Solving binary $\mathcal{MQ}$ with Grover's algorithm

Peter Schwabe Bas Westerbaan

bas@westerbaan.name

Radboud Universiteit

Nijmegen, The Netherlands

December 23, 2016

You can be a Quantum cryptanalysist too

Peter Schwabe Bas Westerbaan

bas@westerbaan.name

Radboud Universiteit

Nijmegen, The Netherlands

December 23, 2016

Convential wisdom

With stable $\sim$4000 qubit quantum computer:

Convential wisdom

With stable $\sim$4000 qubit quantum computer:

1. All popular asymmetric crypto is completely broken with Shor's algorithm.

Convential wisdom

With stable $\sim$4000 qubit quantum computer:

1. All popular asymmetric crypto is completely broken with Shor's algorithm.
2. Bits-of-security of all symmetric crypto is halved by Grover's algorithm.

Convential wisdom

With stable $\sim$4000 qubit quantum computer:

1. All popular asymmetric crypto is completely broken with Shor's algorithm.
2. Bits-of-security of all symmetric crypto is halved by Grover's algorithm.
   Example: pre-image of SHA1 takes 2^{80} time instead of 2^{160}.

Convential wisdom

With stable $\sim$4000 qubit quantum computer:

1. All popular asymmetric crypto is completely broken with Shor's algorithm.
2. Bits-of-security of all symmetric crypto is halved by Grover's algorithm.
   Example: pre-image of SHA1 takes 2^{80} time instead of 2^{160}.

This is almost true.

Jintai Ding, Bo-Yin Yang et al in ASIACRYPT 2015:

Jintai Ding, Bo-Yin Yang et al in ASIACRYPT 2015:

> *We believe that [Grover's algorithm] is no major threat (...) because of the large number of qubits needed in this case: (...) the number of qubits (...) needed to attack Gui by Grovers algorithm is in the order of a million (...)*

Jintai Ding, Bo-Yin Yang et al in ASIACRYPT 2015:

> *We believe that [Grover's algorithm] is no major threat (...) because of the large number of qubits needed in this case: (...) the number of qubits (...) needed to attack Gui by Grovers algorithm is in the order of a million (...)*

This is completely wrong:

Jintai Ding, Bo-Yin Yang et al in ASIACRYPT 2015:

> *We believe that [Grover's algorithm] is no major threat (...) because of the large number of qubits needed in this case: (...) the number of qubits (...) needed to attack Gui by Grovers algorithm is in the order of a million (...)*

This is completely wrong:
137 qubits are enough!

This talk

1. Binary $\mathcal{MQ}$
2. Grover's algorithm
3. Writing an oracle

This talk

1. Binary $\mathcal{MQ}$
2. Grover's algorithm
3. Writing an oracle

Binary $\mathcal{MQ}$

Underling hard problem of Gui

Find $x_i = 0, 1$ such that

$$x_1x_3 + x_1x_2 \text{ is even}$$
$$x_1x_2 + x_2x_3 + x_3x_4 \text{ is odd}$$
$$x_1 + x_1x_3 + x_2x_3 \text{ is even}$$

Binary $\mathcal{MQ}$

Underling hard problem of Gui

Find $x_i = 0, 1$ such that

$$x_1x_3 + x_1x_2 \text{ is even}$$
$$x_1x_2 + x_2x_3 + x_3x_4 \text{ is odd}$$
$$x_1 + x_1x_3 + x_2x_3 \text{ is even}$$

Here $n = 4$ variables and $m = 3$ equations.

Binary $\mathcal{MQ}$

Underling hard problem of Gui

Find $x_i \in \mathbb{F}_2$ such that

$$x_1x_3 + x_1x_2 = 0$$
$$x_1x_2 + x_2x_3 + x_3x_4 = 1$$
$$x_1 + x_1x_3 + x_2x_3 = 0$$

Here $n = 4$ variables and $m = 3$ equations.

Binary $\mathcal{MQ}$

Underling hard problem of Gui

Find $x_i \in \mathbb{F}_2$ such that

$$x_1x_3 + x_1x_2 + 1 = 1$$
$$x_1x_2 + x_2x_3 + x_3x_4 = 1$$
$$x_1 + x_1x_3 + x_2x_3 + 1 = 1$$

Here $n = 4$ variables and $m = 3$ equations.

Binary $\mathcal{MQ}$

Underling hard problem of Gui

Find $x_i \in \mathbb{F}_2$ such that

$$\begin{aligned} x_1x_3 + x_1x_2 + x_5 &= 1 \\ x_1x_2 + x_2x_3 + x_3x_4 &= 1 \\ x_1 + x_1x_3 + x_2x_3 + x_5 &= 1 \\ x_5 &= 1 \end{aligned}$$

Here $n = 5$ variables and $m = 4$ equations.

Binary $\mathcal{MQ}$

Underling hard problem of Gui

Find $x_i \in \mathbb{F}_2$ such that

$$\begin{aligned} x_1(x_3 + x_2) + x_5 &= 1 \\ x_1x_2 + x_2x_3 + x_3x_4 &= 1 \\ x_1(1 + x_3) + x_2x_3 + x_5 &= 1 \\ x_5 &= 1 \end{aligned}$$

Here $n = 5$ variables and $m = 4$ equations.

