Introduction to Quantum Algorithms and Code-Based Cryptography Implementation

Gustavo Banegas

Department of Mathematics and Computer Science
Technische Universiteit Eindhoven
gustavo@cryptme.in
https://www.cryptme.in

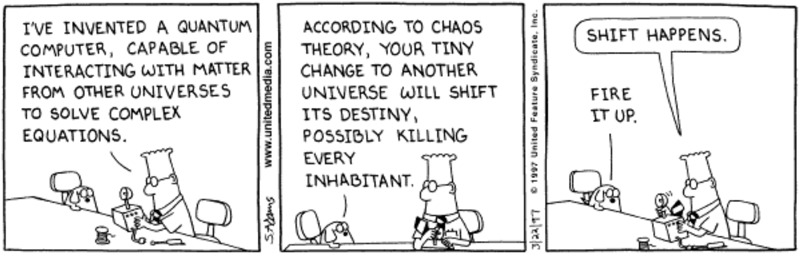
February 11, 2019

Outline

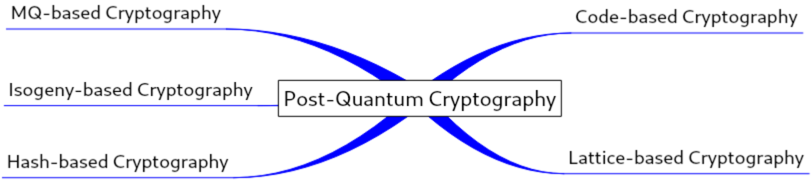
Why study quantum algorithms?

"Somebody announces that he's built a large quantum computer. RSA is dead. DSA is dead. Elliptic curves, hyperelliptic curves, class groups, whatever, dead, dead, dead."(Bernstein, 2005)

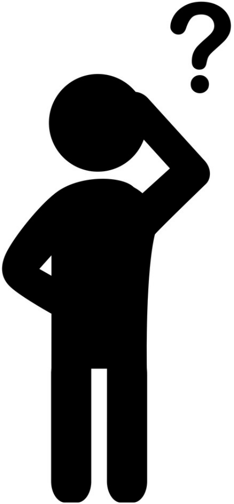
In other words..



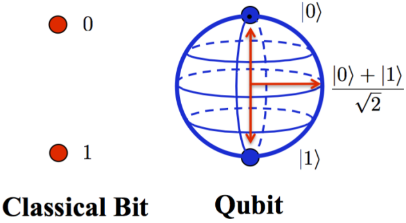
There is already an alternative



Ok! How can we use a quantum computer?



Classical bit vs Qubit



$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

$$\alpha |0\rangle + \beta |1\rangle ,$$

$$|\alpha|^2 + |\beta|^2 = 1$$

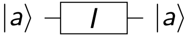
.

Measure quantum state

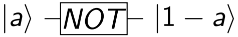
Measuring collapses the state.

Quantum gates

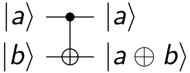
Identity gate:



NOT gate:

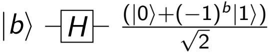


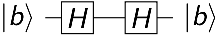
CNOT gate:



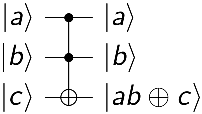
Hadamard Gate:

- $H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$





Toffoli gate:



n-Qubit system

Definition

$|\psi\rangle \in \mathbb{C}^2$ such that $|| \, |\psi\rangle \, || = 1$,

$$|\psi\rangle = \sum_{x \in \{0,1\}^n} \alpha_x |x\rangle$$

where

$$\sum_{x \in \{0,1\}^n} |\alpha_x|^2 = 1.$$

Example 2-qubit system

- 4 basis states: $|0\rangle \otimes |0\rangle$, $|0\rangle \otimes |1\rangle$,$|1\rangle \otimes |0\rangle$, $|1\rangle \otimes |1\rangle$.
- It is common to use just: $|0\rangle \, |1\rangle$,$|10\rangle$

