Quantum Algorithms for Exponentiation in a Group

Xavier Bonnetain Pierrick Gaudry **Medhi Kermaoui**

Centre Inria de l'Université de Lorraine
CARAMBA

Context and motivation

- Shor's algorithm $\implies$ broken public key cryptography
- NIST(US institute for standardisation) call for quantum resistant schemes
- Isogeny based crypto: exponentiation is a costly operation performed in some quantum attacks (Kuperberg for CSIDH)

Context and motivation

- Shor's algorithm $\Longrightarrow$ broken public key cryptography
- NIST(US institute for standardisation) call for quantum resistant schemes
- Isogeny based crypto: exponentiation is a costly operation performed in some quantum attacks (Kuperberg for CSIDH)

$\Longrightarrow$ Goal: improve the exponentiation to improve the quantum cryptanalysis/update security parameters for CSIDH

Context and motivation

- Shor's algorithm $\implies$ broken public key cryptography
- NIST(US institute for standardisation) call for quantum resistant schemes
- Isogeny based crypto: exponentiation is a costly operation performed in some quantum attacks (Kuperberg for CSIDH)

$\implies$ Goal: improve the exponentiation to improve the quantum cryptanalysis/update security parameters for CSIDH

Problem

*Let $(G, \times)$ be a finite group and N a **fixed** integer, from $|x\rangle$ we want to compute $|x^N\rangle$ "efficiently" on a quantum computer.*

The circuit model of computation

- Classical computing: algorithm can be represented as a boolean circuit
  $\Longrightarrow$ logic gates AND, OR and NOT acting on bits.

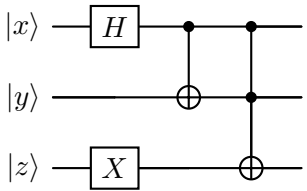
The circuit model of computation

- Classical computing: algorithm can be represented as a boolean circuit
  $\Longrightarrow$ logic gates AND, OR and NOT acting on bits.
- Quantum computing: **quantum algorithm** is represented as a **quantum circuit**
  $\Longrightarrow$ quantum gates (**reversible**) acting on qubits/registers.

The circuit model of computation

- Classical computing: algorithm can be represented as a boolean circuit
  $\Longrightarrow$ logic gates AND, OR and NOT acting on bits.
- Quantum computing: **quantum algorithm** is represented as a **quantum circuit**
  $\Longrightarrow$ quantum gates (**reversible**) acting on qubits/registers.

Example



How can we describe the complexity of a quantum circuit?

Metrics to measure the complexity of a circuit:

- the **size**: the total number of gates (time)
- the **width**: the number of qubits of the circuit (memory)
- the **depth**: the number of gates on the longest path (latency/ parallelisation)

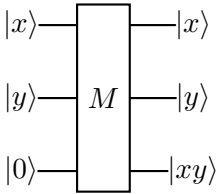
Outline of talk

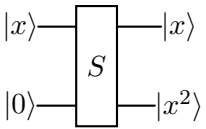
1. Algorithms from classical computing
   - Square and Multiply
   - Windowed version
2. Fibonacci exponentiation
   - In place multiplication gate
   - Fibonacci exponentiation
3. Hybrid algorithm
4. Analysis and comparison

Algorithms from classical computing

Square and Multiply

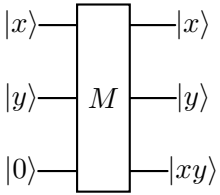
We suppose we have access to the **out of place** operations:

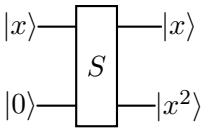




Square and Multiply

We suppose we have access to the **out of place** operations:

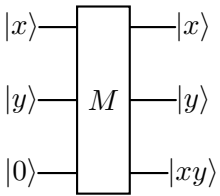


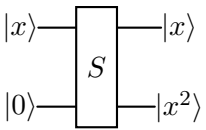


$\Longrightarrow$ Each time we use a S or M gate we add one register

Square and Multiply

We suppose we have access to the **out of place** operations:





$\Longrightarrow$ Each time we use a S or M gate we add one register

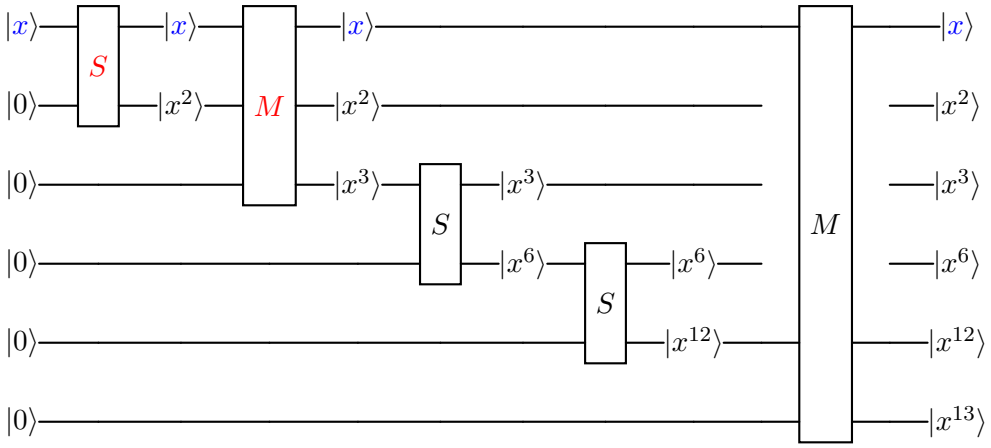
$\Longrightarrow$ for a random n bit integer N, around $3n/2$ registers/gates to compute $|x^N\rangle$

A circuit to compute $|x^{13}\rangle$

$13 = (1101)_2$ so we build the following circuit:

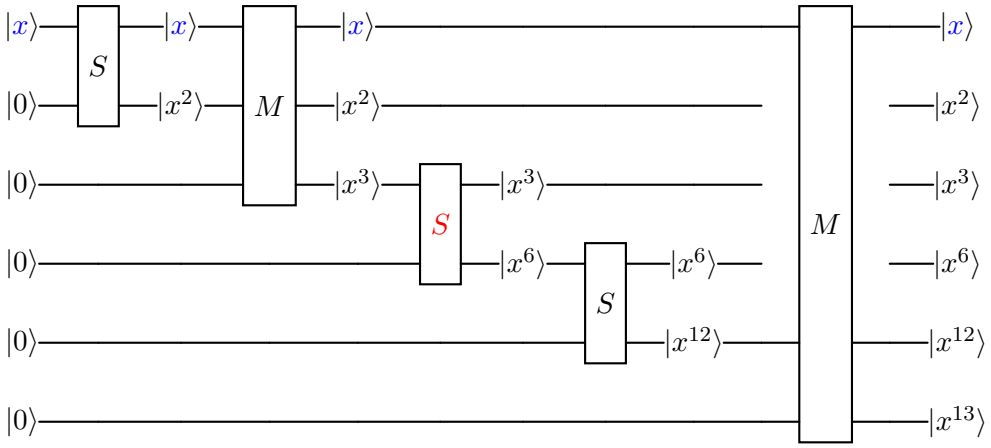
A circuit to compute $|x^{13}\rangle$

$13 = (1101)_2$ so we build the following circuit:



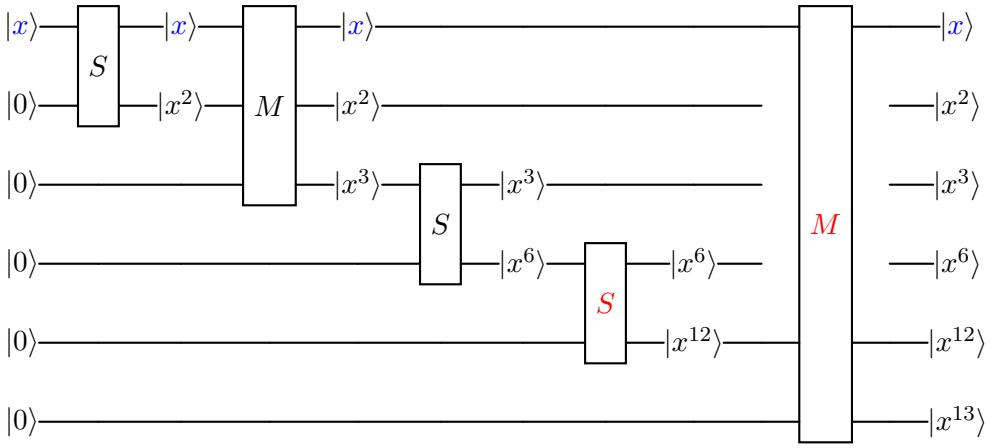
A circuit to compute $|x^{13}\rangle$

$13 = (1101)_2$ so we build the following circuit:



A circuit to compute $|x^{13}\rangle$

$13 = (1101)_2$ so we build the following circuit:



Windowed variants

Variant of Square and Multiply: time-space tradeoff $\implies$ reduce the number of multiplications (density of the decomposition) by using a precomputation table $\tilde{T}$ (takes space).