This talk

1. Binary $\mathcal{MQ}$
2. Grover's algorithm
3. Writing an oracle

Single needle problem

Given *any* $f : \mathbb{F}_2^n \to \mathbb{F}_2$ *(by a Boolean circuit) which is valued* 0 *everywhere except for one* w_1*,*
find *this* w_1 *with* $f(w_1) = 1$*.*

Single needle problem

Given *any* $f : \mathbb{F}_2^n \to \mathbb{F}_2$ *(by a Boolean circuit) which is valued* 0 *everywhere except for one* w_1*,*
find *this* w_1 *with* $f(w_1) = 1$*.*

Classically: expected 2^{n-1} calls to f.

Single needle problem

Given *any* $f : \mathbb{F}_2^n \to \mathbb{F}_2$ *(by a Boolean circuit) which is valued* 0 *everywhere except for one* w_1*,*
find *this* w_1 *with* $f(w_1) = 1$*.*

Classically: expected 2^{n-1} calls to f.

Quantum (Grover): if f can be given* as a quantum circuit then it takes $2^{\frac{n}{2}}$ calls to f.

Grover teaser

```
-- |Reflects (in place) over the standard uniform superposition of qubits.
-- This is used as the second part of the Grover iteration.
reflect_over_a :: [Qubit] -> Circ ()
reflect_over_a qs = do
  with_basis_change (forM_ qs hadamard_at) $ do
    with_basis_change (forM_ qs qnot_at) $ do
      with_basis_change (hadamard_at $ head qs) $ do
        qnot_at (head qs) `controlled` tail qs

-- |Grover's algorithm.  'oracle' is a circuit that should map exactly
-- 'm' 'n'-qubit-words to |1> and the others to |0>.  'grover' returns
-- a circuit that creates a superposition over all 'n'-qubit words that
-- are mapped to |1> by the oracle.
grover :: ([Qubit] -> Circ (Qubit)) -> Int -> Int -> Circ ()
grover oracle n m = do
  -- Create a standard uniform superposition over all qubits.
  qs <- qinit $ replicate n False
  forM_ qs hadamard_at
  iterations qs
  return ()
    where
      n_iters = floor $ pi / 4 / asin(sqrt((fromIntegral m) / (2^n)))
      iterations :: [Qubit] -> Circ ([Qubit])
      iterations = nbox "grover-iteration" n_iters $ \qs -> do
        -- First, the adapted oracle.  The following will send a computational
        -- basis vector w to -w if its tagged by the original oracle and
        -- leave it in place otherwise.
        with_computed (oracle qs) $ \r -> do
          gate_Z_at (head qs) `controlled` r
        -- Then, reflect over the standard uniform superposition.
        reflect_over_a qs
        return qs
```

Quantum circuits

No time to define, suffice to know

1. Every invertible Boolean circuit is a quantum circuit

Quantum circuits

No time to define, suffice to know

1. Every invertible Boolean circuit is a quantum circuit
2. There are more quantum circuits (but we don't need them here for f)

Quantum circuits

No time to define, suffice to know

1. Every invertible Boolean circuit is a quantum circuit
2. There are more quantum circuits (but we don't need them here for f)
3. Every quantum circuit is invertible

Quantum circuits

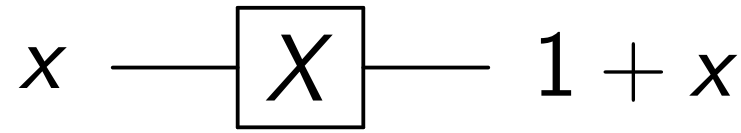
No time to define, suffice to know

1. Every invertible Boolean circuit is a quantum circuit
2. There are more quantum circuits (but we don't need them here for f)
3. Every quantum circuit is invertible

Because of the invertibility requirement,

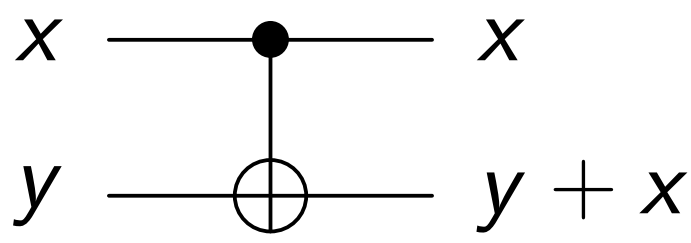
some things are way slower on a quantum computer than on a classical computer!

Gates (1): NOT gate



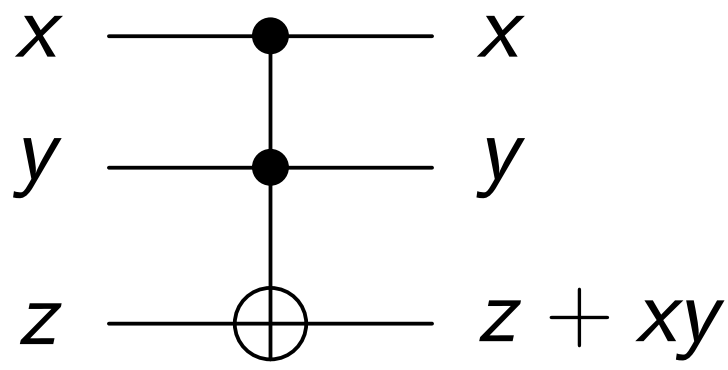
Quantum people like to call this the X-gate to confuse you.