Deutsch-Jozsa problem

- Input: $f : \{0,1\}^n \rightarrow \{0,1\}$ either constant or balanced
- Output: 0 iff f is constant
- Constrains: f is a black box

Query complexity

- Deterministic: $2^{n-1} + 1$

Deutsch-Jozsa problem

- Input: $f : \{0,1\}^n \to \{0,1\}$ either constant or balanced
- Output: 0 iff f is constant
- Constrains: f is a black box

Query complexity

- Deterministic: $2^{n-1} + 1$
- Quantum: 1

Deutsch-Jozsa quantum circuit

Simple quantum circuit:

$$|b\rangle - \boxed{S_f} - (-1)^{f(b)}|b\rangle$$

Deutsch-Jozsa quantum circuit

Simple quantum circuit:

$$|b\rangle \; -\boxed{\;S_f\;}- \; (-1)^{f(b)}\,|b\rangle$$

"Real" quantum circuit:

$$|b\rangle \; -\boxed{H}-\boxed{\;S_f\;}-\boxed{H}- \; ?$$

Deutsch-Jozsa quantum circuit analysis

$$|0\rangle \;-\!\boxed{H}\!-\!\boxed{\;S_f\;}\!-\!\boxed{H}\!-\; ?$$

- Initialization: $|0\rangle$.

Deutsch-Jozsa quantum circuit analysis

$$|0\rangle \; - \boxed{H} - \boxed{S_f} - \boxed{H} - \; ?$$

- Initialization: $|0\rangle$.
- Parallelization: $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$.

Deutsch-Jozsa quantum circuit analysis

$$|0\rangle \;-\!\boxed{H}\!-\!\!-\!\boxed{\;S_f\;}\!-\!\!-\!\boxed{H}\!-\; ?$$

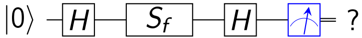
- Initialization: $|0\rangle$.
- Parallelization: $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$.
- Query: $\frac{1}{\sqrt{2}}((-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle)$.

Deutsch-Jozsa quantum circuit analysis

$$|0\rangle \; - \boxed{H} - \boxed{\;S_f\;} - \boxed{H} - \; ?$$

- Initialization: $|0\rangle$.
- Parallelization: $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$.
- Query: $\frac{1}{\sqrt{2}}((-1)^{f(0)}\,|0\rangle + (-1)^{f(1)}\,|1\rangle)$.
- Interferences: $\frac{1}{2}((-1)^{f(0)}(|0\rangle + |1\rangle) + (-1)^{f(1)}(|0\rangle - |1\rangle))$.

Deutsch-Jozsa quantum circuit analysis

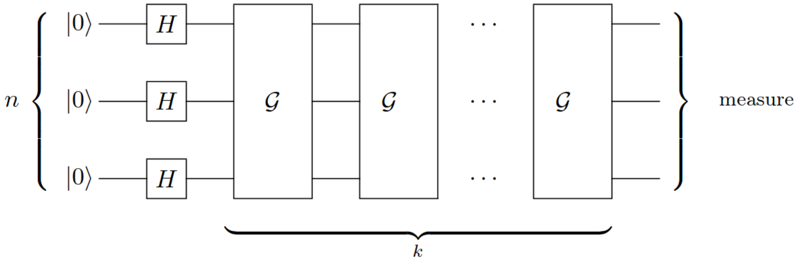


- Initialization: $|0\rangle$.
- Parallelization: $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$.
- Query: $\frac{1}{\sqrt{2}}((-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle)$.
- Interferences: $\frac{1}{2}((-1)^{f(0)}(|0\rangle + |1\rangle) + (-1)^{f(1)}(|0\rangle - |1\rangle))$.
- Final State:
  $\frac{1}{2}(((-1)^{f(0)} + (-1)^{f(1)})|0\rangle + ((-1)^{f(0)} - (-1)^{f(1)})|1\rangle)$.