Algorithm 1 Window variants of SM

1: **Input:** $\tilde{T} = \{x^i, i \in T\}$ a table of precomputed exponents of x and a $T-$decomposition of $N = (n_{r-1}, \ldots, n_0)_T$
2: **Output:** The element $x^N \in G$.
3: $y \leftarrow 1$
4: **for** $i = (r-1) \cdots 0$ **do**
5: $\quad y \leftarrow y^2 \times x^{n_i}$ {*// skip mult. if* $n_i = 0$}
6: **end for**
7: **return** y

Windowed variants

Variant of Square and Multiply: time-space tradeoff $\implies$ reduce the number of multiplications (density of the decomposition) by using a precomputation table $\tilde{T}$ (takes space).

Algorithm 1 Window variants of SM

1: **Input:** $\tilde{T} = \{x^i, i \in T\}$ a table of precomputed exponents of x and a $T-$decomposition of $N = (n_{r-1}, \ldots, n_0)_T$
2: **Output:** The element $x^N \in G$.
3: $y \leftarrow 1$
4: **for** $i = (r-1) \cdots 0$ **do**
5: $\quad y \leftarrow y^2 \times x^{n_i}$ {*// skip mult. if* $n_i = 0$}
6: **end for**
7: **return** y

$\implies$ the number of registers/gates is still around $O(n)$

Question

Can we reduce the number of qubits used to compute $|x^N\rangle$?

Question

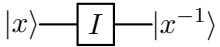
Can we reduce the number of qubits used to compute $|x^N\rangle$?

Yes! $\implies$ build a quantum exponentiation circuit using $O(1)$ registers.

Fibonacci exponentiation

In place multiplication gate [Jr17; RV24; Rag24]

As in the previous section, we suppose having access to the out of place S and M gates, but also the inversion gate

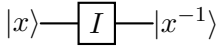


Notation

$|\widehat{x}\rangle := |x\rangle\,|x^{-1}\rangle$ *(augmented register).*

In place multiplication gate [Jr17; RV24; Rag24]

As in the previous section, we suppose having access to the out of place S and M gates, but also the inversion gate



Notation

$|\widehat{x}\rangle := |x\rangle\,|x^{-1}\rangle$ *(augmented register).*

Proposition

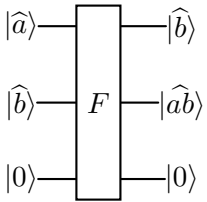
We can build an in place multiplication

$$F : |\widehat{a}\rangle\,|\widehat{b}\rangle\,|0\rangle \mapsto |\widehat{b}\rangle\,|\widehat{ab}\rangle\,|0\rangle$$

using at most 4 calls to M.

"$\boldsymbol{a \leftarrow b, b \leftarrow ab}$"

Fibonacci sequence appears

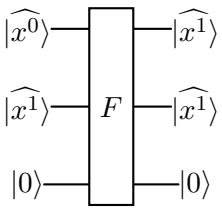


What happens if we iterate F multiple times?

$$"a \leftarrow b, b \leftarrow ab"$$

$$a = x^0$$
$$b = x^1$$

Fibonacci sequence appears

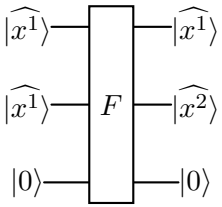


What happens if we iterate F multiple times?

$$"a \leftarrow b, b \leftarrow ab"$$

$$a = x^0, x^1$$
$$b = x^1, x^1$$

Fibonacci sequence appears

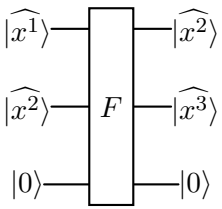


What happens if we iterate F multiple times?

$$"a \leftarrow b, b \leftarrow ab"$$

$$a = x^0, x^1, x^1$$
$$b = x^1, x^1, x^2$$

Fibonacci sequence appears



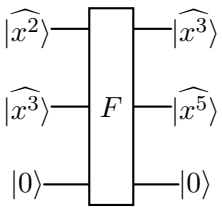
What happens if we iterate F multiple times?

$$\text{"}a \leftarrow b, b \leftarrow ab\text{"}$$

$$a = x^0, x^1, x^1, x^2$$

$$b = x^1, x^1, x^2, x^3$$

Fibonacci sequence appears

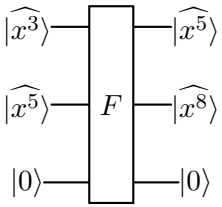


What happens if we iterate F multiple times?

