Attacking binary elliptic curves on a quantum computer

On quantum arithmetic and space-time trade-offs

Martin Roetteler
Microsoft Research

Based on joint work with Brittanney Amento and Rainer Steinwandt [arXiv.org: 1209.5491, 1209.6348, 1306.1161]

DIMACS Workshop on the Mathematics of Post-Quantum Cryptography
January 15, 2015

Motivation

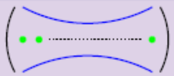
- Analyze resources needed to implement Shor
- Focus: Computing dlogs over abelian groups
- Possible circuit optimizations
- Scaling of space (=#qubits) and time (=depth)?

Please ask questions during talk!

Background:
Quantum resources

Quantum bits and registers

Quantum register of n qubits

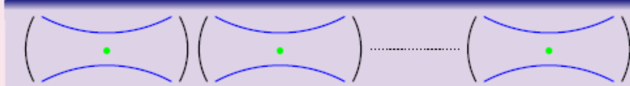
 can hold any coherent superposition

$$|\Psi\rangle = \sum_{\underline{\epsilon} \in \{0,1\}^n} \alpha_{\epsilon_1 \cdots \epsilon_n} |\epsilon_1\rangle \otimes |\epsilon_2\rangle \otimes \cdots \otimes |\epsilon_n\rangle$$

in the 2^n dimensional space $\mathcal{H}_{2^n} = \mathbb{C}^2 \otimes \mathbb{C}^2 \otimes \cdots \otimes \mathbb{C}^2 = \mathbb{C}^{2^n}$.

$$\neq$$

Product states of n qubits

 can only hold a product state

$$|\Psi_1\rangle \otimes |\Psi_2\rangle \otimes \cdots \otimes |\Psi_n\rangle = (\alpha_1 |\uparrow\rangle + \beta_1 |\downarrow\rangle) \otimes \cdots \otimes (\alpha_n |\uparrow\rangle + \beta_n |\downarrow\rangle)$$

(thus, only linear scaling of system and dimension)

Measurements

von Neumann measurements of one qubit

First, specify a basis B for $\mathbb{C}^2$, e. g. $\{|0\rangle, |1\rangle\}$. The outcome of measuring the state $\alpha|0\rangle + \beta|1\rangle$ is described by a random variable X. The probabilities to observe "0" or "1" are given by

$$\Pr(X = 0) = |\alpha|^2, \quad \Pr(X = 1) = |\beta|^2.$$

Measuring a state in $\mathbb{C}^n$ in an orthonormal basis B

- Recall: Orthonormal Basis of $\mathbb{C}^N$

  $$B = \{|\psi_i\rangle : i = 1, \ldots, N\}, \quad \text{where } \langle\psi_i|\psi_j\rangle = \delta_{i,j}$$

- Let $|\varphi\rangle = \sum_{i=1}^N \alpha_i |i\rangle$, where $\sum_{i=1}^N |\alpha_i|^2 = 1$. Then measuring $|\varphi\rangle$ in the basis B gives random variable X_B taking values $1, \ldots, N$:

  $$\Pr(X_B = 1) = |\langle\psi_1|\varphi\rangle|^2, \ \ldots\ , \Pr(X_B = N) = |\langle\psi_N|\varphi\rangle|^2.$$

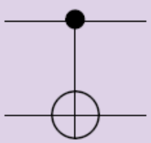
Examples: local operations and CNOT

Local operations

$$\mathbf{1}_N \otimes U \quad = \begin{pmatrix} \boxed{U} & & & \\ & \boxed{U} & & \\ & & \ddots & \\ & & & \boxed{U} \end{pmatrix}, \quad \text{where} \quad U \in \mathcal{U}(2).$$

Conditioned operation: the controlled NOT (CNOT)

$$\boxed{\begin{array}{l} |00\rangle \mapsto |00\rangle \\ |01\rangle \mapsto |01\rangle \\ |10\rangle \mapsto |11\rangle \\ |11\rangle \mapsto |10\rangle \end{array}} \quad \widehat{=} \quad \begin{pmatrix} 1 & \cdot & \cdot & \cdot \\ \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & 1 & \cdot \end{pmatrix} \quad \widehat{=}$$

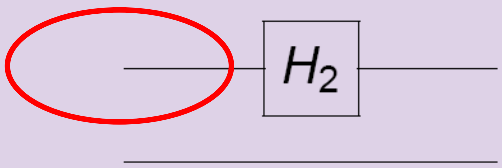


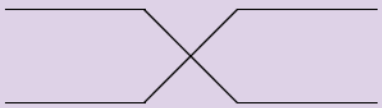
Notation for unitary matrices

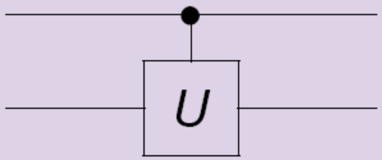
Gate in Feynman notation

Wire = qubit

$$H_2 = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$







Corresponding transformation

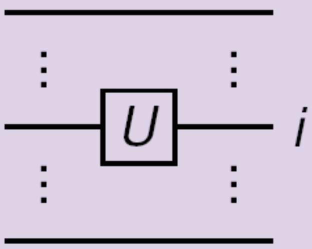
$$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & \cdot & 1 & \cdot \\ \cdot & 1 & \cdot & 1 \\ 1 & \cdot & -1 & \cdot \\ \cdot & 1 & \cdot & -1 \end{bmatrix}$$

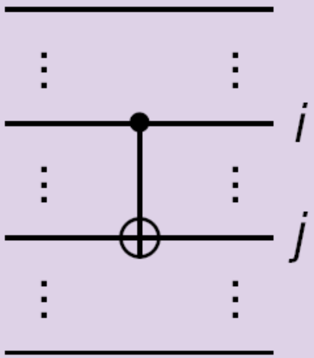
$$\begin{bmatrix} 1 & \cdot & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot \\ \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & \cdot & \cdot & \cdot \\ \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & u_{11} & u_{12} \\ \cdot & \cdot & u_{21} & u_{22} \end{bmatrix}$$

Universality theorem

Elementary quantum gates

$U^{(i)} =$ 

$\mathrm{CNOT}^{(i,j)} =$ 

Universal set of gates

Theorem (Barenco et al., 1995):

$$\mathcal{U}(2^n) = \langle U^{(i)}, \mathrm{CNOT}^{(i,j)} \quad : \quad i,j = 1,\ldots,n, \quad i \neq j \rangle$$

Quantum gates: main problem

Find efficient factorizations for given $U \in \mathcal{U}(2^n)$!

Levels of abstraction

Unitary matrix

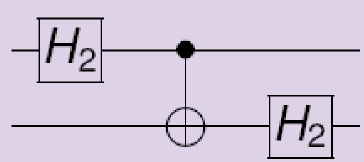
$$U = \frac{1}{2}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ -1 & 1 & 1 & -1 \end{pmatrix}$$

Factorized unitary matrix

$$U = \quad (I \otimes H_2) \quad (I \oplus \sigma_x) \quad (H_2 \otimes I)$$

$$= \frac{1}{2}\begin{pmatrix} 1 & 1 & & \\ 1 & -1 & & \\ & & 1 & 1 \\ & & 1 & -1 \end{pmatrix}\left(\begin{array}{c|c} I & \\ \hline & \sigma_x \end{array}\right)\begin{pmatrix} 1 & & 1 & \\ & 1 & & 1 \\ 1 & & -1 & \\ & 1 & & -1 \end{pmatrix}$$

Quantum circuit



Many more levels down (FTQECC, q control) and up (prog lang)

Operations on subspaces

Rotation on a subspace spanned by the states s_1 and s_2

$$T(s_1,s_2) = \begin{array}{cc}
\begin{array}{ccccccccccc} & & & s_1 & & & & s_2 & & & \end{array} & \\
\left(\begin{array}{ccccccccccc}
1 & & & & & & & & & & \\
 & \ddots & & & & & & & & & \\
 & & 1 & & & & & & & & \\
 & & & * & & & & * & & & \\
 & & & & 1 & & & & & & \\
 & & & & & \ddots & & & & & \\
 & & & & & & 1 & & & & \\
 & & & * & & & & * & & & \\
 & & & & & & & & 1 & & \\
 & & & & & & & & & \ddots & \\
 & & & & & & & & & & 1
\end{array}\right) & \begin{array}{c} \\ \\ \\ s_1 \\ \\ \\ \\ s_2 \\ \\ \\ \\ \end{array}
\end{array}$$

Theorem

Every $U \in \mathcal{U}(2^n)$ can be written in the form

$$U = \prod_{s_1,s_2 \in \{0,1\}^n} T(s_1,s_2).$$

Controlled rotations

Conditional gates with multiple controls

Let $U \in \mathcal{U}(2)$. Then $\Lambda_k(U) \in \mathcal{U}(2^{k+1})$ is defined by

$$\Lambda_k := \begin{pmatrix} 1 & & & \\ & \ddots & & \\ & & 1 & \\ & & & \boxed{U} \end{pmatrix} = \mathbf{1}_{2^{k+1}-2} \oplus U.$$

Alternative description of $\Lambda_k(U)$

$$\Lambda_k(U)\,|x_1,\ldots,x_n\rangle\,|y\rangle = \begin{cases} |x_1,\ldots,x_n\rangle\,|y\rangle & \text{if } \exists i : x_i \neq 1 \\ |x_1,\ldots,x_n\rangle\,U\,|y\rangle & \text{if } \forall i : x_i = 1 \end{cases}$$

Remark: For $U = NOT$, the gate $\Lambda_1(NOT)$ is the CNOT gate. The gate $\Lambda_2(NOT)$ is called the Toffoli gate.

Discrete universal gate sets

Important universal gate set "**Clifford + T**" (for logical operations):

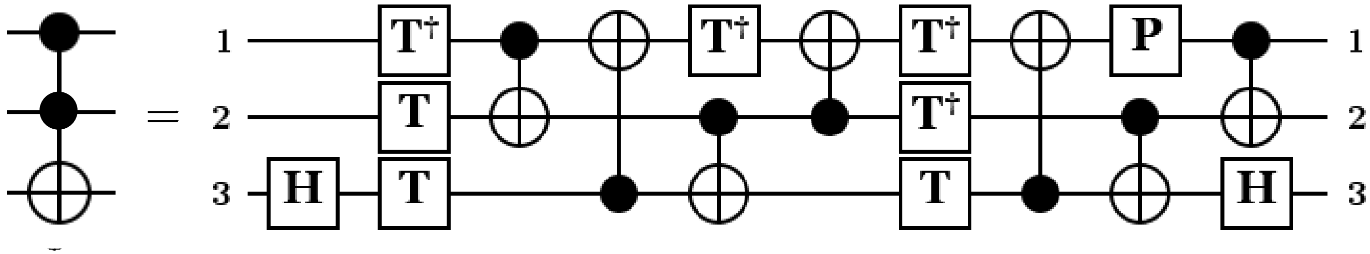
Consists of all Clifford operations (i.e., the group generated by $H_2, CNOT$ and $diag(1,i)$) and the "T gate" (T = $diag(1,\omega_8)$). Can be shown to be universal, i.e., for any unitary U and any given $\epsilon > 0$, there exists an element A in the Clifford+T group such that $|| U - A || \leq \epsilon$.

- This gate set arises naturally in the context of fault-tolerant quantum computing for several quantum codes, e.g., Steane code, surface code.
- T gate usually implemented via a process called "magic state distillation" which is very expensive. Much more expensive than Clifford gates.
- Common metrics used to measure resources:
  - T-count = total number of T gates used in a circuit
  - T-depth = number of T-layers when a circuit is written as C T C … T C
  - #qubits = total number of qubits used, including "ancillas" (=scratch space)

Typically, single-qubit rotations account for most of the cost!

Bounding resources: T gates

A useful factorization:



Lemma: If a unitary U can be implemented exactly over Clifford+T, then also Λ(U) can be implemented exactly. [arxiv.org:1206.0758]

This Lemma be used in some situations to avoid all errors due to single qubit approximations.

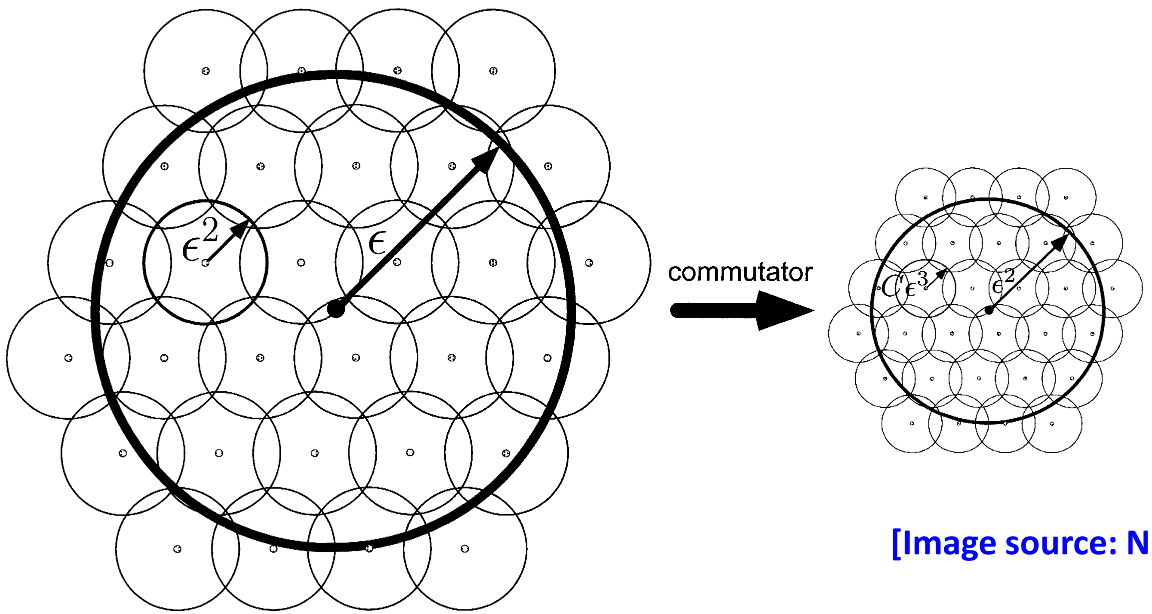
Cost of controlled unitaries:

- Tracking v=[#loc, #CNOT,#H, #P, #T]
- From U to Λ(U): matrix vector multiplication Mv.

$$M = \begin{bmatrix} 0 & 0 & 2 & 0 & 0 \\ 1 & 6 & 3 & 16 & 16 \\ 0 & 2 & 2 & 4 & 4 \\ 0 & 1 & 2 & 3 & 2 \\ 0 & 7 & 2 & 14 & 15 \end{bmatrix}$$

Solovay-Kitaev algorithm

Goal: Approximate unitaries by elements of dense subgroup $G \leq U(N)$
Basic idea: Successive refining of a "net" using commutators



