A resource estimation framework for quantum attacks against cryptographic functions - improvements

GRI quantum risk assessment report Sep. 2017 - Feb. 2018

Vlad Gheorghiu and Michele Mosca

evolutionQ Inc., Waterloo ON, Canada

February 28, 2018

Abstract. We extend our analysis of the security of several symmetric-key cryptographic functions and hash functions against an attack from a full-scale fault-tolerant quantum computer. We investigate the advantages provided by parallelizing Grover's algorithm and compute the security parameters, wall-time and physical footprint as a function of number of processors for each cryptographic primitive, assuming a fault-tolerant implementation on a surface code architecture, as a function of various physical error rates ranging from 10^{-3} to 10^{-7}.

1 Introduction

Symmetric cryptography is a fundamental pillar of modern cryptography. It includes symmetric-key encryption, where a shared secret key is used for both encryption and decryption. It also includes cryptographic hash functions, which map arbitrarily long strings to strings of a fixed finite length. In contrast to asymmetric schemes (or public-key schemes) based on factoring or solving the discrete logarithm problem and which are completely broken by a quantum adversary via Shor's algorithm [1], symmetric schemes and hash functions are less vulnerable to quantum attacks. The best known quantum attacks against them are based on Grover's quantum search algorithm [2], which offers a quadratic speedup compared to classical brute force searching. Given a search space of size N, Grover's algorithm finds, with high probability, an element x for which a certain property such as $f(x) = 1$ holds, for some function f we know how to evaluate (assuming such a solution exists). The algorithm evaluates f a total of $\mathcal{O}(\sqrt{N})$ times. It applies a simple operation in between the evaluations of f, so the $\mathcal{O}(\sqrt{N})$ evaluations of f account for most of the complexity. In contrast, any classical algorithm that evaluates f in a similar "black-box" way requires on the order of N evaluations of f to find such an element.

In [3] we analyzed the security of symmetric schemes and hash functions against quantum adversaries. In the current report we improve the analysis by studying the effects of parallelization on Grover's searching algorithm, down to the fault-tolerant layer. Naively, one might hope that K quantum computers (or quantum "processors", as we will call them later in the paper) running in parallel reduce the number of steps in Grover down to $\mathcal{O}(\sqrt{N})/K$ steps, similar to the classical case of distributing a search space across K classical processors. However Grover's algorithm does not parallelize so well, and the required number of steps for parallel quantum searching is of the order $\mathcal{O}(\sqrt{N/K})$ [4]. This is a factor of $\sqrt{K}$ larger than $\mathcal{O}(\sqrt{N})/K$. As show in [4], the optimal way of parallelizing Grover's algorithm is to partition the search space into N/K parts, and to perform independent quantum searches on each part.

In the current report we investigate the effects of parallelization on various parameters, such as circuit physical footprint (number of qubits), total running time and security parameter. As in our previous report, we assume a surface-code based fault-tolerant architecture [5], using Reed-Muller distillation schemes [6]. For each scheme we vary the possible physical error rates per gate from 10^{-4} to 10^{-7}. We believe that this range of physical error rates is wide enough to cover both first generation quantum computers as well as more advanced future machines. We also believe that it is highly unlikely that the physical error rates will be reduced to less than 10^{-7}.

2 Methodology

The methodology, sketched in Fig. 1 and Fig. 2, follows the same lines as the one described in detail in our earlier report [3]. We refer the reader to the Ref. [3] for more details. All of our results were produced via our Python resource estimate software toolkit.

For each cryptographic primitive, we display four plots, in the following order:

1. We plot the total number of surface code cycles per CPU (where a CPU is a quantum computer capable of executing a single instance of Grover's quantum search algorithm) as a function of the number of CPUs. We directly tie the quantum security parameter to the total number of surface code cycles (see [3] for more details). We also add to the plot the theoretical lower bound achievable by quantum search in the cases of: a) considering the oracle a black box of unit cost (lower line), and b) considering the oracle as composed of ideal quantum gates, each of unit cost (upper line). Note that the difference between b) and a) represents the intrinsic cost of logical overhead (i.e. the overhead introduced by treating the oracle as a logical circuit and not a blackbox), whereas the difference between the upper lines and b) represents the intrinsic cost introduced by the fault-tolerant layer.
2. We plot the total wall-time per CPU (i.e. how long will the whole computation take on a parallel quantum architecture) as a function of the number of CPUs. The horizontal dashed line represents the one-year time line, i.e. the

