Reversing cryptographic primitives using quantum computing

Renaud Lifchitz, Econocom digital.security

HITB Amsterdam, May 10th, 2019

digital security | econocom

Outline (1/2)

Quantum computing basics

- Principles
- Simple quantum gates
- Challenges

Quantum computing simulators

Overview of public quantum cloud computing services

Outline (2/2)

Quantum computing & cryptography

- P-Box modeling & implementation
- 2 ways to reverse a cryptographic primitive
- CRC-8 modeling & optimal implementation
- AES (Rijndael's) S-box modeling & implementation
- Reversing XOR encryption using an oracle
- Quantum threats against current cryptography

Post-quantum cryptography

Speaker's bio

- French security expert @ Econocom digital.security
- Main activities:
  - Penetration testing & security audits
  - Security research
  - Security trainings
- Main interests:
  - Security of protocols (authentication, cryptography, information leakage, zero-knowledge proofs...)
  - Number theory (integer factorization, primality testing, elliptic curves...)

Quantum computing basics

Principles

Quantum principles

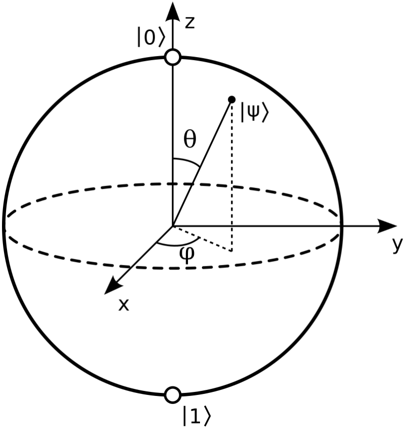
1. Small-scale physical objects (atom, molecule, photon, electron, ...) both behave as particles and as waves during experiments (quantum duality principle)
2. Main characteristics of these objects (position, spin, polarization, ...) are not determined, have multiple values according to a probabilistic distribution (quantum superposition principle / Heisenberg's uncertainty principle)
3. Further interaction or measurement will collapse this probability distribution into a single, steady state (quantum decoherence principle)
4. Consequently, copying a quantum state is not possible (no-cloning theorem)

- We can still take advantage of the first 3 principles to do powerful non-classical computations

Qubit representations (1/2)

- Constant qubits 0 and 1 are represented as $|0\rangle$ and $|1\rangle$
- They form a 2-dimension basis, e.g. $|0\rangle = \begin{bmatrix}1\\0\end{bmatrix}$ and $|1\rangle = \begin{bmatrix}0\\1\end{bmatrix}$
- An arbitrary qubit q is a linear superposition of the basis states: $|q\rangle = \alpha|0\rangle + \beta|1\rangle = \begin{bmatrix}\alpha\\\beta\end{bmatrix}$ where $\alpha \in \mathbb{C}$, $\beta \in \mathbb{C}$
- When q is measured, the real probability that its state is measured as $|0\rangle$ is $|\alpha|^2$ so $|\alpha|^2 + |\beta|^2 = 1$
- Combination of qubits forms a quantum register and can be done using the tensor product: $|10\rangle = |1\rangle \otimes |0\rangle = \begin{bmatrix}0\\0\\1\\0\end{bmatrix}$
- First qubit of a combination is usually the most significant qubit of the quantum register

Qubit representations (2/2)



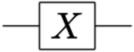
Bloch sphere: a qubit can also be viewed as a unit vector within a sphere - 3 angles (2 angles and a phase)

Basics of quantum gates

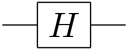
- For thermodynamic reasons, a quantum gate must be reversible
- It follows that quantum gates have the same number of inputs and outputs
- A n-qubit quantum gate can be represented by a 2^nx2^n unitary matrix
- Applying a quantum gate to a qubit can be computed by multiplying the qubit vector by the operator matrix on the left
- Combination of quantum gates can be computed using the matrix product of their operator matrix
- In theory, quantum gates don't use any energy nor give off any heat

Simple quantum gates

Pauli-X gate

<table>
<tr><td>Pauli-X gate</td><td>Number of qubits: 1</td><td>Symbol: 
</td></tr>
<tr><td colspan="3">Description: Quantum equivalent of a NOT gate. Rotates qubit around the X-axis by Π radians. $X.X = I$.</td></tr>
<tr><td colspan="3">Operator matrix: $X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$</td></tr>
</table>

