Escuela de Doctorado de la UCA

V Jornadas Doctorales

Programa de Doctorado en Matemáticas

Post-quantum Cryptography

S. Xambó

UPC/BSC

19/11/2019

Acknowledgements

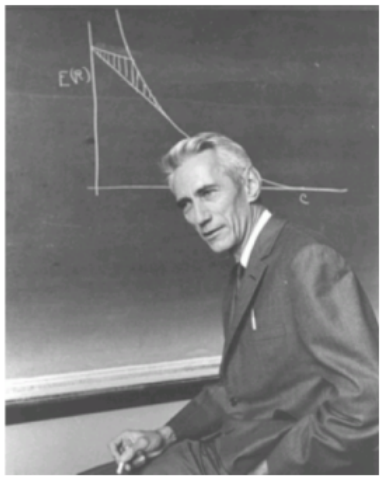
To Jaume **Puigbò**, for his wide aperture view on scientific and techological developments and his unfailing generosity in sharing his findings.

To Narcís **Sayols**, for his collaboration in computational endeavors and particularly in the development of the PyECC system.

The digital era

A Mathematical Theory of Communication (1948)

- Mathematical foundations of communication systems.
- Definition of information (*bit*), of channel capacity and of error-correcting codes.
- Source and channel coding theorems.



Claude Shannon (1916-2001)

- Master's thesis (1937): logic circuits
- II WWII: Cryptography

- **Founder of modern mathematical cryptography**

Euclides

Euler

Leibniz

Gauss

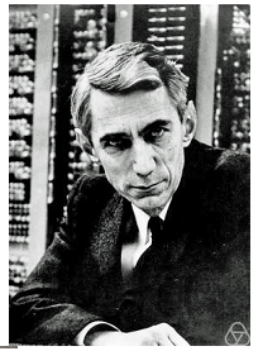
Galois

Ada Lovelace

Church

von Neumann

Turing



Shannon

Diffie-Hellman-Merkel

McEliece

Cocks

Rivest-Shamir-Adleman

Courtesi of

Euclid (325 BC – 265 BC).

Pierre de Fermat (1607-1665). Blaise Pascal (1623-1662). Gottfried Leibniz (1646-1716).

Leonhard Euler (1707-1783). Karl Friedrich Gauss (1777-1855). Charles Babbage (1791-1871).

Évariste Galois (1811-1832). Ada Lovelace (1815-1852). George Boole (1815-1864). Alonzo Church (1903-1995).

John von Neuman (1903-1957). Alan Turing (1912-1954). Claude Shannon (1916-2001). Elwyn Berlekamp (1940-2019).

Whitfield Diffie (1944). Ralph Merkel (1952). Martin Hellman (1945). Robet McElice (1942). Ronald Rivest (1947). Adi Shamir (1952). Leonard Adleman (1945). Cliford Cocks (1950).

- **Remarks on mathematical computation.**
- **RSA in a nutshell.**
- **The code-based McEliece cryptosystem.**
- **Quantum computing.**
- **Post-quantum cryptography.**
- **Urmila Mahadev, for outstanding achievements.**
- **Further references.**

Remarks on mathematical computation

The PyECC system · How can you get it · Examples

- A purely Python computational environment. Small, powerful, and free. Initially designed to cover the computational needs of a book such as [1] (*Error-Correcting Codes. A computational Primer*).
- It is being extended to meet other related purposes, such as, among others, the second edition of [1] (planned to have additional chapters on *post-quantum cryptography*, *quantum codes*, and *convolutional codes*) and [2] (A second edition of *Using intersection theory* and planned to cover a wide range of computations in intersection theory and enumerative geometry).

- For more information, including instructions for how to install it, visit the Web page PyECC.

Py Error-Correcting Coding (PyECC)

S. Xambó

Description

This is a venture in collaboration with *Narcís Sayols Baixeras*. Its aim is to produce a **Python** package (**PyECC**) enabling the construction, coding and decoding of error-correcting codes and make it freely available for students, teachers and researchers.

Initially (October 2015) the idea was to match the funcionality of the CC system developed to deal with the computational tasks related to the book **Xambó-2003**, but we soon found that we could go beyond that system in several directions. The purpose of this page (work in progress) is to document the current state of the project.

INDEX

- For online resources related to this talk, visit the Web page PyACA, which provides access to Python sources and Jupyter notebooks that are companion materials to the paper [3].

Index of links

- *Modular arithmetic* [**py** | **nb**]
- *Computation of $I_q(m)$* [**py** | **nb**]
- *Construction of rings and fields* [**py** | **nb**]
- *Vectors and matrices* [**py** | **nb**]
- *Random pemutation matrix* [**py** | **nb**]
- *rd_GL(k,F)* [**py** | **nb**]
- *The Hamming code [7,4,3]* [**py** | **nb**]
- *Coding and decoding the Hamming code [7,4,3]* [**py** | **nb**]
- *Example of blow and prune* [**py** | **nb**]
- *Example of a Goppa code* [**py** | **nb**]
- *An illustration of the McEliece system, step by step* [**py** | **nb**]

```
def NOT(j, x):
    x[j] -= 1
    return x

# Controlled-not. For j=i, x[i] is set to 0.
def CNOT(i,j, x):
    x[j] += x[i]
    return x

# Remark: for i=j it agrees with CNOT(j,k, x)
def TOFFOLI(i,j,k, x):
    x[k] += x[i]*x[j]
    return x
```

Fact: Any classical computation can be embedded in a reversible computation (see [4]) and any reversible computation can be achieved with a sequence of Toffoli gates.

```
# To pick a random n-digit prime number
def rd_prime(n):
    while True:
        r = ri(n) # a random integer of n digits
        if even(r): r += 1
        if is_prime(r): return r
```

```
rd_prime(10) => 31716 43721
rd_prime(100) =>
75549 06889 06549 88619 83249 61562 84269 54436 19389 71081
98583 48723 01531 63041 11630 94137 11913 03811 57707 35373
```