It is easy to expand for *n*-qubits.

Grover's Algorithm

Grover's algorithm in a nutshell



- Originally described as search element in an unoreded database.
- Needs $O(\sqrt{N})$ queries in database of size $N = 2^n$ elements.

Preimage search

Security of a hash function

Given a hash-function H. The following three security properties should hold:

- Collision resistance: It is computationally infeasible to find any two distinct inputs x, x' which hash to the same output, i.e., such that $H(x) = H(x')$.
- Preimage resistance: It is computationally infeasible to find any preimage x' such that $H(x') = y$ when given any image y.
- 2nd preimage resistance: It is computationally infeasible to find any second input which has the same output as any specified input, i.e., given x, to find a 2nd-preimage $x' \neq x$ such that $H(x) = H(x')$.

Pre-quantum preimage search

Threat to AES

- van Oorschot–Wiener "parallel rho method".
  - Uses a mesh of p small processors.
  - Each running $2^{128}/pt$ fast steps, to find one of t independent AES keys $k_1, \ldots, k_t$, using a fixed plaintext, e.g, AES(0).

NIST has claimed that AES-128 is secure enough.

"Grover's algorithm requires a long-running serial computation, which is difficult to implement in practice. In a realistic attack, one has to run many smaller instances of the algorithm in parallel, which makes the quantum speedup less dramatic."

Introduction - Parallel rho method

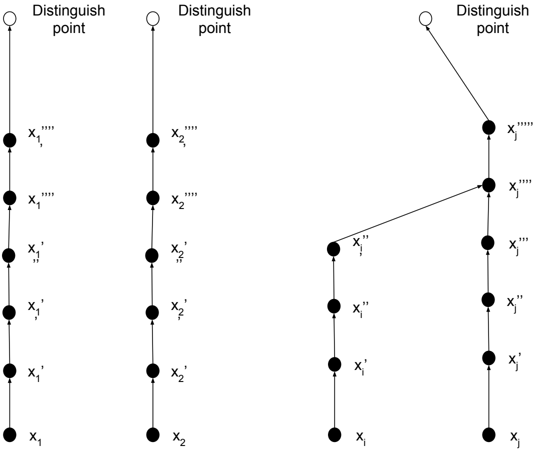
Distinguish Point

Consider $H : \{0,1\}^b \to \{0,1\}^b$
Take x an input of H, $x' = H(x)$.
Thereafter, take x' and apply H again, $x'' = H(x')$.
It is possible to do it n times (H^n), until a given condition is satified. In our case, we want the first $0 < d < b/2$ bits as 0.
$H^n_d(x)$ means d bits of x, computed n times.

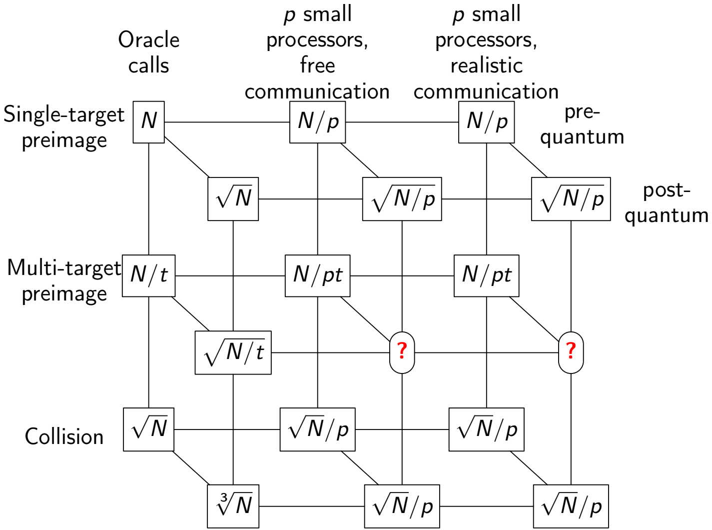
$$H^n_d(x) = \underbrace{0\ldots0}_{d \text{ zeros}}\{0,1\}^{b/2}$$

Introduction - Parallel rho method

Distinguish Point



Results in pre and post-quantum preimage search



Apply Grover's algorithm to find a preimage

Grover's algorithm to find a preimage

- Design AES as a quantum circuit.
- Design a quantum circuit for Grover's algorithm that uses the AES quantum circuit.
- Put the previous circuits in p processors using t keys.

