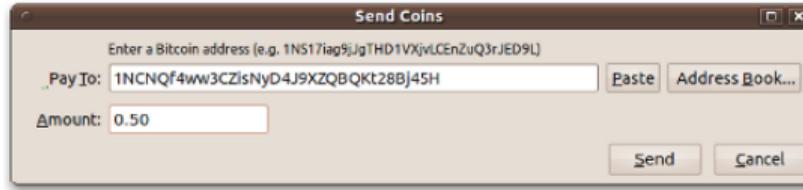
- Authenticates account holders and performs transactions
 - Provides authenticity of transaction log
 - Resolves disputes

How to do that without a trusted central authority?

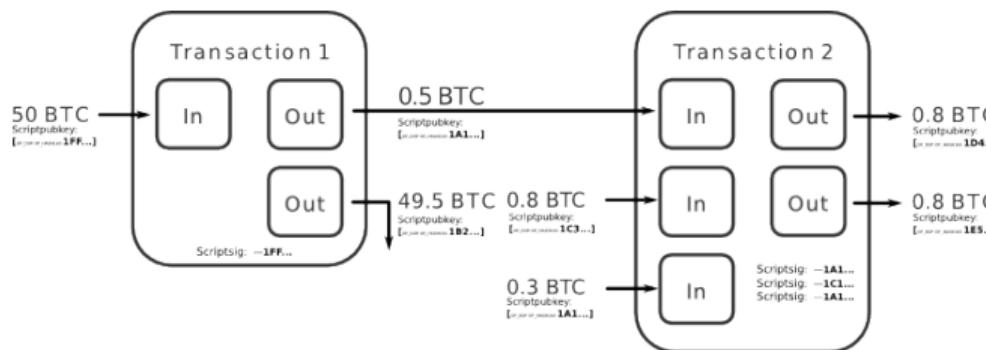
Bitcoin

- How to maintain the transaction log without central authority?
 - Distribute to everyone over a peer-to-peer network
- How to verify account holder's intent without central authority?
 - Account holder signs transactions using digital signature
- How to bind account holder's identity to public key?
 - Public key is an identity / account number itself
 - Anyone who can sign using the key can respond the coins
 - Transactions are made between public keys
- How to verify the integrity of transaction log without central authority?
 - By majority vote using computing power
 - Requires active participation by honest majority
- How to get coins into the system?
 - Deterministic amount of money supplied through a lottery

Transaction

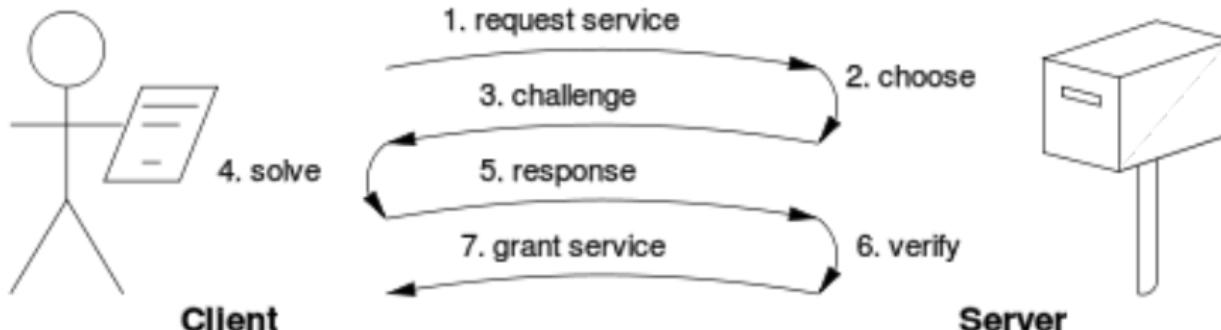


- Address is a hash of EC public key
- One who can produce the signature can claim the money



- Every input must be unlocked by a signature
- Transaction is valid if signatures are valid and inputs unspent
- The difference between inputs and outputs is a transaction fee

