KU LEUVEN

CAMPUS DIEPENBEEK

KU LEUVEN
EMBEDDED SYSTEMS & SECURITY

Mini Onderzoeks-Labo

Help choosing the next crypto-standard

Embedded Systems and Security

People

Professors

Teaching Staff

PhD students

Research Expert

Postdocs

Preparation

Preparation

Q1

What **did** you find out about this "formula" ?

cryptology = cryptography + cryptanalysis

First of all we look at the definitions of all these words separately. Cryptology is the study, writing and solving of codes. Cryptography is the art of writing or deciphering messages in code. It hides the message in code. Cryptanalysis is the science of analyzing and breaking of codes and ciphers.

Now we can see that if we combine the definitions of cryptography and cryptoanalysis we become cryptology. We need both of them because cryptography is all about making codes and cryptanalysis is about breaking them or solving them.

Preparation

Q2

What is the difference between Symmetric key and Public key cryptography ?

The fundamental difference between the keys is how they are distributed. The symmetric key is only given to the people that are allowed to encrypt and decrypt the message, no one else gets the key. In contrary, the public key has two different keys, a public key, which is distributed to encrypt the message and a private key, which is used by the receiver to decrypt the message. The advantage of public key encryption is, that in order to send data, the recipient never has to share some important information. The recipient just sends the public key and when they receive the message, only the recipient with the private key can decrypt it. **That is the reason why most of the internet relies on public keys,** with this method, the information sent by a web browser is encrypted and not modifiable without a private key.

Preparation

Q3

What is a cryptographic algorithm ?

A cryptographic algorithm is a mathematical formula that is applied to each digit of something in order to encrypt it. This encrypted message can then be decrypted by the receiver with an encryption key.

Preparation

Q3

What is a cryptographic algorithm ?

Symmetric-key cryptography

- AES
- Twofish
- RC4

Public-key cryptography

- RSA
- DSS
- YAK

Preparation

Q4

Why would you optimise some code or a design towards binary size, **anno 2020** ?

Optimization is something you do to make something go **faster** and to take **less storage space**. The packing and unpacking of information can be made quicker. The quicker it is, the faster people can communicate with each other or with machinery. With more technology we also need more binary code to pass around. More technology (like electric cars) also means we need more protection against hacking. If you take an electric car for example, if this gets hacked it could do major damage and kill people. Therefore we need to optimise the design and code for more safety. We don't want someone intercepting your command and change it with another command.

Preparation Q5

How do you compile a static library in C, and how do you link with it ?

When compiling C source code with a C compiler (like GCC, VC++, Turbo C) which uses some external declared functions, it is necessary to link the required static library to this program. We can achieve this by passing it to the compiler arguments. For this text we'll use the most popular C compiler, namely GCC.

Firstly we'll have to explain what a compiler is.

GCC stands for GNU Compiler Collection which includes compilers for the C, C++, Objective-C, Fortran, Ada, Go and D programming languages. In order to use this compiler, we can type in the command "gcc" inside a Linux shell.

This command will attempt to compile the C file "sourcefile.c" and output the binary into the "destinationfile" which is what a compiler does. It compiles source code into binary data which your computer can read. We can later execute this binary by using the ./ operator.

Preparation Q5

How do you compile a static library in C, and how do you link with it ?

- what is the difference between a static and a dynamic

Now that we understand how we can use the compiler. We'll talk about static libraries.

Static libraries are sets of routines, external functions and variables which are resolved in a caller at **compile-time**, these have file names like liblibrary.a. Essentially this causes the linker to copy all of the external variables and functions into your binary when compiling the sourcefile. This way, your program knows where and how to call certain external functions. We add these libraries as arguments when executing gcc with –l.

Cryptography

Two important cryptographic primitives

Hash function

- one-way function
- map data of arbitrary length to fixed-length string

- examples:

Authenticated Encryption with Associated Data

- hide the data from eavesdroppers
- protect plain-text data

- examples:

Two important cryptographic primitives

Hash function

- one-way function
- map data of arbitrary length to fixed-length string

- examples:
  - MD5
  - Whirlpool
  - SHA-3

Authenticated Encryption with Associated Data

- hide the data from eavesdroppers
- protect plain-text data

- examples:
  - ...