- Quantum computer work in a way that requires all algorithms to be reversible.
  - Need to design AES circuit and Grover circuit as reversible circuits.

Apply Grover's algorithm to find a preimage

Grover's algorithm to find a preimage

- Design AES as a quantum circuit.
- Design a quantum circuit for Grover's algorithm that uses the AES quantum circuit.
- Put the previous circuits in p processors using t keys.

- Quantum computer work in a way that requires all algorithms to be reversible.
  - Need to design AES circuit and Grover circuit as reversible circuits.
- Want to have low memory.

Distinguish point in quantum setting

Trade-off from Bennett–Tompa

Example to compute $H^4(x)$:

Distinguish point in quantum setting

Trade-off from Bennett–Tompa

Example to compute $H^4(x)$:

time 0:	x	0	0	0	0

Distinguish point in quantum setting

Trade-off from Bennett–Tompa

Example to compute $H^4(x)$:

time 0:	x	0	0	0	0
time 1:	x	0	$H(x)$	0	0

Distinguish point in quantum setting

Trade-off from Bennett–Tompa

Example to compute $H^4(x)$:

time 0:	x	0	0	0	0
time 1:	x	0	$H(x)$	0	0
time 2:	x	0	$H(x)$	$H^2(x)$	0

Distinguish point in quantum setting

Trade-off from Bennett–Tompa

Example to compute $H^4(x)$:

time 0:	x	0	0	0	0
time 1:	x	0	$H(x)$	0	0
time 2:	x	0	$H(x)$	$H^2(x)$	0
time 3:	x	0	$H(x)$	$H^2(x)$	$H^3(x)$

Distinguish point in quantum setting

Trade-off from Bennett–Tompa

Example to compute $H^4(x)$:

time 0:	x	0	0	0	0
time 1:	x	0	$H(x)$	0	0
time 2:	x	0	$H(x)$	$H^2(x)$	0
time 3:	x	0	$H(x)$	$H^2(x)$	$H^3(x)$
time 4:	x	$H^4(x)$	$H(x)$	$H^2(x)$	$H^3(x)$

Distinguish point in quantum setting

Trade-off from Bennett–Tompa

Example to compute $H^4(x)$:

time 0:	x	0	0	0	0
time 1:	x	0	$H(x)$	0	0
time 2:	x	0	$H(x)$	$H^2(x)$	0
time 3:	x	0	$H(x)$	$H^2(x)$	$H^3(x)$
time 4:	x	$H^4(x)$	$H(x)$	$H^2(x)$	$H^3(x)$
time 5:	x	$H^4(x)$	$H(x)$	$H^2(x)$	0

Distinguish point in quantum setting

Trade-off from Bennett–Tompa

Example to compute $H^4(x)$:

time 0:	x	0	0	0	0
time 1:	x	0	$H(x)$	0	0
time 2:	x	0	$H(x)$	$H^2(x)$	0
time 3:	x	0	$H(x)$	$H^2(x)$	$H^3(x)$
time 4:	x	$H^4(x)$	$H(x)$	$H^2(x)$	$H^3(x)$
time 5:	x	$H^4(x)$	$H(x)$	$H^2(x)$	0
time 6:	x	$H^4(x)$	$H(x)$	0	0