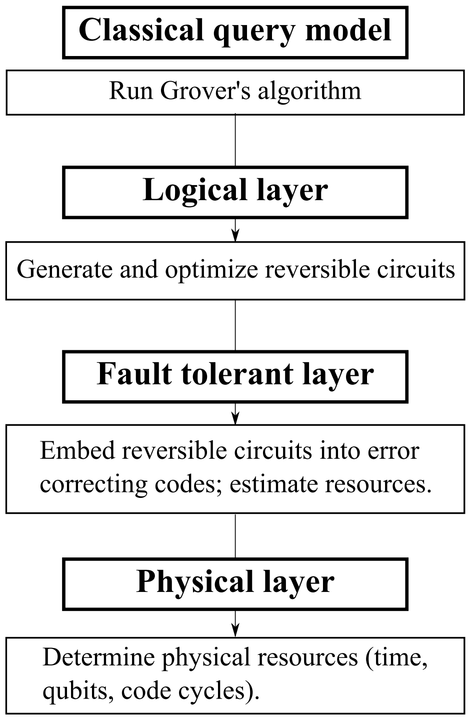
Classical query model
Run Grover's algorithm
Logical layer
Generate and optimize reversible circuits
Fault tolerant layer
Embed reversible circuits into error correcting codes; estimate resources.
Physical layer
Determine physical resources (time, qubits, code cycles).

Fig. 1. Analyzing an attack against a symmetric cryptographic function with a fault-tolerant quantum adversary. Our resource estimation methodology takes into account several of the layers between the high level description of an algorithm and the physical hardware required for its execution. Our approach is modular should assumptions about any of these layers change, and hence it allows one to calculate the impact of improvements in any particular layer.

X coordinate of the intersection point between the "Total time per CPU" line and the one-year time line provides the number of processors required to break the system within one year (in $\log_2$ units).

3. We plot the total physical footprint (number of qubits) per CPU, as a function of the number of CPUs.
4. Finally we plot the total physical footprint (number of qubits) of all quantum search machines (CPUs) running in parallel.

In the following sections we proceed to analyze symmetric ciphers (AES, Sec. 3), hash functions (SHA-256, SHA3-256, Sec. 4, Bitcoin's hash function, Sec. 5), and finally the minimal resources required for running Grover's algorithm with a trivial oracle 6 (e.g. the identity gate) on search spaces of various sizes.

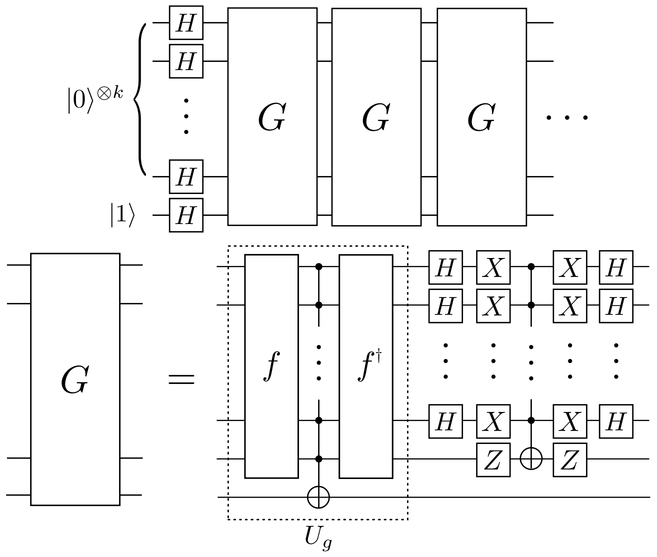

Fig. 2. Grover searching with an oracle for $f : \{0,1\}^k \to \{0,1\}^k$. The algorithm makes $\lfloor \frac{\pi}{4} 2^{N/2} \rfloor$ calls to G, the *Grover iteration*, or, if parallelized on K processors, $\lfloor \frac{\pi}{4} 2^{N/(2K)} \rfloor$ calls to G. The Grover iteration has two subroutines. The first, U_g, implements the predicate $g : \{0,1\}^k \to \{0,1\}$ that maps x to 1 if and only if $f(x) = y$. Each call to U_g involves two calls to a reversible implementation of f and one call to a comparison circuit that checks whether $f(x) = y$.

3 Symmetric ciphers

Below we analyze the effect of parallelization on the AES family of symmetric ciphers. We used the highly optimized logical circuits produced in [7].