Gates (2): CNOT gate



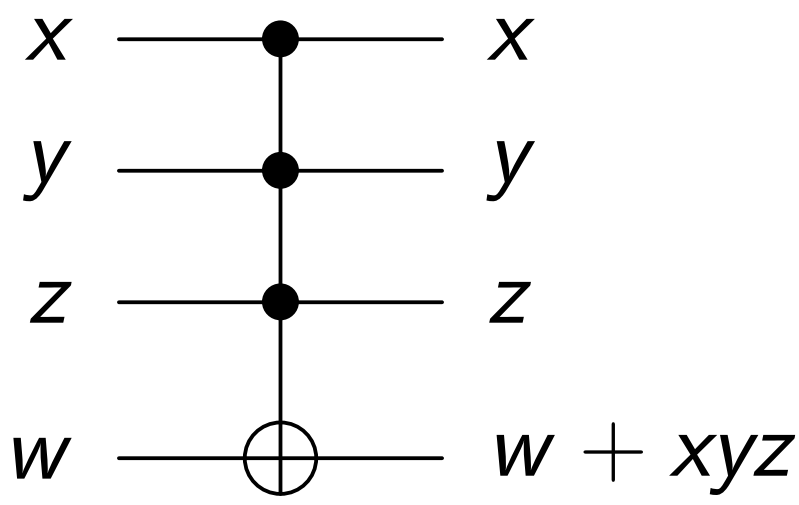
This is the invertible version of the XOR-gate.

Gates (3): Tofolli gate



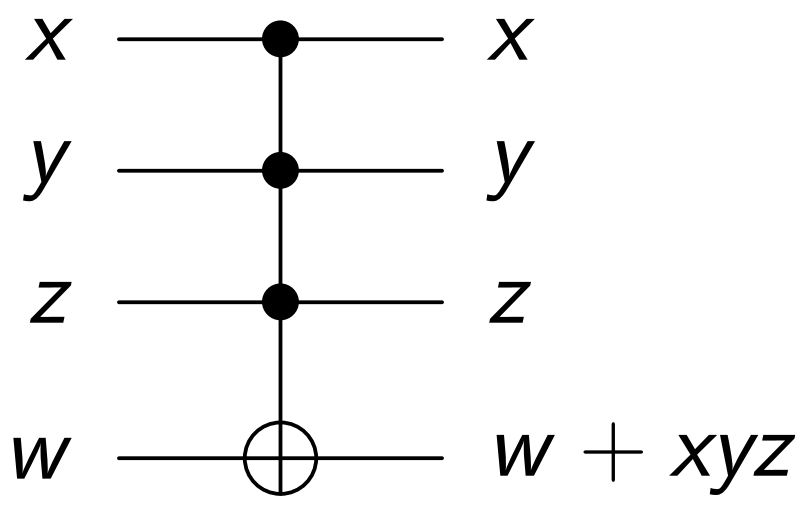
This is the invertible version of the AND-gate.

Gates (4): bigger Tofolli gates



This is for bigger AND-gates.

Gates (4): bigger Tofolli gates



This is for bigger AND-gates.

Clasically big AND-gates are easy. For quantum computers it's more costly.