The density of n-digit prime numbers is $\sim 0.48/(n+1)$, which roughly means an average of $2n$ trials to produce a prime number.

```
# x, the message, is a binary vector
# p is a random binary vector of the same length as x.
# The encrypted message is y = x+p
# Decryption is y + p = x + p + p = x

def one_time_pad(x):
    n = len(x)
    p = rd_vector(Zn(2),n)
    return (x+p,p)

x = rd_vector(Zn(2),25)
one_time_pad(x) =>
([0,0,0,0,0,1,0,0,1,0,1,1,0,0,1,0,0,1,0,0,0,1,1,1,0],
 [1,1,1,0,1,0,1,1,1,1,1,1,0,1,1,0,1,1,1,1,0,0,0,0,0])
```

```
# Pseudo-random generator of n bits
# (Blum-Blum-Shup, 1996)
def PRG(n,d=30):
    N = rd_prime(d)*rd_prime(d)
    x = rd_int(1,N-1)
    s = str(x%2)
    for _ in range(1,n):
        x = x**2 % N
        s += str(x%2)
    return s

PRG(40) => '1000101110101010111000111011111010011101'
```

The function permutation_matrix(n) creates a random permutation matrix of order n.

```
def rd_permutation_matrix(n):
    N = list(range(n))
    p = rd_permutation(n)
    P = matrix(ZZ(),n,n)
    for j in range(n):
        P[j,p[j]] = 1
    return P

rd_permutation_matrix(5) =>
[[1  0  0  0  0]
 [0  0  0  0  1]
 [0  1  0  0  0]
 [0  0  0  1  0]
 [0  0  1  0  0]]
```

The construction of uniformly random invertible matrix S of order k with entries in a finite field F is a bit involved (see [3], particularly the function rd_GL(k,F)). Instead, let us consider an easier function scramble_matrix(k,A) which creates a random $S \in A(k)$ with $\det(S) = \pm 1$. Note: rd(A) returns an element of A selected uniformly at random.

```
def scramble_matrix(k,A):
    U = matrix(A,k,k)
    L = matrix(A,k,k)
    for i in range(k):
        U[i,i] = L[i,i] = 1
        for j in range(i+1,k):
            U[i,j] = rd(A)
            L[j,i] = rd(A)
    P = permutation_matrix(k)
    return P*L*U
```

RSA in a nutshell

RSA pairs of prime numbers · RSA keys · Encryption and decryption · RSA numbers

```
def rsa_pair(n,m=3):
    while True:
        p = rd_prime(n); q = rd_prime(n)
        if igcd(m,p-1)>1 or igcd(m,q-1)>1 or p==q:
            continue
        else: break
    return (p,q)
```

```
rsa_pair(20) =>
(215695130859086603 39, 155261035906218761 57)
```

```
rsa_pair(20,17) =>
(339964144178897188 49, 566107449004098854 59)
```

Fix a positive integer n (in practice today, a three digit number, say 120).

(p, q) an RSA pair of n-digit prime numbers for some m (say $m = 3$ for concreteness).

k the inverse of m mod $(p-1)(q-1)$

Secret keys: (p, q, m, k).

Public keys: (N, m), where $N = pq$.

```
(p,q) = rsa_pair(10) => (2952363761, 6768633203)
k = inverse(3,(p-1)*(q-1)) => 13322311580211706347
N = p*q = 19983467380038556483
```

x: a number in $(1, N-1)$, the *message*.

Encryption: $y = x^m \mod N$.

Decryption: $z = y^k \mod N$.

Fact: $z = x$.

Proof: $y^k = (x^m)^k = x^{mk} = x^{1+r(p-1)(q-1)} = x$, as $(p-1)(q-1) = \varphi(N)$ and hence $x^{(p-1)(q-1)} = 1$ (Euler).

Security

- *RSA is secure if factoring N is unfeasible*.
- The best known algorithms factor N (a $2n$-digit number) in subexponential expected time $e^{(1+o(1))r(n)}$, $r(n) = \sqrt{2n\log(2n)}$.

n	100	200	300	400	500	1000
$\lfloor \log_2 e^{r(n)} \rfloor$	46	70	89	105	119	177

RSA-100	RSA-170	RSA-230	RSA-290 *	RSA-360 *	RSA-450 *
RSA-110	174 RSA-576	RSA-232 *	RSA-300 *	RSA-370 *	RSA-460 *
RSA-120	RSA-180	232 RSA-768	RSA-309 *	RSA-380 *	463 RSA-1536 *
RSA-129	RSA-190	RSA-240 *	309 RSA-1024 *	RSA-390 *	RSA-470 *
RSA-130	193 RSA-640	RSA-250 *	RSA-310 *	RSA-400 *	RSA-480 *
RSA-140	RSA-200	RSA-260 *	RSA-320 *	RSA-410 *	RSA-490 *
RSA-150	RSA-210	RSA-270 *	RSA-330 *	RSA-420 *	RSA-500 *
RSA-155	212 RSA-704	270 RSA-896 *	RSA-340 *	RSA-430 *	RSA-617 *
RSA-160	RSA-220	RSA-280 *	RSA-350 *	RSA-440 *	617 RSA-2048 *

RSA Factoring Challenge: 1991-2007. Each number is the products of two primes of similar size. Those underlined in red are quoted in bits, with the number of decimal digits written in grey above them. The numbers with * have not been factored yet. Source: https://en.wikipedia.org/wiki/RSA_numbers.

RSA-768[232] [edit]

RSA-768 has 232 decimal digits (768 bits), and was factored on December 12, 2009 over the span of two years, by Thorsten Kleinjung, Kazumaro Aoki, Jens Franke, Arjen K. Lenstra, Emmanuel Thomé, Pierrick Gaudry, Alexander Kruppa, Peter Montgomery, Joppe W. Bos, Dag Arne Osvik, Herman te Riele, Andrey Timofeev, and Paul Zimmermann.[36]

```
RSA-768 = 123018668453011775513049495838496272077285356959533479219732245215172640050726365751874520219978646938995647494277406384592519255732630345373154826850791702612214291346167042921431160222124047927473779408066535141959745985690214341 3
```

```
RSA-768 = 33478071698956898786044169848212690817704794983713768568912431388982883793878002287614711652531743087737814467999489
        × 36746043666799590428244633799627952632279158164343087642676032283815739666511279233373417143396810270092798736308917
```