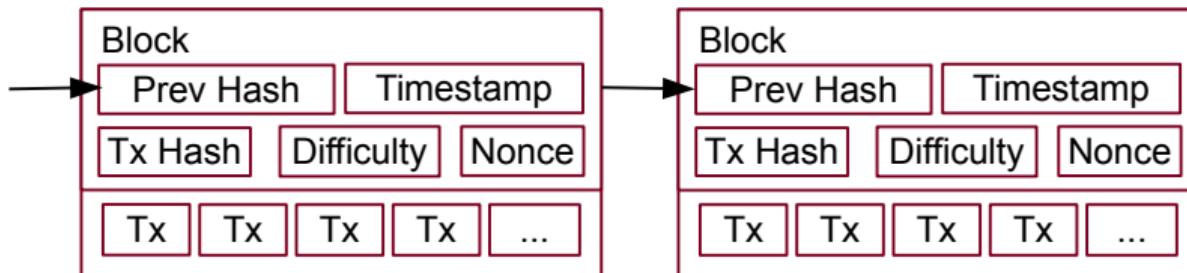
Proof-of-work system



Hashcash:

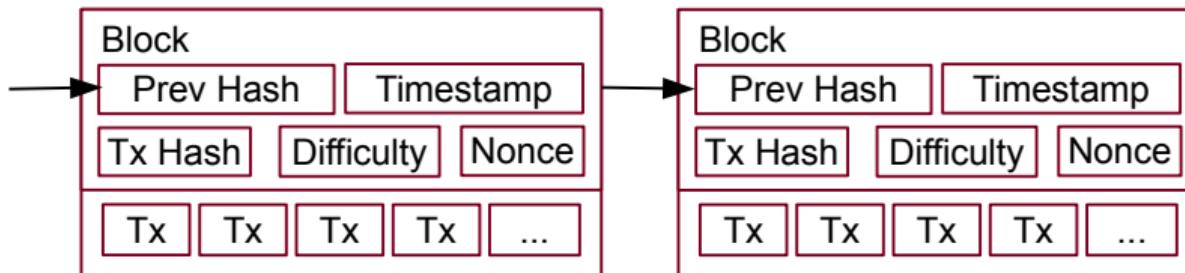
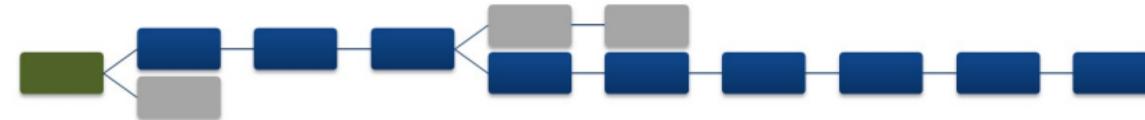
- Challenge: find a *nonce* such that first x bits of $\text{hash}(\text{random_challenge} \parallel \text{nonce})$ are zero bits.
- Solution requires a brute force: 2^x tries on average
- Verification requires a single hash operation
 $\text{hash}(\text{random_challenge} \parallel \text{nonce}) == "000000..."$?
- Non-interactive proof-of-work to fight spam

Blockchain (transaction log)



- Blocks are produced by miners who solve proof-of-work
- Miner earns 12.5 BTC “out of thin air”
 - Halved every 210'000 blocks (4 years)
- Miner collects all transaction fees

Blockchain (transaction log)



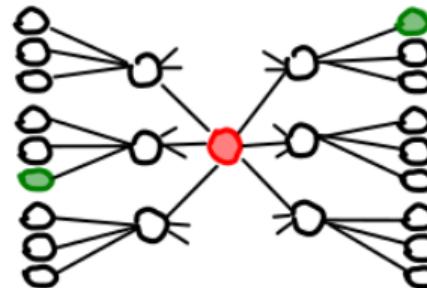
- The chain with the largest total difficulty is the consensus chain
- The proof-of-work difficulty is recalculated every 2016 blocks
 - To produce one block in 10 minutes
 - Difficulty cannot change more than by a factor of 4
 - Current difficulty – 76 bits