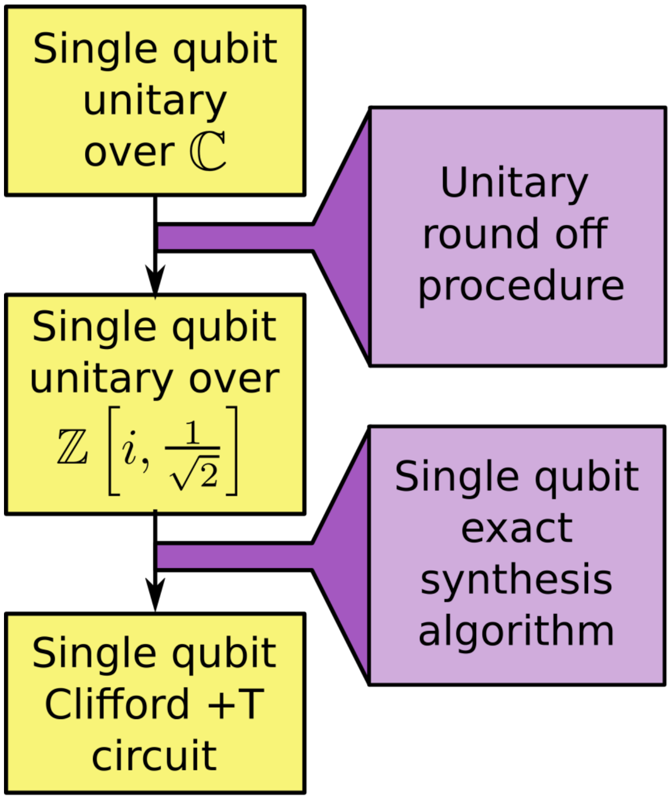
[Image source: Nielsen/Chuang, CUP 2000]

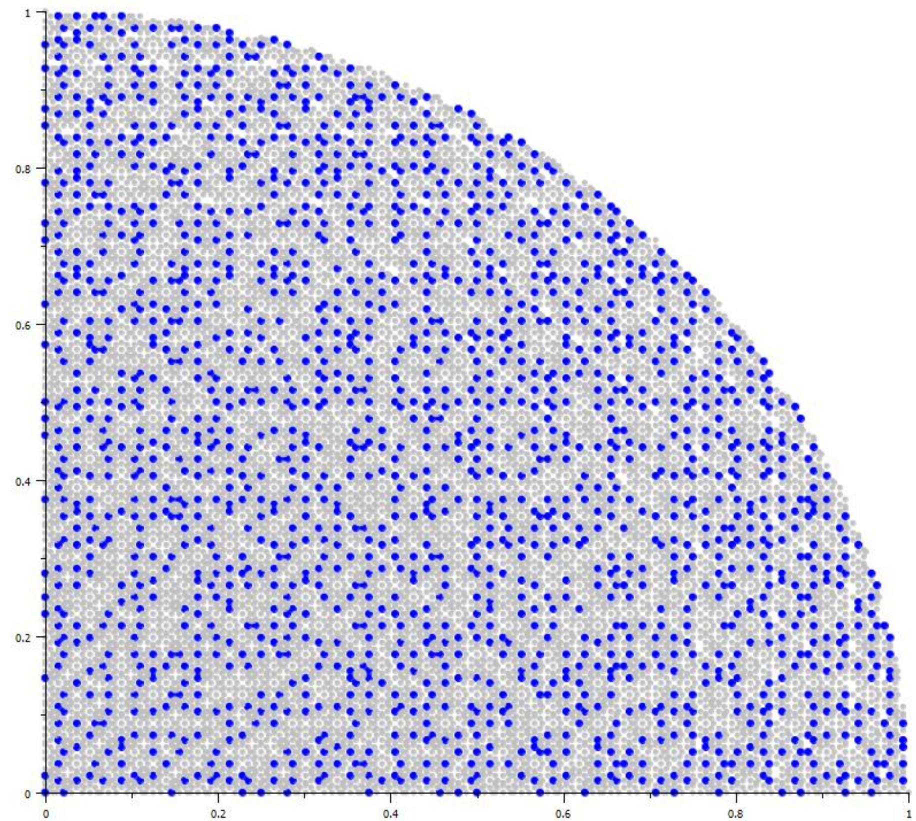
Implementations:

- **[Kitaev, Shen, Vyialyi, AMS 2002]:** $\log^{3+\delta}(1/\varepsilon)$ time, $\log^{3+\delta}(1/\varepsilon)$ length
- **[Dawson, Nielsen, quant-ph/0505030]:** $\log^{2.71}(1/\varepsilon)$ time, $\log^{3.97}(1/\varepsilon)$ length
- **[Harrow, Recht, Chuang, quant-ph/0111031]:** non-constructive, $\log(1/\varepsilon)$ length

Single qubit gates: synthesis methods

Basic idea: [Kliuchnikov/Maslov/Mosca 2012], [Selinger 2012]





Number of T gates required is
$O(\log(1/\varepsilon))$ vs $O\left(\log^{3+\delta}(1/\varepsilon)\right)$
(for the Solovay-Kitaev algorithm)

Shown are all unitaries in $\langle H, T\rangle$ that are obtainable from a simple round-off procedure and have T-count ≤ 12.

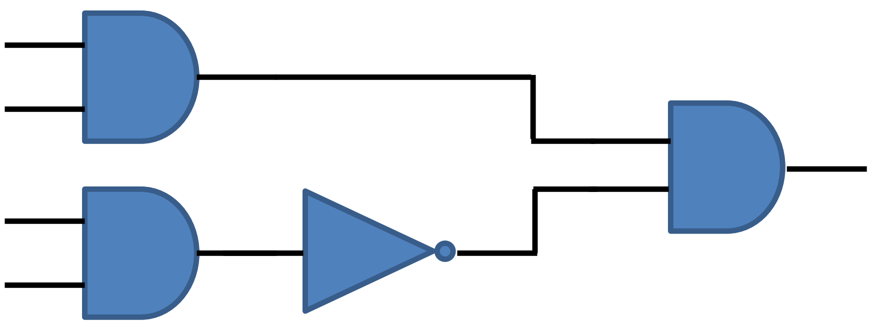
[Slide concept by V. Kliuchnikov]

Tools from the theory of reversible computing

Classical circuits

- Consider functions from n≥1 bits to m≥1 bits. We are interested in implementing functions by **combinational circuits**, i.e., circuits that do not make use of memory elements or feedback.
- Universal families of gates exist, i.e., sets of elementary gates from which any circuit can be built.

- We can compose gates together to make larger circuits.



- Problem for quantum computing: many gates are <u>not</u> reversible!

[Slide concept by M. Mosca, Waterloo]

How to invert an irreversible operation?

Reversible computation

Basic issue of reversible computing

Suppose, we want to compute a function $f : \{0,1\}^n \to \{0,1\}^m$ that is not reversible. How can we do this?

One possible solution

Define a new Boolean function which takes $n+m$ inputs and $n+m$ outputs as follows:

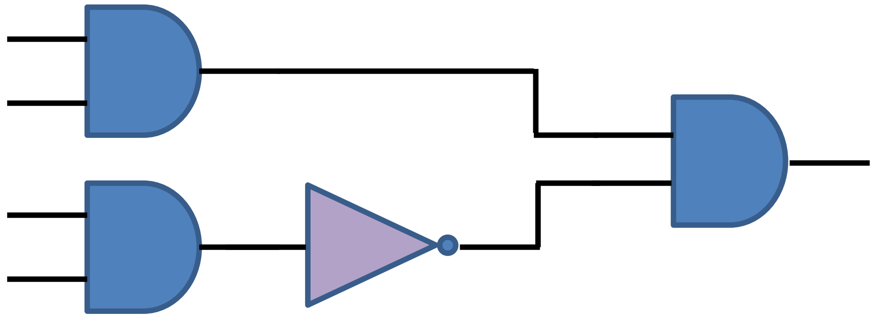
$$F(x,y) := (x, y \oplus f(x))$$

Properties of $F(x,y)$

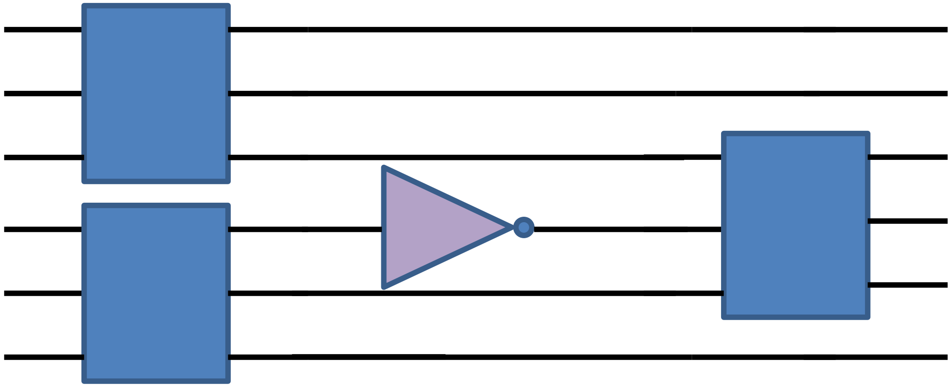
- On the special inputs $(x,0)$, where $x \in \{0,1\}^n$ we obtain that $F(x,0) = (x, f(x))$. Furthermore, F is reversible.
- Theorem (Bennett): If f can be computed using K gates, then F can be computed using $2K+m$ gates.

How to make circuits reversible?

Example:



Replace each gate with a reversible one:



[Slide concept by M. Mosca, Waterloo]

How to avoid garbage?

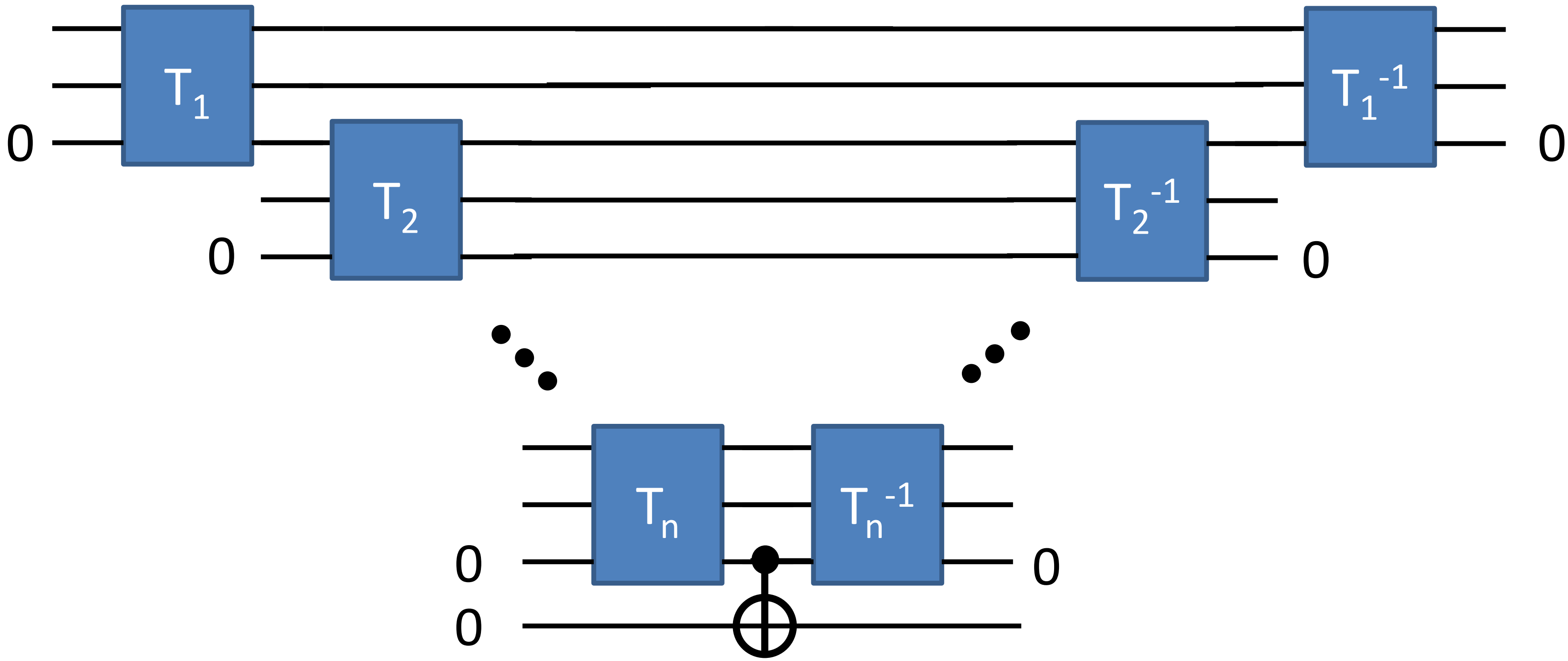
- Replacing each gate with a reversible one works fine, however, it produces “garbage”, i.e., help registers will be in a state different from 0 at the end.
- While this is fine for reversible computing, it is bad for quantum computing (it will prevent interference).
- There is a way out of this dilemma: the Bennett trick

$$
\begin{aligned}
|x\rangle\,|0\rangle\,|0\rangle\,|0\rangle &\mapsto |x\rangle\,|f(x)\rangle\,|garbage(x)\rangle\,|0\rangle \\
&\mapsto |x\rangle\,|f(x)\rangle\,|garbage(x)\rangle\,|f(x)\rangle \\
&\mapsto |x\rangle\,|0\rangle\,|0\rangle\,|f(x)\rangle
\end{aligned}
$$

Idea: compute forward, copy the result, “uncompute” the garbage by running the computation backwards.

Uncomputing the garbage

Replace each gate with a reversible one:

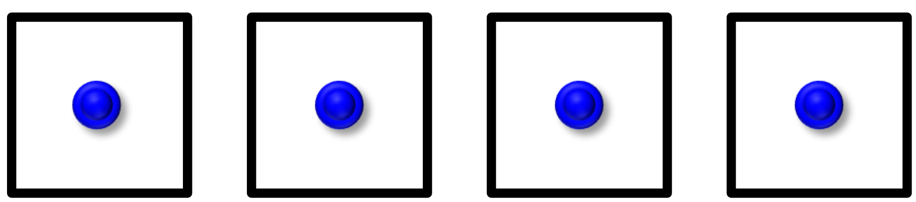


The pebble game

Rules of the game: **[Bennett, SIAM J. Comp., 1989]**

- n boxes, labeled i = 1, …, n
- in each move, either add or remove a pebble
- a pebble can be added or removed in i=1 at any time
- a pebble can be added of removed in i>1 if and only if there is a pebble in i-1.

Example:



1 2 3 4

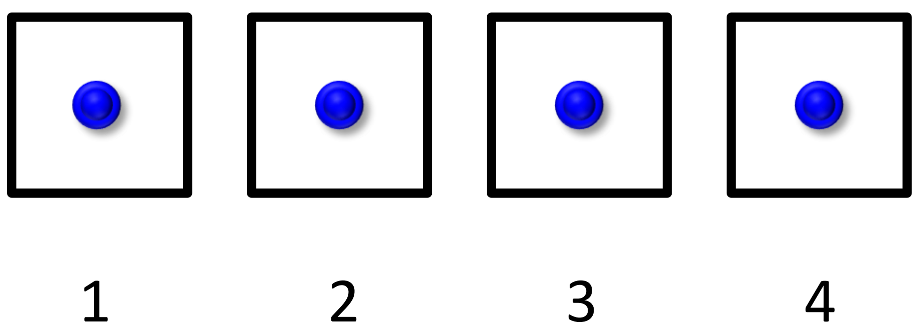
#	i
1	1
2	2
3	3
4	4
5	3
6	2
7	1

The pebble game

Imposing resource constraints:

- only a total of S pebbles are allowed
- corresponds to reversible algorithm with at most S ancilla qubits

Example: (n=3, S=3)



#	i
1	1
2	2
3	3
4	1
5	4
6	3
7	1
8	2
9	1

Optimal pebbling strategies

Definition: Let X be solution of pebble game. Let T(X) be # steps and Let S(X) be #pebbles. Define F(n,S) = min { T(X) : S(X) ≤ S }.