The CPU time spent on finding these factors by a collection of parallel computers amounted approximately to the equivalent of almost 2000 years of computing on a single-core 2.2 GHz AMD Opteron-based computer.[36]

RSA-240 [edit]

RSA-240 has 240 decimal digits (795 bits), and has not been factored so far.

```
RSA-240 =
124620366781718784065835044608106590434820374651678805754818788883289666801188210855036039570272508747509864768438458621054865537970253930571891217684318286362846948405301614416430468066875699415246993185704183030512549594371372159029236099
```

RSA-250 [edit]

RSA-250 has 250 decimal digits (829 bits), and has not been factored so far.

```
RSA-250 = 21403246502407449612644230728393335630086147151447550177977549208814180234471401366433455190958046796109928518724709145876873962619215573630474547705208051190564931066876915900197594056934574522305893259766974716817380693648946998715784949759374979 37
```

McEliece cryptosystems

Ingredients of a McECS · Encryption · Decryption · Construction of McECS · Security analysis

- $F = F_q$, a finite field of cardinal q (*base field*). The most important case will be $F = \mathbf{Z}_2$.
- k a positive integer. The vectors of F^k are called *information vectors*, or *messages*.
- $n > k$ an integer. The vectors of F^n are called *transmission vectors*.

Notations

If $\mathbf{x} \in F^n$, we let $|\mathbf{x}|$ denote the number of non-zero components of $\mathbf{x}$ and we say that it is the *weight* of $\mathbf{x}$.

$F(r, s)$ denotes the space of matrices of type $r \times s$ with entries in F and $F(r) = F(r, r)$.

A receiving user needs the following data:

- $G \in F(k, n)$ such that $\text{rank}(G) = k$;
- $S \in F(k)$ invertible and chosen uniformly at random;
- $P \in F(n)$ a random permutation matrix;
- t, a positive integer; and
- $g : X \to F^k$, $X \subseteq F^n$, such that for any $\boldsymbol{u} \in F^k$ and all $\boldsymbol{e} \in F^n$ with $|\boldsymbol{e}| \leqslant t$,

$$\mathbf{x} = \boldsymbol{u}G + \boldsymbol{e} \in X \quad \text{and} \quad g(\mathbf{x}) = \boldsymbol{u}. \tag{1}$$

The map g is called an *t-error-correcting G-decoder*, or simply *decoder*, and the vectors of X are said to be *g-decodable*.

- **Private key**: $\{G, S, P\}$.
- **Public key**: $\{G', t\}$, where $G' = SGP$.

Encryption protocol

The protocol that a user has to follow to encript and send a message $\boldsymbol{u}$ to the user whose public key is $\{G', t\}$ consists of two steps:

- Random generation of a transmission vector $\boldsymbol{e}$ of weight t;
- Sending the vector $\mathbf{x} = \boldsymbol{u}G' + \boldsymbol{e} = \boldsymbol{u}SGP + \boldsymbol{e}$ to that user.

Decryption protocol

Consists of four steps that only use private data of the receiver and the vector $\mathbf{x}$ sent by the emitter:

- Set $\mathbf{y} = \mathbf{x}P^{-1}$, so that $\mathbf{y} = (\boldsymbol{u}S)G + \boldsymbol{e}P^{-1}$.
- Set $\mathbf{x}' = g(\mathbf{y})$. Since P is a permutation matrix, $|\boldsymbol{e}P^{-1}| = |\boldsymbol{e}| = t$, and hence $\mathbf{x}'$ is well defined, as g corrects t errors. The result is $\mathbf{x}' = (\boldsymbol{u}S)G$, which says that $\mathbf{x}'$ is the linear combination of the rows of G with coefficients $\boldsymbol{u}' = \boldsymbol{u}S$.
- Since G has rank k, $\boldsymbol{u}'$ is uniquely determined by $\mathbf{x}'$ and can be obtained by solving the system of linear equations $\mathbf{x}' = \boldsymbol{u}'G$, where $\boldsymbol{u}'$ is the unknown vector.
- Let $\boldsymbol{u} = \boldsymbol{u}'S^{-1}$, which agrees with the message sent by the emitter.

- $F = F_2$ (most constructions also for F_q, $q > 2$).
- $\bar{F} = F_{q^m}$, m a positive integer. If $\beta \in \bar{F}$, $[\beta]$ will denote the colum vector of its components with respect to a basis of $\bar{F}/F$.
- $\boldsymbol{\alpha} = \alpha_1, \ldots, \alpha_n \in \bar{F}$ distinct elements, so that $n \leqslant q^m$.
- $p \in \bar{F}[X]$ a polynomial of degree $r > 0$ such that $p(\alpha_j) \neq 0$ $(j = 1, \ldots, n)$.
- Set $h_j = 1/p(\alpha_j)$ $(j = 1, \ldots, n)$ and

$$\bar{H} = \begin{pmatrix} h_1 & \cdots & h_n \\ h_1\alpha_1 & \cdots & h_n\alpha_n \\ \vdots & & \vdots \\ h_1\alpha_1^{r-1} & \cdots & h_n\alpha_n^{r-1} \end{pmatrix} \in \bar{F}(r, n).$$