Distinguish point in quantum setting

Trade-off from Bennett–Tompa

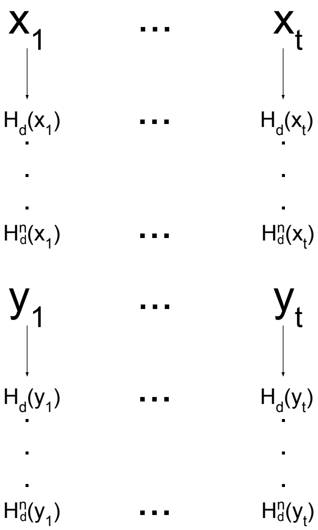
Example to compute $H^4(x)$:

time 0:	x	0	0	0	0
time 1:	x	0	$H(x)$	0	0
time 2:	x	0	$H(x)$	$H^2(x)$	0
time 3:	x	0	$H(x)$	$H^2(x)$	$H^3(x)$
time 4:	x	$H^4(x)$	$H(x)$	$H^2(x)$	$H^3(x)$
time 5:	x	$H^4(x)$	$H(x)$	$H^2(x)$	0
time 6:	x	$H^4(x)$	$H(x)$	0	0
time 7:	x	$H^4(x)$	0	0	0

$$
\begin{array}{ccc}
x_1 & \cdots & x_t \\
\downarrow & & \downarrow \\
H_d(x_1) & \cdots & H_d(x_t) \\
\vdots & & \vdots \\
H_d^n(x_1) & \cdots & H_d^n(x_t)
\end{array}
$$

$$\begin{array}{ccc} x_1 & \cdots & x_t \\ \downarrow & & \downarrow \\ H_d(x_1) & \cdots & H_d(x_t) \\ \vdots & & \vdots \\ H_d^n(x_1) & \cdots & H_d^n(x_t) \end{array}$$

$$\begin{array}{ccc} y_1 & \cdots & y_t \\ \downarrow & & \downarrow \\ H_d(y_1) & \cdots & H_d(y_t) \\ \vdots & & \vdots \\ H_d^n(y_1) & \cdots & H_d^n(y_t) \end{array}$$



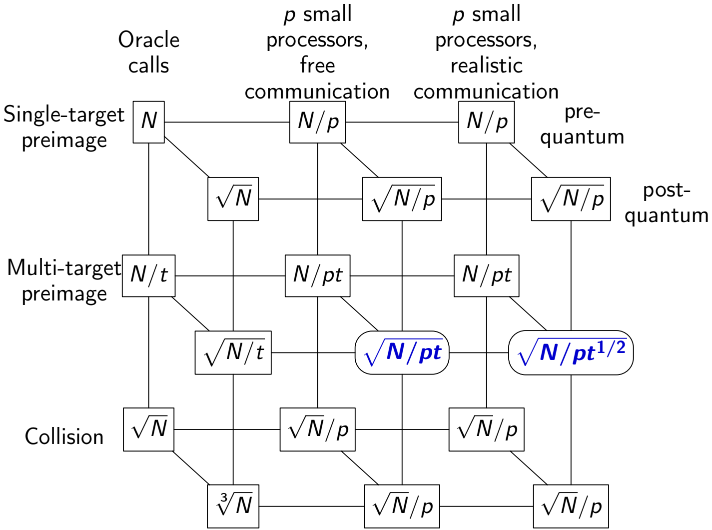
$$H_d^n(y_i) \stackrel{?}{=} H_d^n(x_j)$$

Low-communication parallel quantum multi-target preimage search

Gustavo Banegas & Daniel J. Bernstein

- Bennett-Tompa technique to build a reversible circuit for distinguished points.
- Possible to achieve using low communication costs and no memory.