$$"a \leftarrow b, b \leftarrow ab"$$

$$a = x^0, x^1, x^1, x^2, x^3$$
$$b = x^1, x^1, x^2, x^3, x^5$$

Fibonacci sequence appears

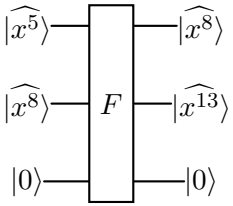


What happens if we iterate F multiple times?

$$"a \leftarrow b, b \leftarrow ab"$$

$$a = x^0, x^1, x^1, x^2, x^3, x^5$$
$$b = x^1, x^1, x^2, x^3, x^5, x^8$$

Fibonacci sequence appears

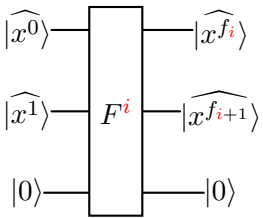


What happens if we iterate F multiple times?

$$\text{"}a \leftarrow b, b \leftarrow ab\text{"}$$

$$a = x^0, x^1, x^1, x^2, x^3, x^5, x^8$$
$$b = x^1, x^1, x^2, x^3, x^5, x^8, x^{13}$$

Fibonacci sequence appears



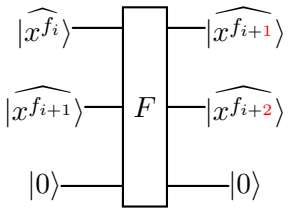
What happens if we iterate F multiple times?

$$\text{"}a \leftarrow b, b \leftarrow ab\text{"}$$

$$a = x^0, x^1, x^1, x^2, x^3, x^5, x^8 \dots$$
$$b = x^1, x^1, x^2, x^3, x^5, x^8, x^{13} \dots$$

Fibonacci sequence $f_0 = 0, f_1 = 1$
$f_{i+2} = f_{i+1} + f_i$ for $i \geq 0$

Fibonacci sequence appears



What happens if we iterate F multiple times?

$$\text{"}a \leftarrow b, b \leftarrow ab\text{"}$$

$$a = x^0, x^1, x^1, x^2, x^3, x^5, x^8 \dots$$
$$b = x^1, x^1, x^2, x^3, x^5, x^8, x^{13} \dots$$

Fibonacci sequence $f_0 = 0, f_1 = 1$
$f_{i+2} = f_{i+1} + f_i$ for $i \geq 0$

$\implies$ increment the Fibonacci sequence

How to compute $|x^N\rangle$ using the gate F?

If N is a Fibonacci number, it is clear how to proceed, but if not we rely on the following theorem:

Theorem (Zeckendorf representation [Zec72])

Let N be a positive integer and $(f_i)_{i\geq 0}$ the Fibonacci sequence, then N can be uniquely written as

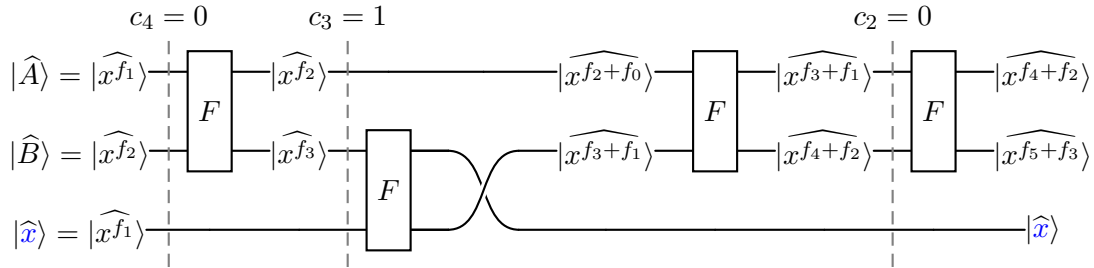
$$N = \sum_{i=2}^{k} c_i f_i$$

with $c_i \in \{0,1\}$ and $c_{i+1}c_i = 0$. We write by $N = (c_k, \ldots, c_2)_Z$ this representation.

$\Longrightarrow$ greedy algorithm to get the decomposition

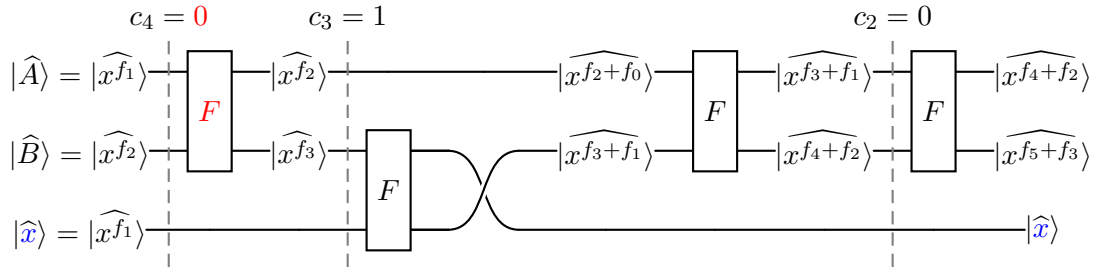
An example with the computation of $|x^7\rangle$