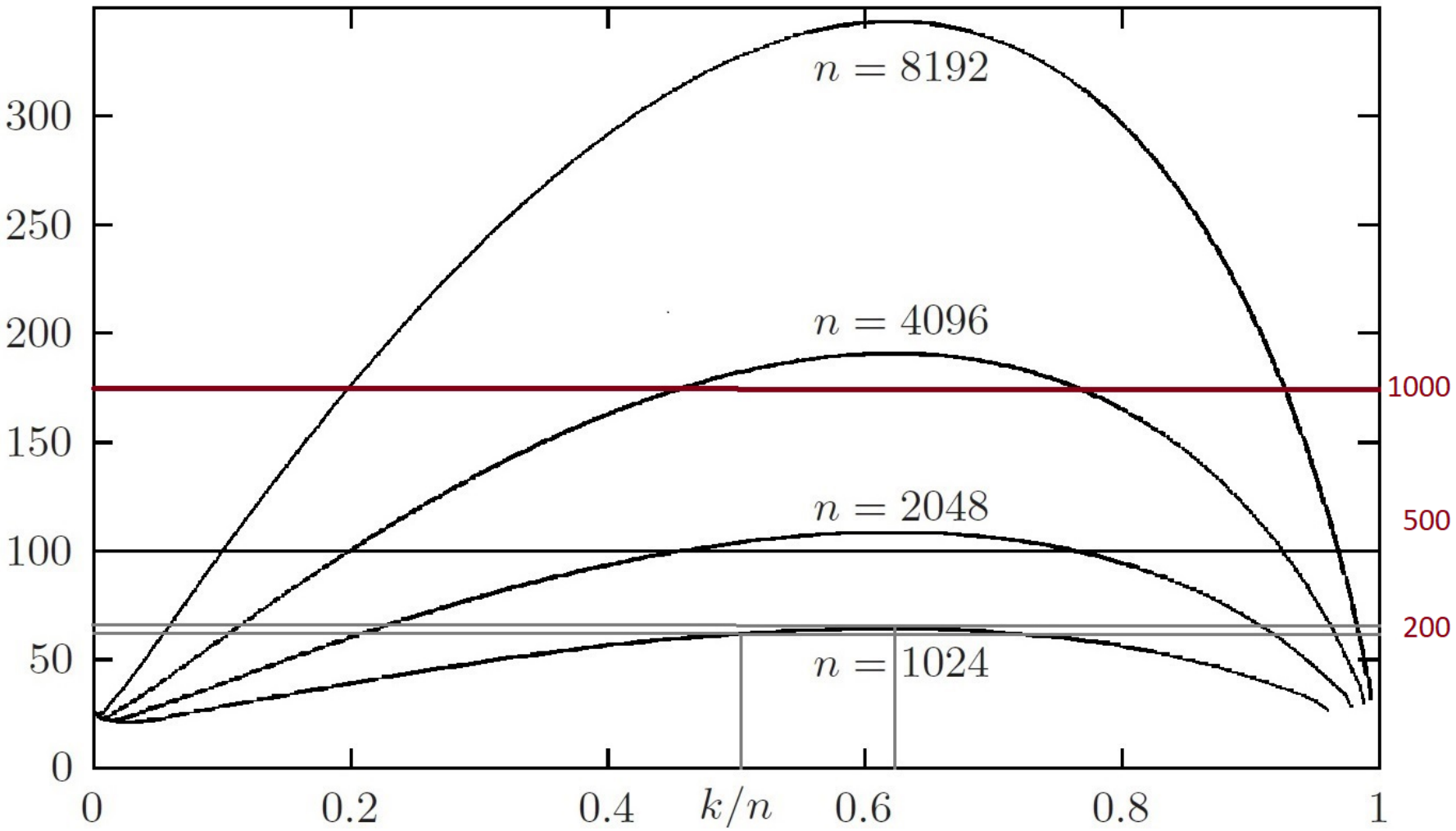
- Let $H \in F(r', n)$ be the result of replacing each entry β of $\bar{H}$ by $[\beta]$ (this yields a matrix $[H] \in F(mr, n)$), followed by deleting from $[H]$ any row that is in the span of the previous ones. Note that $r' \leqslant mr$. It also holds that $r \leqslant r'$, as the $\langle H \rangle_{\bar{F}} = \langle \bar{H} \rangle_{\bar{F}}$.
- Let $\Gamma = \Gamma(p, \alpha) = \{\mathbf{x} \in F^n : \mathbf{x}H^T = 0\}$. It is a **code** of type $[n, k = n - r']$. This code is called the *classical Goppa code* associated to p and $\boldsymbol{\alpha}$.
- We have $n - mr \leqslant k \leqslant n - r$.
- **Fact**: If $G \in F(k, n)$ is a generating matrix of Γ, *there is G-decoder that corrects r errors* ($r/2$ for $q > 2$) *provided p has no multiple roots in $\bar{F}$*. See, for example, [1, P.4.7]

Practical specification

- Let $\boldsymbol{\alpha}$ be the set of elements of $\bar{F}$. Hence $n = 2^m$.
- Let $p \in \bar{F}[X]$ be a monic irreducible of degree $t > 1$. Then p has no roots in $\bar{F}$ and so a generating matrix G of $\Gamma(p, \boldsymbol{\alpha})$ *has a decoder g that corrects t errors*.

This ends the theoretical construction of a McECS with the following parameters:

- $n = 2^m$, where m is any posivive integer, and p is monic irreducible of degree t.
- $\bar{H} \in \bar{F}(t, n)$ and $G \in F(k, n)$, where $k = n - \text{rank}(H)$ $(n - tm \leqslant k \leqslant n - t)$.
- Original example: $m = 10$, $n = 1024$, $t = 50$, $k = 524$ (in this case $k = n - tm$, the minimum possible given m and t).



Horizontal axis $R = k/n$, WF curves for $n = 2^j$, $j = 10, \ldots, 13$. The red line represents the WF needed to break RSA with 1000-digit prime numbers. The similar 500-digit and 200-digit levels are also shown. The latter is comparable to the original McECS.