3.1 AES-128

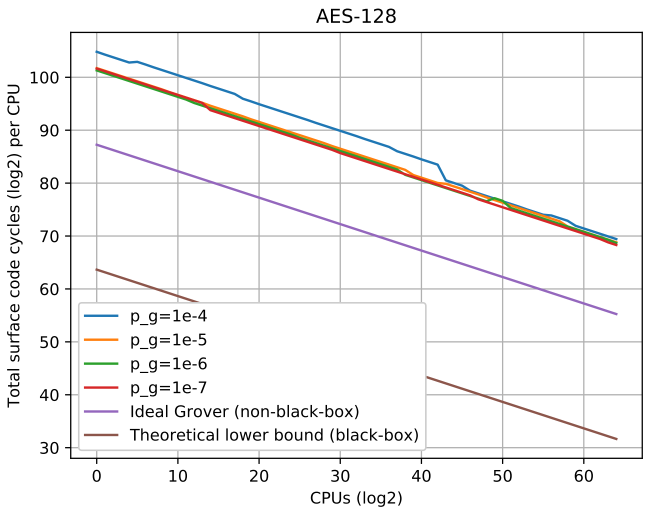

Fig. 3. AES-128 block cipher. Required surface clock cycles per processor, as a function of the number of processors ($\log_2$ scale). The bottom brown line (theoretical lower bound, black box) represents the minimal number of queries required by Grover's algorithm, the cost function being the total number of queries to a black-box oracle, each query assumed to have unit cost, and a completely error-free circuit. The purple line (ideal grover, non-black-box) takes into consideration the structure of the oracle, the cost function being the total number of gates in the circuit, each gate having unit cost; the quantum circuit is assumed error-free as well. The curves above the purple line show the overhead introduced by fault tolerance (in terms of required surface code cycles, each surface code cycle assumed to have unit cost). More optimization at the logical layer will shift the purple line down, whereas more optimization at the fault-tolerant layer will move the upper curves closer to the purple line. Similar remarks to the above hold for the remaining plots in this manuscript.

For example, the plots in Fig. 3 tells us that if we have 2^{50} quantum computers running Grover's algorithm in parallel, with no physical errors, then it would take about 2^{63} gate calls (where the purple line intersects the vertical line at 50), where we assume each gate to have unit cost. Still with no errors, a trivial cost for implementing the cryptographic function (oracle) would bring the cost down to about 2^{38} oracle calls. Keeping the actual function implementation, but adding the fault-tolerant layer with a physical error rate of 10^{-7} (with appropriate assumptions and using state-of-the-art quantum error correction) pushes the cost up to around 2^{76} surface code cycles (where now each code cycle is assumed to have unit cost). Similar remarks hold for the remaining plots in this manuscript.

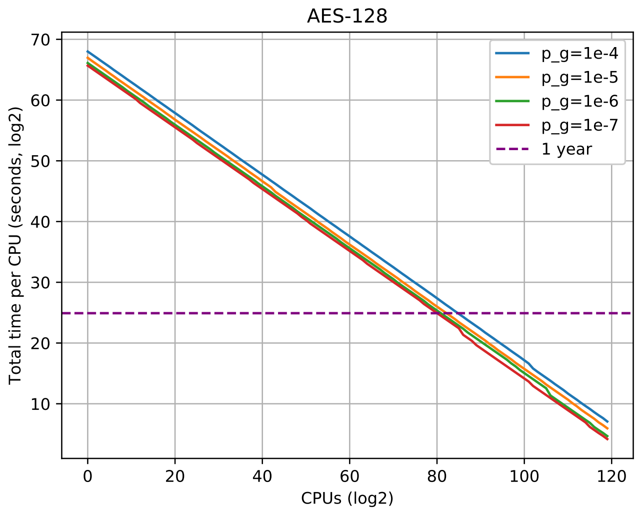

Fig. 4. AES-128 block cipher. Required time per processor, as a function of the number of processors ($\log_2$ scale). The horizontal dotted line indicates one year. The X-axis is deliberately extended to show the necessary number of CPUs for a total time of one year. Thus the figure shows that it would take, with the stated assumptions, over 2^{80} parallel quantum searches to break AES-128 in a year. Similar remarks to the above hold for the remaining plots in this manuscript.

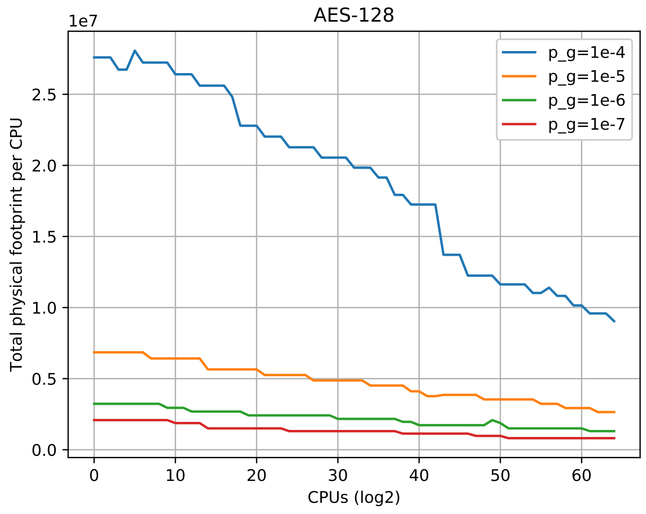

Fig. 5. AES-128 block cipher. Physical footprint (physical qubits) per processor, as a function of the number of processors ($\log_2$ scale). Note that in some ranges of the plot the total physical footprint increases slightly with the number of processors, which may seem counter-intuitive. This happens due to the fact that with more processors the required code distances decrease, and in some instances one can pipeline more distilleries in parallel into the surface code, which in effect causes an increase in the overall physical footprint. Note that the total time per CPU is monotonically decreasing, as parallelizing distilleries does not increase the wall time. For more details see [8]. Similar remarks to the above hold for the remaining plots in this manuscript.