Result:



Oracle calls
p small processors, free communication
p small processors, realistic communication
Single-target preimage
Multi-target preimage
Collision
pre-quantum
post-quantum
N
N/p
N/p
$\sqrt{N}$
$\sqrt{N/p}$
$\sqrt{N/p}$
N/t
N/pt
N/pt
$\sqrt{N/t}$
$\sqrt{N/pt}$
$\sqrt{N/pt^{1/2}}$
$\sqrt{N}$
$\sqrt{N}/p$
$\sqrt{N}/p$
$\sqrt[3]{N}$
$\sqrt{N}/p$
$\sqrt{N}/p$

Quantum Algorithms

- Deutsch–Jozsa's Algorithm;
- Grover's Algorithm (Search in unoreded database);
- Simon's Algorithm (QFT);
- Shor's Algorithm (Factoring numbers);
- Ambaini's Algorithm (Element disticness);
- Claw finding Algorithm;
- Kuperberg's Algorithm (dihedral hidden subgroup problem);

Post-quantum cryptography

A little bit of history in Post-quantum cryptography

- 2003: Small community of post-quantum researchers.
- 2014: PQCrypto conference reaches more than 100 people.
- 2015: NSA admits that the world needs post-quantum crypto.
- 2016: Other agencies also react (NCSC UK, NCSC NL, NSA).
- 2016: NIST calls for submissions to "Post-Quantum Cryptography Standardization Project".
- 2017: NIST receives 69 proper submissions.
- 2018: PQCrypto conference reaches more than 350 people.

Introduction to error correction

First a little bit of theory in error correction

- Enable data recovery after noisy transmission.
- In general, k bits of data get stored in n bits, adding redundancy.
- If no error occurred, these n bits satisfy $n - k$ parity check equations; else can correct some errors from the error pattern.
- Check equations can be represented by a matrix.
- Good codes can correct many errors without blowing up storage too much; offer guarantee to correct t errors (often can correct or at least detect more).

Introduction to Code-based cryptography

McEliece cryptosystem

- Use Goppa codes for public-key cryptography.
- Oldest (1978) code-based cryptosystem.
- Easily scale up for higher security.
- Big public key: at least $\approx 256KB$.

Alternative Codes

Other codes that can be used

- Quasi-cyclic codes (QC).
- Quasi-Dyadic Codes (Misoczki, Barreto '09).
- Generalized Srivastava (Persichetti '11).

Use subfield subcode construction to encrypt in the subcode and decrypt using parent code.

$\mathbb{F}_{q^m}$ - Decryption

$\mathbb{F}_q$ - Encryption

DAGS is Key Encapsulation using Dyadic GS Codes

DAGS is a joint project by:
Gustavo Banegas, Paulo S. L. M. Barreto, Brice Odilon Boidje, Pierre-Louis Cayrel, Gilbert Ndollane Dione, Kris Gaj, Cheikh Thiécoumba Gueye, Richard Haeussler, Jean Belo Klamti, Ousmane N'diaye, Duc Tri Nguyen, Edoardo Persichetti and Jefferson E. Ricardini
https://www.dags-project.org

DAGS: Key Encapsulation using Dyadic GS Codes

DAGS cryptosystem

- Use Generalized Srivastava codes.
- No decoding errors.
- Smaller keys.

DAGS: Key Encapsulation using Dyadic GS Codes

DAGS cryptosystem

- Use Generalized Srivastava codes.
- No decoding errors.
- Smaller keys.

DAGS Sizes (in bytes)

Parameter Set	Public Key	Private Key	Ciphertext
DAGS 1	8112	2496	656
DAGS 3	11264	4864	1248
DAGS 5	19712	6400	1632

About DAGS implementation

Key generation

- Operations in $\mathbb{F}_{2^{16}}$ and in $\mathbb{F}_{2^8}$.
  - Additions are "cheap".
  - Multiplications and inversions are costly.
    Originally with log **and i**-log **tables**.
  - Transformation from $\mathbb{F}_{2^{16}}$ to $\mathbb{F}_{2^8}$ and vice-versa.
- Random generation of a polynomial in $\mathbb{F}_{2^{16}}$.

About DAGS implementation

Key generation

- Operations in $\mathbb{F}_{2^{16}}$ and in $\mathbb{F}_{2^8}$.
  - Additions are "cheap".
  - Multiplications and inversions are costly.
    Originally with log **and i**-log **tables**.
  - Transformation from $\mathbb{F}_{2^{16}}$ to $\mathbb{F}_{2^8}$ and vice-versa.
- Random generation of a polynomial in $\mathbb{F}_{2^{16}}$.