This talk

1. Binary $\mathcal{MQ}$
2. Grover's algorithm
3. Writing an oracle

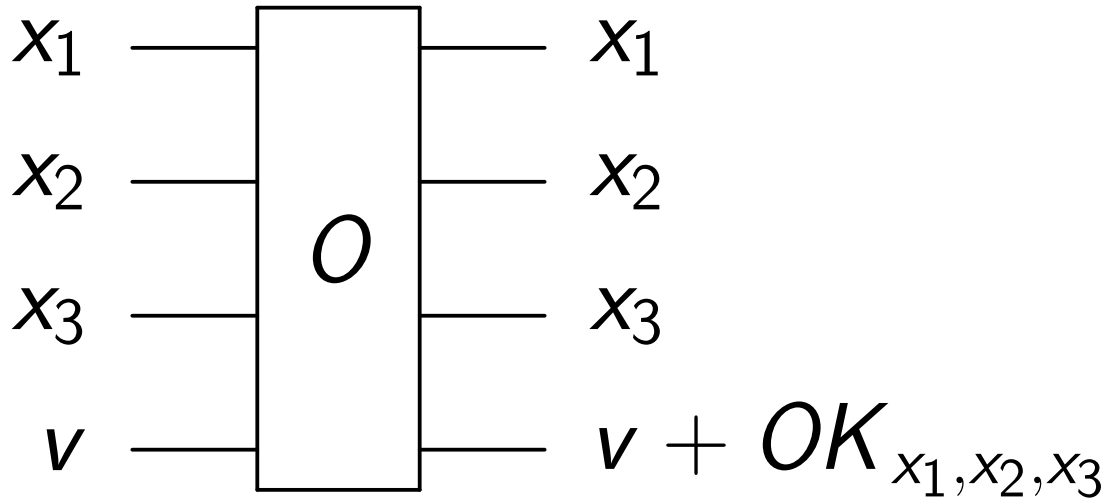
Running example

$$x_1(1 + x_2 + x_3) + x_2x_3 = 1$$
$$x_2(1 + x_3) = 1$$

Running example

$$x_1(1 + x_2 + x_3) + x_2x_3 = 1$$
$$x_2(1 + x_3) = 1$$

we must construct a circuit O with



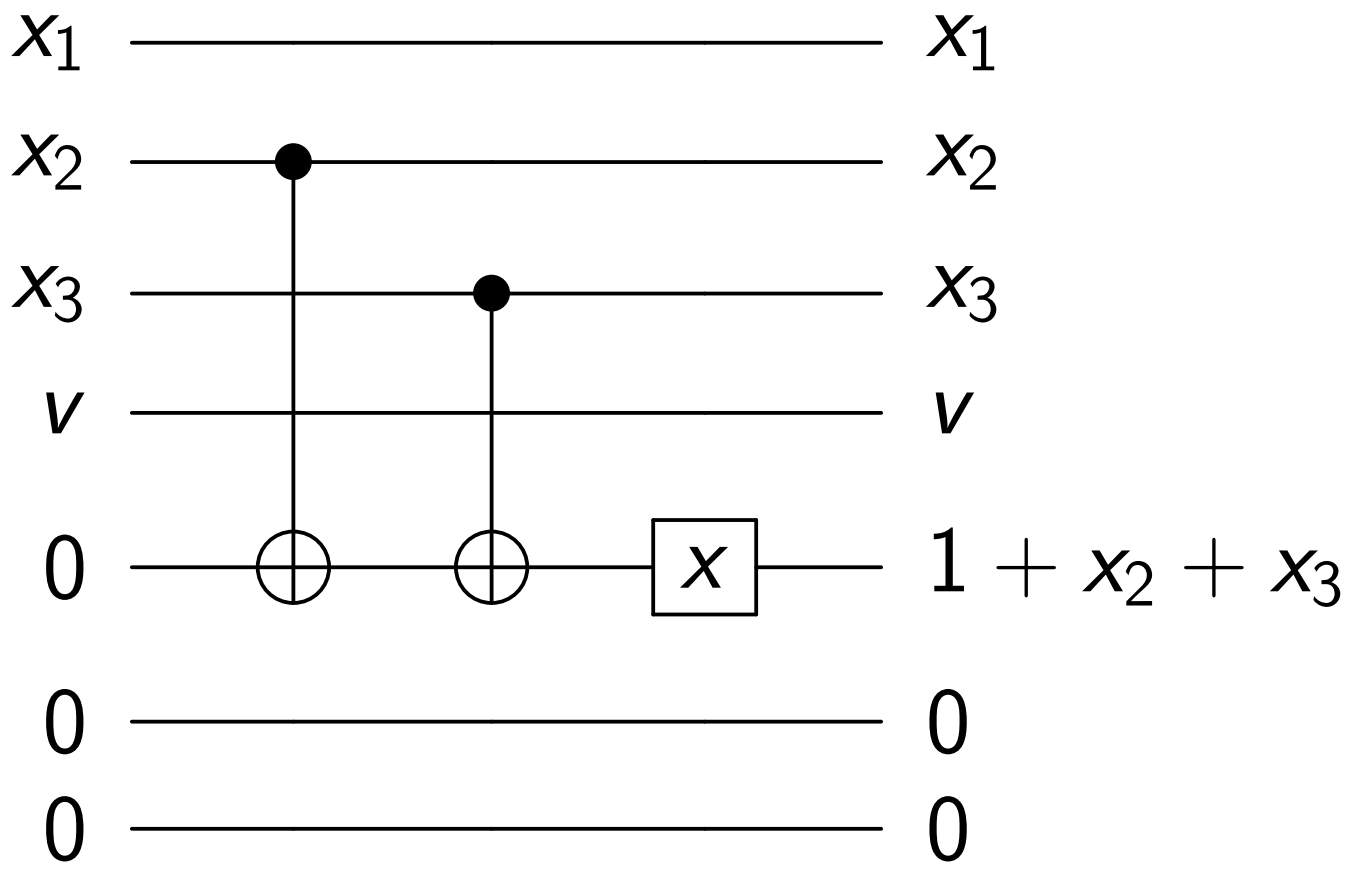
Building the circuit (1)

We will use 3 extra qubits:

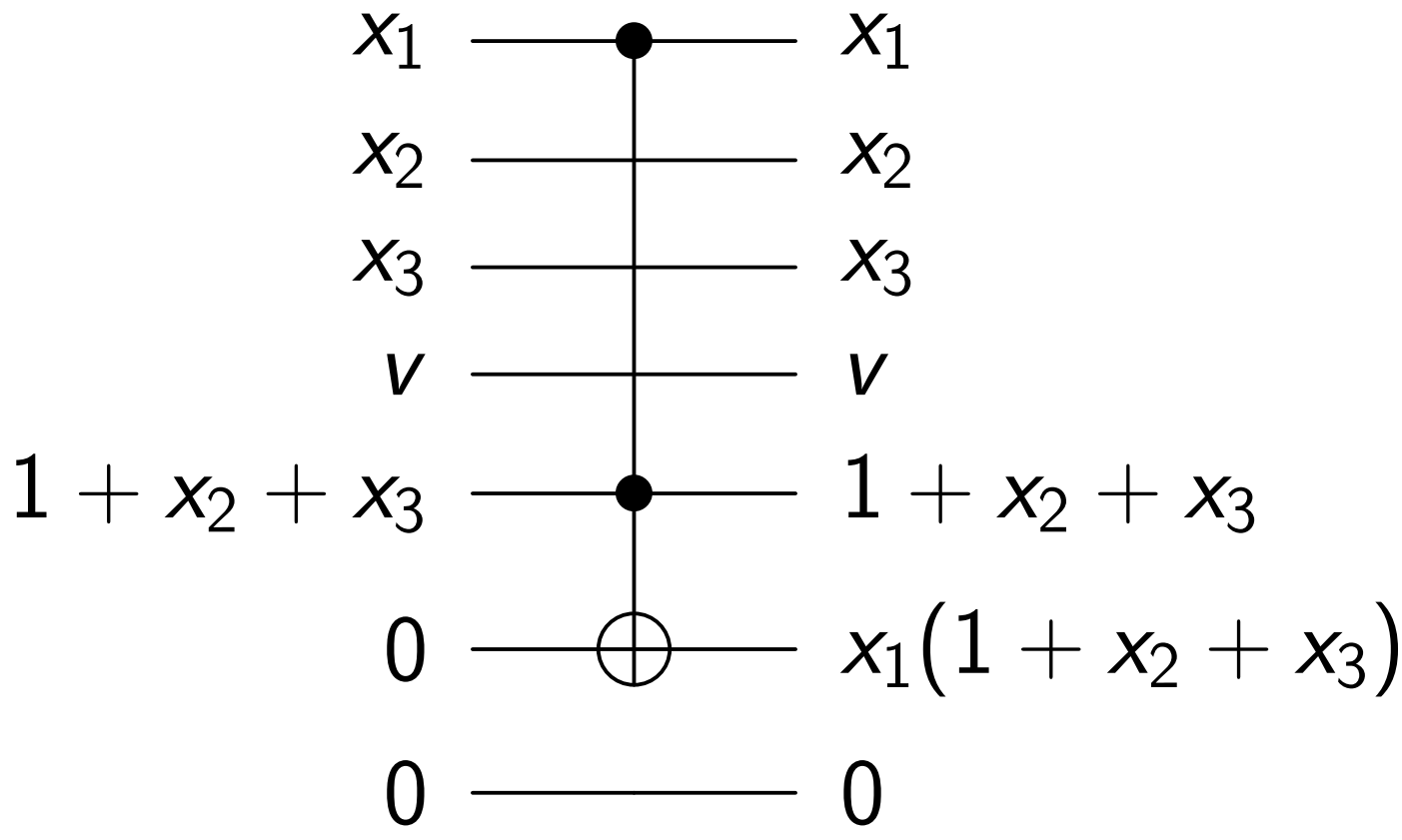
1. One for each equation to store its value.
2. One temporary.