$7 = f_5 + f_3 = (1010)_Z$



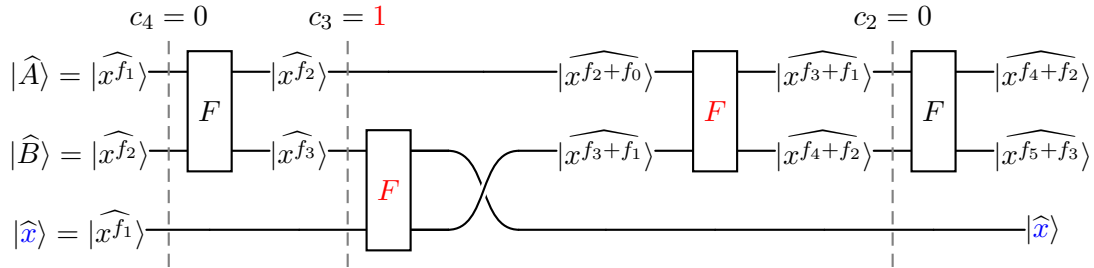
An example with the computation of $|x^7\rangle$

$7 = f_5 + f_3 = (1010)_Z$



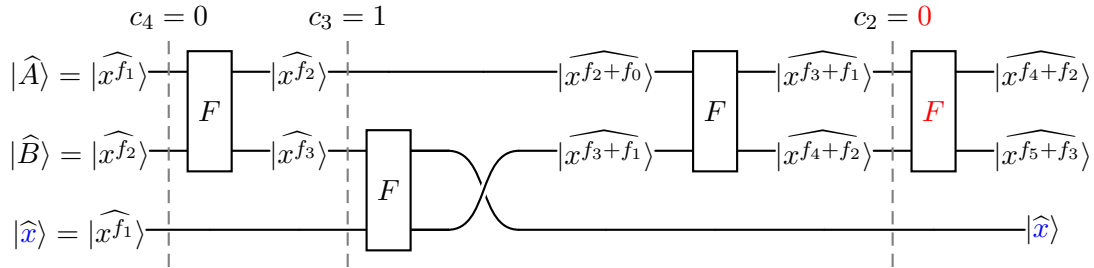
An example with the computation of $|x^7\rangle$

$7 = f_5 + f_3 = (1010)_Z$

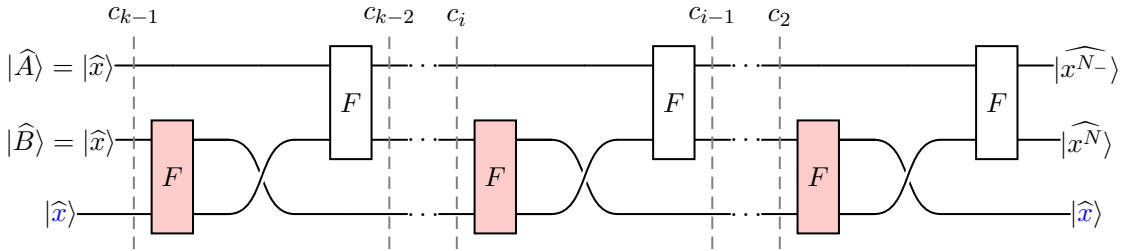


An example with the computation of $|x^7\rangle$

$7 = f_5 + f_3 = (1010)_Z$

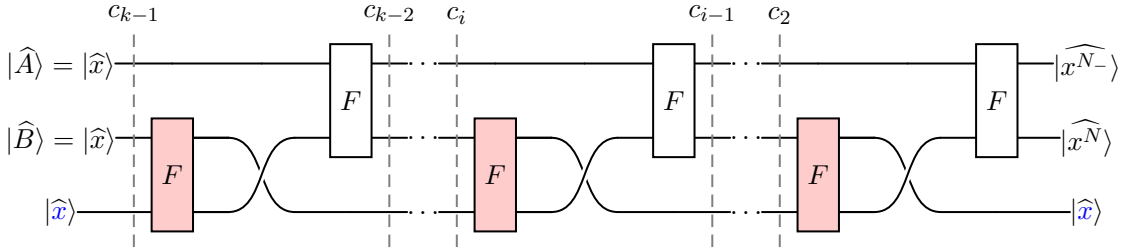


FibExp, with $N = (c_k, \ldots, c_2)_Z$



$|\widehat{x}\rangle$ acts as precomputed table

FibExp, with $N = (c_k, \ldots, c_2)_Z$



$|\widehat{x}\rangle$ acts as precomputed table

Cost of the Fibonacci exponentiation

- Size: $k - 3 + h_Z$ calls to F
- Width: 3 augmented registers and one ancilla register ($|0\rangle$ used for F)

where h_Z denote the Hamming weight of $(c_k, \ldots, c_2)$.

