# Post-Quantum Cryptography:
# From construction to destruction

**Gustavo Banegas**

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

January 18, 2019

# Outline

Introduction

Code-based cryptography

DAGS: Key Encapsulation

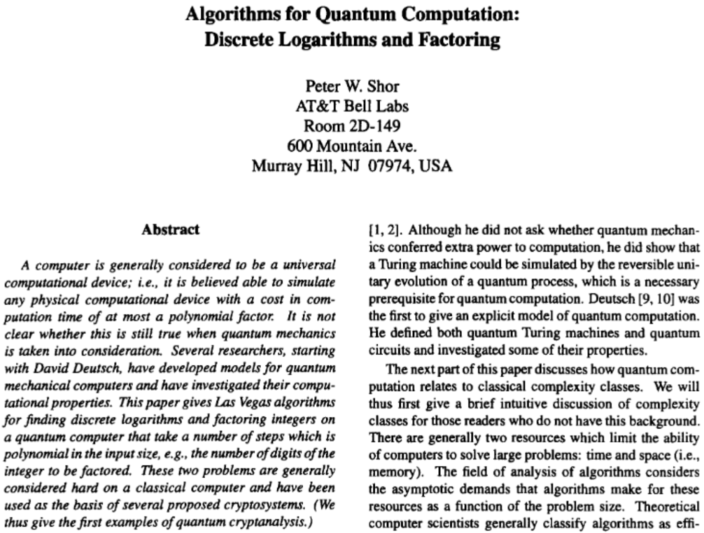
Quantum algorithms

# Deployed Cryptography

## Currently deployed cryptography

- Symmetric Cryptography:
  - AES - 128 bits and 256 bits.
  - ChaCha20 - 256 bits.
- Asymmetric Cryptography:
  - RSA - 1024 bits, 2048 bits and 4096 bits.
  - ECC - Curve25519 (256 bits), NIST Curves and others.

# Quantum Algorithms



## Algorithms for Quantum Computation: Discrete Logarithms and Factoring

Peter W. Shor
AT&T Bell Labs
Room 2D-149
600 Mountain Ave.
Murray Hill, NJ 07974, USA

### Abstract

*A computer is generally considered to be a universal computational device; i.e., it is believed able to simulate any physical computational device with a cost in computation time of at most a polynomial factor. It is not clear whether this is still true when quantum mechanics is taken into consideration. Several researchers, starting with David Deutsch, have developed models for quantum mechanical computers and have investigated their computational properties. This paper gives Las Vegas algorithms for finding discrete logarithms and factoring integers on a quantum computer that take a number of steps which is polynomial in the input size, e.g., the number of digits of the integer to be factored. These two problems are generally considered hard on a classical computer and have been used as the basis of several proposed cryptosystems. (We thus give the first examples of quantum cryptanalysis.)*

[1, 2]. Although he did not ask whether quantum mechanics conferred extra power to computation, he did show that a Turing machine could be simulated by the reversible unitary evolution of a quantum process, which is a necessary prerequisite for quantum computation. Deutsch [9, 10] was the first to give an explicit model of quantum computation. He defined both quantum Turing machines and quantum circuits and investigated some of their properties.

The next part of this paper discusses how quantum computation relates to classical complexity classes. We will thus first give a brief intuitive discussion of complexity classes for those readers who do not have this background. There are generally two resources which limit the ability of computers to solve large problems: time and space (i.e., memory). The field of analysis of algorithms considers the asymptotic demands that algorithms make for these resources as a function of the problem size. Theoretical computer scientists generally classify algorithms as effi-

# Deployed Cryptography

## Currently deployed cryptography after a quantum computer

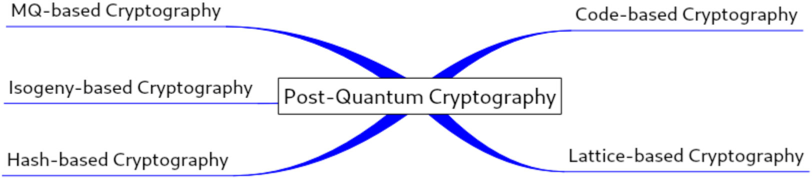
- Symmetric Cryptography:
  - AES - 128 bits and 256 bits.
  - Chacha20 - 256 bits.
- Asymmetric Cryptography:
  - ~~RSA - 1024 bits, 2048 bits and 4096 bits.~~ Dead
  - ~~ECC - Curve25519 (256 bits), NIST Curves and others.~~ Dead

# Post-quantum cryptography

## Another Motivation

- 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.