Planck

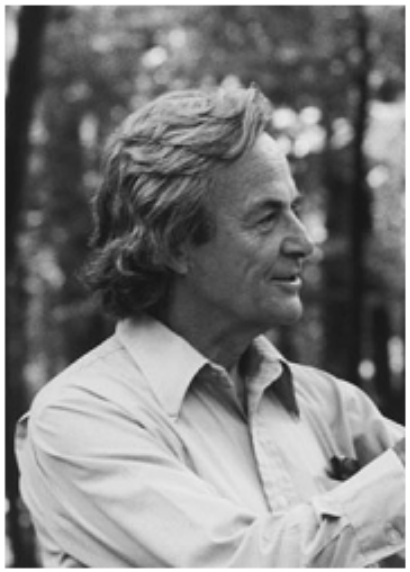
Einstein

Schrödinger

de Broglie

Dirac

Pauli



Feynman

Shor

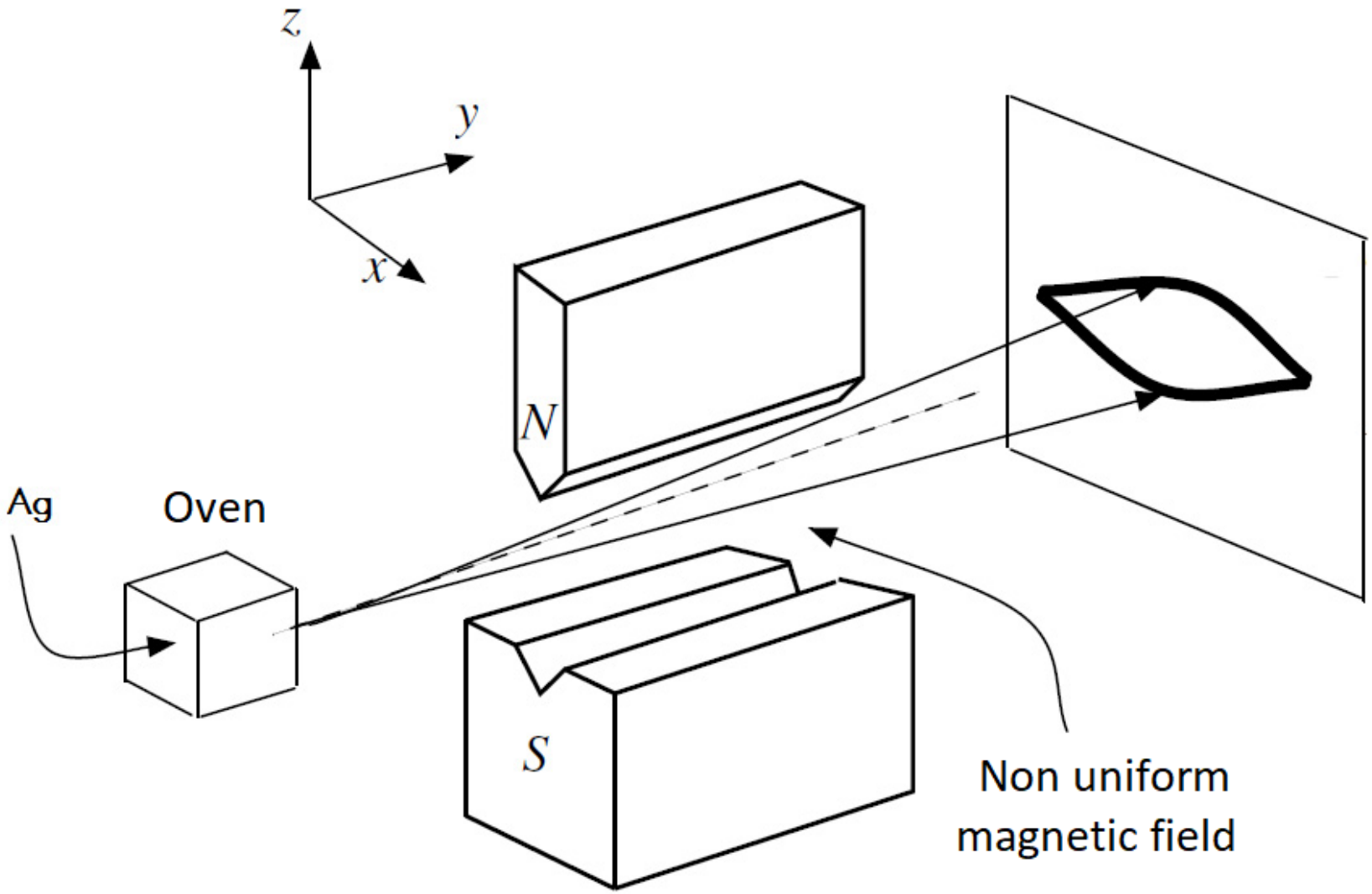
Max Planck (1858-1947), Albert Einstein (1879-1955), Erwing Schrödinger (1887-1966), Louis de Broglie (1892-1987).

Paul Dirac (1902-1984), Wolfgang Pauli (1909-1958).

Richard Feynman (1918-1988), Peter Shor (1959).

Quantum computing

q-bits · q-registers · q-computations · q-gates · q-algorithms · Shor's factoring q-algorithm · Grover's searching q-algorithm · The power of q-computing.



Stern-Gerlach experiment, which uncovered the quantum nature of the electron *spin*.

- To move from classical to quantum computation, replace $B = \{0, 1\}$, the set of classical bits, by all their 'superpositions' (linear combinations with complex coefficients), i.e. by the complex space $E = E^{(1)}$ generated by B.

(Superpositions of waves occur in classical physics, particularly in many wave phenomena, and in the related notion of polarization statates of electromagnetic waves. Its general validity in the context of quantum physics is one of the main tenets of this theory)

Thus the elements of E have the form $\psi = \psi_0 e_0 + \psi_1 e_1$, where e_0 and e_1 is the basis corresponding to 0 and 1 and $\psi_0, \psi_1 \in \mathbf{C}$.

The vectors ψ of norm 1 form a sphere $S^3 \subset E^{(1)} \simeq \mathbf{R}^4$ and are called *Pauli spinors* ($|\psi|^2 = |\psi_0|^2 + |\psi_1|^2$).

- The q-bit *state* corresponding to a spinor ψ is the point (in *Dirac's ket notation*) $|\psi\rangle \in S^2 \subset \mathbf{R}^3$ defined by the formulas

$$x = (\bar{\psi}_0\psi_1 + \psi_0\bar{\psi}_1), y = -i(\bar{\psi}_0\psi_1 - \psi_0\bar{\psi}_1), z = (\bar{\psi}_0\psi_0 - \bar{\psi}_1\psi_1).$$

The map $S^3 \to S^2$, $\psi \mapsto |\psi\rangle$ is the *Hopf fibration*.

In fact, $|\psi\rangle = |\psi'\rangle$ if and only if $\psi' = \xi\psi$ for some unit complex number ξ, a relation that we will denote $\psi \sim \psi'$.