Algorithms in standards

A standard uses a specific (set of) **algorithm**(s). The algorithm within AES (the advanced encryption standard):

Rijndael

$$x^i \bmod (x^4 + 1) = x^{i \bmod 4}. \tag{4.10}$$

The modular product of $a(x)$ and $b(x)$, denoted by $a(x) \otimes b(x)$, is given by the four-term polynomial $d(x)$, defined as follows:

$$d(x) = d_3 x^3 + d_2 x^2 + d_1 x + d_0 \tag{4.11}$$

with

$$\begin{aligned} d_0 &= (a_0 \bullet b_0) \oplus (a_3 \bullet b_1) \oplus (a_2 \bullet b_2) \oplus (a_1 \bullet b_3) \\ d_1 &= (a_1 \bullet b_0) \oplus (a_0 \bullet b_1) \oplus (a_3 \bullet b_2) \oplus (a_2 \bullet b_3) \\ d_2 &= (a_2 \bullet b_0) \oplus (a_1 \bullet b_1) \oplus (a_0 \bullet b_2) \oplus (a_3 \bullet b_3) \\ d_3 &= (a_3 \bullet b_0) \oplus (a_2 \bullet b_1) \oplus (a_1 \bullet b_2) \oplus (a_0 \bullet b_3) \end{aligned} \tag{4.12}$$

When $a(x)$ is a fixed polynomial, the operation defined in equation (4.11) can be written in matrix form as:

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} \tag{4.13}$$

Because $x^4 + 1$ is not an irreducible polynomial over GF(2^8), multiplication by a fixed four-term polynomial is not necessarily invertible. However, the AES algorithm specifies a fixed four-term polynomial that *does* have an inverse (see Sec. 5.1.3 and Sec. 5.3.3):

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\} \tag{4.14}$$

$$a^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\} \tag{4.15}$$

Algorithms in standards

A standard uses a specific (set of) **algorithm**(s). The algorithm within AES (the advanced encryption standard):

Rijndael

$$x^i \bmod (x^4 + 1) = x^{i \bmod 4}. \tag{4.10}$$

The modular product of $a(x)$ and $b(x)$, denoted by $a(x) \otimes b(x)$, is given by the four-term polynomial $d(x)$, defined as follows:

$$d(x) = d_3 x^3 + d_2 x^2 + d_1 x + d_0 \tag{4.11}$$

with

$$\begin{aligned} d_0 &= (a_0 \bullet b_0) \oplus (a_3 \bullet b_1) \oplus (a_2 \bullet b_2) \oplus (a_1 \bullet b_3) \\ d_1 &= (a_1 \bullet b_0) \oplus (a_0 \bullet b_1) \oplus (a_3 \bullet b_2) \oplus (a_2 \bullet b_3) \\ d_2 &= (a_2 \bullet b_0) \oplus (a_1 \bullet b_1) \oplus (a_0 \bullet b_2) \oplus (a_3 \bullet b_3) \\ d_3 &= (a_3 \bullet b_0) \oplus (a_2 \bullet b_1) \oplus (a_1 \bullet b_2) \oplus (a_0 \bullet b_3) \end{aligned} \tag{4.12}$$

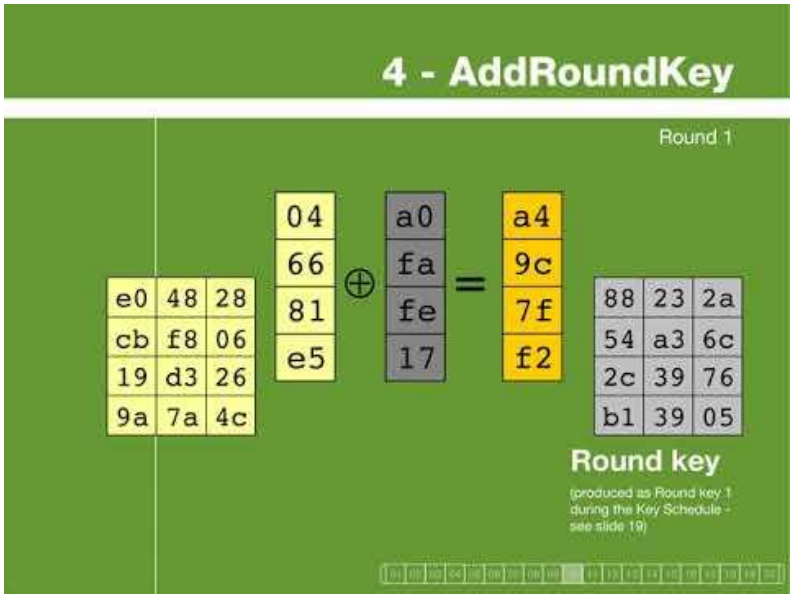
When $a(x)$ is a fixed polynomial, the operation defined in equation (4.11) can be written in matrix form as:

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} \tag{4.13}$$

Because $x^4 + 1$ is not an irreducible polynomial over GF(2^8), multiplication by a fixed four-term polynomial is not necessarily invertible. However, the AES algorithm specifies a fixed four-term polynomial that *does* have an inverse (see Sec. 5.1.3 and Sec. 5.3.3):

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\} \tag{4.14}$$

$$a^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\} \tag{4.15}$$



Who chooses these algorithms ?

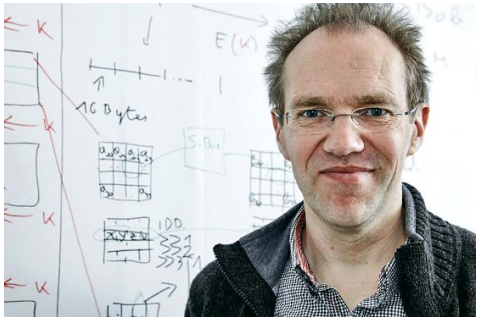
The National Institute of Standards and Technology

- American institute
- organises “competitions”

Who chooses these algorithms ?

Past competitions:

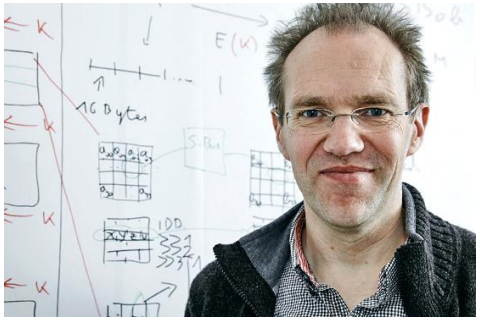
- AES competition
  - 1997 - 2000
  - winner: **Rijndael**
  - authors: Vincent Rijmen & Joan Daemen



Who chooses these algorithms ?

Past competitions:

- AES competition
- SHA-3 competition
  - 2007 - 2012
  - winner: **Keccak**
  - authors: Bertoni, Joan Daemen, Peeters, Van Assche



Who chooses these algorithms ?

Ongoing competitions:

- Post-Quantum Cryptography
- Lightweight Cryptography
  - growing number of constraint devices
  - more interconnecting devices
  - 2017 - ...

Lightweight Cryptography

Who chooses these algorithms ?

Lightweight Cryptography:

- **57** submissions
- **32** candidates to round 2
- October 19-21 2020: new workshop

- ACE
- ASCON
- COMET
- DryGASCON
- Elephant
- ESTATE
- ForkAE
- GIFT-COFB
- Gimli
- Grain-128AEAD
- HyENA
- ISAP
- KNOT
- LOTUS/LOCUS
- mixFeed
- ORANGE
- Oribatida
- PHOTON-beetle
- Pyjamask
- Romulus
- SAEAES
- Saturnin
- SKINNY
- SPARKLE
- SPIX
- SpoC
- Spook
- Subterranean 2.0
- SUNDAE-GIFT
- TinyJambu
- Wage
- Xoodyak

Who chooses these algorithms ?