# What is Post-quantum Cryptography?


Post-Quantum Cryptography
MQ-based Cryptography
Isogeny-based Cryptography
Hash-based Cryptography
Code-based Cryptography
Lattice-based Cryptography

# 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 error correction

## Hamming Code

Parity check matrix ($n = 7$, $k = 4$):

$$H = \begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}$$

An error-free string of 7 bits $\mathbf{b} = (b_0, b_1, b_2, b_3, b_4, b_5, b_6)$ satisfies these three equations:

$$\begin{array}{ccccccccc} b_0 + & b_1 + & & b_3 + & b_4 & & & = 0 \\ b_0 + & & b_2 + & b_3 + & & b_5 & & = 0 \\ & b_1 + & b_2 + & b_3 + & & & b_6 & = 0 \end{array}$$

If one error occurred at least one of these equations will not hold.

# Introduction to error correction

## Hamming Code

For example, if we have as the result $1, 0, 1$:

$$\begin{array}{cccccccl}
b_0+ & b_1+ & & b_3+ & b_4 & & & = 1 \\
b_0+ & & b_2+ & b_3+ & & b_5 & & = 0 \\
 & b_1+ & b_2+ & b_3+ & & & b_6 & = 1
\end{array}$$

What does it mean?

# Introduction to error correction

## Hamming Code

For example, if we have as the result $1, 0, 1$:

$$\begin{array}{cccccccc} b_0+ & b_1+ & & b_3+ & b_4 & & & =1 \\ b_0+ & & b_2+ & b_3+ & & b_5 & & =0 \\ & b_1+ & b_2+ & b_3+ & & & b_6 & =1 \end{array}$$

What does it mean?
A: The bit $b_1$ flipped.

# Introduction to Code-based cryptography

## Generator Matrix

The parity check matrix $H$ is used for correcting errors. However, we need a generator matrix $G$ to encode and decode messages. A codeword is formed by multiplying a message $m = (m_1, \ldots, m_k)$ with $G$. After that, the codeword can be corrected using $H$. We have

$$GH^T = 0.$$

It is easy to compute $G$ from $H$.

# Introduction to Code-based cryptography

## Code-based cryptography

What happens if we put the errors on purpose?

- The error needs to be random.
- We need a way to recover the correct message.
- We need to ensure that it is hard for an attacker to correct the error without the proper decoder.

# Code-based cryptography 101

Key Generation:

- Choose $\omega$-error correcting code $\mathcal{C}$.
- $SK$: nice code description for $\mathcal{C}$.
- $PK$: pertubed code description given by generator matrix $G$.

Encryption:

- Message is a vector $m \in \mathbb{F}_{q^m}^k$.
- Select random error vector $e \in \mathbb{F}_{q^m}^n$ of weight $\omega$.
- $c = mG + e$.

Decryption:

- Map $c$ to equivalent vector $c'$ in nice code representation.
- Set $m =$Decode$(c)$ and return $m$.

# 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.

$$\begin{tikzcd}\mathbb{F}_{q^m} \text{ - Decryption} \arrow[d, no head] \\ \mathbb{F}_q \text{ - Encryption}\end{tikzcd}$$

# 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 |

# DAGS

Select hash functions $\mathbf{G}$,$\mathbf{H}$,$\mathbf{K}$

# DAGS

Select hash functions **G**,**H**,**K**

## Key Generation

- Generate a QD-GS code $C$.
- SK: description for $C$ (in alternant form) over $\mathbb{F}_{q^m}$.
- PK: generator matrix $G$ in systematic form $G = [I|G']$ for $C$ over $\mathbb{F}_q$.

# DAGS

Select hash functions **G**,**H**,**K**

## Key Generation

- Generate a QD-GS code $C$.
- SK: description for $C$ (in alternant form) over $\mathbb{F}_{q^m}$.
- PK: generator matrix $G$ in systematic form $G = [I|G']$ for $C$ over $\mathbb{F}_q$.

## Encapsulation

- Choose random word $m \in \mathbb{F}_q^k$.
- Compute $(\rho \parallel \sigma) = \mathbf{G}(m)$, $d = \mathbf{H}(m)$ and set $\mu = (\rho \parallel m)$.
- Generate $e \in \mathbb{F}_q^n$ of weight $\omega$ from seed $\sigma$.
- Output $(c, d)$ where $c = \mu G + e$.

# DAGS

Select hash functions **G**,**H**,**K**

## Key Generation

- Generate a QD-GS code $C$.
- SK: description for $C$ (in alternant form) over $\mathbb{F}_{q^m}$.
- PK: generator matrix $G$ in systematic form $G = [I|G']$ for $C$ over $\mathbb{F}_q$.

## Encapsulation

- Choose random word $m \in \mathbb{F}_q^k$.
- Compute $(\rho \parallel \sigma) = \mathbf{G}(m)$, $d = \mathbf{H}(m)$ and set $\mu = (\rho \parallel m)$.
- Generate $e \in \mathbb{F}_q^n$ of weight $\omega$ from seed $\sigma$.
- Output $(c, d)$ where $c = \mu G + e$.

## Decryption

- Set $(\mu', e') = Decode(c)$ and parse $\mu' = (\rho' \parallel m')$.
- Recompute $\mathbf{G}(m')$, $d = \mathbf{H}(m')$ and $e''$, then compare.
- Return $\perp$ if decoding fails or any check fails.
- Else return $K = \mathbf{K}(m')$.