- If we set

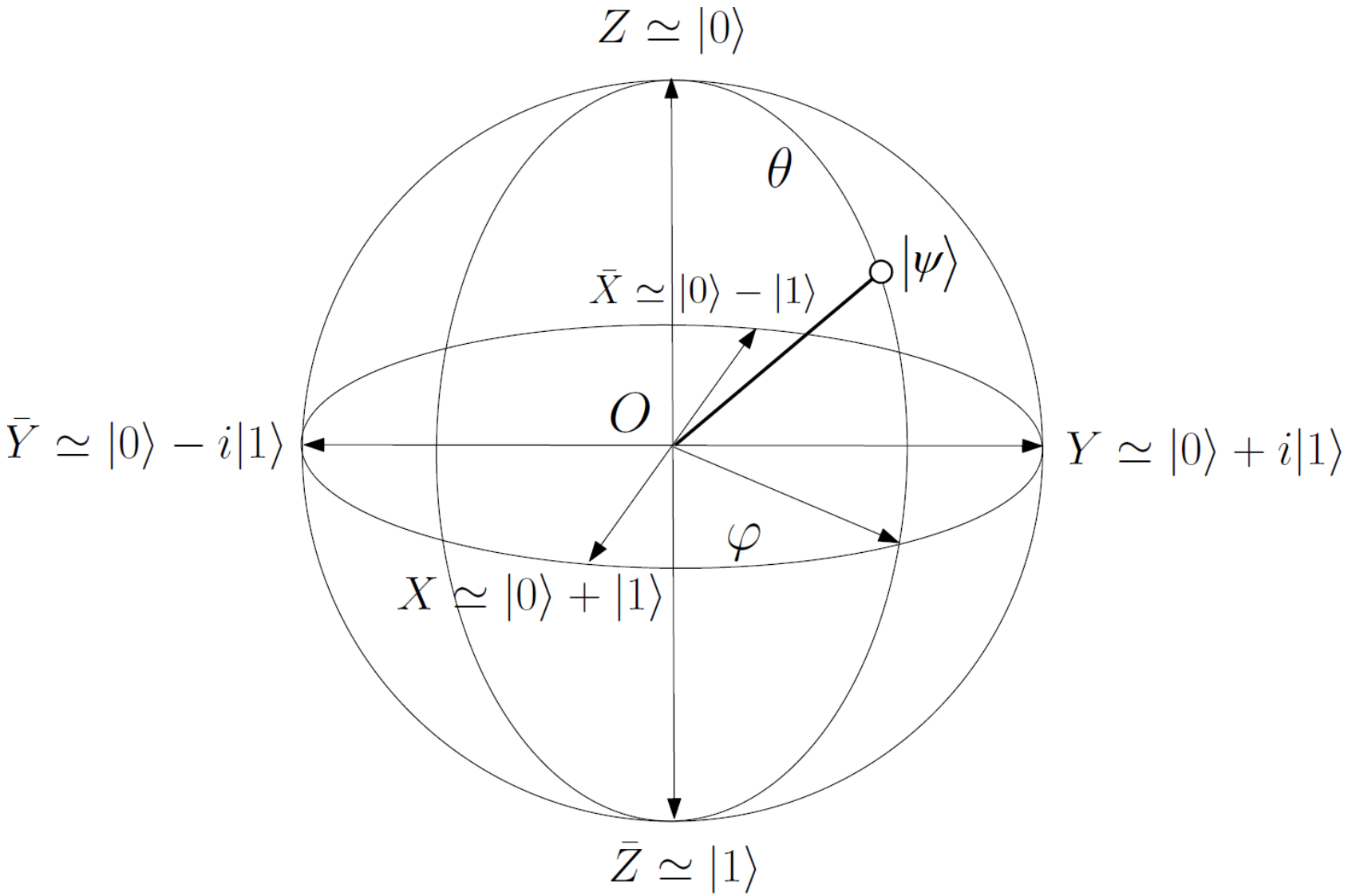
$$\psi_{\phi,\theta} = e^{-i\phi/2}\cos\frac{\theta}{2}e_0 + e^{i\phi/2}\sin\frac{\theta}{2}e_1$$

$(0 \leqslant \phi < 2\pi,\ 0 \leqslant \theta \leqslant \pi)$, then $S_{\phi,\theta} = |\psi_{\phi,\theta}\rangle \in S^2$ is the point at longitude ϕ and colatitude θ. In particular,

$\pm X = (\pm 1, 0, 0) = |\frac{\sqrt{2}}{2}(e_0 \pm e_1)\rangle$,

$\pm Y = (0, \pm 1, 0) = |\frac{\sqrt{2}}{2}(e_0 \pm ie_1)\rangle$,

$Z = |e_0\rangle = (0, 0, 1)$ (North pole), $-Z = |e_1\rangle = (0, 0, -1)$ (South pole).



$$\psi = e^{-i\varphi/2}\cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\varphi/2}\sin\left(\frac{\theta}{2}\right)|1\rangle$$

- The *measurement* of a q-bit that is in the state $S_{\phi,\theta}$, which corresponds to the *reading* of a classical bit, produces the state $|e_0\rangle = N = \uparrow$ or the state $|e_1\rangle = -N = \downarrow$, and this result is a random event with *probabilities* $p_S(\uparrow) = \cos^2 \frac{\theta}{2}$ and $p_S(\downarrow) = \sin^2 \frac{\theta}{2}$.

These states $\uparrow$ and $\downarrow$ are classical, in that $p_\uparrow(\uparrow) = p_\downarrow(\downarrow) = 1$ (in agreement with the Stern-Gerlach findings), and they are the only states having this property. For states S on the equator ($\theta = \pi/2$), $\uparrow$ and $\downarrow$ are equiprobable, and they are the only ones having this property.

- More generally, the model of a *q-register of length n*, Q_n, is based on replacing B^n (the space of n-bit strings) by $E^{(n)} = \langle B^n \rangle_{\mathbf{C}}$, the complex vector space with basis B^n, represented as the tensor product (*q-entanglement* of the q-bits in the q-register) of $E^{(1)}$ with itself n times.

So the vectors of $E^{(n)}$ have the form $\psi = \sum_{b \in B^n} \psi_b e_b$, where $e_b = e_{b_1} \otimes \cdots \otimes e_{b_n} \equiv e_{b_1} \cdots e_{b_n}$. A vector ψ is *normalized* if $|\psi|^2 = \sum_{b \in B^n} |\psi_b|^2 = 1$.