Table (small values of F):

$n \setminus S$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	∞	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
3	∞	∞	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
4	∞	∞	9	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
5	∞	∞	∞	11	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9
6	∞	∞	∞	15	13	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11
7	∞	∞	∞	19	17	15	13	13	13	13	13	13	13	13	13	13	13	13	13	13
8	∞	∞	∞	25	21	19	17	15	15	15	15	15	15	15	15	15	15	15	15	15
9	∞	∞	∞	∞	25	23	21	19	17	17	17	17	17	17	17	17	17	17	17	17
10	∞	∞	∞	∞	29	27	25	23	21	19	19	19	19	19	19	19	19	19	19	19
11	∞	∞	∞	∞	33	31	29	27	25	23	21	21	21	21	21	21	21	21	21	21
12	∞	∞	∞	∞	39	35	33	31	29	27	25	23	23	23	23	23	23	23	23	23
13	∞	∞	∞	∞	45	39	37	35	33	31	29	27	25	25	25	25	25	25	25	25
14	∞	∞	∞	∞	53	43	41	39	37	35	33	31	29	27	27	27	27	27	27	27
15	∞	∞	∞	∞	61	47	45	43	41	39	37	35	33	31	29	29	29	29	29	29
16	∞	∞	∞	∞	71	51	49	47	45	43	41	39	37	35	33	31	31	31	31	31
17	∞	∞	∞	∞	∞	57	53	51	49	47	45	43	41	39	37	35	33	33	33	33
18	∞	∞	∞	∞	∞	63	57	55	53	51	49	47	45	43	41	39	37	35	35	35
19	∞	∞	∞	∞	∞	69	61	59	57	55	53	51	49	47	45	43	41	39	37	37
20	∞	∞	∞	∞	∞	77	65	63	61	59	57	55	53	51	49	47	45	43	41	39
21	∞	∞	∞	∞	∞	85	69	67	65	63	61	59	57	55	53	51	49	47	45	43
22	∞	∞	∞	∞	∞	93	73	71	69	67	65	63	61	59	57	55	53	51	49	47
23	∞	∞	∞	∞	∞	101	79	75	73	71	69	67	65	63	61	59	57	55	53	51
24	∞	∞	∞	∞	∞	109	85	79	77	75	73	71	69	67	65	63	61	59	57	55

[E.Knill, arxiv:math/9508218]

Time-space tradeoffs

Let A be an algorithm with time complexity T and space complexity S.

- Using reversible pebble game, **[Bennett, SIAM J. Comp. 1989]** showed that for any ε>0 there is a reversible algorithm A' with time complexity $O(T^{1+\varepsilon})$ and space complexity O(S ln(T)).
- Issue: one cannot simply take the limit ε→0. The space would grow in an unbounded way (as $O(\varepsilon 2^{1/\varepsilon} S \ln(T))$).
- Improved analysis **[Levine, Sherman, SIAM J. Comp. 1990]** showed that for any ε>0 there is a reversible algorithm A' with time complexity $O(T^{1+\varepsilon}/S^{\varepsilon})$ and space complexity O(S (1+ln(T/S))).
- Other time/space tradeoffs: **[Buhrman, Tromp, Vitányi, ICALP'01]**

Research topic: develop a "compiler" that takes a classical combinational circuit as input and translates it into a reversible circuit, with respect to various resource constraints.

Shor

Reducing factoring to period finding

- **Modular exponentiation:** Let N be an integer and let a be in Z_N. Modular exponentiation is the map $f(x) := a^x \bmod N$.

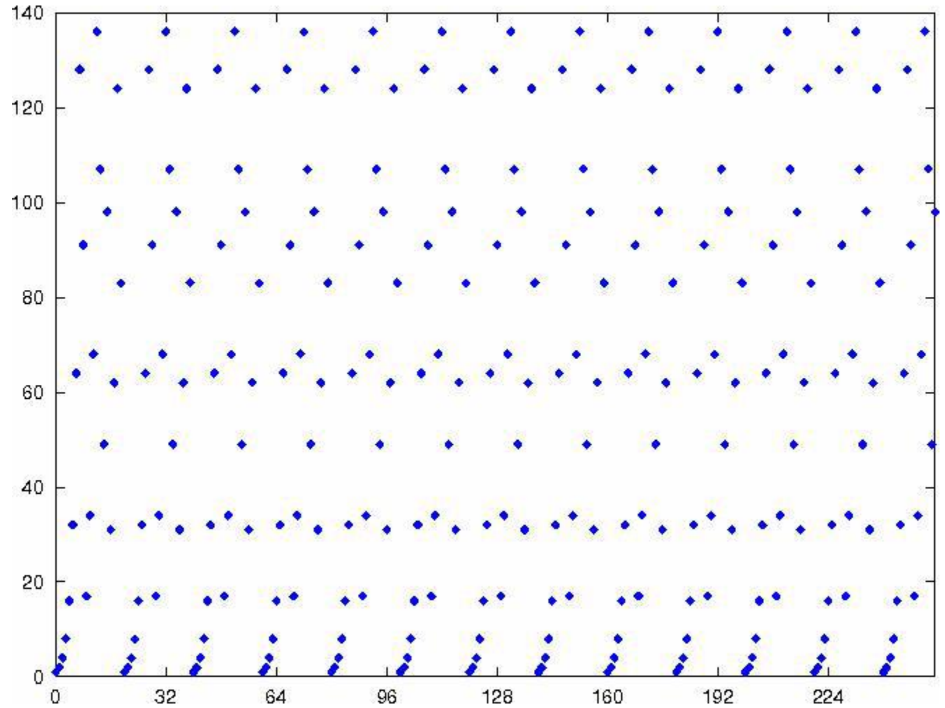
- **Fact:** The map f can be implemented in O(poly(log N)) ops.
- **Fact:** It can be shown that it can also be implemented efficiently on a quantum computer.
- **More facts:**
  - Recall that the order of a is defined as the smallest integer r such that $a^r = 1 \bmod N$.
  - The function $f(x) := a^x \bmod N$ is periodic with period r equal to the order of a, i. e., $f(x) = f(x + r)$ for all x.
  - The problem of factoring N can be reduced to period finding for modular exponentiation f (for random a).

Setting up a periodic state

- **Observation:** The function f(x) = a^x mod N is periodic and has period length r, i. e., f (x) = f (x + r) for all inputs x.

- **Example:** graph of the function f (x) = 2x mod 165:

$$|\, y = f(x) \rangle$$



$$|\, x \rangle$$

Shor's algorithm for period finding

Computing the modular exponentiation

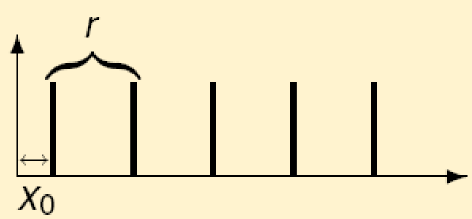
Let $f(x) = a^x \bmod N$ be modular exponentiation, let $M \gg N$, and compute:

$$|0\rangle\,|0\rangle \mapsto \frac{1}{\sqrt{M}} \sum_{x \in Z_M} |x\rangle\,|0\rangle \overset{f}{\mapsto} \frac{1}{\sqrt{M}} \sum_{x \in Z_M} |x\rangle\,|f(x)\rangle\,.$$

Collapsing this state

Now, measuring the second register will yield a random $s \in Z_N$ in the image of f. The state collapses to (suppose that $r|M$)

$$\frac{1}{\sqrt{M/r}} \sum_{k=0}^{M/r-1} |x_0 + k \cdot r\rangle$$



This is an example of a coset state!

Period finding using coset states

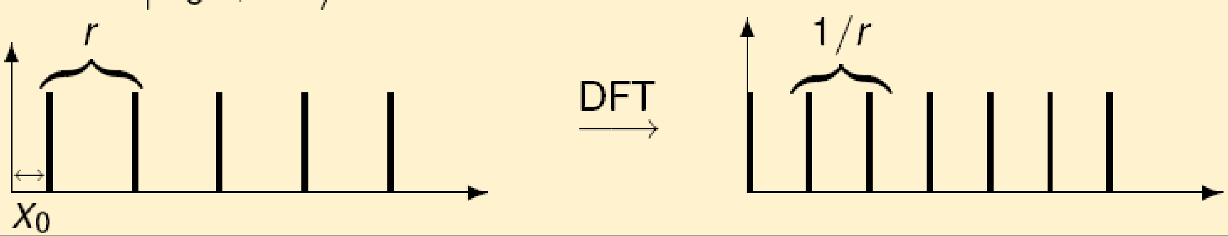
Coset state for the cyclic group

Let $G = Z_M$, $x_0 \in G$, $H = \langle r \rangle$, where r is the order of a. Then:

$$|x_0 + H\rangle = \frac{1}{\sqrt{M/r}} \sum_{k=0}^{M/r-1} |x_0 + k \cdot r\rangle$$

Period finding (Shor'94)

Coset state $|x_0 + H\rangle$ and its Fourier transform:



Coset states in the abelian case

We can compute H efficiently from coset states!

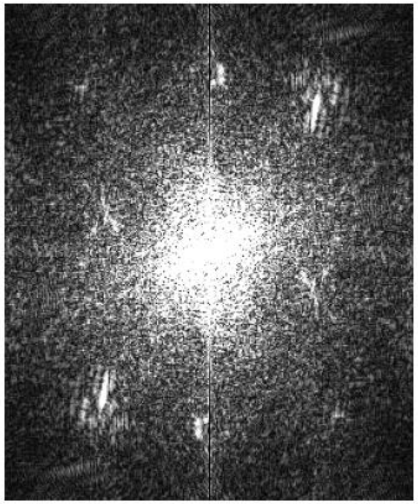
Discrete Fourier Transforms

Definition:

$$\mathsf{DFT}_N := \frac{1}{\sqrt{N}} \left[\omega_N^{k \cdot \ell} \right]_{k,\ell=0\ldots N-1}, \quad \omega_N = e^{2\pi i/N}$$

Example:

DFT



Discrete Fourier Transform (DFT/QFT)

Definition: $\mathsf{DFT}_N = \frac{1}{\sqrt{N}} \left[\, \omega_N^{k \cdot \ell} \, \right]_{k,\ell=0\ldots N-1}, \quad \omega_N = e^{2\pi i/N}$

Cooley-Tukey FFT:

$$\mathsf{DFT}_4 \quad = \quad \Pi_{rev} \quad \cdot (\mathbf{1}_2 \otimes \mathsf{DFT}_2) \cdot \mathrm{diag}(1,1,1,i) \cdot \quad (\mathsf{DFT}_2 \otimes \mathbf{1}_2)$$

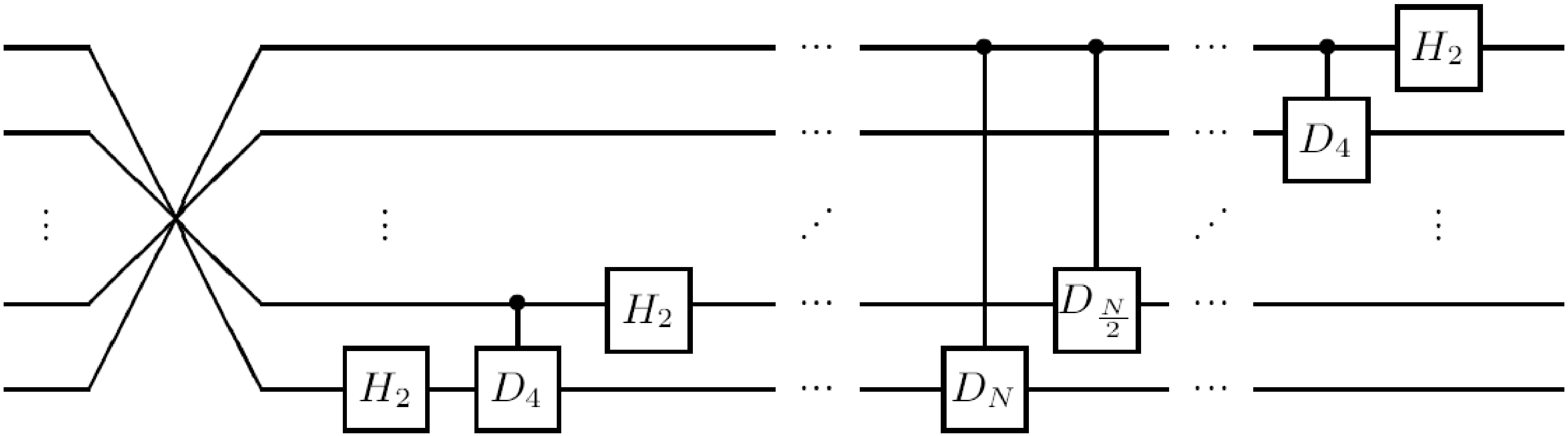
$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{bmatrix} = \begin{bmatrix} 1 & & & \\ & & 1 & \\ & 1 & & \\ & & & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & & \\ 1 & -1 & & \\ & & 1 & 1 \\ & & 1 & -1 \end{bmatrix} \cdot \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & i \end{bmatrix} \cdot \begin{bmatrix} 1 & & 1 & \\ & 1 & & 1 \\ 1 & & -1 & \\ & 1 & & -1 \end{bmatrix}$$

Theorem: Multiplication with DFT_N can be performed classically in $O(N \log N)$ elementary operations.

We can do much better on a quantum computer!

Quantum Fast Fourier Transform

Quantum circuit for DFT_N



Cost:

Classical Computer	Quantum Computer
$T(N) = 2\,T(N/2) + O(N)$	$T(N) = T(N/2) + O(\log N))$
$T(N) = O(N \log N)$	$T(N) = O(\log^2 N)$

The Hidden Subgroup Problem

Definition of the problem

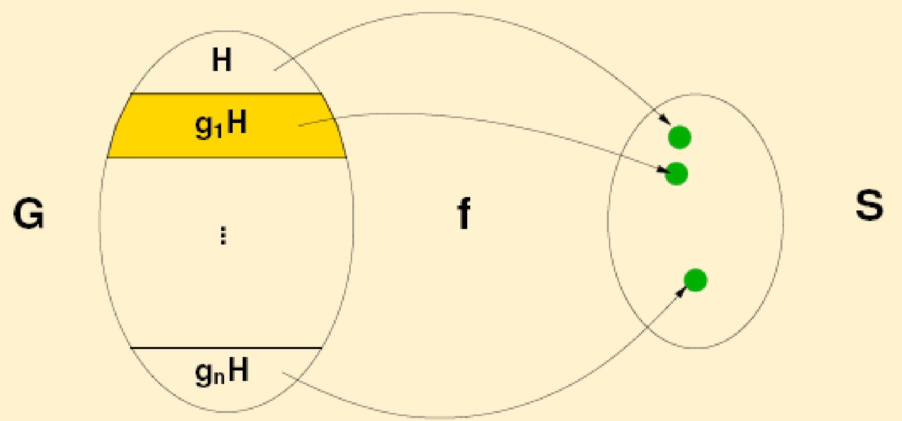
Given: Group G, set S, map $f : G \to S$ given as black box

Promise: There exists subgroup $H \leq G$ with

- f constant on each coset of H
- $g_1 H \neq g_2 H$ implies $f(g_1) \neq f(g_2)$

Problem: Find generators for H (input size: $\log |G|$)

Visualization of the cosets of H in G



Caveat

Difficulty of HSP depends crucially on the structure of the group G.