So $n + m + 1$ total in general.

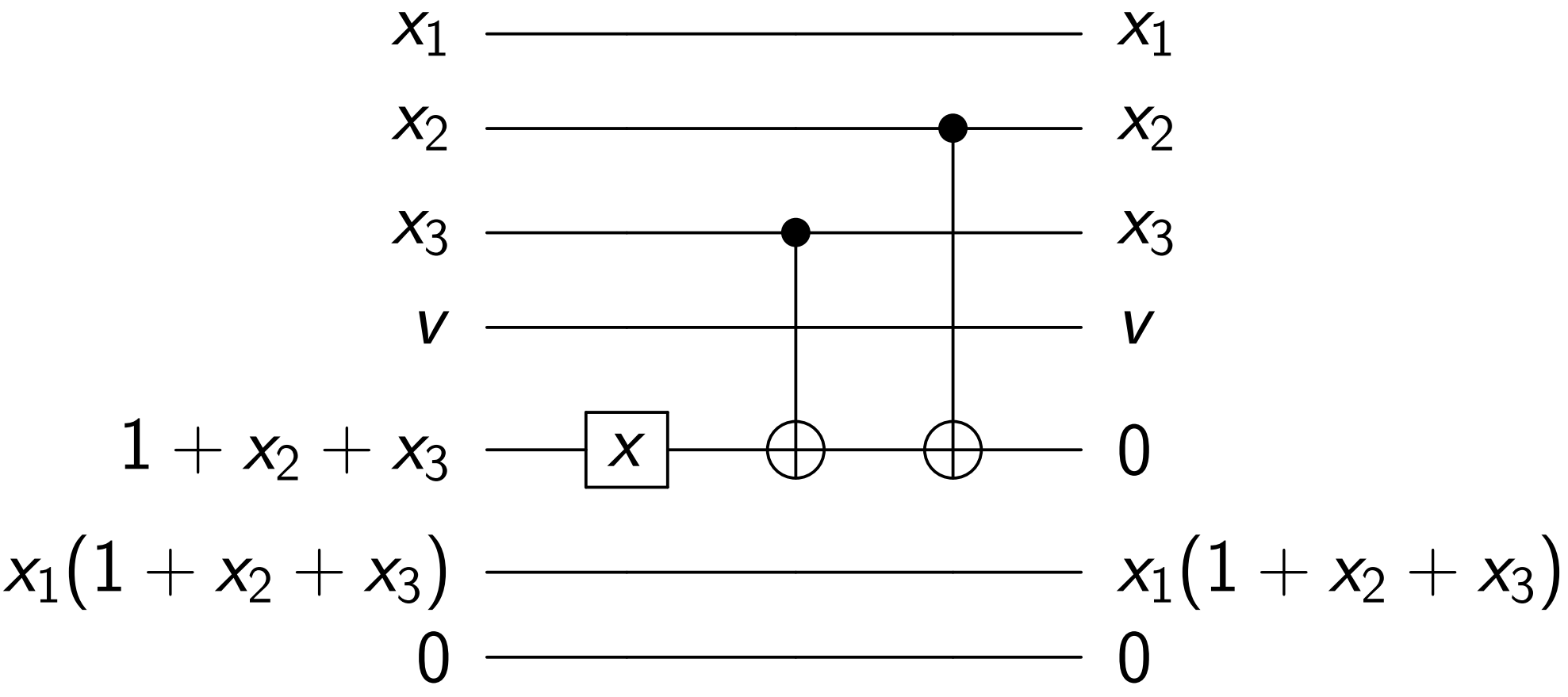
Building the circuit (2)