Hadamard gate

<table>
<tr><td>Hadamard gate</td><td>Number of qubits: 1</td><td>Symbol:

</td></tr>
<tr><td colspan="3">Description: Mixes qubit into an equal superposition of $|0\rangle$ and $|1\rangle$.</td></tr>
<tr><td colspan="3">Operator matrix: $H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$</td></tr>
</table>

Hadamard gate

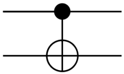
- The Hadamard gate is a special transform mapping the qubit-basis states $|0\rangle$ and $|1\rangle$ to two superposition states with "50/50" weight of the computational basis states $|0\rangle$ and $|1\rangle$:

$$H.|0\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$$

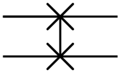
$$H.|1\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle$$

- For this reason, it is widely used for the first step of a quantum algorithm to work on all possible input values in parallel

CNOT gate

<table>
<tr><td>CNOT gate</td><td>Number of qubits: 2</td><td>Symbol:
</td></tr>
<tr><td colspan="3">Description: Controlled NOT gate. First qubit is control qubit, second is target qubit. Leaves control qubit unchanged and flips target qubit if control qubit is true. CNOT gates with more than one control qubit are called Toffoli gates.</td></tr>
<tr><td colspan="3">Operator matrix: $CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$</td></tr>
</table>

SWAP gate

<table>
<tr><td>SWAP gate</td><td>Number of qubits: 2</td><td>Symbol: </td></tr>
<tr><td colspan="3">Description: Swaps the 2 input qubits.</td></tr>
<tr><td colspan="3">Operator matrix: $SWAP = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$</td></tr>
</table>

Universal gates

A set of quantum gates is called **universal** if any classical logic operation can be made with only this set of gates. Examples of universal sets of gates:

- Hadamard gate, Phase shift gate (with $\theta = \frac{\Pi}{4}$ and $\theta = \frac{\Pi}{2}$) and Controlled NOT gate
- Toffoli gate only

Challenges

Challenges (1/2)

- Qubits and qubit registers cannot be independently copied in any way
- In simulation like in reality, number of used qubits must be limited (qubit reuse wherever possible)
- Qubit registers shifts are costly, moving gates “reading heads” is somehow easier
- In reality, quantum error codes should be used to avoid partial decoherence during computation

Challenges (2/2)

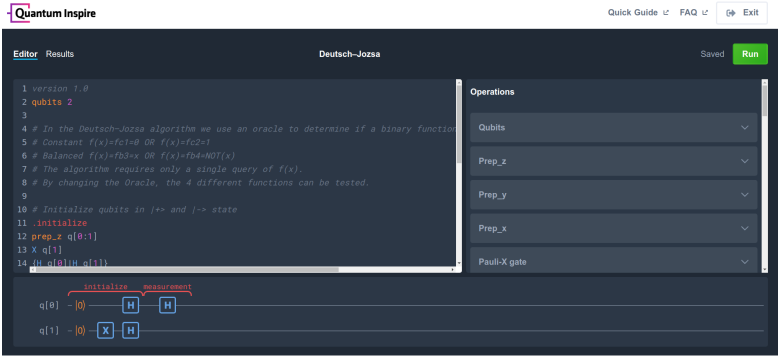
For serious purposes we need:

- A high number of qubits
  (about 50 qubits is enough for quantum supremacy)
- A good qubit and gate fidelity (low-error rate)
- Optionally, error correction

High number of qubits is not the most important, most algorithms are limited by circuit depth ($\approx$ 20-30 gates) because of qubit and gate fidelity.