Shor's algorithm for dlogs:

Step 1: Create $\sum_{k\in\{0,1\}^n}|k_1,\ldots,k_n\rangle \otimes \sum_{\ell\in\{0,1\}^n}|\ell_1,\ldots,\ell_n\rangle \otimes |\mathcal{O}\,\rangle$ by applying Hadamard gates to 2 registers of n qubits; $n = \lceil\log(ord_P)\rceil$

Step 2: For fixed generator P and fixed target $Q \in \langle P\rangle$ compute the transformation that maps this state to

$$\sum_{k\in\{0,1\}^n}|k\rangle \otimes \sum_{\ell\in\{0,1\}^n}|\ell\rangle \otimes |kP+\ell Q\rangle$$

Step 3: Measure the 3rd register. Obtain a result R. Letting $Q = \alpha P$ and $R = \beta P$, we obtain a state corresponding to a "line"

$$\sum_{\substack{k,\ell\in\{0,1\}^n:\\ k+\alpha\ell=\beta}}|k\rangle \otimes |\ell\rangle \otimes |R\rangle = \sum_{\ell\in\{0,1\}^n}|\beta-\alpha\ell\rangle \otimes |\ell\rangle$$

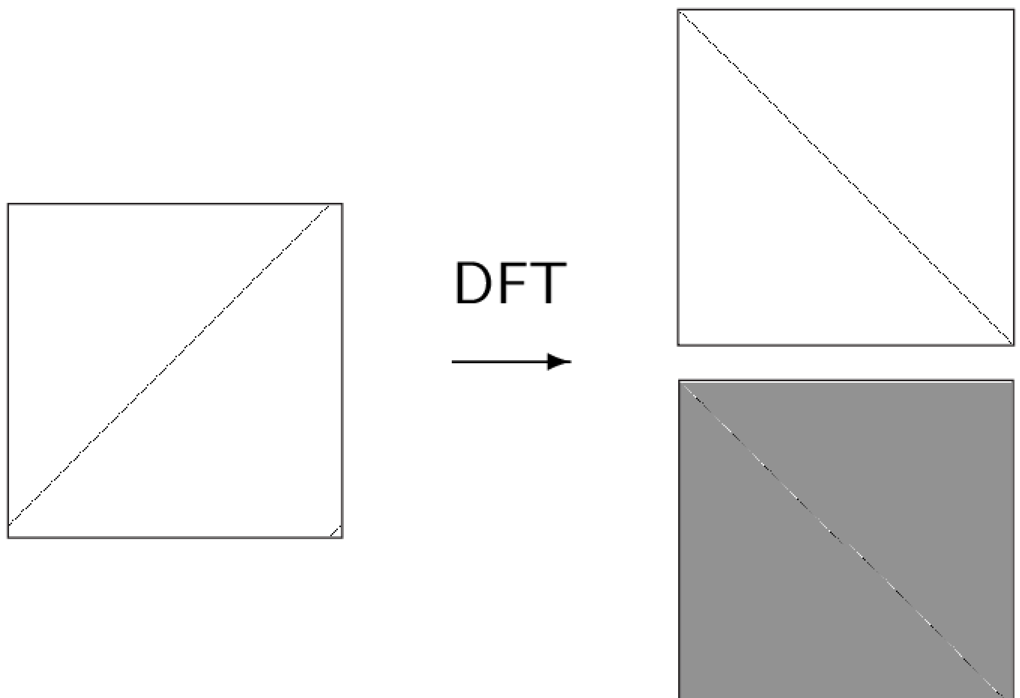
Step 4: Apply $QFT \otimes QFT$ and measure to sample from the line $\{(x,\alpha x), x \in \{0,..,2^n-1\}$. If x is a unit, we obtain α.

Visualizing Fourier duality

Abelian groups:

$$\mathsf{DFT}_A\Big(\frac{1}{\sqrt{|U|}}\sum_{\mathbf{x}\in U+c}|\mathbf{x}\rangle\Big) = \frac{1}{\sqrt{|U^{\perp}|}}\sum_{\mathbf{y}\in U^{\perp}}\varphi_{c,y}\,|\mathbf{y}\rangle$$

Shifted line

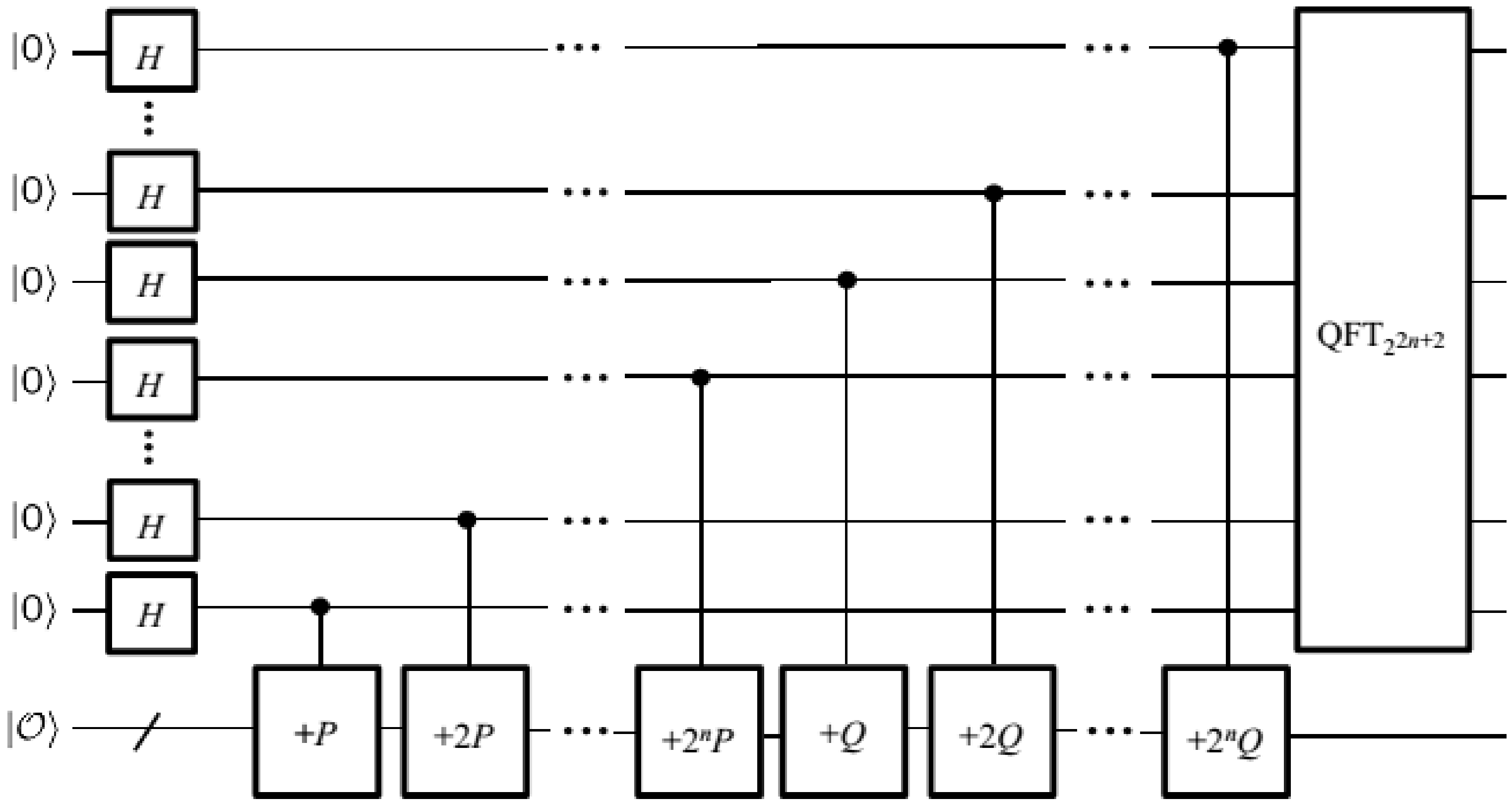


Amplitude

Phase

Circuit for Shor's dlog algorithm

Phase estimation circuit layout:



Simple circuit optimizations

Double & Add

Input: binary string $(x_{n-1}, x_{n-2}, \dots, x_1, x_0)$

Output: $x = \sum_i x_i 2^i = \mathrm{x}_0 + 2(\mathrm{x}_1 + 2(\mathrm{x}_2 + \ \dots))$

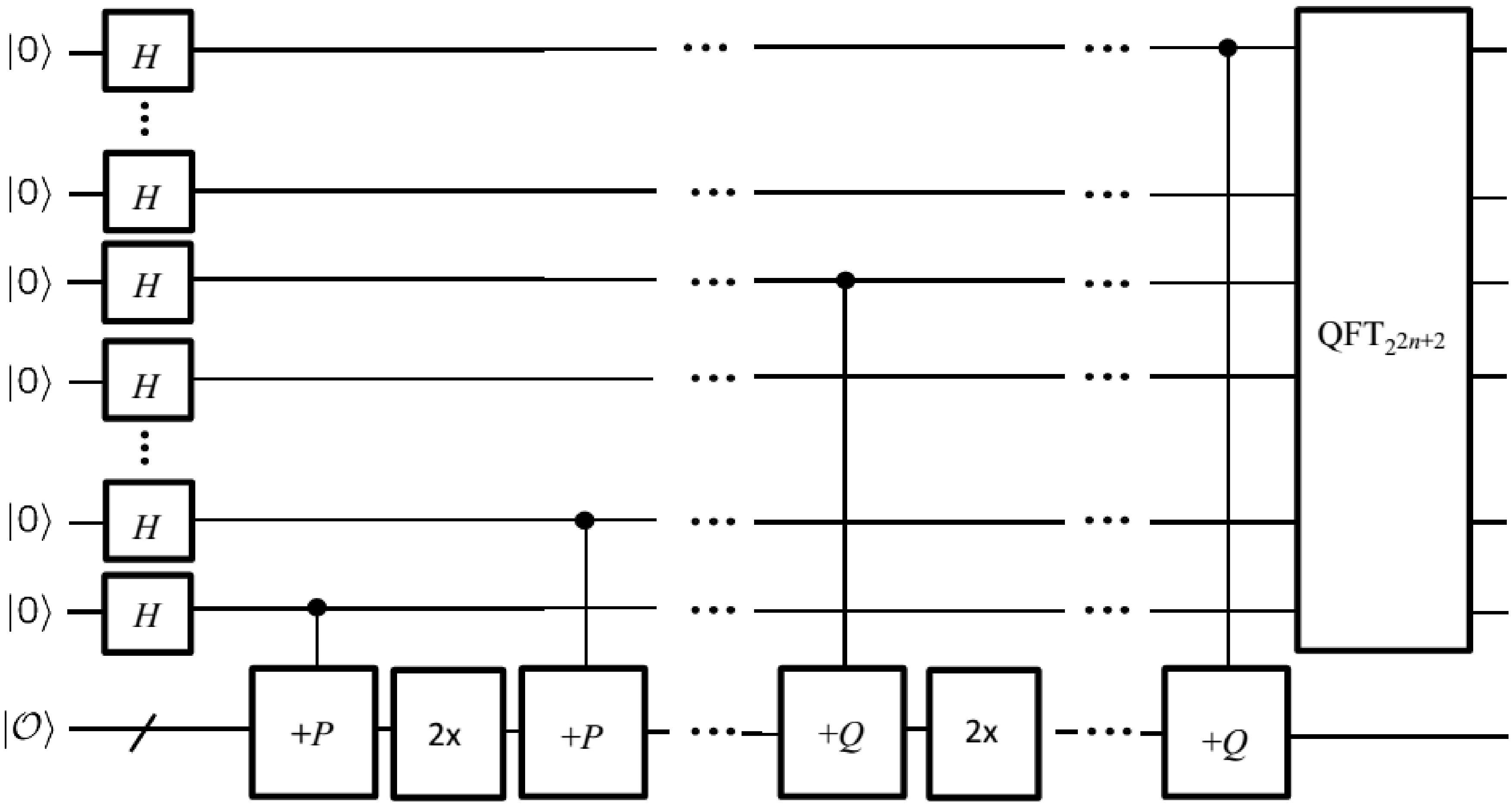
<u>Method 1 ("evaluate left-to-right")</u>

x $\leftarrow x_0$
for i = 1 … n − 1 do
  x $\leftarrow x + 2^i x_i$
end for
return x

<u>Method 2 ("evaluate right-to-left")</u>

x $\leftarrow x_{n-1}$
for i = n − 2 … 1 do
  x $\leftarrow 2x + x_i$
end for
return x

Rewriting the ECC dlog circuit



Rewriting the ECC dlog circuit

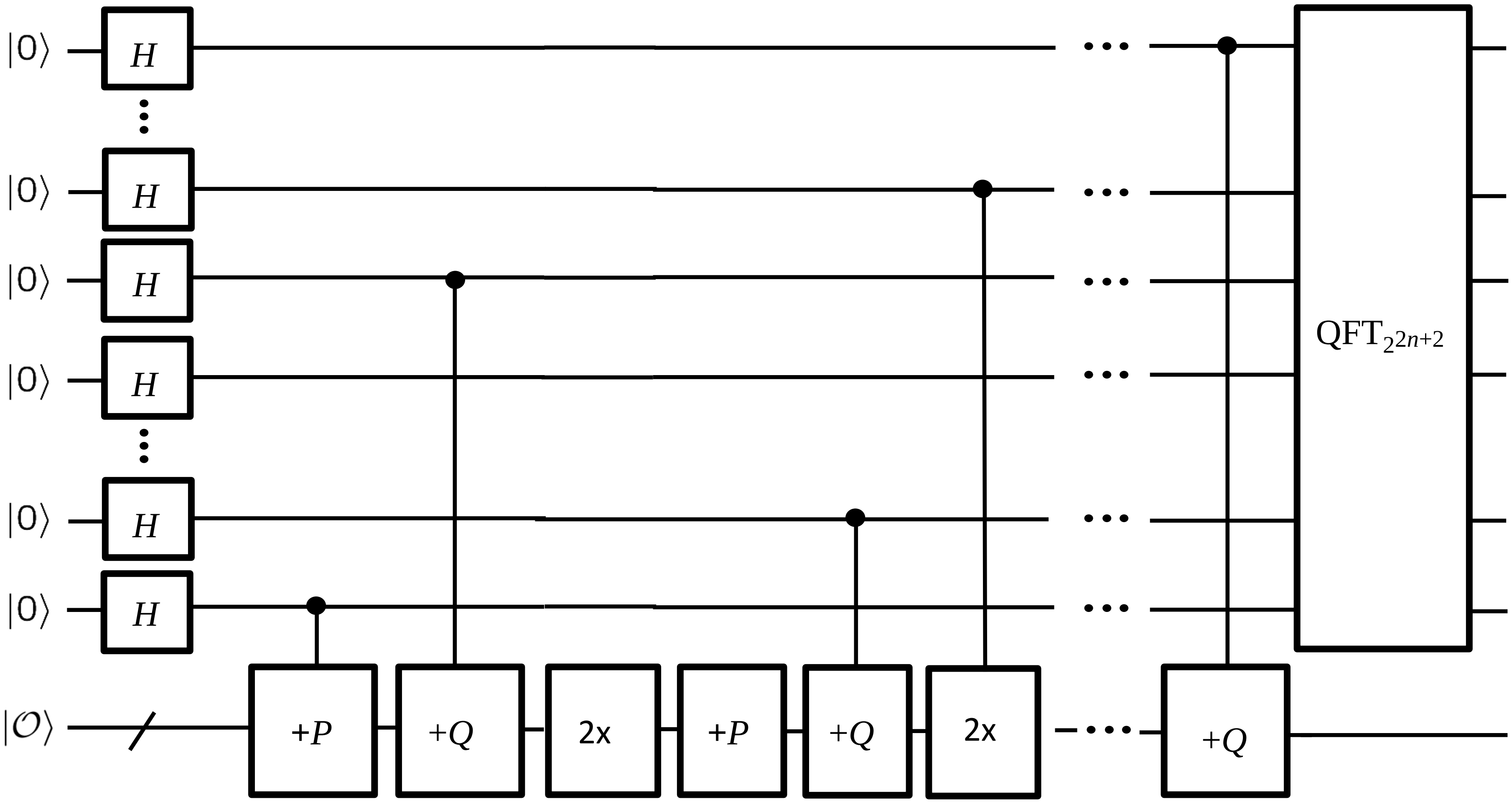
Improvement 2: use Shamir's trick to combine double& add for P and Q