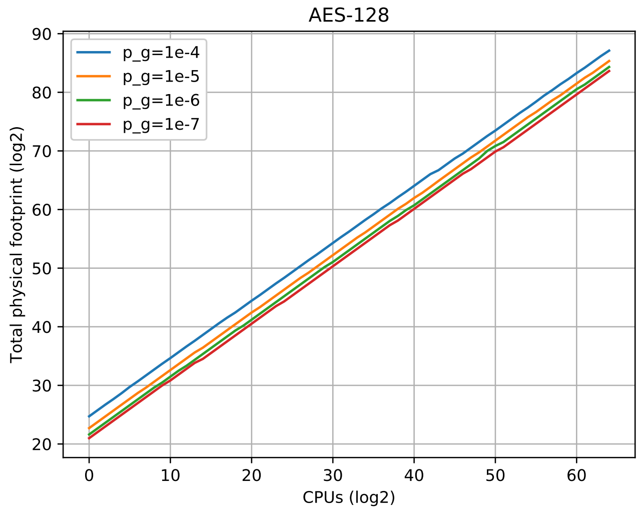

Fig. 6. AES-128 block cipher. Total physical footprint (physical qubits), as a function of the number of processors ($\log_2$ scale). Note that the qubits are not correlated across processors.

3.2 AES-192

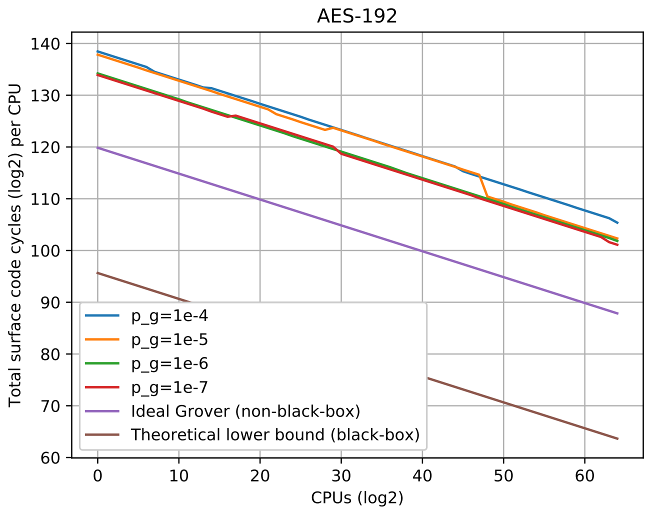

Fig. 7. AES-192 block cipher. Required surface clock cycles per processor, as a function of the number of processors ($\log_2$ scale).

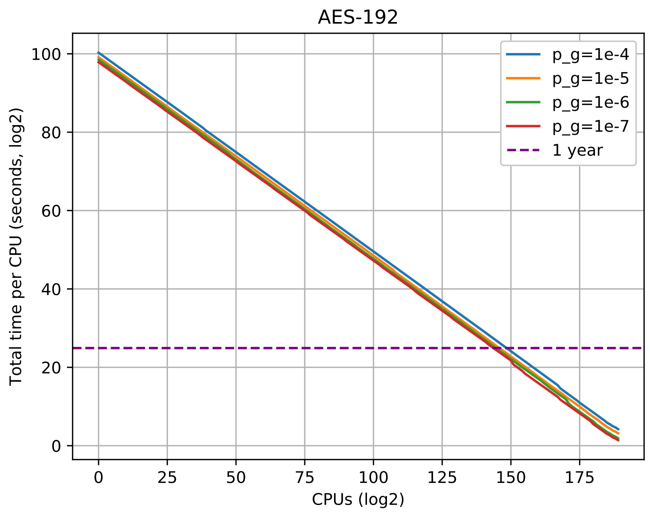

Fig. 8. AES-192 block cipher. Required time per processor, as a function of the number of processors ($\log_2$ scale).