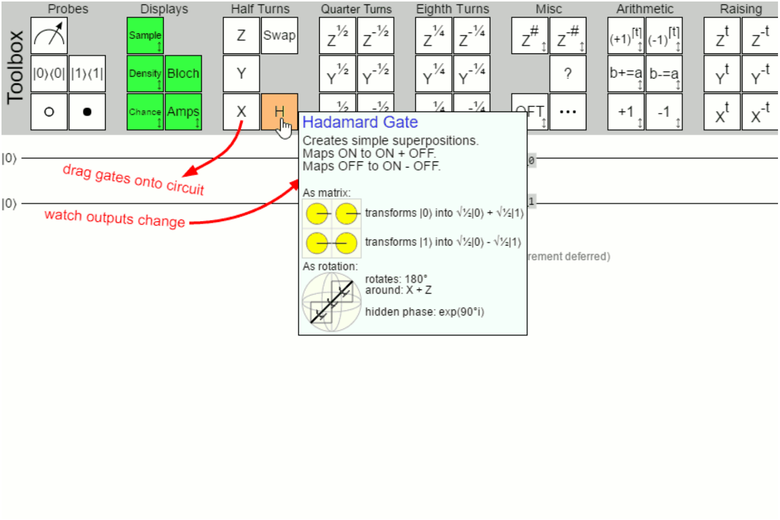
Quantum computing simulators

Quantum Inspire



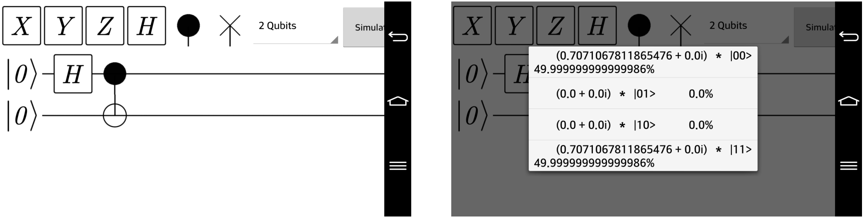
https://www.quantum-inspire.com/

Quirk



http://algassert.com/quirk

Quantum Circuit Simulator (Android)



Design and simulation of a qubit entanglement circuit

`https://play.google.com/store/apps/details?id=mert.qcs`

Quantum computing simulators

A longer list:

`https://quantiki.org/wiki/list-qc-simulators`

Overview of public quantum cloud computing services

Public quantum cloud computing services

- Bristol University "Quantum in the Cloud" (http://www.bristol.ac.uk/physics/research/quantum/engagement/qcloud/): up to 2-3 qubits
- Alibaba Quantum Computing Cloud Service (http://quantumcomputer.ac.cn): up to 11 qubits
- IBM "Q Experience" (https://www.research.ibm.com/ibm-q/technology/devices/): up to 14 qubits, 20 qubits for private clients
- Rigetti "Quantum Cloud Services" (https://www.rigetti.com/qpu): up to 19 qubits, 128 qubits to come
- D-Wave "Leap" (https://cloud.dwavesys.com/leap/): up to 1000 qubits, adiabatic quantum chip, not universal, mainly for optimization problems

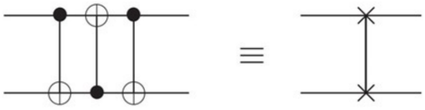
Quantum computing & cryptography

P-Box modeling & implementation

Modeling permutations and their reverse

Modeling a complex permutation and its reverse requires:

- Decomposing the permutation in single (two-elements) permutations
- Implementing it using several SWAP gates
- Converting SWAP gates to CNOT gates for practical reasons



- Inverting the whole circuit (most gates are their own inverse!)
- Simplifying the circuit

2 ways to reverse a cryptographic primitive

2 ways to reverse a cryptographic primitive

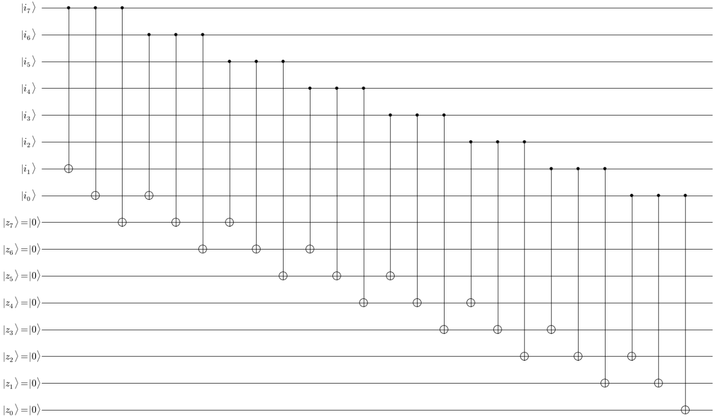
- Implement a reversible circuit and execute it in the reverse way. Problems:
  - Function is not often reversible, solutions: embed function (add input bits as output bits and various other simple techniques)
  - Ancilla qubits are often numerous (but efficient if they are in minority)
- Grover oracle: implement the primitive in the direct way and query a Grover oracle (specific quantum-only algorithm) to find the correct input