Double & Add: Shamir's Trick

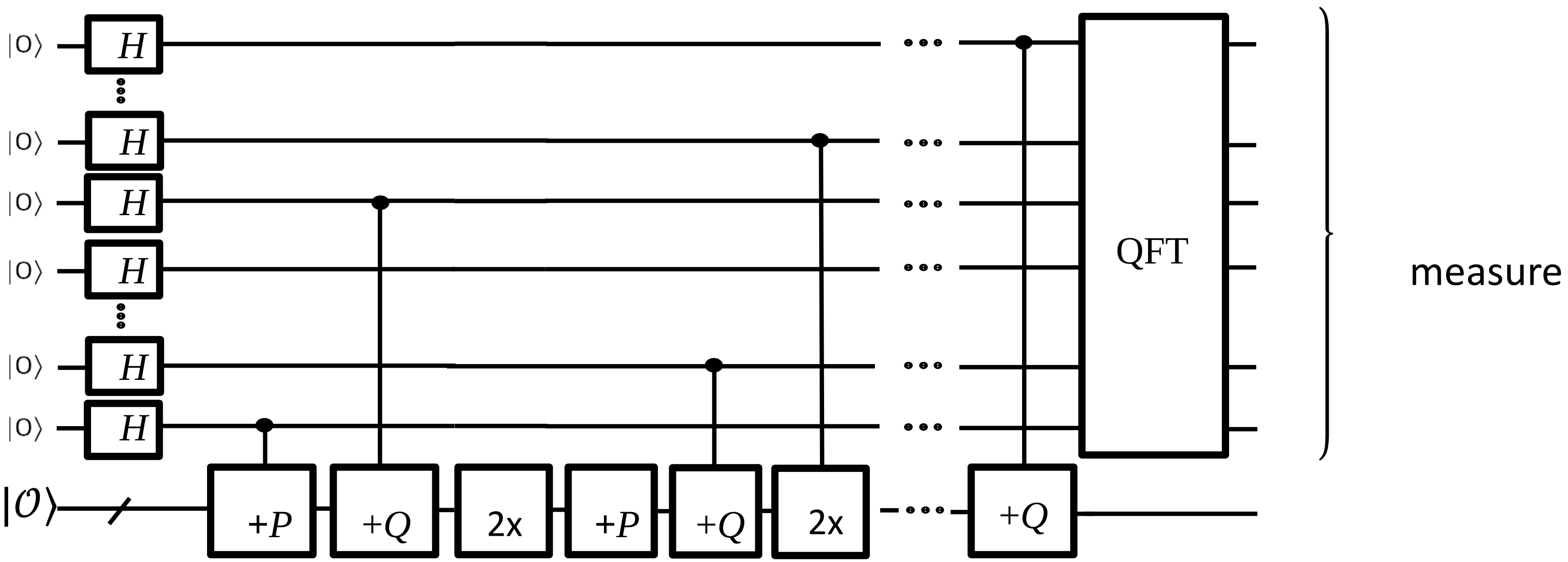
$R \leftarrow \mathcal{O}$ # initialize result to identity
if $k_n = 1$ then $R \leftarrow R + P$ # adjust starting value based on most significant bit
if $\ell_n = 1$ then $R \leftarrow R + Q$
for $i = n - 1$ to 0 step -1
    $R \leftarrow 2 \cdot R$
    if $k_i = 1$ then $R \leftarrow R + P$
    if $\ell_i = 1$ then $R \leftarrow R + Q$
return R # $R = kP + \ell Q$

Saves 50% of the doublers

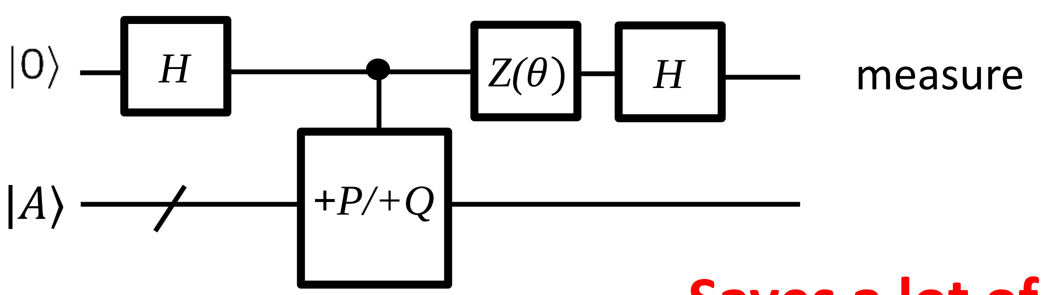
More rewriting: Shamir's trick



Semi-classical QFT



Equivalent protocol:



Saves a lot of qubits!

Example: ECC point addition

Consider elliptic curve in short Weierstrass form over $GF(2^m)$

$$y^2 + xy = x^3 + a_2 x^2 + a_6$$

Adding 2 projective points $P_1 = (\mathrm{X}_1, \mathrm{Y}_1, \mathrm{Z}_1)$ and $P_2 = (\mathrm{X}_2, \mathrm{Y}_2, \mathrm{Z}_2)$ can be done with 12 $GF(2^m)$-mults—of which 9 are generic—7 $GF(2^m)$-adds, and 1 squaring *(madd-2008-bl):*

$$\begin{array}{lllllllll} A & = & Y_1 + Z_1 \cdot y_2, & B & = & X_1 + Z_1 \cdot x_2, & AB & = & A + B, \\ C & = & B^2, & E & = & B \cdot C, & F & = & (A \cdot AB + a_2 \cdot C) \cdot Z_1 + E, \\ \hline X_3 & = & B \cdot F, & & & & & & \\ Y_3 & = & C \cdot (A \cdot X_1 + B \cdot Y_1) + AB \cdot F, & & & & & & \\ Z_3 & = & E \cdot Z_1. & & & & & & \end{array}$$

[Bernstein, Lange: http://www.hyperelliptic.org/EFD/]

Complete binary Edwards curves

[Bernstein, Lange, Farashahi, 2008]: For n≥3 each ordinary binary elliptic curve is birationally equivalent to a complete binary Edwards curve: (d_1, $d_2 \in GF(2^n)$ with $Tr(d_2)=1$).

$$d_1(x+y) + d_2(x^2+y^2) = xy + xy(x+y) + x^2y^2$$

Point addition / group law:

$$x_3 = \frac{d_1(x_1+x_2) + d_2(x_1+y_1)(x_2+y_2) + (x_1+x_1^2)(x_2(y_1+y_2+1)+y_1y_2)}{d_1 + (x_1+x_1^2)(x_2+y_2)} \text{ and}$$

$$y_3 = \frac{d_1(y_1+y_2) + d_2(x_1+y_1)(x_2+y_2) + (y_1+y_1^2)(y_2(x_1+x_2+1)+x_1x_2)}{d_1 + (y_1+y_1^2)(x_2+y_2)},$$

- no projective closure needed
- one formula to implement group law for all points
- identity: (0,0)

Complete binary Edwards curves

Consider complete binary Edwards curve:

$$d_1(x+y)+d_2(x^2+y^2)=xy+xy(x+y)+x^2y^2$$

- One can work projectively to avoid inversions.
- Adding projective points $P_1=(\mathrm{X}_1,\mathrm{Y}_1,\mathrm{Z}_1)$ and $P_2=(\mathrm{X}_2,\mathrm{Y}_2,\mathrm{Z}_2)$ can be done with 21 $GF(2^m)$-mults—of which 17 are generic—15 $GF(2^m)$-adds, and 1 squaring:

$$\begin{array}{rclrclrclrcl}
W_1 & = & X_1+Y_1, & W_2 & = & X_2+Y_2, & A & = & X_1\cdot(X_1+Z_1), & B & = & Y_1\cdot(Y_1+Z_1),\\
C & = & Z_1\cdot Z_2, & D & = & W_2\cdot Z_2, & E & = & d_1C^2, & H & = & (d_1Z_2+d_2W_2)\cdot W_1\cdot C,\\
I & = & d_1Z_1\cdot C, & U & = & E+A\cdot D, & V & = & E+B\cdot D, & S & = & U\cdot V,\\
X_3 & = & \multicolumn{10}{l}{S\cdot Y_1+(H+X_2\cdot(I+A\cdot(Y_2+Z_2)))\cdot V\cdot Z_1,}\\
Y_3 & = & \multicolumn{10}{l}{S\cdot X_1+(H+Y_2\cdot(I+B\cdot(X_2+Z_2)))\cdot U\cdot Z_1,}\\
Z_3 & = & \multicolumn{10}{l}{S\cdot Z_1.}
\end{array}$$

Example: higher genus

Algorithm 1 Projective doubling for general divisors on the Jacobian of $C : y^2 = x^5 + f_3x^3 + f_2x^2 + f_1 + f_0$.

Input: $P = (U_1 : U_0 : V_1 : V_0 : Z)$ and f_2, f_3 (curve constants)
Output: $[2]P = (U_1'' : U_0'' : V_1'' : V_0'' : Z'')$.

$$
\begin{array}{ll}
U_0'' \leftarrow U_0 \cdot Z, & t_1 \leftarrow Z^2, \\
t_2 \leftarrow U_1^2, & t_3 \leftarrow 2 \cdot t_2, \\
t_4 \leftarrow 2 \cdot U_0'', & t_5 \leftarrow t_3 + t_4, \\
t_5 \leftarrow t_5 \cdot U_1, & t_6 \leftarrow V_1^2, \\
t_7 \leftarrow f_2 \cdot t_1, & t_6 \leftarrow t_7 - t_6, \\
t_6 \leftarrow t_6 \cdot Z, & t_6 \leftarrow t_6 + t_5, \\
t_1 \leftarrow f_3 \cdot t_1, & t_1 \leftarrow t_1 + t_2, \\
t_4 \leftarrow t_1 - t_4, & t_4 \leftarrow t_4 + t_3, \\
V_0'' \leftarrow V_0 \cdot Z, & t_1 \leftarrow U_1 \cdot V_1, \\
t_2 \leftarrow 2 \cdot t_1, & t_1 \leftarrow t_1 + V_0'', \\
t_2 \leftarrow t_2 - V_0'', & t_3 \leftarrow t_3 + U_0'', \\
t_3 \leftarrow V_1 \cdot t_3, & t_5 \leftarrow t_3 \cdot t_4, \\
t_7 \leftarrow t_6 \cdot t_2, & t_5 \leftarrow t_5 - t_7, \\
t_6 \leftarrow t_6 \cdot V_1, & t_4 \leftarrow t_4 \cdot t_1, \\
t_4 \leftarrow t_4 - t_6, & t_3 \leftarrow t_3 \cdot V_1, \\
t_1 \leftarrow t_1 \cdot t_2, & t_3 \leftarrow t_3 - t_1, \\
t_1 \leftarrow t_5 \cdot t_4, & t_2 \leftarrow t_3 \cdot t_4, \\
t_4 \leftarrow t_4^2, & t_6 \leftarrow U_0'' \cdot t_4, \\
t_7 \leftarrow t_4 \cdot Z, & t_4 \leftarrow t_4 \cdot U_1, \\
t_3 \leftarrow 2 \cdot t_3, & t_3 \leftarrow t_3^2, \\
t_3 \leftarrow t_3 \cdot Z, & t_2 \leftarrow 2 \cdot t_2, \\
U_0'' \leftarrow t_2 \cdot Z, & V_1'' \leftarrow V_1 \cdot U_0'', \\
V_0'' \leftarrow V_0'' \cdot t_2, & t_2 \leftarrow t_1 - t_4, \\
t_5 \leftarrow t_5^2, & t_8 \leftarrow 2 \cdot t_3, \\
t_8 \leftarrow t_8 - t_2, & t_8 \leftarrow t_8 - t_1, \\
t_8 \leftarrow t_8 \cdot U_1, & t_8 \leftarrow t_8 + t_5, \\
t_5 \leftarrow 2 \cdot V_1'', & t_8 \leftarrow t_8 + t_5, \\
V_1'' \leftarrow t_6 + V_1'', & t_6 \leftarrow t_6 \cdot t_2, \\
U_1'' \leftarrow 2 \cdot t_2, & U_1'' \leftarrow U_1'' - t_3, \\
t_2 \leftarrow U_1'' - t_2, & t_4 \leftarrow t_4 - U_1'', \\
t_4 \leftarrow t_4 \cdot t_2, & t_4 \leftarrow t_4 \cdot Z, \\
Z'' \leftarrow U_0'' \cdot Z, & t_1 \leftarrow t_1 - U_1'', \\
U_1'' \leftarrow U_1'' \cdot Z'', & U_0'' \leftarrow t_8 \cdot U_0'', \\
V_1'' \leftarrow V_1'' - t_8, & V_1'' \leftarrow V_1'' \cdot t_7, \\
V_1'' \leftarrow t_4 - V_1'', & V_0'' \leftarrow V_0'' \cdot t_7, \\
t_1 \leftarrow t_1 \cdot t_8, & t_1 \leftarrow t_1 - t_6, \\
V_0'' \leftarrow t_1 - V_0'', & Z'' \leftarrow Z'' \cdot t_7
\end{array}
$$

Algorithm 2 Projective addition between general divisors on the Jacobian of $C : y^2 = x^5 + f_3x^3 + f_2x^2 + f_1 + f_0$.