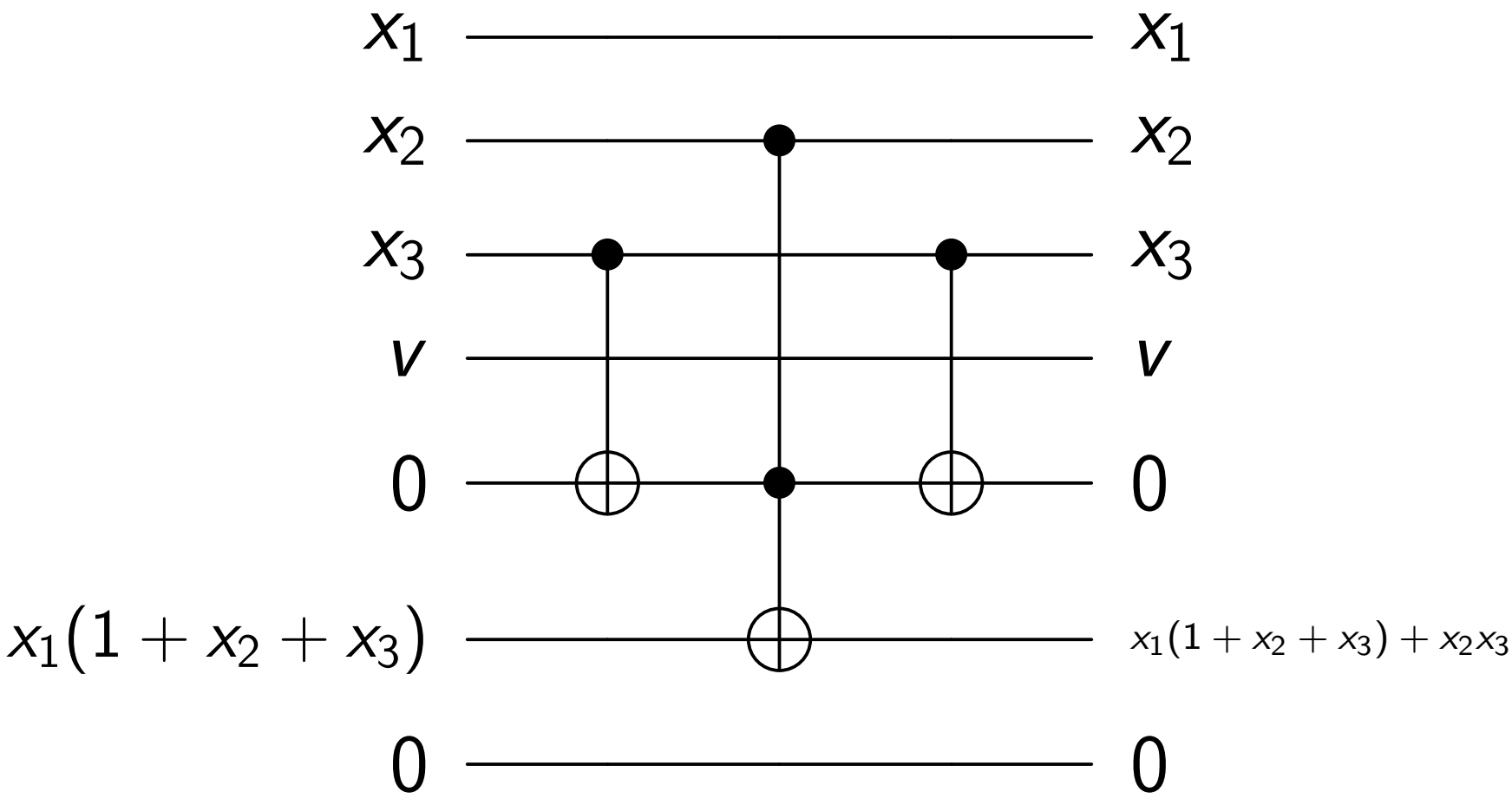
Building the circuit (3)