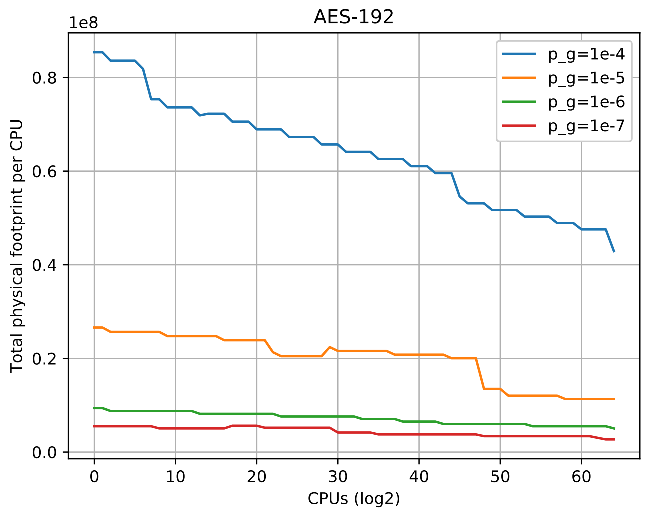

Fig. 9. AES-192 block cipher. Physical footprint (physical qubits) per processor, as a function of the number of processors ($\log_2$ scale).

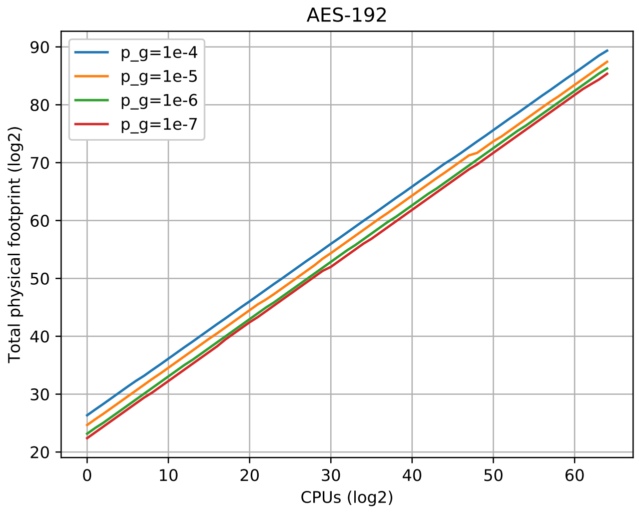

Fig. 10. AES-192 block cipher. Total physical footprint (physical qubits), as a function of the number of processors ($\log_2$ scale). Note that the qubits are not correlated across processors.

3.3 AES-256

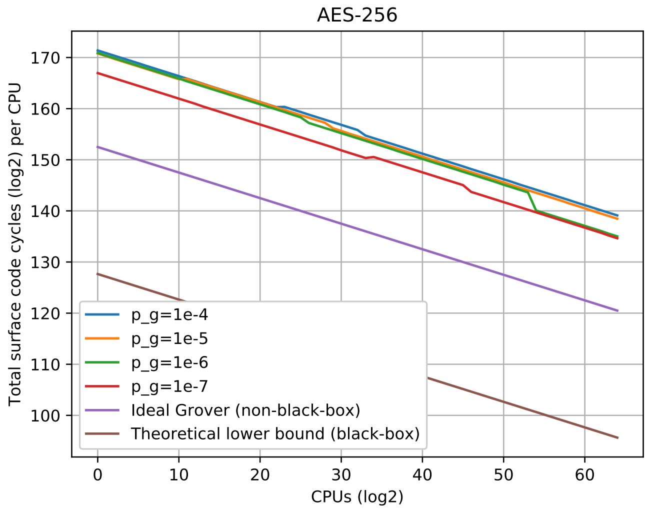

Fig. 11. AES-256 block cipher. Required surface clock cycles per processor, as a function of the number of processors ($\log_2$ scale).

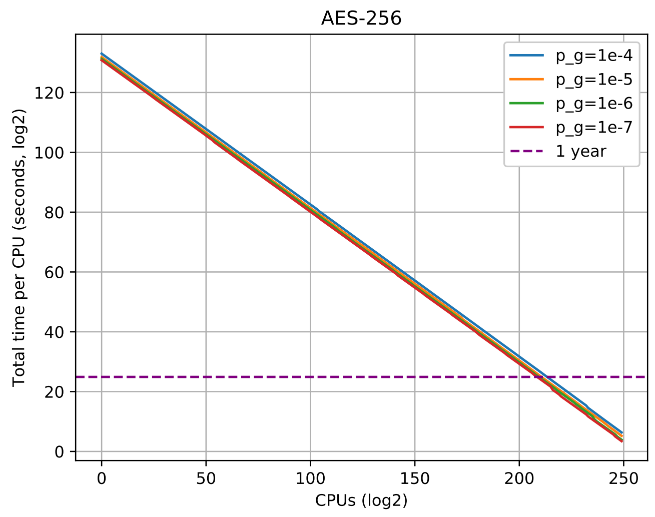

Fig. 12. AES-256 block cipher. Required time per processor, as a function of the number of processors ($\log_2$ scale).