CRC-8 modeling & optimal implementation

Reverse CRC-8 modeling: the steps

- Naive CRC-8 implementation (moving “reading heads” to shift qubits) using ancilla qubits
- Simplify if possible
- Compute the CRC-8 truth table
- Use a reversible computation framework to find a (optimum) circuit

CRC-8: a nearly naive implementation



A quantum CRC-8 circuit with only CNOT gates

revkit: a useful framework for reversible computation

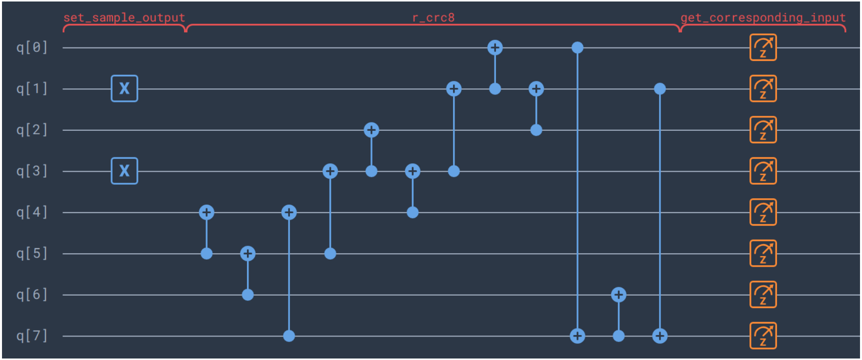
- Interesting framework for reversible & quantum circuits
- Takes various kinds of inputs (truth tables, circuits, boolean functions)
- Has different synthesis & optimization strategies
- Able to embed non-reversible functions into reversible ones
- Sometimes able to find optimum circuits (if not too big)
- `https://msoeken.github.io/revkit.html`

Reverse-CRC-8 optimal implementation (1/2)

```
version 1.0
qubits 8
error_model depolarizing_channel, 0.001
.set_sample_output
{X q[1]|X q[3]}
.R_CRC8
CNOT q[5],q[4]
CNOT q[6],q[5]
CNOT q[7],q[4]
CNOT q[5],q[3]
CNOT q[3],q[2]
CNOT q[4],q[3]
CNOT q[3],q[1]
CNOT q[1],q[0]
CNOT q[2],q[1]
CNOT q[0],q[7]
CNOT q[7],q[6]
CNOT q[1],q[7]
.get_corresponding_input
Measure_all
```

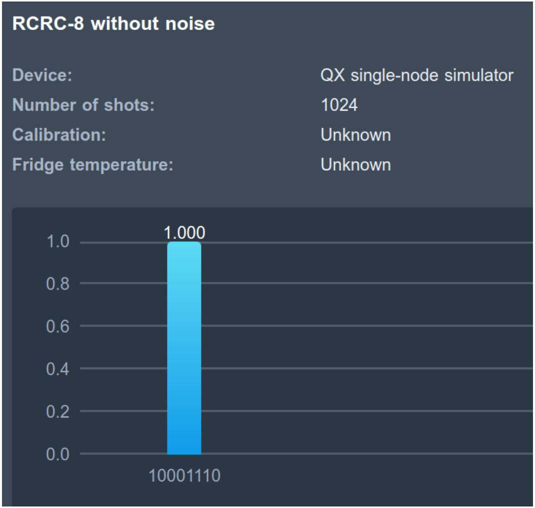
Our optimal reverse-CRC-8 circuit instructions
using Quantum Inspire

Reverse-CRC-8 optimal implementation (2/2)