Encapsulation

- Operations in $\mathbb{F}_{2^8}$.
- Random generation of a polynomial in $\mathbb{F}_{2^8}$.
- Hash function calls.

About DAGS implementation

Key generation

- Operations in $\mathbb{F}_{2^{16}}$ and in $\mathbb{F}_{2^8}$.
  - Additions are "cheap".
  - Multiplications and inversions are costly.
    Originally with log **and i**-log **tables**.
  - Transformation from $\mathbb{F}_{2^{16}}$ to $\mathbb{F}_{2^8}$ and vice-versa.
- Random generation of a polynomial in $\mathbb{F}_{2^{16}}$.

Encapsulation

- Operations in $\mathbb{F}_{2^8}$.
- Random generation of a polynomial in $\mathbb{F}_{2^8}$.
- Hash function calls.

Decapsulation

- Operations in $\mathbb{F}_{2^{16}}$ and in $\mathbb{F}_{2^8}$.
- Random generation of a polynomial in $\mathbb{F}_{2^8}$.
- Hash function calls.

Implementation details

How to represent elements in $\mathbb{F}_{2^8}$ or $\mathbb{F}_{2^16}$?

Elements in $\mathbb{F}_2$ are just: $\{0, 1\}$.

Implementation details

How to represent elements in $\mathbb{F}_{2^8}$ or $\mathbb{F}_{2^16}$?

Elements in $\mathbb{F}_2$ are just: $\{0, 1\}$.
Elements in $\mathbb{F}_{2^m}$ can be seen as a vector of m bits. For example: $\mathbb{F}_{2^8}$ is a vector of 8 bits, i.e., 1 byte. In a program language we can represent as an integer, element $x^3 + x + 1 = 11$.

Implementation details

How to implement operations in $\mathbb{F}_{2^8}$ or $\mathbb{F}_{2^16}$?

Multiplications can be trick since we want to avoid "side-channel" attacks such as timing attacks or cache attacks. We want avoid this:

Algorithm 1: Square and multiply in "RSA".

Data: C as integer, d as private exponent, n as length of d in bits and N
Result: $x = C^d \mod N$
$x \leftarrow C$;
for $j \leftarrow 1$ *to* N **do**
    $x \leftarrow mod(x^2, N)$;
    if $d_j == 1$ **then**
        $x \leftarrow mod(xC, N)$;
    end
    next j;
end
return x;

Implementation details

How to implement multiplication in $\mathbb{F}_{2^8}$ or $\mathbb{F}_{2^16}$ avoiding timing attacks?

Algorithm 2: Constant-time multiplication and reduction using $f(x) = x^8 + x^4 + x^3 + x^2 + 1$.

Data: a, b elements in $\mathbb{F}_{2^8}$

Result: $c = abmodx^8 + x^4 + x^3 + x^2 + 1$

$c \leftarrow 0$;

for $i \leftarrow 0$ *to* 7 **do**

  $c \leftarrow c \oplus (a * (b(1 << i)))$;

end

$c \leftarrow c \& 0xFFF$;

$c \leftarrow c \oplus (c >> 6)$;

$c \leftarrow c \oplus (c >> 5) \& 0x3E$;

$c \leftarrow c \& 0x3F$;

return c;

Implementation details

Other operations

Inversions can be implemented using exponentiation:

$$\alpha^{-1} = \alpha^{m-2} \in \mathbb{F}_{2^m}$$

Division can be implemented as:

$$\beta * \alpha^{-1} = \beta/\alpha$$

Implementation details

Techniques to avoid side-channel attacks

- Avoid branches (if, while), use masking ;
- Avoid big tables;
- Check for time variations.

Questions

Thank you for your attention.
Questions?
gustavo@cryptme.in