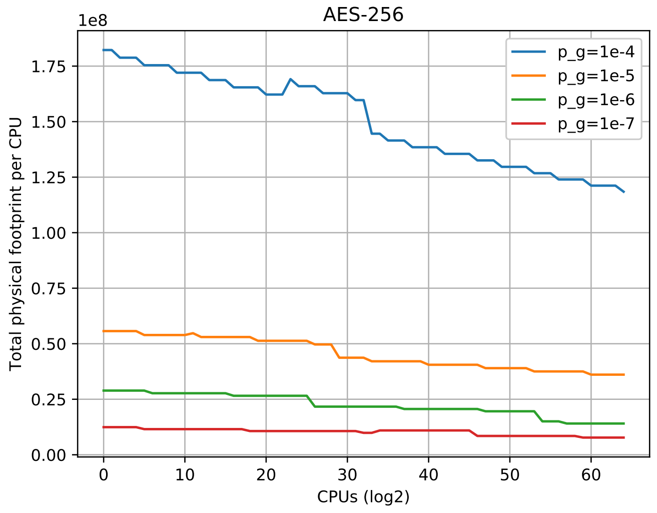

Fig. 13. AES-256 block cipher. Physical footprint (physical qubits) per processor, as a function of the number of processors ($\log_2$ scale).

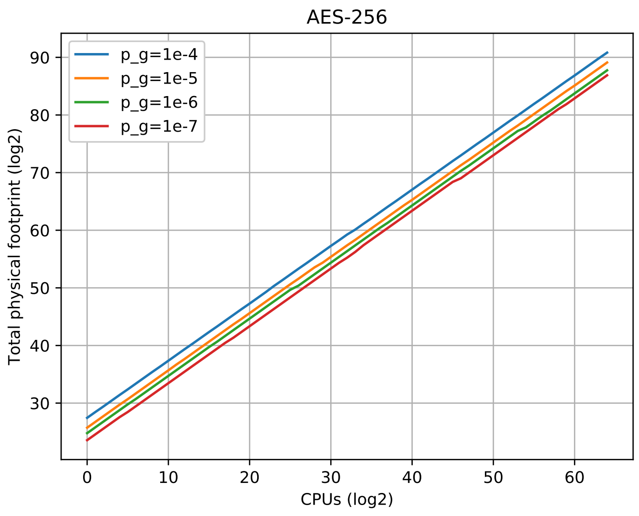

Fig. 14. AES-256 block cipher. Total physical footprint (physical qubits), as a function of the number of processors ($\log_2$ scale). Note that the qubits are not correlated across processors.

4 Hash functions

In this section we study the effect of Grover parallelization on the SHA-256[9] snd SHA3-256[10] family of hash functions. We used the highly optimized logical circuits produced in [8].

4.1 SHA-256

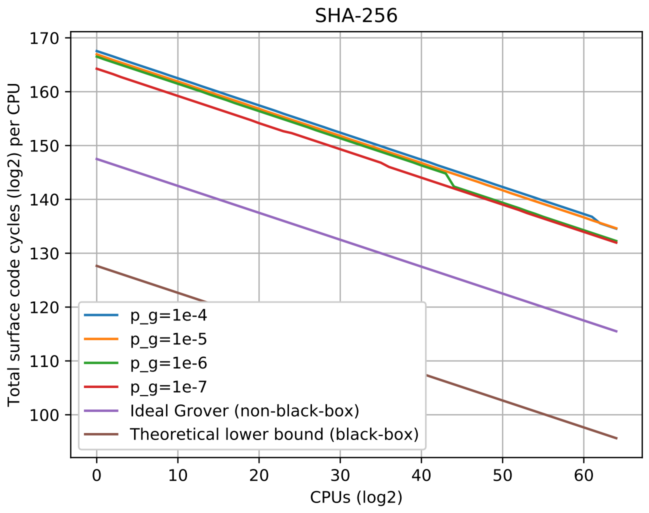

Fig. 15. SHA-256 cryptographic hash function. Required surface clock cycles per processor, as a function of the number of processors ($\log_2$ scale).

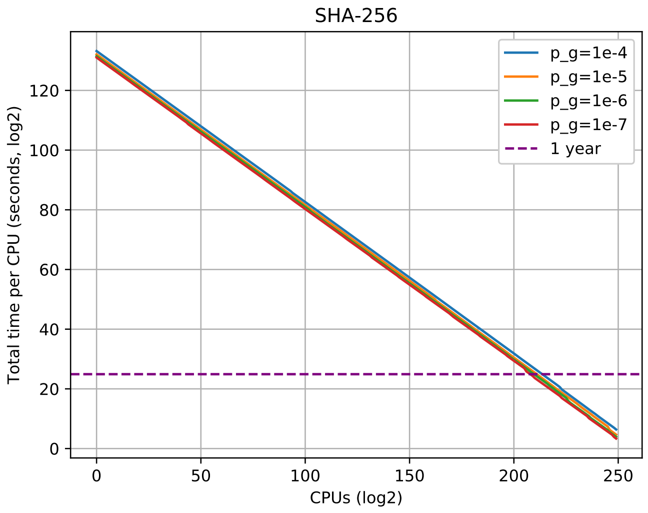

Fig. 16. SHA-256 cryptographic hash function. Required time per processor, as a function of the number of processors ($\log_2$ scale).