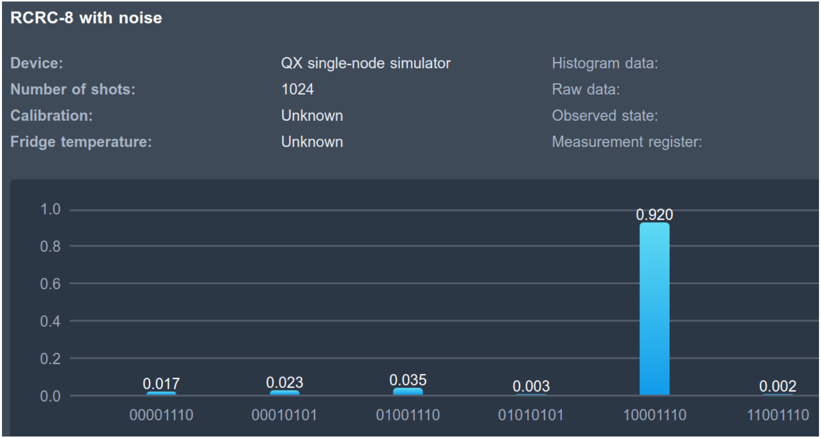
Optimal circuit visualized using Quantum Inspire

Reversing a single CRC-8 using quantum computing (1/4)



Quantum simulation without noise using Quantum Inspire

Reversing a single CRC-8 using quantum computing (2/4)



Quantum simulation with typical noise using Quantum Inspire

Reversing a single CRC-8 using quantum computing (3/4)

```
def go():
    q = QuantumRegister(8, 'q'); b = ClassicalRegister(8, 'b'); qc1 = QuantumCircuit(q, b)
    qc1.x(q[1]); qc1.x(q[3]); qc1.barrier(q) # Input value
    qc1.cx(q[5], q[4]); qc1.cx(q[6], q[5]); qc1.cx(q[7], q[4])
    qc1.cx(q[5], q[3]); qc1.cx(q[3], q[2]); qc1.cx(q[4], q[3]);
    qc1.cx(q[3], q[1]); qc1.cx(q[1], q[0]); qc1.cx(q[2], q[1]);
    qc1.cx(q[0], q[7]); qc1.cx(q[7], q[6]); qc1.cx(q[1], q[7])
    qc1.barrier(q); qc1.measure(q, b)
    job_sim = execute([qc1,], Aer.get_backend('qasm_simulator'))
    sim_result = job_sim.result(); print("simulation: ",sim_result.get_counts(qc1))
    print("\n(IBMQ Backends)", IBMQ.backends())
    try:
      #least_busy_device = least_busy(IBMQ.backends(simulator=False))
      least_busy_device = IBMQ.get_backend('ibmq_16_melbourne')
      print("Running on current least busy device: ", least_busy_device)
      # running the job
      job_exp = execute([qc1,], backend=least_busy_device, shots=1024)
      interval = 10
      while job_exp.status().name != 'DONE':
        print(job_exp.status().name)
        time.sleep(interval)
      exp_result = job_exp.result()
      d=exp_result.get_counts(qc1)
      print(sorted([(v,k) for k,v in d.items()], reverse=True))
    except ValueError:
        print("All devices are currently unavailable.")
```

Reversing a single CRC-8 on real quantum hardware
(program, IBM Q 14 Melbourne)

Reversing a single CRC-8 using quantum computing (4/4)