- Each normalized vector ψ defines a state $|\psi\rangle$ in the *state space* Σ_n of a Q_n, with the rule that $|\psi\rangle = |\psi'\rangle$ if and only if $\psi' = \xi\psi$ for some unit complex number ξ. Again, we denote this relation by $\psi \sim \psi'$. As we have seen, $\Sigma_1 = S^2$.

- The *measuring* of Q_n in the state $S = |\psi\rangle$ returns one of the basis states $|e_b\rangle \equiv |b\rangle$ at random with probabilities $|\psi_b|^2$.

- The notion of a classical (reversible) computation on n bits is replaced by a unitary matrix U of order 2^n, which can be viewed as a unitary transformation $U : E^{(n)} \to E^{(n)}$.
- To any classical computation $f : B^n \to B^n$ we can associate the q-computation $U_f : E^{(n)} \to E^{(n)}$ defined by $e_b \mapsto e_{f(b)}$. It is a permutation matrix.
- In particular we can regard the logical gates CNOT and TOFFOLI as *quantum gates*. Thus, for instance, CNOT(1,4) leaves e_b fixed if $b_1 = 0$ and changes it to $e_{b'}$ if $b_1 = 1$, where $b'_4 = 1 + b_4$ and otherwise $b'_i = b_i$.

- For a q-bit, the *Hadamard gate*, H, is defined by $e_0 \mapsto \frac{\sqrt{2}}{2}(e_0 + e_1)$, $e_1 \mapsto \frac{\sqrt{2}}{2}(e_0 - e_1)$:

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

This is a genuine q-gate, as the states $\pm X$ defined by $H(e_0)$ and $H(e_1)$ are not classical.

In the case of Q_n, H can be applied to any q-bit, or to all, in which case we will denote it by $H^{(n)}$. For example:

$$H^{(2)}e_{00} = \tfrac{1}{2}(e_0 + e_1)(e_0 + e_1) = \tfrac{1}{2}(e_{00} + e_{01} + e_{10} + e_{11}).$$

Thus, on measuring, the four possible results 00, 01, 10, 11 are equiproblable.

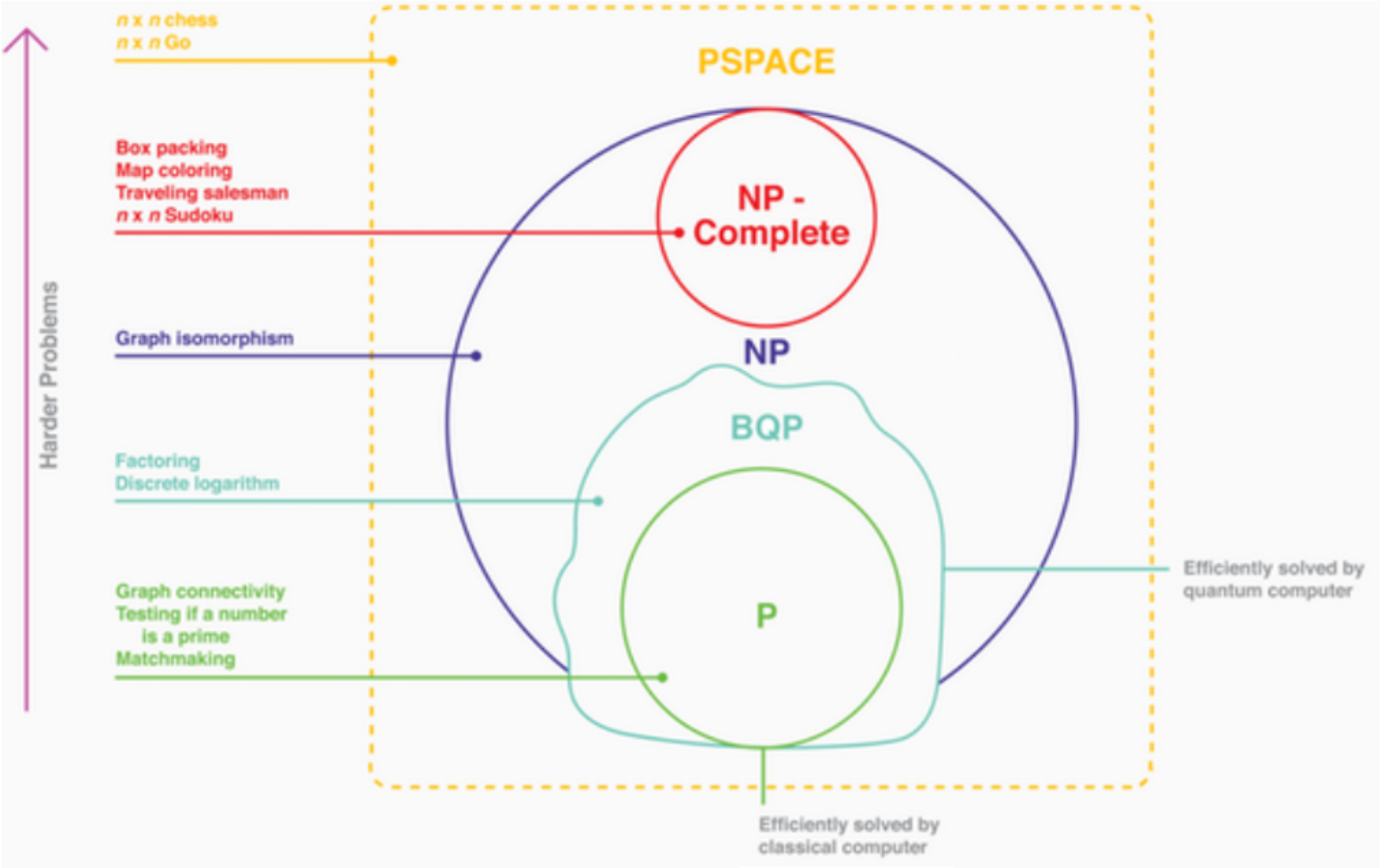
In general, $H^{(n)}e_{0\ldots0} = \left(\frac{\sqrt{2}}{2}\right)^n \sum_{b\in B^n} e_b$, a superpostion for which all states $|b\rangle$ are equiprobable. This embodies the so-called *quantum parallelism*.

- A *q-algorithm* is a sequence of Hadamard and Toffoli gates, followed by a measure of the final state, which is the (classical) bit string returned by the algorithm.