Lightweight Cryptography:

Belgian (co-)authors

- **57** submissions
- **32** candidates to round 2
- October 19-21 2020: new workshop

- ACE
- ASCON
- COMET
- DryGASCON
- Elephant
- ESTATE
- ForkAE
- GIFT-COFB
- Gimli
- Grain-128AEAD
- HyENA
- ISAP
- KNOT
- LOTUS/LOCUS
- mixFeed
- ORANGE
- Oribatida
- PHOTON-beetle
- Pyjamask
- Romulus
- SAEAES
- Saturnin
- SKINNY
- SPARKLE
- SPIX
- SpoC
- Spook
- Subterranean 2.0
- SUNDAE-GIFT
- TinyJambu
- Wage
- Xoodyak

Who chooses these algorithms ?

Lightweight Cryptography:

- **57** submissions
- **32** candidates to round 2
- October 19-21 2020: new workshop

Belgian (co-)authors

Joan Daemen

- ACE
- ASCON
- COMET
- DryGASCON
- Elephant
- ESTATE
- ForkAE
- GIFT-COFB
- Gimli
- Grain-128AEAD
- HyENA
- ISAP
- KNOT
- LOTUS/LOCUS
- mixFeed
- ORANGE
- Oribatida
- PHOTON-beetle
- Pyjamask
- Romulus
- SAEAES
- Saturnin
- SKINNY
- SPARKLE
- SPIX
- SpoC
- Spook
- Subterranean 2.0
- SUNDAE-GIFT
- TinyJambu
- Wage
- Xoodyak

How to evaluate ?

Evaluation criteria

- cost:
  - area, energy, memory
- performance:
  - latency, throughput, power
- side channel resistance
- fault attack resistance

Requirements

- AEAD capabilities
- Hash capabilities
- SUPERCOP

How to evaluate ?

Requirements

- SUPERCOP

The file `encrypt.c` has the following structure:

```
#include "crypto_aead.h"

int crypto_aead_encrypt(
  unsigned char *c,unsigned long long *clen,
  const unsigned char *m,unsigned long long mlen,
  const unsigned char *ad,unsigned long long adlen,
  const unsigned char *nsec,
  const unsigned char *npub,
  const unsigned char *k
)
{
  ...
  ... the code for the cipher implementation goes here,
```

How to evaluate ?

Requirements

- SUPERCOP

The file `encrypt.c` has the following structure:

```
#include "crypto_aead.h"

int crypto_aead_encrypt(
  unsigned char *c,unsigned long long *clen,
  const unsigned char *m,unsigned long long mlen,
  const unsigned char *ad,unsigned long long adlen,
  const unsigned char *nsec,
  const unsigned char *npub,
  const unsigned char *k
)
{
  ...
  ... the code for the cipher implementation goes here,
```

Library exercise

Library exercise

Users have been made:

bmaurissen iwarnants mgiardina tgysen mgielkens

SSH to <machine>

Library exercise

demo_v1.c

```c
#include <stdio.h>

int sum(int x, int y) {
  return (int)(x+y);
}

int main(void) {
  int a, b, c;

  a = 3;
  b = 2;

  c = sum(a, b);

  printf("The sum of a + b = %d + %d = %d\n", a, b, c);

  return 0;
}
```