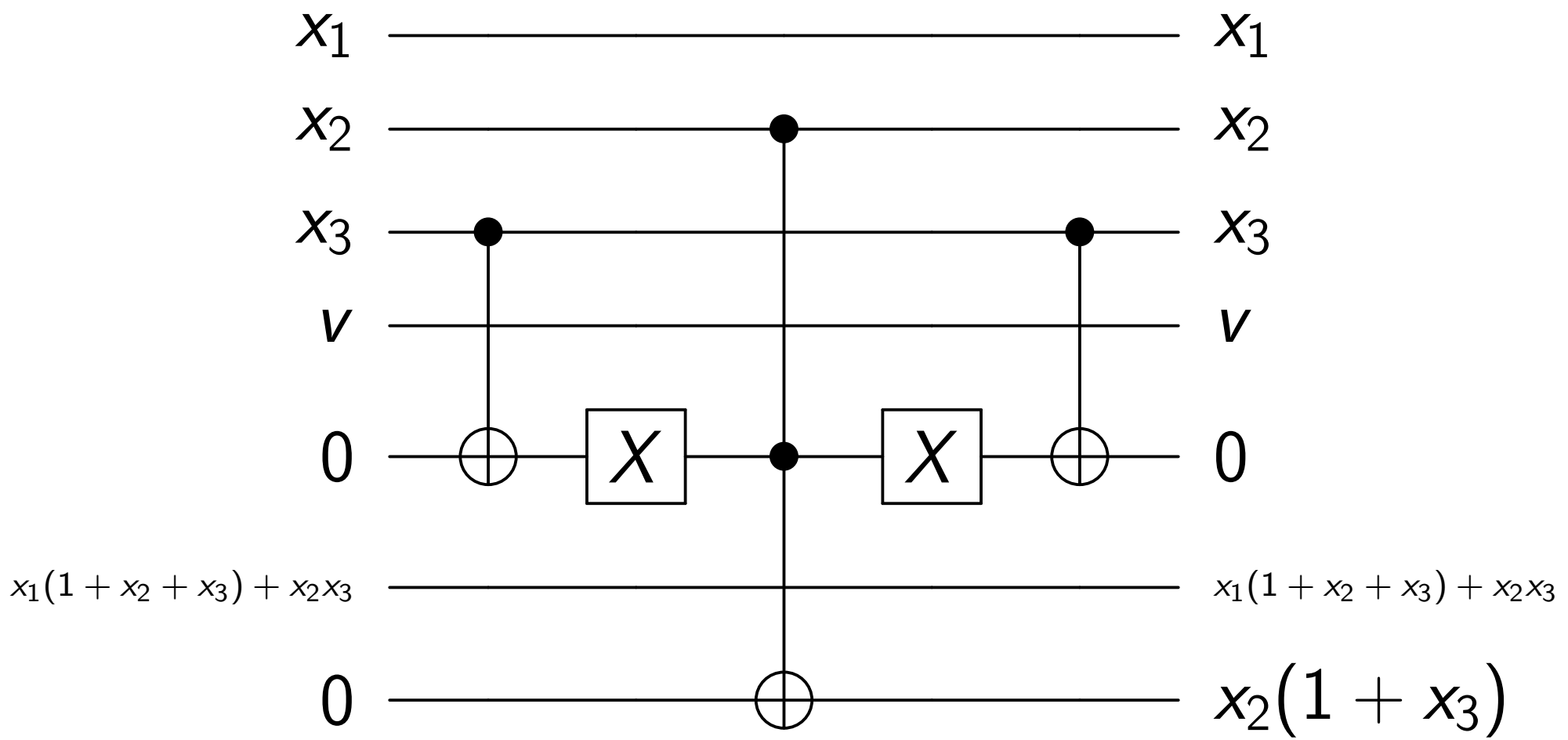
Building the circuit (4)



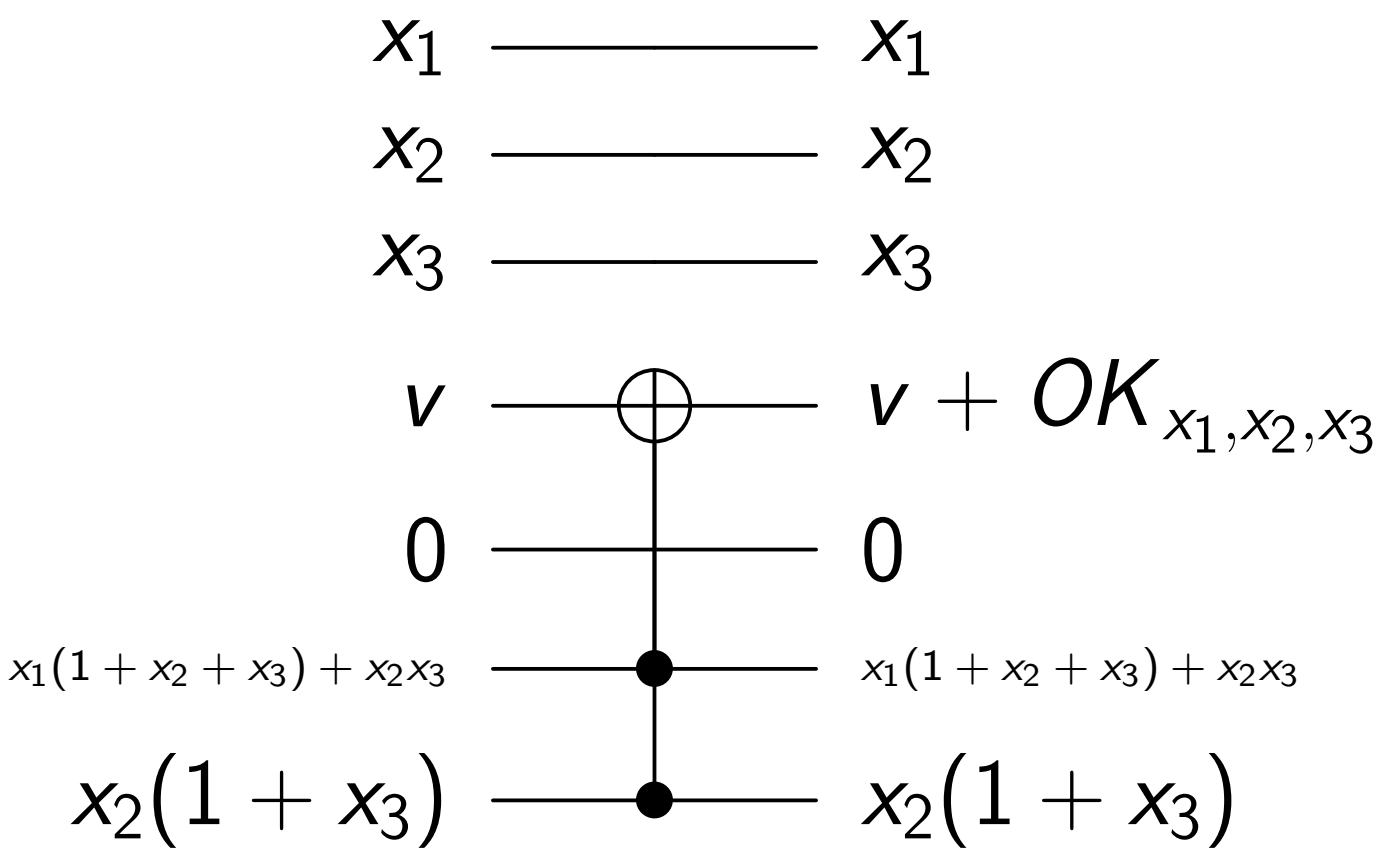
Building the circuit (5)