Bitcoin peer-to-peer network

- Node listens on TCP port 8333
- Node connects to few other nodes
- Node advertises to other peers:
 - new transactions
 - new blocks
 - new peer addresses
 - blocks (on request)
 - block headers (on request)
 - peer addresses (on request)
- Node must not relay invalid blocks/transactions
- Node must implement DoS protection

Anonymity

- All transactions are public and traceable
- Transactions occur between public keys
- Backward and forward privacy is needed
- Solution: mixing services



Security assumptions

- ECDSA scheme and SHA256/RIPEMD160 are secure
- Attacker does not control majority of the hashpower
 - Attacker could execute double-spending attacks
 - Attacker could destroy the network
 - Attacker could gain more by following the rules
 - Hashpower is not distributed uniformly
 - Litecoin's use of scrypt()
- Attacker cannot partition the network or isolate participants
 - Sybil attack
 - Forked chains cannot be merged
 - Profit by isolating other miners

Requirements

- Participants are able to store and verify the transaction log
 - Transaction log size is 276 GB (excluding indexes)
 - Thin clients must trust power nodes
- Participants are rational
 - Indirect incentive to keep the network healthy
- No one can impose regulation
 - Regulation needed to fix security flaws
 - Changes without unanimous support will fork blockchain
 - Bitcoin software developers have an advantage here
 - Regulation needed to stop bitcoin thefts

Bitcoin security depends on a lot more than cryptography

Opening Pandora's box

NOTICE OF EXTORTION

Your business, 900 Degrees Neapolitan Pizzeria, has been targeted for extortion. The selection process is random, and was not triggered by any event under your control.

Should you fail to pay the one-time monetary tribute, by the deadline provided below, your business will be **severely and irreparably damaged**. The following methods are commonly employed in cases of non-compliance:

- Negative Online Reviews
- BBB Complaints
- Harassing Telephone Calls
- Threatening Delivery Orders
- Telephone Denial-of-Service
- Bomb Threats
- Vandalism
- Mercury Contamination

- Health Code Violations
- OSHA Violations
- Criminal Tax Evasion
- Money Laundering
- Illegal Drug Sales
- Marijuana Grow Operations
- Methamphetamine Production
- Terrorist Training Activity

The tribute price is only One Bitcoin (1 BTC), but must be paid by **August 15, 2014**. Payment is to be made to the Bitcoin Wallet Address listed below.

If payment is not received, our team will begin taking the actions listed above. Once engagement has begun, it can only be stopped for a tribute of Three Bitcoin (3 BTC). Because many of the actions we take are catastrophic and irreversible, is it advised to pay the tribute before the deadline is reached.

Payment Type: Bitcoin

Deadline: August 15, 2014

Amount Due: 1 Bitcoin
(If paid before deadline)

Amount Due: 3 Bitcoin
(If paid after deadline)

Purchase Bitcoin @

<https://www.coinbase.com/>



17gt1BancvtnnJwy4BA41VBUH3pfbUvzE

Extortion in Estonia

Shoppers evacuated from multiple Rimi supermarkets in Western and Eastern Estonia following bomb threats



On Wednesday afternoon, a bomb threat prompted Rimi supermarket locations in Pärnu, Kuressaare, Haapsalu and Jõhvi to be evacuated and searched.

At 4:15 p.m. on Wednesday afternoon, the Police and Border Guard Board (PPA) was sent a bomb threat via email regarding the Rimi location in Jõhvi's Tsentraal Shopping Centre, Delfi news portal reported.

According to a spokesperson for the PPA's East Prefecture, the store had been searched and shoppers thereafter allowed to reenter the store by 5:55 p.m.