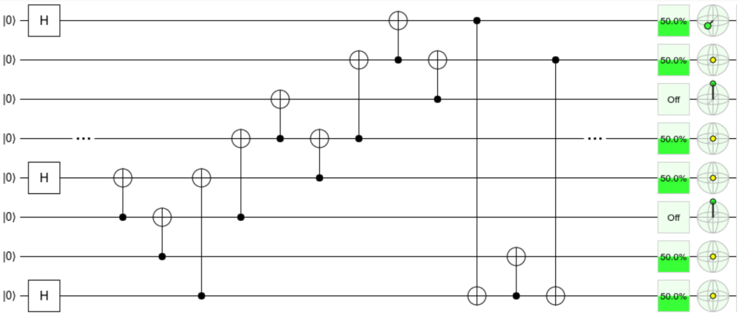
```
go()
```

```
simulation:  {'10001110': 1024}

(IBMQ Backends) [<IBMQBackend('ibmqx4') from IBMQ()>, <IBMQBackend('ibmqx5') from IBMQ()>, <IBMQBackend('ibmqx2') f
rom IBMQ()>, <IBMQBackend('ibmq_16_melbourne') from IBMQ()>, <IBMQBackend('ibmq_qasm_simulator') from IBMQ()>]
Running on current least busy device:  ibmq_16_melbourne
INITIALIZING
RUNNING
RUNNING
[(95, '00000000'), (56, '01000011'), (54, '00100000'), (43, '10001110'), (34, '01000010'), (26, '10000110'), (25,
'01100011'), (25, '00001000'), (24, '11000001'), (24, '10000010'), (24, '00000010'), (24, '00000001'), (23, '110011
01'), (23, '00101000'), (20, '01010011'), (20, '01010010'), (19, '11000101'), (18, '11100001'), (18, '11011100'),
(18, '01001011'), (17, '10101110'), (17, '10001010'), (17, '00000011'), (16, '11000000'), (16, '00001010'), (15, '1
1010000'), (15, '01011111'), (15, '00100001'), (14, '11010001'), (14, '01100010'), (14, '01000111'), (14, '0100000
1'), (14, '00010100'), (14, '00010001'), (14, '00000100'), (13, '11011101'), (13, '10100010'), (13, '10011010'), (1
3, '01001010'), (13, '00011000'), (12, '10010010'), (12, '10000000'), (12, '01101111'), (12, '01010111'), (12, '001
00011'), (11, '11110100'), (11, '11011001'), (11, '11010100'), (11, '01101011'), (11, '00010000'), (10, '1101010
1'), (10, '11001001'), (10, '10110010'), (10, '10011110'), (10, '10010000'), (10, '10001111'), (10, '10001100'), (1
0, '10000111'), (10, '01111111'), (10, '01010000'), (10, '01000000'), (10, '00110001'), (9, '11101100'), (9, '11100
101'), (9, '11100000'), (9, '11000100'), (9, '10100110'), (9, '10100011'), (9, '10011111'), (9, '10010011'), (9, '0
1110111'), (9, '01110010'), (9, '01001111'), (9, '01000110'), (9, '00111100'), (9, '00110000'), (8, '11110001'),
```

Reversing a single CRC-8 on real quantum hardware
(results, IBM Q 14 Melbourne)

Reversing multiple CRC-8s with fixed and unfixed bits



Quantum simulation & results using Quirk: fixed null bits have been found in the input for 8 different outputs! (`https://tinyurl.com/rcrc8multi`)

AES (Rijndael's) S-box modeling & implementation

AES S-Box implementation

Forward S-box [edit]

The S-box maps an 8-bit input, c, to an 8-bit output, $s = S(c)$. Both the input and output are interpreted as polynomials over GF(2). First, the input is mapped to its multiplicative inverse in $GF(2^8) = GF(2)[x]/(x^8 + x^4 + x^3 + x + 1)$, Rijndael's finite field. Zero, which has no inverse, is mapped to zero. This transformation is known hift-q] the "Nyberg S-box" after its inventor Kaisa Nyberg.[2] The multiplicative inverse is then transformed using the following affine transformation:

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \\ s_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

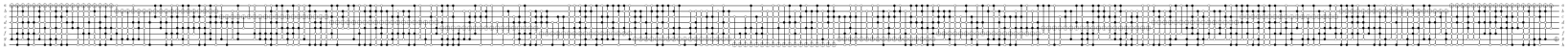
where $[s_7, ..., s_0]$ is the S-box output and $[b_7, ..., b_0]$ is the multiplicative inverse as a vector.

AES S-Box. The column is determined by the least significant nibble, and the row by the most significant nibble. For example, the value `0x9a` is converted into `0xb8`.

	00	**01**	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**	**0a**	**0b**	**0c**	**0d**	**0e**	**0f**
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Source: Wikipedia

Reverse AES S-Box implementation



Our reverse AES S-Box circuit
with 281 Pauli-X, CNOT and Toffoli gates
(optimal circuit requires at least 14 gates)

Reversing XOR encryption using an oracle

Reversing XOR encryption using an oracle

- Idea: for a given key size, implement a direct XOR encryption and find the candidate keys by minimizing the bytes MSBs (for ASCII text encryption)

Quantum threats against current cryptography

Quantum threats against symmetric cryptography

Main threat is Grover algorithm:

- Pure quantum algorithm for searching among N unsorted values
- Complexity: $\mathcal{O}(\sqrt{N})$ operations and $\mathcal{O}(\log N)$ storage place
- Probabilistic, iterating and optimal algorithm