Input: $P = (U_1 : U_0 : V_1 : V_0 : Z)$, $Q = (U_1' : U_0' : V_1' : V_0' : Z')$.
Output: $P + Q = (U_1'' : U_0'' : V_1'' : V_0'' : Z'')$.

$$
\begin{array}{ll}
U_1'' \leftarrow U_1 \cdot Z', & U_0'' \leftarrow U_0 \cdot Z', \\
t_1 \leftarrow V_0 \cdot Z', & t_2 \leftarrow V_0' \cdot Z, \\
t_1 \leftarrow t_1 - t_2, & t_2 \leftarrow U_0' \cdot Z, \\
t_3 \leftarrow U_1' \cdot Z, & t_4 \leftarrow t_3 \cdot t_2, \\
t_2 \leftarrow t_2 - U_0'', & t_5 \leftarrow U_1'' - t_3, \\
t_6 \leftarrow U_1'' \cdot U_0'', & t_4 \leftarrow t_4 - t_6, \\
t_6 \leftarrow V_1' \cdot Z, & Z'' \leftarrow Z \cdot Z', \\
t_7 \leftarrow V_1 \cdot Z', & t_8 \leftarrow t_7 - t_6, \\
t_6 \leftarrow t_7 + t_6, & t_9 \leftarrow U_1''^2, \\
t_{10} \leftarrow Z'' \cdot t_2, & t_{10} \leftarrow t_9 + t_{10}, \\
t_{11} \leftarrow t_3^2, & t_3 \leftarrow U_1'' + t_3, \\
t_{12} \leftarrow t_{10} - t_{11}, & t_{11} \leftarrow t_9 + t_{11}, \\
t_9 \leftarrow t_4 \cdot t_8, & t_4 \leftarrow t_4 \cdot t_5, \\
t_5 \leftarrow t_1 \cdot t_5, & t_1 \leftarrow t_1 \cdot t_{12}, \\
t_8 \leftarrow t_2 \cdot t_8, & t_2 \leftarrow t_2 \cdot t_{12}, \\
t_1 \leftarrow t_9 + t_1, & t_5 \leftarrow t_5 + t_8, \\
t_2 \leftarrow t_2 - t_4, & t_4 \leftarrow t_5 \cdot Z'', \\
t_8 \leftarrow t_2 \cdot t_4, & t_2 \leftarrow t_2^2, \\
t_5 \leftarrow t_5 \cdot t_4, & t_4 \leftarrow t_1 \cdot t_4, \\
U_1'' \leftarrow U_1'' \cdot t_5, & t_9 \leftarrow 2 \cdot t_4, \\
t_9 \leftarrow t_9 - t_2, & t_{12} \leftarrow t_5 \cdot t_3, \\
t_9 \leftarrow t_9 - t_{12}, & t_2 \leftarrow t_9 - t_2, \\
t_2 \leftarrow t_2 \cdot t_3, & t_{11} \leftarrow t_5 \cdot t_{11}, \\
t_2 \leftarrow t_2 + t_{11}, & t_2 \leftarrow t_2/2, \\
t_{12} \leftarrow Z'' \cdot t_5, & U_0'' \leftarrow U_0'' \cdot t_{12}, \\
t_{12} \leftarrow t_8 \cdot t_{12}, & t_{11} \leftarrow Z' \cdot t_{12}, \\
V_0'' \leftarrow t_{11} \cdot V_0, & V_1'' \leftarrow t_{11} \cdot V_1, \\
t_{11} \leftarrow t_4 - t_9, & t_4 \leftarrow U_1'' - t_4, \\
t_1 \leftarrow t_1^2, & t_6 \leftarrow t_8 \cdot t_6, \\
t_1 \leftarrow t_1 \cdot Z'', & t_1 \leftarrow t_1 + t_6, \\
t_1 \leftarrow t_1 - t_2, & t_2 \leftarrow t_1 - U_0'', \\
t_5 \leftarrow t_2 \cdot t_5, & t_2 \leftarrow t_9 \cdot t_{11}, \\
t_{11} \leftarrow t_1 \cdot t_{11}, & t_6 \leftarrow U_1'' \cdot t_4, \\
t_6 \leftarrow t_6 + t_2, & t_5 \leftarrow t_6 + t_5, \\
t_4 \leftarrow U_0'' \cdot t_4, & t_{11} \leftarrow t_4 + t_{11}, \\
t_9 \leftarrow t_9 \cdot t_8, & U_1'' \leftarrow t_9 \cdot Z'', \\
U_0'' \leftarrow t_1 \cdot t_8, & t_5 \leftarrow t_5 \cdot Z'', \\
V_1'' \leftarrow t_5 - V_1'', & V_0'' \leftarrow t_{11} - V_0'', \\
Z'' \leftarrow Z'' \cdot t_{12} &
\end{array}
$$

Algorithm 3 Mixed addition between general divisors on the Jacobian of $C : y^2 = x^5 + f_3x^3 + f_2x^2 + f_1 + f_0$.

Input: $P = (U_1 : U_0 : V_1 : V_0 : Z)$, $Q = (u_1, u_0, v_1, v_0)$.
Output: $P + Q = (U_1'' : U_0'' : V_1'' : V_0'' : Z'')$.

$$
\begin{array}{ll}
t_1 \leftarrow v_0 \cdot Z, & V_0'' \leftarrow V_0 - t_1, \\
t_1 \leftarrow v_1 \cdot Z, & t_2 \leftarrow t_1 + V_1, \\
t_1 \leftarrow t_1 - V_1, & V_1'' \leftarrow u_1 \cdot Z, \\
t_3 \leftarrow V_1'' + U_1, & t_4 \leftarrow u_0 \cdot Z, \\
t_5 \leftarrow V_1'' \cdot t_4, & t_6 \leftarrow U_1 \cdot U_0, \\
t_6 \leftarrow t_6 - t_5, & U_0'' \leftarrow U_0 - t_4, \\
t_5 \leftarrow V_1''^2, & t_7 \leftarrow U_1^2, \\
U_1'' \leftarrow V_1'' - U_1, & t_8 \leftarrow t_5 - t_7, \\
t_5 \leftarrow t_5 + t_7, & t_7 \leftarrow Z \cdot U_0'', \\
t_8 \leftarrow t_7 + t_8, & t_7 \leftarrow t_6 \cdot t_1, \\
t_1 \leftarrow U_0'' \cdot t_1, & U_0'' \leftarrow U_0'' \cdot t_8, \\
t_6 \leftarrow t_6 \cdot U_1'', & U_1'' \leftarrow V_0'' \cdot U_1'', \\
t_8 \leftarrow V_0'' \cdot t_8, & t_7 \leftarrow t_7 - t_8, \\
t_1 \leftarrow t_1 - U_1'', & U_0'' \leftarrow U_0'' - t_6, \\
t_8 \leftarrow U_0''^2, & t_6 \leftarrow t_1 \cdot Z, \\
U_0'' \leftarrow U_0'' \cdot t_6, & t_1 \leftarrow t_1 \cdot t_6, \\
V_1'' \leftarrow t_1 \cdot V_1'', & t_5 \leftarrow t_1 \cdot t_5, \\
V_0'' \leftarrow t_7 \cdot t_6, & t_6 \leftarrow t_6^2, \\
t_7 \leftarrow t_7^2, & t_4 \leftarrow t_4 \cdot t_6, \\
t_6 \leftarrow U_0'' \cdot t_6, & U_1'' \leftarrow 2 \cdot V_0'', \\
U_1'' \leftarrow U_1'' - t_8, & t_2 \leftarrow U_0'' \cdot t_2, \\
t_7 \leftarrow t_7 \cdot Z, & t_7 \leftarrow t_7 + t_2, \\
t_2 \leftarrow t_1 \cdot t_3, & U_1'' \leftarrow U_1'' - t_2, \\
t_8 \leftarrow U_1'' - t_8, & t_3 \leftarrow t_3 \cdot t_8, \\
t_3 \leftarrow t_3 + t_5, & t_3 \leftarrow t_3/2, \\
t_7 \leftarrow t_7 - t_3, & t_8 \leftarrow V_1'' - V_0'', \\
V_0'' \leftarrow V_0'' - U_1'', & t_5 \leftarrow t_7 - t_4, \\
V_1'' \leftarrow V_1'' \cdot t_8, & t_1 \leftarrow t_1 \cdot t_5, \\
t_1 \leftarrow t_1 + V_1'', & V_1'' \leftarrow U_1'' \cdot V_0'', \\
V_1'' \leftarrow V_1'' + t_1, & t_4 \leftarrow t_4 \cdot t_8, \\
V_0'' \leftarrow V_0'' \cdot t_7, & V_0'' \leftarrow t_4 + V_0'', \\
t_4 \leftarrow t_6 \cdot v_1, & V_1'' \leftarrow V_1'' - t_4, \\
U_1'' \leftarrow U_1'' \cdot Z, & U_1'' \leftarrow U_1'' \cdot U_0'', \\
U_0'' \leftarrow t_7 \cdot U_0'', & V_1'' \leftarrow Z \cdot V_1'', \\
Z'' \leftarrow Z \cdot t_6, & t_7 \leftarrow Z'' \cdot v_0, \\
V_0'' \leftarrow V_0'' - t_7 &
\end{array}
$$

Projective coordinates of the points require division at the end to make representation unambiguous

Algorithm 6 Combined doubling and pseudo-addition, $\mathcal{K}(\texttt{DBLADD})$.

Input: $P = (x : y : z : t)$, $Q = (x' : y' : z' : t')$, $P - Q = (\bar{x} : \bar{y} : \bar{z} : \bar{t})$, and $y_0, z_0, t_0, y_0', z_0', t_0'$.
Output: $([2]P, P + Q) = \texttt{DBLADD}(P, Q, P - Q)$.

1. $x, y, z, t \leftarrow \texttt{H}(x, y, z, t)$, $x', y', z', t' \leftarrow \texttt{H}(x', y', z', t')$.
2. $X \leftarrow x \cdot x'$, $Y \leftarrow y \cdot y_0'$, $Z \leftarrow z \cdot z_0'$, $T \leftarrow t \cdot t_0'$.
3. $x \leftarrow x^2$, $y \leftarrow y \cdot Y$, $z \leftarrow z \cdot Z$, $t \leftarrow t \cdot T$.
4. $Y \leftarrow Y \cdot y'$, $Z \leftarrow Z \cdot z'$, $T \leftarrow T \cdot t'$.
5. $x, y, z, t \leftarrow \texttt{H}(x, y, z, t)$, $X, Y, Z, T \leftarrow \texttt{H}(X, Y, Z, T)$.
6. $x \leftarrow x^2$, $y \leftarrow y^2$, $z \leftarrow z^2$, $t \leftarrow t^2$.
7. $X \leftarrow X^2$, $Y \leftarrow Y^2$, $Z \leftarrow Z^2$, $T \leftarrow T^2$.
8. $y \leftarrow y \cdot y_0$, $z \leftarrow z \cdot z_0$, $t \leftarrow t \cdot t_0$.
9. $x' \leftarrow X/\bar{x}$, $y' \leftarrow Y/\bar{y}$, $z' \leftarrow Z/\bar{z}$, $t' \leftarrow T/\bar{t}$.
10. **return** $(x : y : z : t)$, $(x' : y' : z' : t')$.

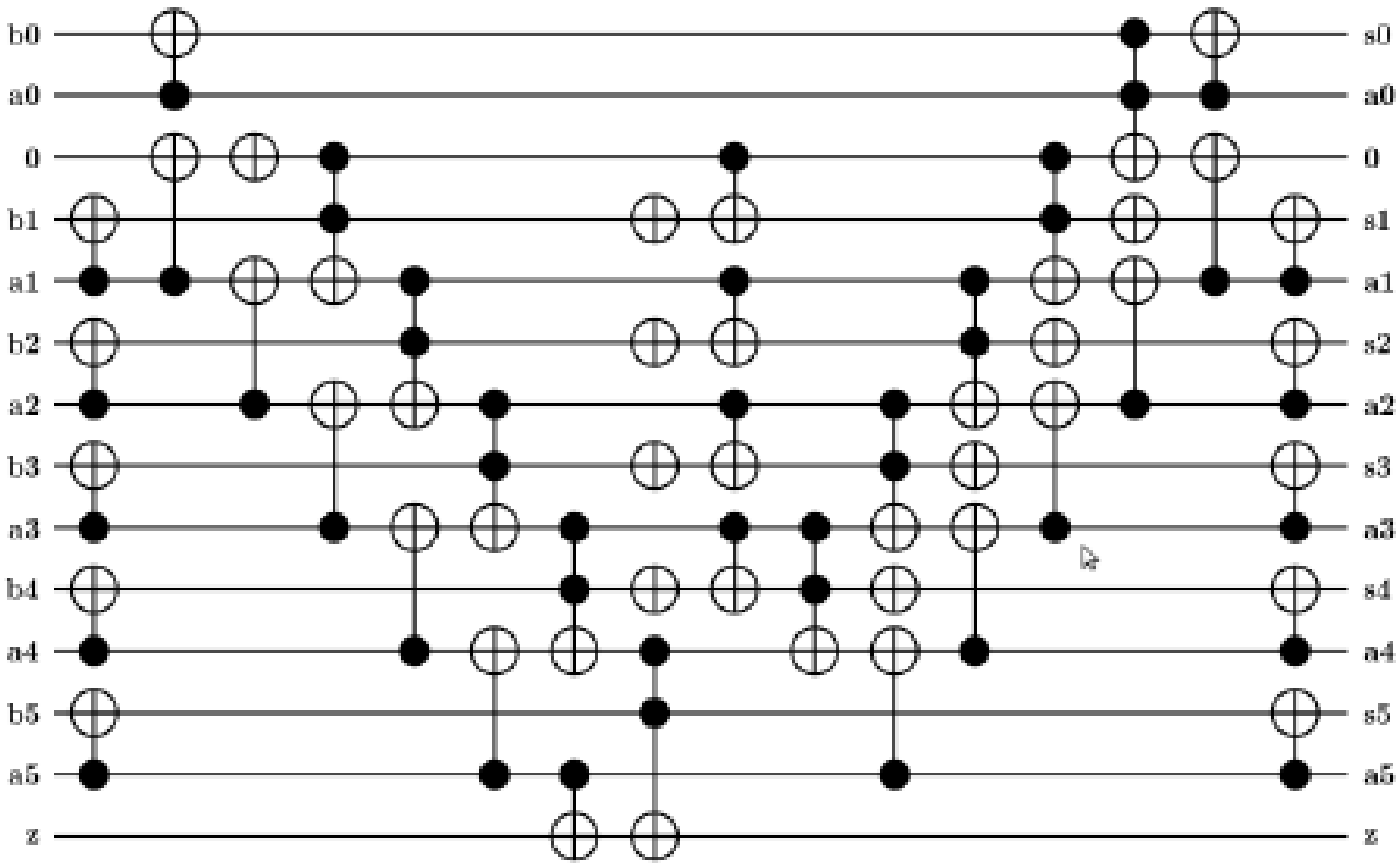
Some formulas require modular division

[Bos, Costello, Hisil, Lauter, 2013]

Quantum arithmetic

what is the problem?
why is this non-trivial?
who cares?

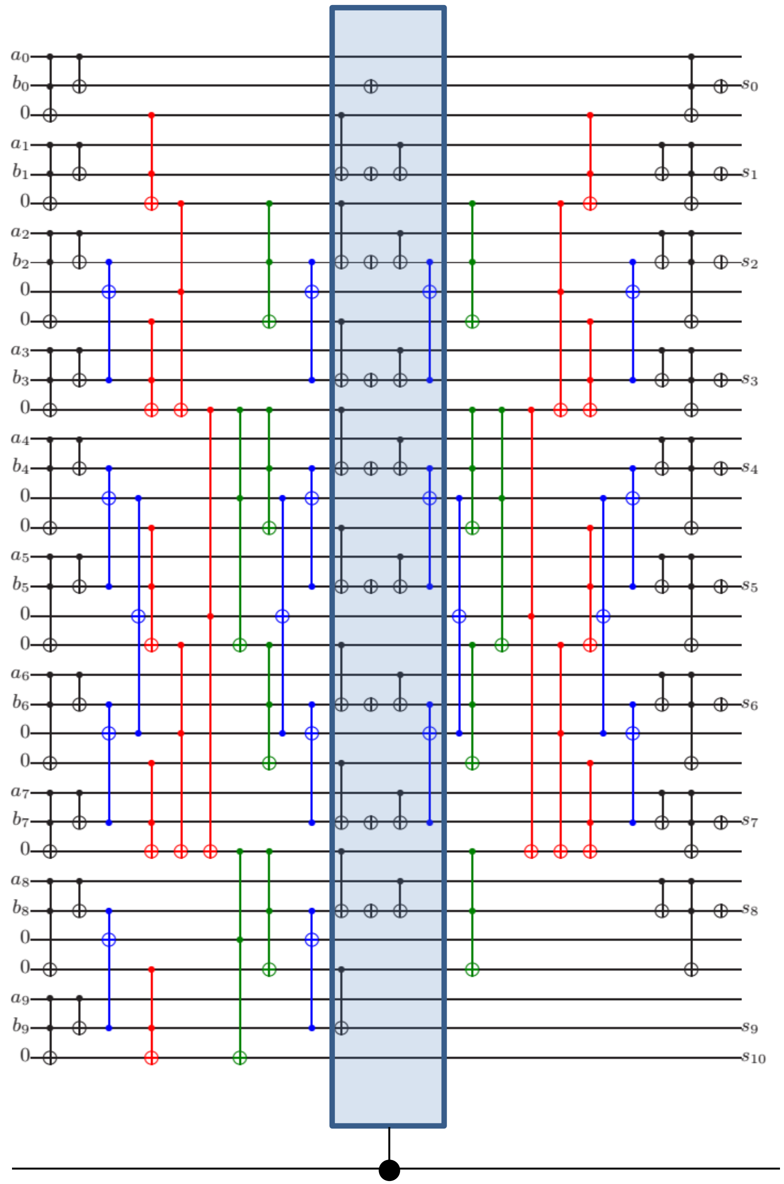
Adders



[CDKM:04] S. A. Cuccaro, T. G. Draper, S. A. Kutin, and D. P. Moulton, quant-ph/0410184 (2004).