Building the circuit (6)



Building the circuit (7)



```haskell
-- Compute y_i^(k) from the coefficients lambda_ii^(k), ..., lambda_in^(k)
-- and the assignment x_1, ..., x_n.
compute_y :: [Bool] -> [Qubit] -> Circ (Qubit)
compute_y cs xs = withM (qinit False) $ \t ->
  unless (null cs) $ do
    when (head cs) $ qnot_at t
    zipWithM_ (\c x ->  when c (qnot_at t `controlled` x)) (tail cs) (tail xs)

-- Computes E^(k) from the "triangle" lambda^(k)_ij (1 <= j <= n)
-- and the assignment x_1, ..., x_n.
compute_e :: [Qubit] -> [[Bool]] -> Circ (Qubit)
compute_e xs css =
  withM (qinit False) $ \e ->
    forM_ (zip css (init $ tails xs))  $ \(cs, xs') ->
      with_computed (compute_y cs xs') $ \t ->
        qnot_at e `controlled` (t, head xs')

-- The first (straight-forward) oracle.  Computes whether the
-- assignment x_1, ..., x_n satisfies the given system of equations.
oracle1 :: [[[Bool]]] -> [Qubit] -> Circ (Qubit)
oracle1 csss xs =
  withM (qinit False) $ \r -> do
    label (r:xs) ("r":["x" ++ (show i) | i <- [1..length xs]])
    es <- mapM (compute_e xs) csss
    label es ["E" ++ (show i) | i <- [1..length es]]
    qnot_at r `controlled` es
```

Breaking $b\mathcal{MQ}$ $n = 80$ $m = 84$

	first
qubits	168
time	

Breaking $b\mathcal{MQ}$ $n = 80$ $m = 84$

	first	second
qubits	168	90
time		

Breaking $b\mathcal{MQ}$ $n = 80$ $m = 84$

	first	second
qubits	168	90
time	1,430,025,554,865,881,938	2,861,116,040,048,158,450

Breaking $b\mathcal{MQ}$ $n = 80$ $m = 84$

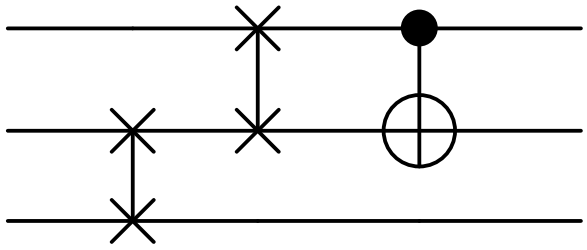
	first	second
qubits	168	90
time	2^{60}	2^{61}

Breaking $b\mathcal{MQ}$ $n = 80$ $m = 84$

	first	second	(third)
qubits	168	90	6484
serial time	2^{60}	2^{61}	2^{60}
parallel time	2^{60}	2^{61}	$\sim 2^{50}$

Effecient quantum counter

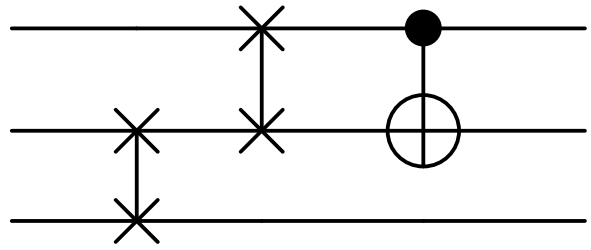
Create efficient counter from primitive polynomial over $\mathbb{F}_2$. E.g. $x^3 + x + 1$:



$$|111\rangle \mapsto |101\rangle \mapsto |100\rangle \mapsto |010\rangle$$
$$\mapsto |001\rangle \mapsto |110\rangle \mapsto |011\rangle \mapsto |111\rangle$$

Effecient quantum counter

Create efficient counter from primitive polynomial over $\mathbb{F}_2$. E.g. $x^3 + x + 1$:



$$|111\rangle \mapsto |101\rangle \mapsto |100\rangle \mapsto |010\rangle$$
$$\mapsto |001\rangle \mapsto |110\rangle \mapsto |011\rangle \mapsto |111\rangle$$

```
oracle2 :: [[[Bool]]] -> [Qubit] -> Circ (Qubit)
oracle2 csss xs = do
  ctr <- init_counter $ length csss
  label xs ["x" ++ (show i) | i <- [1..length xs]]
  forM_ csss $ \css ->
    with_computed (compute_e xs css) $ controlled $ inc_counter ctr
  check_counter ctr
```

Conclusion

1. Choice of parameters in ‘post-quantum’ crypto is important

Conclusion

1. Choice of parameters in ‘post-quantum’ crypto is important
2. ≤ 100 qubits already break some crypto

Conclusion

1. Choice of parameters in ‘post-quantum’ crypto is important
2. ≤ 100 qubits already break some crypto
3. Programming quantum computer isn’t that hard

Conclusion

1. Choice of parameters in ‘post-quantum’ crypto is important
2. ≤ 100 qubits already break some crypto
3. Programming quantum computer isn’t that hard
4. Writing efficient oracles underrated?