At approximately 4:20 p.m. on the same day, the PPA's North Prefecture was notified via email that there may be a bomb in Rimi. Which town's supermarket was not specified in the email, and so Rimi locations in Pärnu, Haapsalu and Kuressaare were searched. Police found nothing suspicious and shoppers were thereafter allowed to return to the stores.

A similar bomb threat was made via email regarding Prisma supermarkets on Dec. 23. Similarly, the exact location was not specified, which prompted police to take action at multiple locations across Estonia.

<http://news.err.ee/120174/shoppers-evacuated-from-multiple-rimi-supermarkets>

Extortion in Estonia

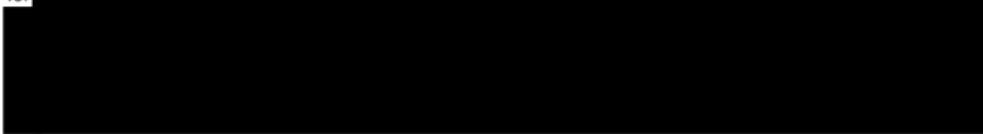
----- Forwarded message -----

From: DD4BC TEAM <[REDACTED]>

Date: Sat, Nov 1, 2014 at 4:57 AM

Subject: DDOS ATTACK!

To: [REDACTED]



Hello

Your site is extremely vulnerable to ddos attacks.

I want to offer you info how to properly setup your protection, so that you can't be ddosed!

My price is 1 Bitcoin only.

Right now I will star small (very small) attack which will not crash your server, but you should notice it in logs.
Just check it.

I want to offer you info on how I did it and what you have to do to prevent it. If interested pay me 1 BTC to
17aLGgw8AwJdqIBtMMG1QtQJgNQQkiyEsp

Thank you.

<https://cybersec.ee/2015/05/14/two-estonian-companies-received-bitcoin-extortion-letters/>

Questions

- How is transaction integrity and authenticity provided in Bitcoin?
- When seeing Bitcoin blockchain how its authenticity can be established?
- Why Bitcoin miners solve blocks?
- Why can't an attacker replace transaction in a solved Bitcoin block?
- How can an adversary which has majority of hashpower destroy Bitcoin system?
- How to open an account in Bitcoin system?
- Who has the control over Bitcoin system?

Task: Proof-of-work solver – 2p

Implement a proof-of-work solving tool:

```
$ ./pow.py --difficulty 26
[+] Solved in 296.456492 sec (0.2112 Mhash/sec)
[+] Input: 41726e69732055540000000003bb67af
[+] Solution: 00000031fc8ad63fa6070e341ccddd55bc36ac0b1e94965f2a8bb624d1a51071
[+]Nonce: 62613423
```

- Hash function – SHA256(SHA256())
- Input: your identity + 8 byte counter
- Difficulty: the number of zero leftmost bits in the solution
- Provide the output for difficulty 26 in the source code comments
 - Must push at least 0.1 Mhash/sec on today's hardware
 - Reject invalid candidates as early as possible
- Verification of proof-of-work:

```
>>> input = codecs.decode('41726e69732055540000000003bb67af', 'hex')
>>> input
b'Arnis UT\x00\x00\x00\x00\x03\xbbg\xaf'
```

```
>>> hashlib.sha256(hashlib.sha256(input).digest()).hexdigest()
'00000031fc8ad63fa6070e341ccddd55bc36ac0b1e94965f2a8bb624d1a51071'
```

Exam

This was the last lecture!

Exam:

- May 21, 2020, 14:15–16:00, online
- Register in OIS
- May 18 23:59 is the final deadline for homework 14
- May 21 is the final deadline for objections regarding homework grading
- Exam will be online – video camera and microphone required
 - <https://button.ut.ee/b/arn-hyy-tmg>
- You will have to show your photo ID (e.g., ID card)
- Closed-book exam

Make sure you are able to authenticate with your ID card to:
<https://cybersec.ee/appcrypto2020/exam/>