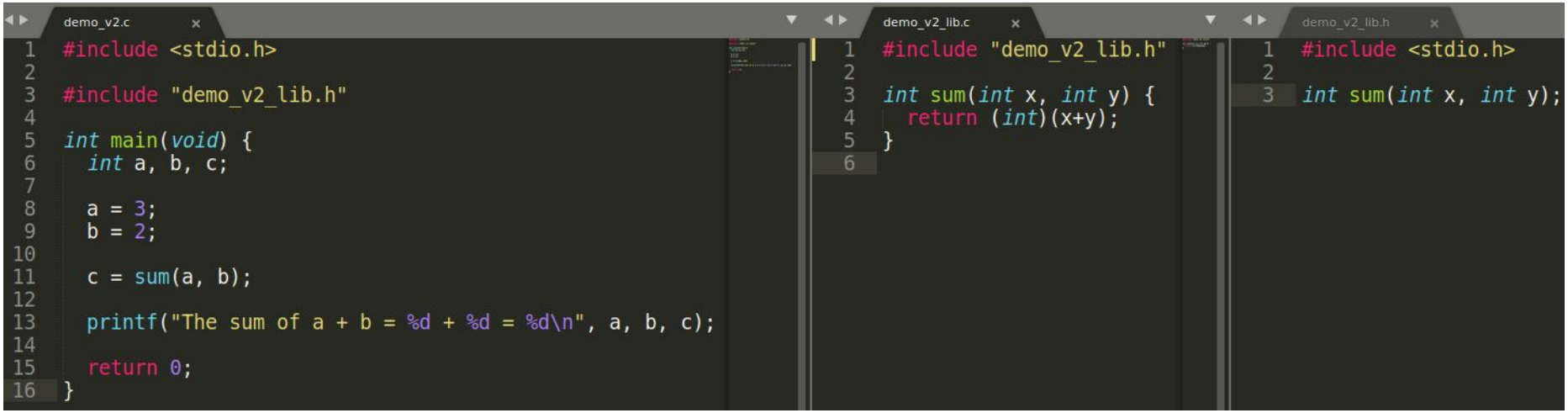
All code in a **single** file

gcc -c demo_v1.c

gcc -o demo_v1 demo_v1.o

C source
object file
binary
static library

Library exercise



demo_v2.c

```c
#include <stdio.h>

#include "demo_v2_lib.h"

int main(void) {
  int a, b, c;

  a = 3;
  b = 2;

  c = sum(a, b);

  printf("The sum of a + b = %d + %d = %d\n", a, b, c);

  return 0;
}
```

demo_v2_lib.c

```c
#include "demo_v2_lib.h"

int sum(int x, int y) {
  return (int)(x+y);
}
```

demo_v2_lib.h

```c
#include <stdio.h>

int sum(int x, int y);
```

All code in a **separate** files

gcc -c demo_v2_lib.c

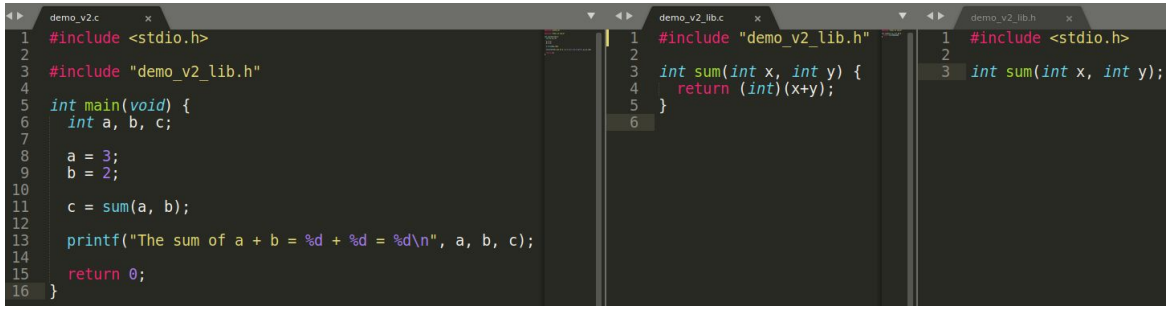
gcc -c demo_v2.c

gcc -o demo_v2 demo_v2.o demo_v2_lib.o

C source
object file
binary
static library

Library exercise

code is unaltered !!



demo_v2.c

```c
#include <stdio.h>

#include "demo_v2_lib.h"

int main(void) {
  int a, b, c;

  a = 3;
  b = 2;

  c = sum(a, b);

  printf("The sum of a + b = %d + %d = %d\n", a, b, c);

  return 0;
}
```

demo_v2_lib.c

```c
#include "demo_v2_lib.h"

int sum(int x, int y) {
  return (int)(x+y);
}
```

demo_v2_lib.h

```c
#include <stdio.h>

int sum(int x, int y);
```

All code in a **separate** files

gcc -c demo_v2_lib.c

ar -rcs libdemo_v2.a demo_v2_lib.o

gcc -c demo_v3.c

gcc -L. -o demo_v3 demo_v3.o -ldemo_v2

C source
object file
binary
static library

Mini Research Lab

Mini Research Lab

Mini Research Lab

1. login into <machine>
2. git clone https://github.com/jvliegen/mol_esands
   a. Download a candidate
   b. Compile candidate as static library
   c. Link in example file (test.c)
   d. while (! MOL == over)
      i. change parameters
      ii. run
      iii. evaluate
      iv. [change candidate]