This is a space optimized adder. Runs in T-depth 2n-1. Quite poor load factor, i.e., most qubits in the computation are idle. Explore time/space trade-offs.

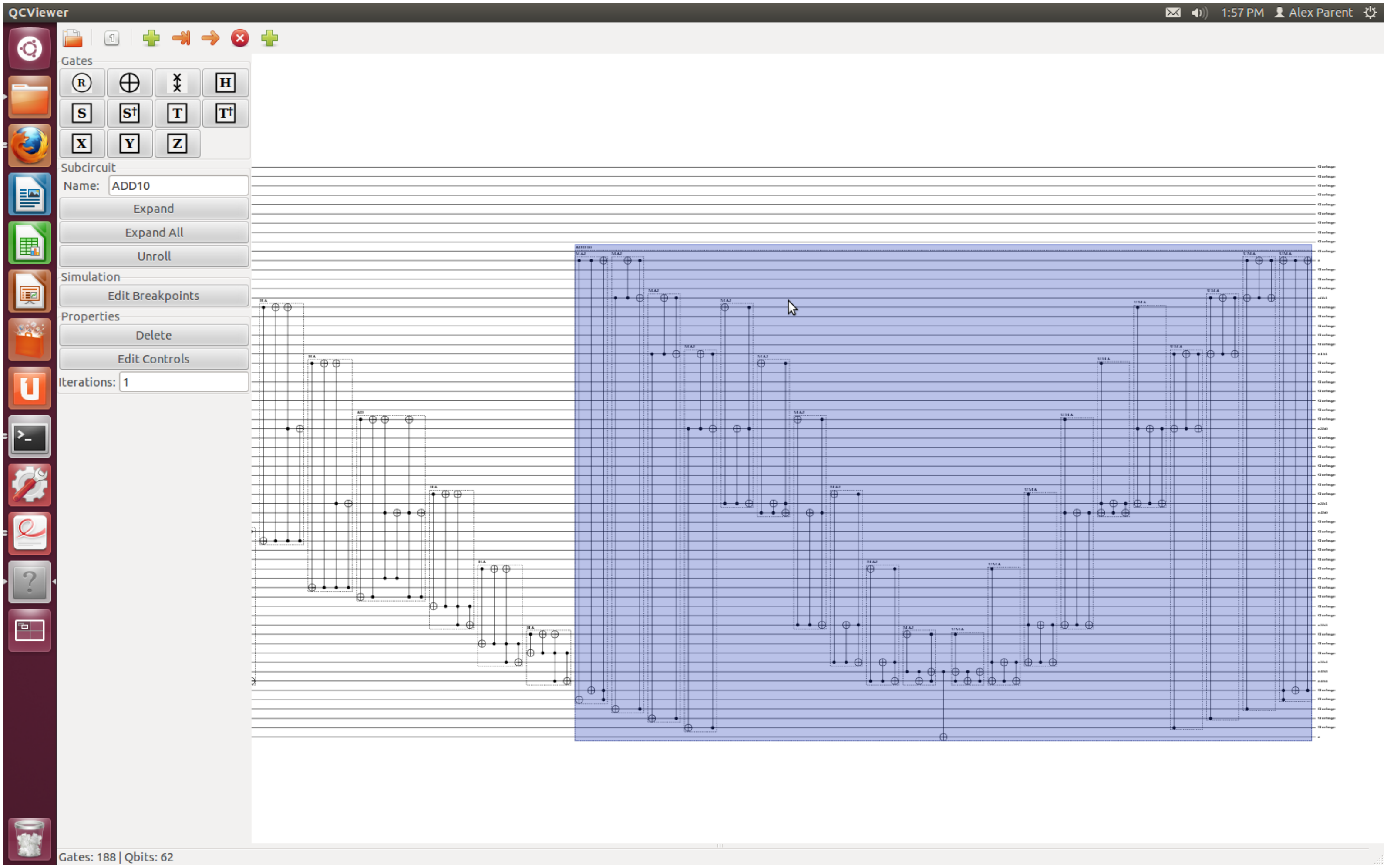
Controlled quantum adder



Resource estimate:
$14n - 11$ Toffoli gates

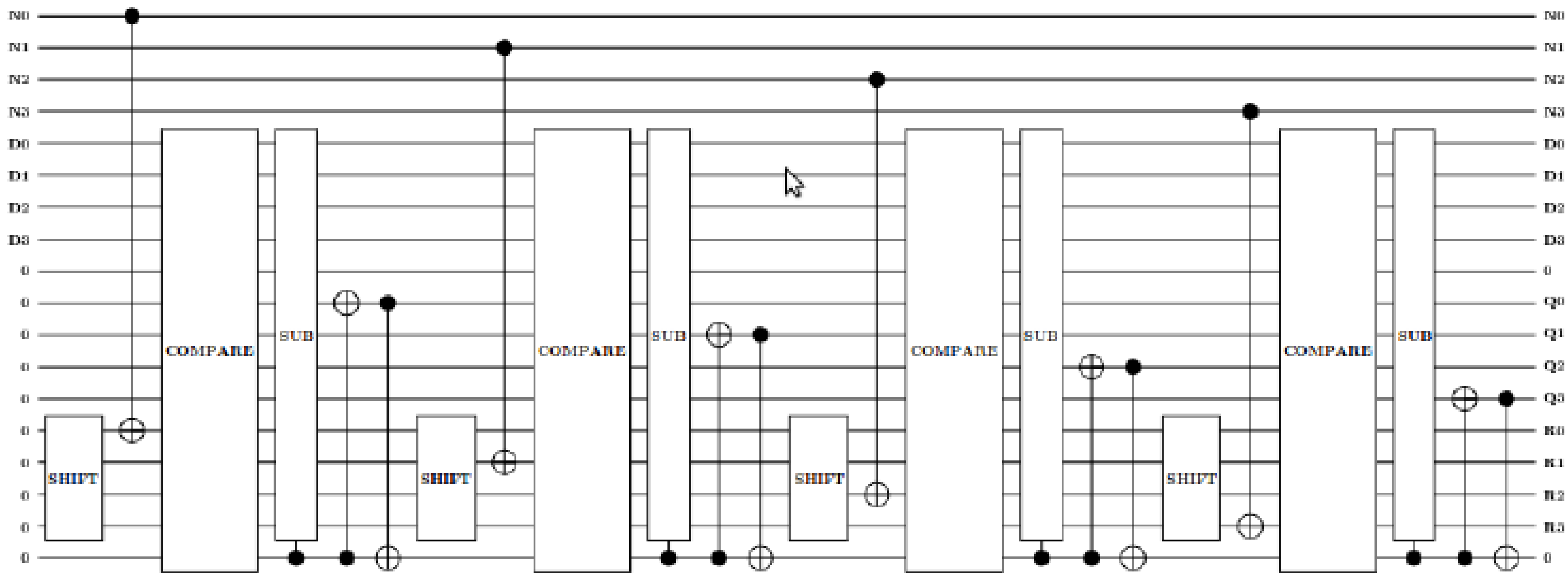
[Draper, Kutin, Rains, Svore, 2004]

Multipliers



Wallace tree multiplier. T-count of $n^2 + 4n\,\log_2(n)$ and T-depth $O(\log_2(n))$. Shown is an implementation in .qc/QCViewer of a circuit generated dynamically by a Haskell library.

Division with remainder



For a division with remainder we obtain the estimate $Divider(n) = n \cdot (CSub(n) + Adder(n) + 3CNOT)$, where $CSub(n)$ is bounded by the cost of one $Adder(n)$, $3n$ local Pauli operations, n CNOT gates, and $(n+1)$ Toffoli gates.

Time-space tradeoffs II

Adders for n bit integers:

- Low depth circuit:
  - [Draper, Kutin, Rains, Svore, quant-ph/0406142].
  - Depth $O(\log n)$, however, requires $O(n)$ ancillas.
  - In-place version exists. Easy to modify into controlled adder
- Space optimized circuit:
  - [Cuccaro, Draper, Kutin, Moulton, quant-ph/0410184].
  - Can be used to implement in-place addition $|x, y\rangle \mapsto |x, x+y\rangle$ with only 1 additional ancilla qubit. Depth scales linear with n.

Multipliers for n bit integers:

- Simple $O(n^2)$ "school" method using controlled adders. Disadvantage: circuit depth scales linear with n. Improvement: Wallace tree in log depth.
- Limitation: only out-of-place multipliers $|x, y, 0\rangle \mapsto |x, y, x \cdot y\rangle$ known.

Arithmetic for modular exponentiation:

- Computing $x \mapsto a^x \ mod\ N$ for **fixed** a, N is relatively easy and can be done using $2n + 3$ qubits and $O(n^3)$ time: [Beauregard, quant-ph/0205095]

Modular inverses:

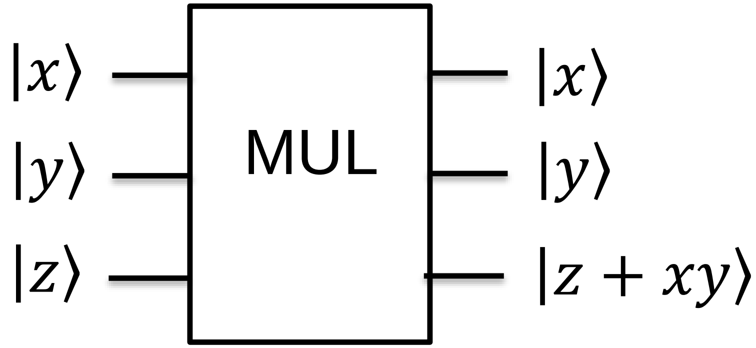
Approaches based on Fermat’s little theorem

Modular Inverse a la Fermat

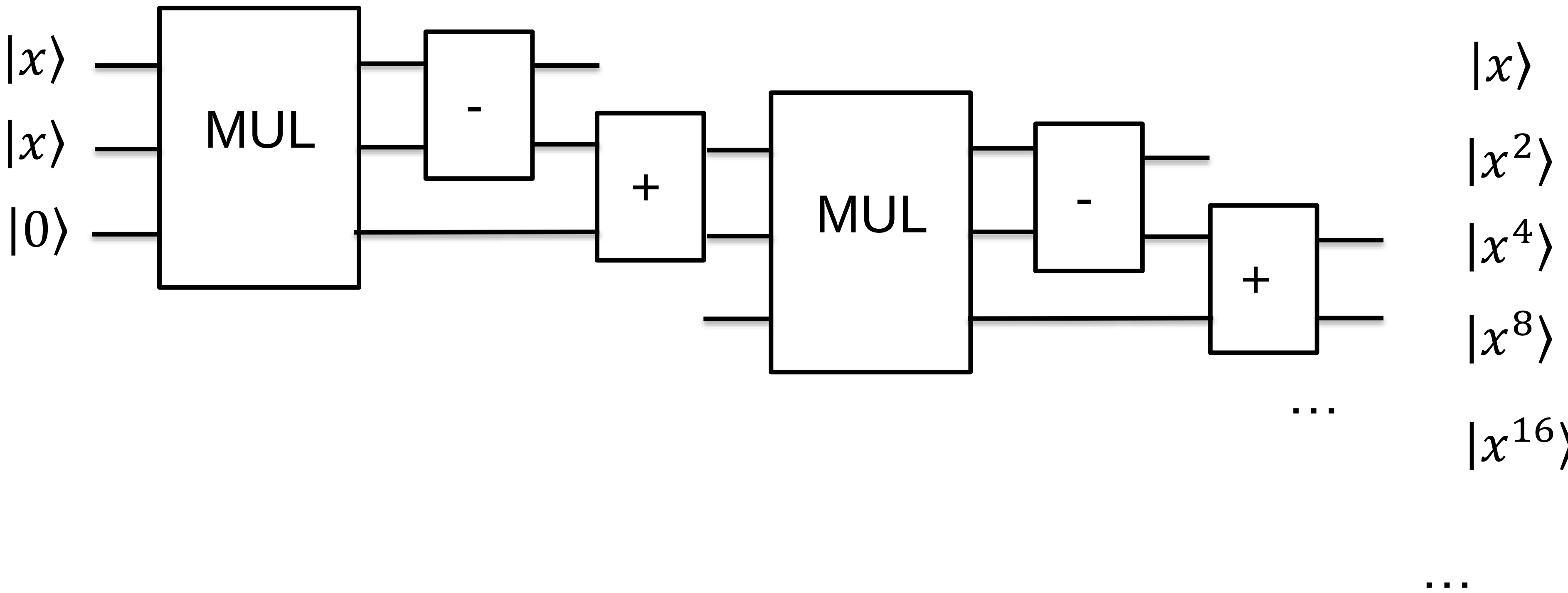
Basic idea:

- Let p be prime, let $x \in \{1, \ldots, p-2\}$.
- Recall that in any finite group: $x^{|G|} = e$.
- When applied to $GF(p)^{\times}$ this implies
  - $x^{p-1} \equiv 1\ (p)$
- Or in other words: $x^{p-2} \cdot x \equiv 1\ (p)$
- Or in other words: $x^{-1} \equiv x^{p-2}\ (p)$
- That means we can compute the inverse by exponentiation of the (unknown) x for the (known, fixed) exponent p.

Modular multiplier



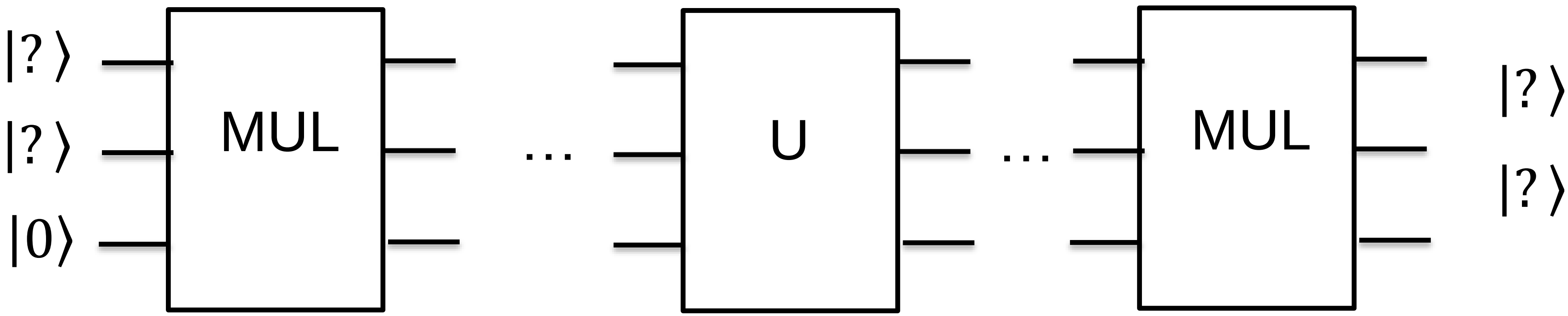
Square & multiply by unrolling



Depth: $2n \times depth(MUL) + 2depth(ADD)) + n$
Width: $2n \times n = n^2$

- Here n is the bit-size of x
- Use binary representation of $p-2$ to compute x^{p-2}

Open problem: improvements?