Comparison SM / FibExp

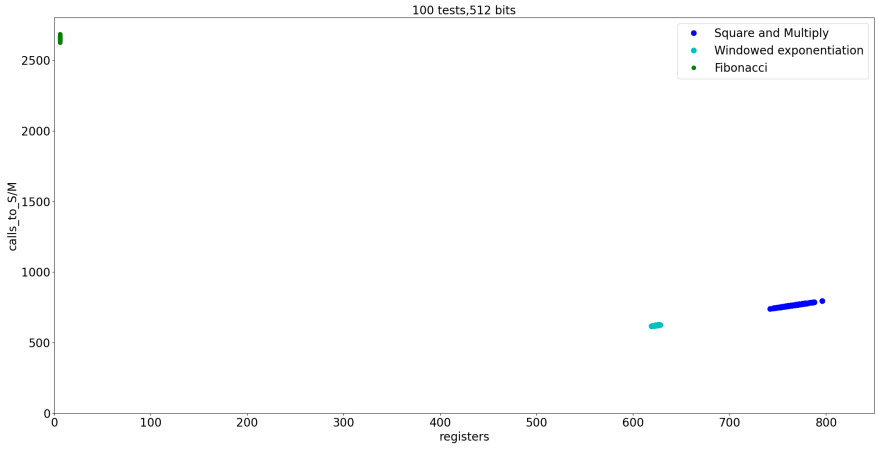


Figure: Cost to perform exponentiation for (random) N of 512-bits (finite field)

Hybrid algorithm

Building block: SMF_c circuit

Notation

$|\widehat{A}\rangle \simeq |\widehat{x^A}\rangle$ *and* $F : |\widehat{A}\rangle\,|\widehat{B}\rangle \mapsto |\widehat{A+B}\rangle\,|\widehat{B}\rangle$

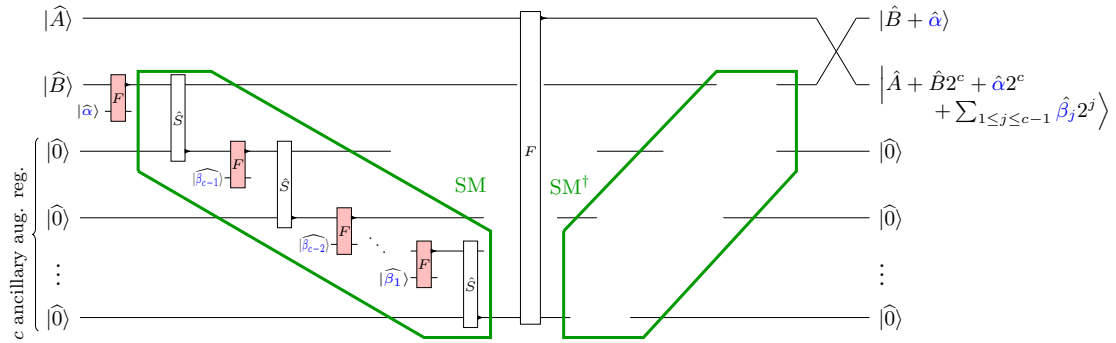


Figure: SMF_c circuit. The extra inputs of the red F-gates are taken from a precomputed table T.

The general idea

Our general circuit $\mathsf{GenCirc}_{c,w,n,d}$ proceeds as follows:

1. Compute a table T of all small multiples up to w in augmented representation, and initialize our **working registers** $|\widehat{A}\rangle$ and $|\widehat{B}\rangle$ to the neutral group element.
2. Apply n times the SMF_c circuit.
3. Apply a SM sequence of length d to $|\widehat{B}\rangle$, with a time-space trade-off.