# 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.

# Current DAGS

## Current status of DAGS[1]

- Awaiting result for the 2nd round from NIST.
- SimpleDAGS: Binary version of DAGS, operations in $\mathbb{F}_2$.
- Implementation for small devices (ARM Cortex-M4) and added AVX implementation[2].

[1] https://eprint.iacr.org/2018/1203

[2] https://git.dags-project.org/dags/dags

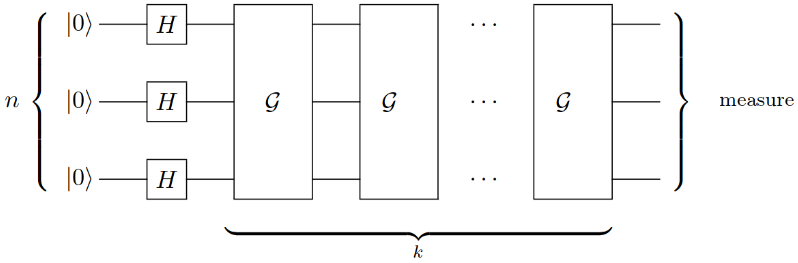
# Quantum algorithms

## Which quantum algorithms exist?

- Shor's algorithm: Integer factorization and discrete logs.
- Grover's algorithm: Originally described as search for element in unsorted database.
- Quantum walks: Element disticness, triangle finding and others applications.

# 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

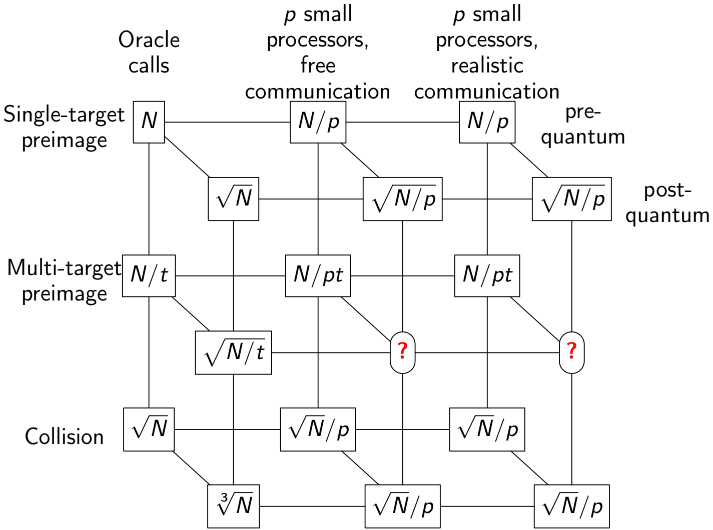
## Threat to AES

- 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.”

# Results in pre and post-quantum preimage search



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

# 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.

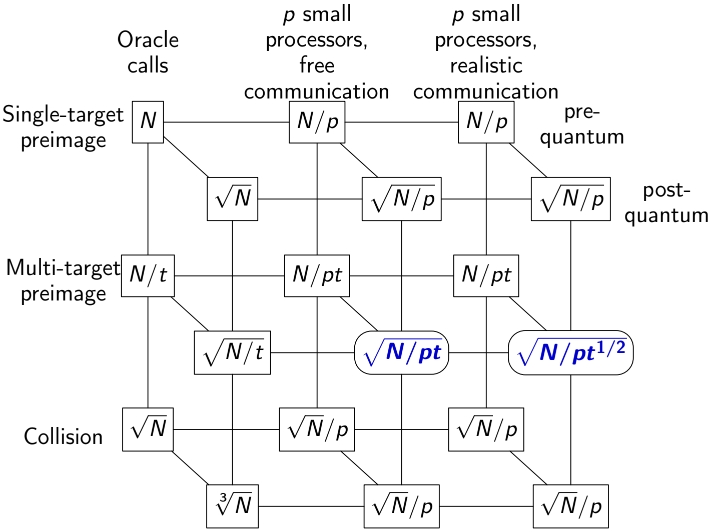
- Quantum computer work in a way that requires all algorithms to be reversible.
  - Need to design AES circuit and Grover circuit as reversible circuits.
- Want to have low memory.

# 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 | pre-quantum | $N$ | $N/p$ | $N/p$ |
| | post-quantum | $\sqrt{N}$ | $\sqrt{N/p}$ | $\sqrt{N/p}$ |
| Multi-target preimage | pre-quantum | $N/t$ | $N/pt$ | $N/pt$ |
| | post-quantum | $\sqrt{N/t}$ | $\boldsymbol{\sqrt{N/pt}}$ | $\boldsymbol{\sqrt{N/pt^{1/2}}}$ |
| Collision | pre-quantum | $\sqrt{N}$ | $\sqrt{N}/p$ | $\sqrt{N}/p$ |
| | post-quantum | $\sqrt[3]{N}$ | $\sqrt{N}/p$ | $\sqrt{N}/p$ |

# Questions

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