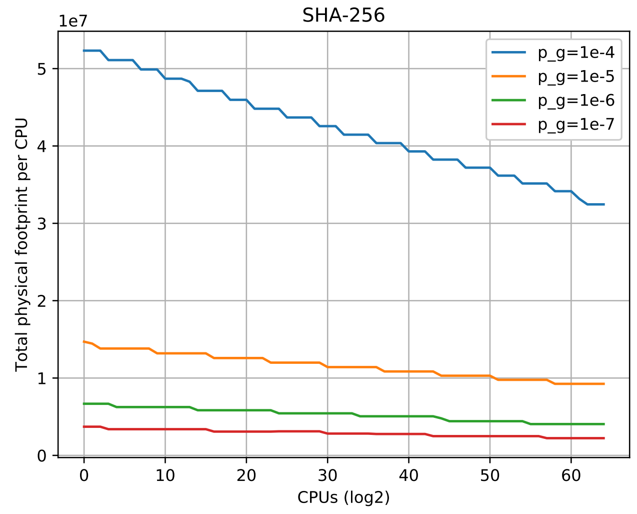

Fig. 17. SHA-256 cryptographic hash function. Physical footprint (physical qubits) per processor, as a function of the number of processors ($\log_2$ scale).

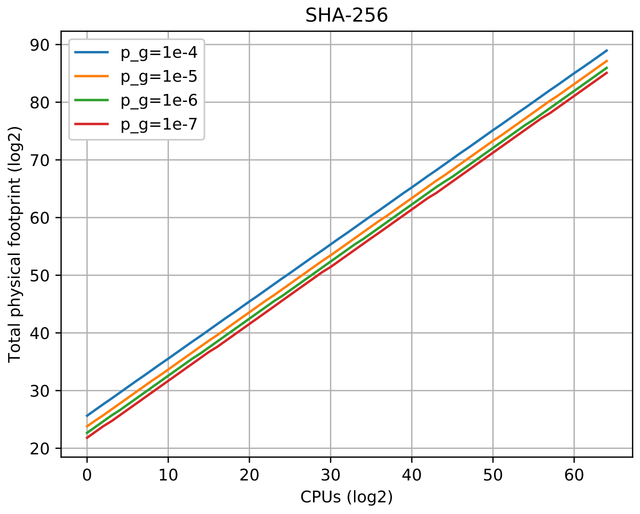

Fig. 18. SHA-256 cryptographic hash function. Total physical footprint (physical qubits), as a function of the number of processors ($\log_2$ scale). Note that the qubits are not correlated across processors.

4.2 SHA3-256

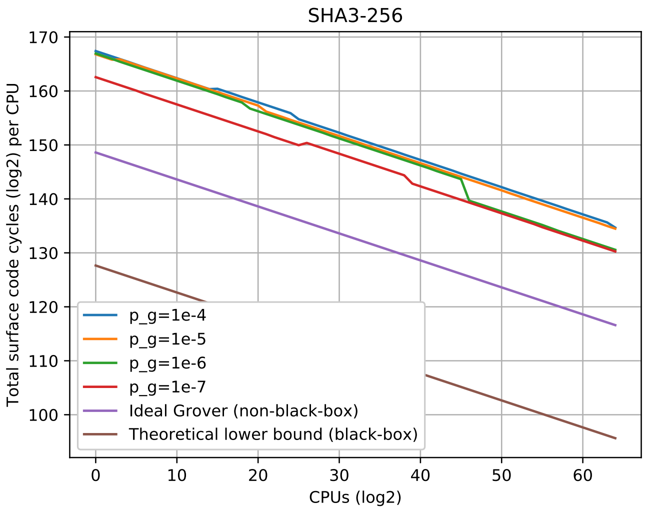

Fig. 19. SHA3-256 cryptographic hash function. Required surface clock cycles per processor, as a function of the number of processors ($\log_2$ scale).