Performance of the circuit

- If c is low, it performs like the Fibonacci (window) exponentiation
- If c is high (around $log_2(N)$), it performs like the Windowed exponentiation

Decomposition

The output value $|\widehat{B}\rangle$ of $\mathsf{GenCirc}_{c,w,n,d}$ circuit is

$$N = \underbrace{2^{d-c}\sum_{i=1}^{n}\left(\alpha_i 2^c g_i + \sum_{j=1}^{c-1}\beta_{i,j}\tilde{h}_{i,j}\right)}_{\mathsf{SMF}_c} + \underbrace{\sum_{i=0}^{d}\gamma_i 2^i}_{SM}$$

where

- $g_{i+2} = 2^c g_{i+1} + g_i$
- $\tilde{h}_{i,j} = 2^{c+j} g_{i-1}$

Decomposition

The output value $|\widehat{B}\rangle$ of $\mathsf{GenCirc}_{c,w,n,d}$ circuit is

$$N = \underbrace{2^{d-c}\sum_{i=1}^{n}\left(\alpha_i 2^c g_i + \sum_{j=1}^{c-1}\beta_{i,j}\tilde{h}_{i,j}\right)}_{\mathsf{SMF}_c} + \underbrace{\sum_{i=0}^{d}\gamma_i 2^i}_{SM}$$

where

- $g_{i+2} = 2^c g_{i+1} + g_i$
- $\tilde{h}_{i,j} = 2^{c+j} g_{i-1}$

$\Longrightarrow$ Greedy algorithm to get the decomposition

Analysis and comparison

Finite field

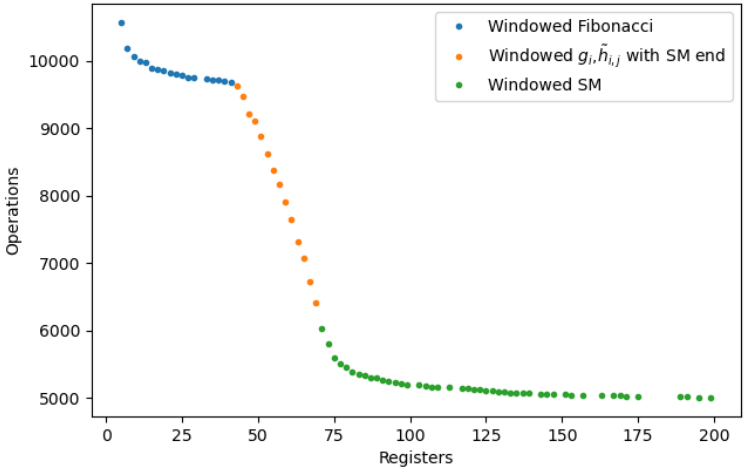


Figure: Tradeoff for an N of 2048 bits, $w \leq 50$, with explicit inverses and F costs $3S$.

EC, $F = 2S$

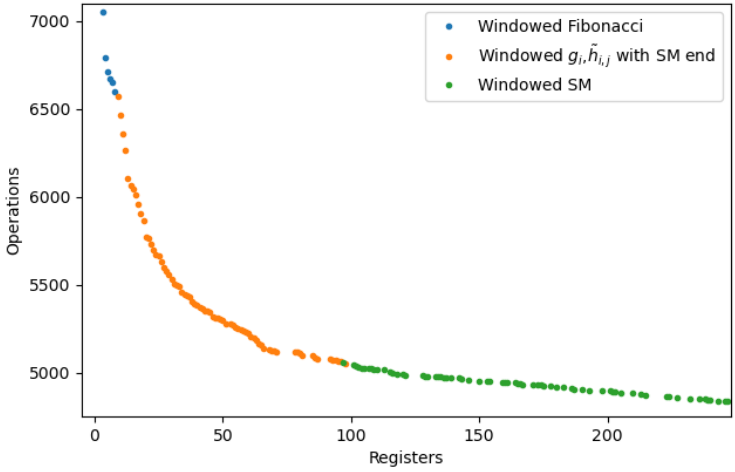


Figure: Tradeoff for an N of 2048 bits, $w \leq 50$, with a free inverse and $F = 2S$.

EC, $F = S$

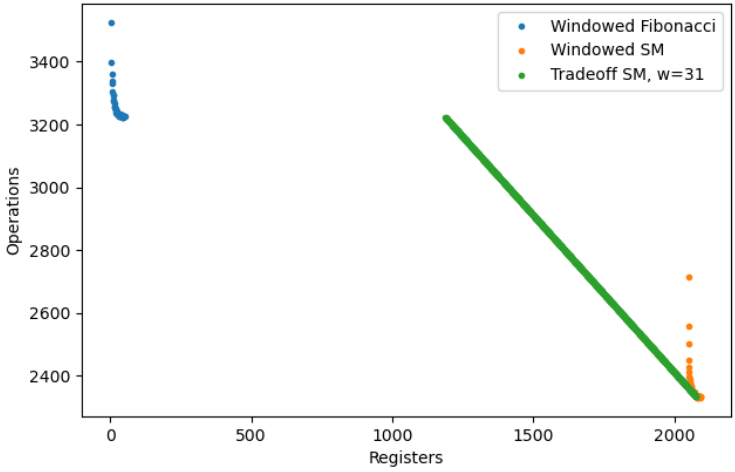


Figure: Tradeoff for an N of 2048 bits, $w \leq 50$, with a free inverse and comparable costs for S and F.