Depth: $O(n^2)$
Width: $O(n)$

$\leftarrow$ Can we achieve this using suitable initial configuration, suitable U?

- Partial success: using MUL and suitable permutations U we can compute the Chebyshev polynomials $T_n(x) \; mod \; p$.
- Unclear whether they allow to efficiently compute monomials x^n

Unknown whether linear space can be achieved by this approach

Modular inverses:

Approaches based on the Euclidean algorithm

Modular Inverse via GCD

Basic idea:

- Let p be prime, let $x \in \{1, \ldots, p-2\}$.
- Compute the greatest common divisor (GCD) of p and x
- … and find the linear representation of the GCD:
$$a\,p \;+\; b\,x \;=\; GCD(p, x) \;=\; 1$$
- This means that modulo p we have that $b\,x \;=\; 1$
- In other words: $x^{-1}\,mod\,p = b$.

How to find a and b? → Extended Euclidean Algorithm

An Orwellian principle (?)

"Ignorance is Strength"

Any computation that a quantum computer carries out must be independent of the input data.

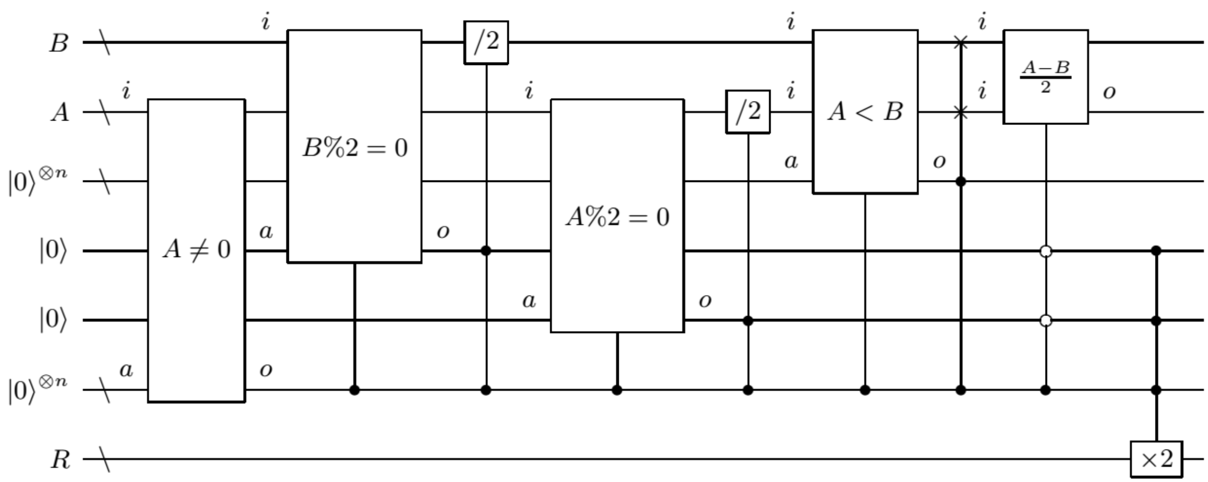
- **Reason:** quantum programs must be able to run on superposition of input data. If the execution flow of the depended on the input in any way that makes 2 or more inputs distinguishable, this can lead unwanted entanglement that destroys interference.
- In quantum context first studied by [Bernstein/Vazirani'93] → path synchronization technique for Quantum TMs.
- Classically studied too: "Oblivious Turing Machines"

Saeedi & Markov's method

Uses binary Euclid:

- If $A\%2 = B\%2 = 0$, $\gcd(A,B) = 2\gcd(A/2, B/2)$
- If $A\%2 = 0 = 1 - B\%2$, $\gcd(A,B) = \gcd(A/2, B)$
  If $A\%2 = 1 = 1 - B\%2$, $\gcd(A,B) = \gcd(A, B/2)$
- If $A\%2 = B\%2 = 1$, then we ensure that $A \geq B$, and use $\gcd(A,B) = \gcd\left(\frac{A-B}{2}, B\right)$

Single round:



Summary: + Easy to circuitize
+ Depth scales as O(n log n)
- But does not yield linear representation of GCD

[Saeedi, Markov arXiv:1304.7516]

Shor for factoring vs ECC dlog

Factoring algorithm (RSA)			EC discrete logarithm (ECC)		
n	$\approx$ # qubits	time	n	$\approx$ # qubits	time
	$2n$	$4n^3$		$f'(n)$ $(f(n))$	$360n^3$
512	1024	$0.54 \cdot 10^9$	110	700 (800)	$0.5 \cdot 10^9$
1024	2048	$4.3 \cdot 10^9$	163	1000 (1200)	$1.6 \cdot 10^9$
2048	4096	$34 \cdot 10^9$	224	1300 (1600)	$4.0 \cdot 10^9$
3072	6144	$120 \cdot 10^9$	256	1500 (1800)	$6.0 \cdot 10^9$
15360	30720	$1.5 \cdot 10^{13}$	512	2800 (3600)	$50 \cdot 10^9$

[Proos, Zalka, quant-ph/0301141]

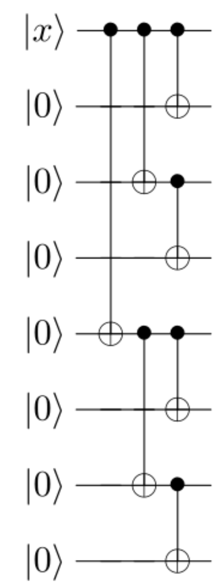
- Suggests that quantum attacks on ECC/dlog can be done more efficiently than RSA/factoring with comparable level of security.
- Circuits are somewhat non-trivial to implement and to layout.
- Only short Weierstrass forms considered, unclear how classical optimizations of point additions can be leveraged.
- Leaves open how to optimize depth for Shor ECC.

Optimizing the circuit depth for the binary case

Low-depth GF(2^n)-arithmetic

Design decision: polynomial basis representation

- **Addition:** depth O(1)
- **Squaring:** matrix-vector mult. → addition trees+“multi-fan-out CNOT w/ |0⟩-input”: O(log n)



- **Multiplication:** Maslov et al.’s construction reduces to 3 matrix-vector multiplications
  parallelization: depth O(log n)

Projective point addition: depth O(log n)

Note: all this is irrelevant for the large p case !!

Inversion: prior work

Beauregard et al. 2003, Kaye-Zalka 2004, Maslov et al. 2009 offer circuits for $GF(2^m)$-inversion:

Inversion: apply extended Euclidean algorithm in depth $O(m^2)$ using $2m + O(\log m)$ qubits.

We can actually do much better in the binary case and achieve poly-log scaling of depth!

Ghost-bit basis representation

[Itoh-Tsujii 1989], [Silverman 1999]:

If $f=1+x+\ldots+x^m \in GF(2^m)[x]$ is irreducible, the maps

$$
\begin{array}{ccc}
GF(2^m)[x]/(f) & \longrightarrow & GF(2^m)[x]/(x^{m+1}+1) \\
\Sigma\alpha_i+(f) & \rightarrow & \Sigma\alpha_i+(x^{m+1}+1) \\
\Sigma(\alpha_i+\alpha_m)x^i+(f) & \leftarrow & \alpha_0x^0+\ldots+\alpha_m x^m+(x^{m+1}+1)
\end{array}
$$

allow to move arithmetic to $GF(2^m)[x]/(x^{m+1}+1)$.

Ghost-bit basis arithmetic

- **Addition:** bit-wise $\oplus$ (i.e., depth 1 with CNOTs)
- **Multiplication:** $(\sum a_i x^i)\cdot(\sum b_i\, x^i) = \sum_i (\sum_j a_j b_{(i-j) \bmod (m+1)})\cdot x^i$
- **Squaring:** $(\sum a_i x^i)^2 = \sum a_{p^{-1}(i)}\cdot x^i$ with
  $p(i)=2\cdot i \bmod (m+1)$

Squaring is a shuffle of the coefficient vector

Gaussian normal basis of type T

Vector space basis $\{h, h^2, h^{2^2},\ldots, h^{2^{m-1}}\}$ of $GF(2^m)$; let $p=Tm+1$, $u\in GF(2^m)^*$ of order T, $F(2^i u^j \bmod p)=i$

- **Addition:** bit-wise $\oplus$
- **Multiplication:** $(\sum a_i \cdot h^{2^i})\cdot(\sum b_i \cdot h^{2^i}) = \sum g_i \cdot h^{2^i}$ with $g_i = a_{F(1+1)+i} \cdot b_{F(p-1)+i} + \ldots + a_{F(Tm-1+1)+i} \cdot b_{F(p-(Tm-1))+i}$
- **Squaring:** $(\sum a_i \cdot h^{2^i})^2 = \sum a_{i-1 \,(\mathrm{mod}\, m)} \cdot h^{2^i}$

Squaring is a rotation of the coefficient vector

Itoh-Tsujii inversion algorithm

For $\alpha \in GF(2^m)^*$ let $\beta_i = \alpha^{2^i - 1}$. Then $\beta_1 = \alpha$, $\alpha^{-1} = (\beta_{m-1})^2$, and

$$\beta_{i+j} = \beta_i \cdot (\beta_j)^{2^i} \,.\ (*)$$

(1) write $m-1 = 2^{k_1} + \ldots + 2^{k_{HW(m-1)}}$ with
$\lfloor \log_2(m-1) \rfloor = k_1 > \ldots > k_{HW(m-1)} \geq 0$

(2) find $\beta_{2^0}, \beta_{2^1}, \ldots, \beta_{2^{k_1}}$ applying (*) with i=j

(3) find $\beta_{2^{k_1}+2^{k_2}}, \ldots, \beta_{2^{k_1}+\ldots+2^{k_{HW(m-1)}}} (= \beta_{m-1})$ with (*)

Total cost:

$\lfloor \log_2(m-1) \rfloor + HW(m-1) - 1$ multiplications (+ squarings)

Inversion in depth $O(\log^2 m)$

(1) find $\beta_{2^0}, \beta_{2^1}, \ldots, \beta_{2^{k_1}}$ from Itoh-Tsujii algorithm with $\lfloor \log_2(m-1) \rfloor$ "single-input" multipliers (squaring is free: permute control positions)

(2) find $\beta_{2^{k_1} + \ldots + 2^{k_{HW(m-1)}}}(=\beta_{m-1})$ with $HW(m-1)-1$ "ordinary" multipliers (not needed for $m=2^n+1$, e.g., a Fermat prime)

(3) Finally, $\alpha^{-1} = (\beta_{m-1})^2$ which is just a shuffle

How not to compute $k \cdot P + \ell \cdot Q$…

Maslov et al.'strategy – right-to-left double-and-add:

$R \leftarrow 0$

for $i = 0$ to n step 1

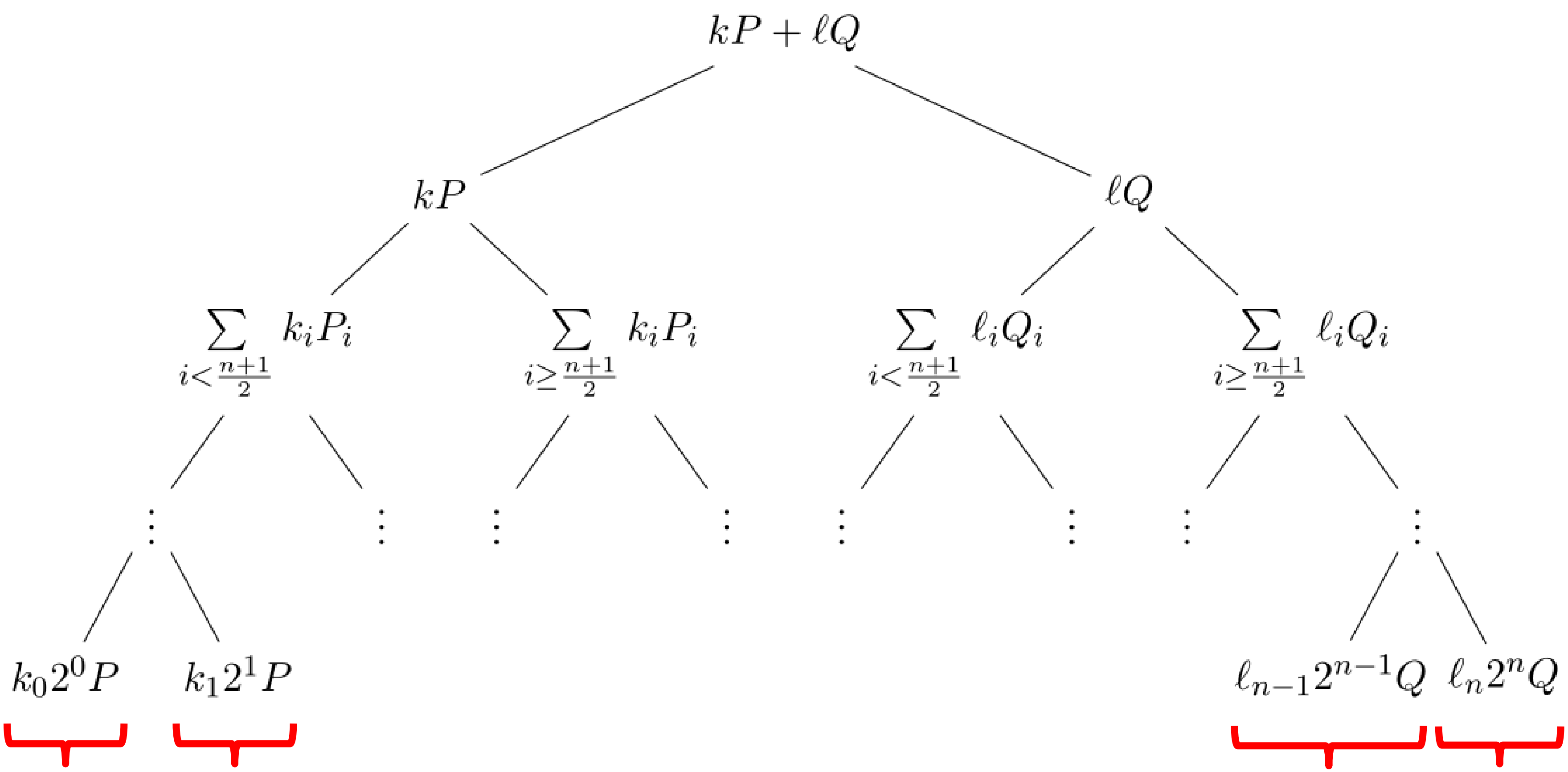
  if $k_i = 1$ then $R \leftarrow R + 2^i \cdot P$

  if $\ell_i = 1$ then $R \leftarrow R + \underbrace{2^i \cdot Q}_{\text{precomputed}}$

return R

… yields depth $O(n \cdot \log n)$ circuit

Instead: Parallelized double-and-add



- requires “multi-fan-out CNOT w/ |0⟩-input”
- depth O($\log^2 n$), using general addition circuits

Open problems

- Can we adapt the methods to a 2D NN architecture?
- Can square&multiply based ideas be modified to make them space efficient?
- Can the "quantum-quantum" techniques based on the quantum Fourier transform (e.g., Draper adder) be applied to the modular inversion problem? Can we avoid modular inversions altogether?
- Can we simplify the (Edwards) point addition circuits? Few T-gates, less T-depth, less qubits?
- Use the resource estimates to obtain resource estimates for quantum attacks on ECC dlog for NIST curves and generalize this to Jacobians of hyperelliptic curves.