Defense: doubling all symmetric key sizes is enough to be out of reach from quantum attacks

Quantum threats against asymmetric cryptography

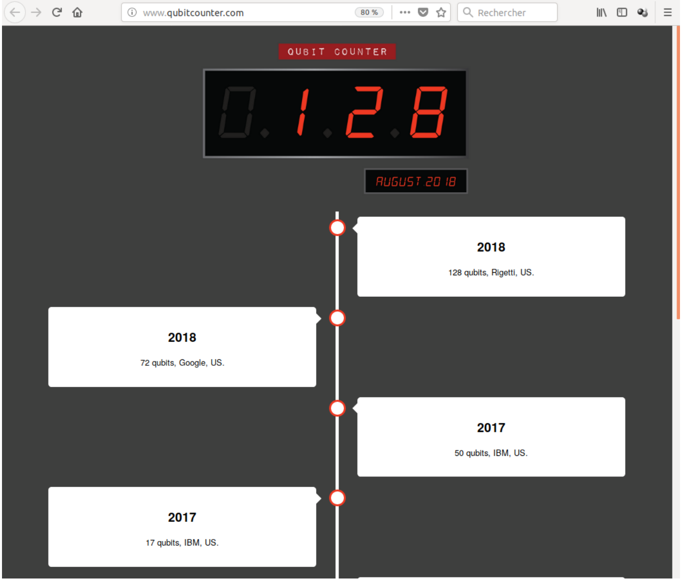
Main threat is Shor algorithm:

- Pure quantum algorithm for integer factorization that runs in polynomial time formulated in 1994
- Complexity: $\mathcal{O}((\log N)^3)$ operations and storage place
- Probabilistic algorithm that basically finds the period of the sequence $a^k \mod N$ and non-trivial square roots of unity mod N
- Uses QFT, some steps are performed on a classical computer
- Breaks RSA, DSA, ECDSA, ECDLP efficiently

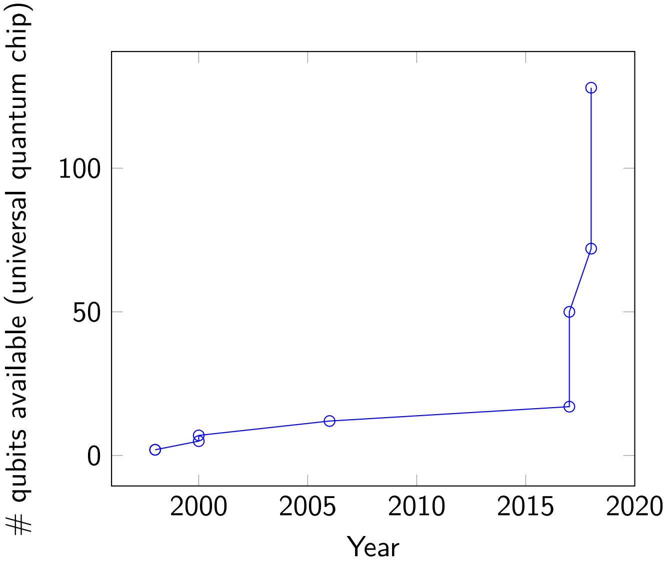
Defense: use a PQC alorithm

Post-quantum cryptography

Progress in number of qubits (1/2)



Progress in number of qubits (2/2)



Looks like a Moore law... ☺

Quantum Resistant Cryptography

Currently there are 6 main different approaches:

- Lattice-based cryptography
- Multivariate cryptography
- Hash-based cryptography
- Code-based cryptography
- Supersingular Elliptic Curve Isogeny cryptography
- Symmetric Key Quantum Resistance

Annual event about PQC: PQCrypto conference
(`https://twitter.com/pqcryptoconf`, 10th edition in 2019)

Quantum Resistant Cryptography

Very few asymmetric PQ algorithms, the most well-known is NTRU, a lattice-based shortest vector problem:

- NTRUEncrypt for encryption (1996)
- NTRUSign for digital signature

`https://www.onboardsecurity.com/products/ntru-crypto`

Thank you!

Questions?
@nono2357 on Twitter
info@digital.security