Key takeways

Previously:

- If restricted memory: FibExp
- If intermediate memory: FibExp
- If unrestricted memory: Window variants of SM

This work: Modular circuit that leverages all the available memory: interpolating smoothly between FibExp and variants of SM.

Key takeways

Previously:

- If restricted memory: FibExp
- If intermediate memory: FibExp
- If unrestricted memory: Window variants of SM

This work: Modular circuit that leverages all the available memory: interpolating smoothly between FibExp and variants of SM.

Future works?

- Spooky pebbling games (allow measurements during the computation)
- Exponent and value in superposition?
- Apply this work to the cryptanalysis of isogeny-based schemes (Qarton)

Key takeways

Previously:

- If restricted memory: FibExp
- If intermediate memory: FibExp
- If unrestricted memory: Window variants of SM

This work: Modular circuit that leverages all the available memory: interpolating smoothly between FibExp and variants of SM.

Future works?

- Spooky pebbling games (allow measurements during the computation)
- Exponent and value in superposition?
- Apply this work to the cryptanalysis of isogeny-based schemes (Qarton)

Thank you for your attention!

[Jr17] Burton S. Kaliski Jr. Targeted Fibonacci Exponentiation. 2017. arXiv: 1711.02491 [math.NT]. URL: https://arxiv.org/abs/1711.02491.

[Rag24] Seyoon Ragavan. "Regev Factoring Beyond Fibonacci: Optimizing Prefactors". In: Cryptology ePrint Archive (2024).

[RV24] Seyoon Ragavan and Vinod Vaikuntanathan. "Space-Efficient and Noise-Robust Quantum Factoring". In: Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology C Ed. by Leonid Reyzin and Douglas Stebila. Vol. 14925. Lecture Notes in Computer Science. Springer, 2024, pp. 107–140. DOI: 10.1007/978-3-031-68391-6_4. URL: https://doi.org/10.1007/978-3-031-68391-6%5C_4.

[Zec72] Édouard Zeckendorf. "Representations des nombres naturels par une somme de nombres de fibonacci on de nombres de lucas". In: Bulletin de La Society Royale des Sciences de Liege (1972), pp. 179–182.

Time-space tradeoff

① compute $U_1, \ldots, U_X$

② uncompute $U_{X-1}, \ldots, U_1$

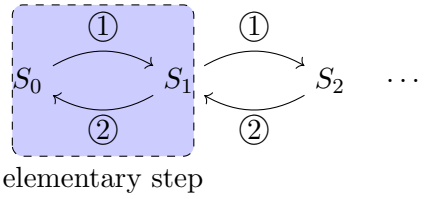


Figure: Non-asymptotic variant of Bennett's time-space tradeoffs. The elementary step, represented in blue, consumes one register and is repeated until the memory is filled.

Asymptotic costs

Algorithm	# registers	# gates
SM	$1.5n\,R$	$nS + 0.5nM$
SM-wNAF	$(1 + O(\frac{1}{\log n}))n\,R$	$nS + O(\frac{n}{\log n})M$
SM-Bennett	$\sqrt{n}\,R$	$2nS + o(n)M$
Fibonacci	$3\,\hat{R} + 1\,R$	$1.73nF$
Ours (no window)	$(c+3)\,\hat{R} + 1\,R$	$2n\hat{S} + \frac{2n}{3}(1 + \frac{1}{c})F$
(window)	$k\,\hat{R}$	$2n\hat{S} + \frac{2n}{\log k}F$

Figure: Summary of asymptotic costs for an exponentiation by a number N of n bits. The "hat" notation corresponds to augmented registers, or operations that deal with them. In the case where inversions are non-free, they are typically doubled compared to non-hat version.

Greedy algorithm to optimize the "cost per bit gained"

Ignoring insertions, we have 4 main cases:

- Square-multiply, costs 1 S/bit,
- Level-1 tradeoff, $\sqrt{n}$-memory square-multiply, costs 2 S/bit,
- Fibonacci, costs $\log_\phi(2) \simeq 1.44$ F/bit,
- Our generalization, costs $\frac{1}{c}$ F/bit and 2 $\hat{S}$/bit.

Thus, if F costs roughly the same as S, Fibonacci is optimal, except compared with a memory-inefficient square-multiply with $\mathcal{O}(n)$ registers. In this case the only way to meaningfully reduce the time complexity is to add windowing.