Example. Initialize Q_3 in the state $|000\rangle$. Apply $H^{(3)}$. End by a measuring operation. This q-algorithm yields a uniform random string of three bits. The generalization to Q_n is obvious.

- *Shor's q-algorithm factors integers in polynomial time*.
- *Grover's q-algorithm searches an item in a list of size N in time* $\sqrt{N}$.

For a systematic introduction to q-computing, including a discussion of these and other algorithms, see [4], and also the references therein.



An excellent discussion about the complexity theory issues is provided by Scott Aaronson' book [7] (*Quantum computing since Democritus*).

Post-quantum cryptography

Quantum threats to cryptographic protocols. McECS as a post-quantum system. Other post-quantum protocols. The NIST initiative for PQ protocols.

- q-computing poses a fundamental threat to widespread cryptographic systems like RSA.
- In principle, it does not pose a threat to McECS (unless P = NP), because the general problem of decoding linear codes is NP complete [8] (Berlekamp, McEliece and van Tilborg, 1978: *On the inherent intractability of certain coding problems*).

However, for the special codes used in McECS it could still exist an astute way of using their structure to crack them. But all the evidence collected in their study so far (see page 32) suggests it is a *post-quantum* protocol, in the sense that no computational power can crack it if the appropriate parameters are used.

In addition to the McEliece cryptosystem, there are other systems that may qualify as post-quantum cryptography:

- Hash-based cryptography;
- Code-based cryptography;
- Lattice-based cryptography;
- Multivariate-quadratic-equations cryptography.

See [9], particularly the introductory paper by D. J. Bernstein.

These, and variations on them, are being considered by NIST with the goal set at defining and standardize one or more post-quantum cryptography protocols. See `http://dx.doi.org/10.6028/NIST.IR.8105`.

Urmila Mahadev

For outstanding achievements

"Urmila Mahadev spent eight years in graduate school solving one of the most basic questions in quantum computation: How do you know whether a quantum computer has done anything quantum at all?" (article by Erika Klarreich, 8 October 2018, in Quantamagazine).

`https://www.quantamagazine.org/graduate-student-solves-quantum-verification-problem-20181008/`

See [10] (U. Mahadev, *Classical Verification of Quantum Computations*).

Abstract. We present the first protocol allowing a classical computer to interactively verify the result of an efficient quantum computation.

We achieve this by constructing a *measurement protocol*, which enables a classical verifier to use a quantum prover as a trusted measurement device.

The protocol forces the prover to behave as follows: (1) the prover must construct an n qubit state of his choice, (2) measure each qubit in the *Hadamard* or standard basis **as directed by the verifier**, and (3) report the measurement results to the verifier.

The soundness of this protocol is enforced *based on the assumption that the learning with errors problem* [one of the post-quantum protocols] is computationally intractable for efficient quantum machines.

Further references

On mathematical cryptography: [11].

For a study of McECS and its security: [3]

On quantum computing: [12]. For an approach to the q-bit based on the stereographic projection of S^2, see [13].

Another important paper of Urmila Mahadev: [14].

References I

[1] S. Xambó-Descamps, *Block error-correcting codes: a computational primer*.

Univesitext, Springer, 2003.

[2] S. Xambó-Descamps, J. M. Miret, and N. Sayols, *Intersection Theory and Enumerative Geometry–A Computational Primer*.

Universitext, Springer, 2020.

Extended second editon of *Using Intersection Theory*, by S. Xambó-Descamps, Sociedad Matemática Mexicana, 1996.

[3] N. Sayols and S. Xambó-Descamps, "Computer Algebra Tales on Goppa Codes and McEliece Cryptography," *Mathematics in Computer Science*, 2019.

13 p. Accepted 12 April 2019.

References II

[4] J. Rué and S. Xambó-Descamps, "Introducció matemàtica a la computació quàntica," *Butlletí de la Societat Catalana de Matemàtiques*, vol. 28, no. 2, pp. 183–231, 2013.

English version:
`https://mat-web.upc.edu/people/sebastia.xambo/QC/qc.pdf`.

[5] D. J. Bernstein, T. Lange, and C. Peters, "Attacking and defending the McEliece cryptosystem," in *Post-Quantum Cryptography* (J. D. E. J. Buchanan, ed.), vol. 5299 of *Lecture Notes Computer Science*, pp. 31–46, Springer, 2008.

Proceedings of the Second PQCrypto international workshop, Cincinnati, OH, USA, October 17-19, 2008.
`https://cr.yp.to/codes/mceliece-20080807.pdf`.

References III

[6] "Post-Quantum Cryptography 2018."
https://csrc.nist.gov/Projects/Post-Quantum-Cryptography.

First PQC Standardization Conference organized by the NIST Computer Security Resource Center.

[7] S. Aaronson, *Quantum computing since Democritus*.

Cambridge University Press, 2013.

[8] E. R. Berlekamp, R. J. McEliece, and H. C. van Tilborg, "On the inherent intractability of certain coding problems," *IEEE Transactions of Information Theory*, vol. 24, no. 3, pp. 384–386, 1978.

[9] D. J. Bernstein, J. Buchmann, and E. D. (eds.), *Post-Quantum Cryptography*.

Springer, 2009.

ix+345 p.

References IV

[10] U. Mahadev, "Classical verification of quantum computations," in *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 259–267, IEEE, 2018.

`https://arxiv.org/pdf/1804.01082.pdf`.

[11] J. Hoffstein, J. Pipher, J. H. Silverman, and J. H. Silverman, *An introduction to mathematical cryptography*.

Undergraduate Texts in Mathematics, Springer, 2008.

[12] M. A. Nielsen and I. Chuang, "Quantum computation and quantum information," 2002.

[13] S. Xambó-Descamps, "Escondidas sendas de la geometría proyectiva a los formalismos cuánticos," in *El legado matemático de Juan Bautista Sancho Guimerá* (A. Campillo and D. Hernández-Ruipérez, eds.), pp. 233–274, Universidad de Salamanca, 2016.

References V

[14] U. Mahadev, "Classical homomorphic encryption for quantum circuits," in *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 332–338, IEEE, 2018.

https://arxiv.org/pdf/1708.02130.pdf.