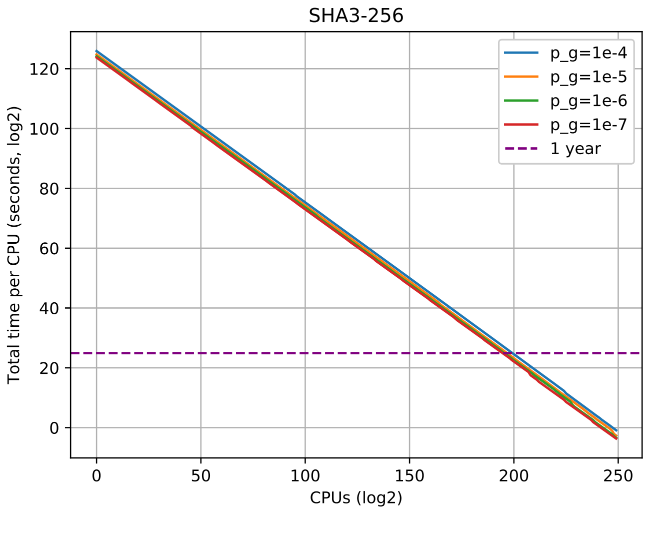

Fig. 20. SHA3-256 cryptographic hash function. Required time per processor, as a function of the number of processors ($\log_2$ scale).

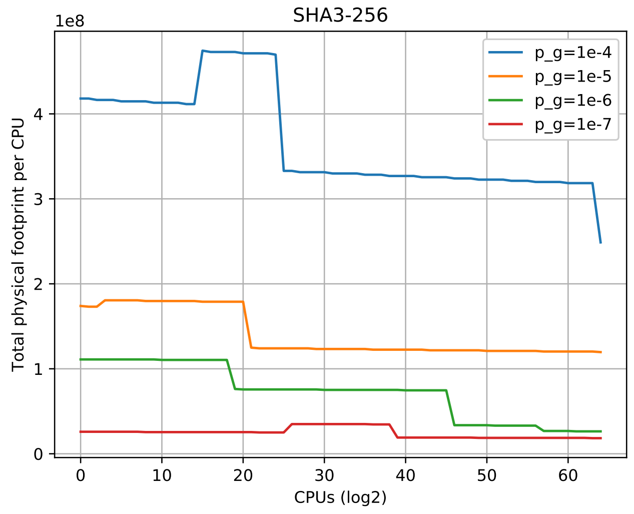

Fig. 21. SHA3-256 cryptographic hash function. Physical footprint (physical qubits) per processor, as a function of the number of processors ($\log_2$ scale).

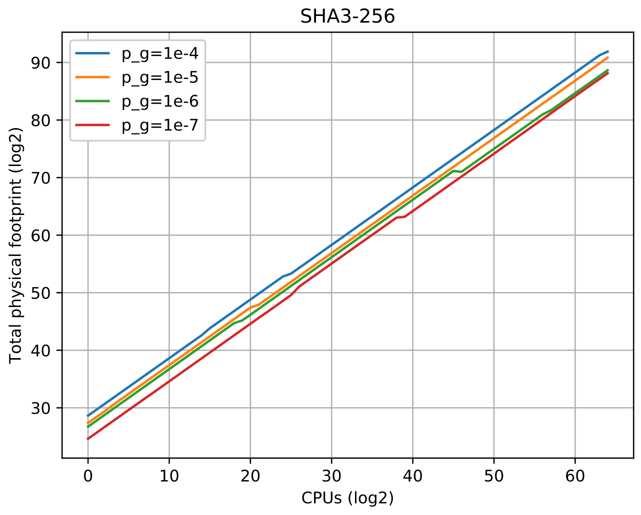

Fig. 22. SHA3-256 cryptographic hash function. Total physical footprint (physical qubits), as a function of the number of processors ($\log_2$ scale). Note that the qubits are not correlated across processors.

5 Bitcoin

In this section we analyze the security of Bitcoin's [11] proof-of-work protocol, which is based on finding a hash[1] pre-image which that starts with a certain number of zeros. The latter is dynamically adjusted by the protocol so that the problem is on average solved by the whole network in 10 minutes. Currently, it takes around 2^{72} operations for finding a desired hash pre-image successfully.

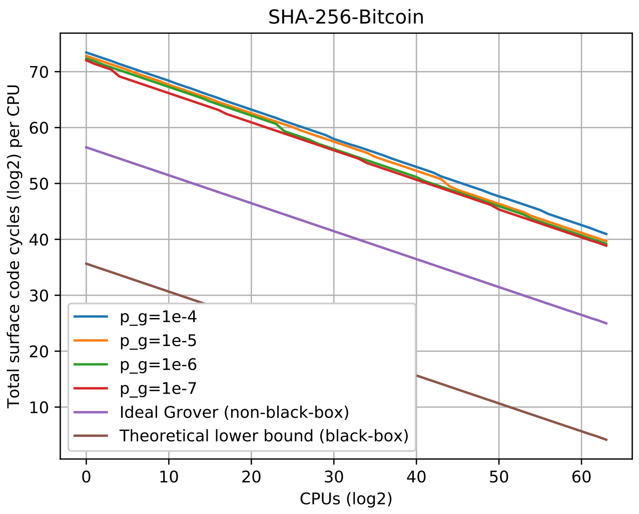

Fig. 23. Bitcoin's cryptographic hash function $\mathrm{H}(x) := \text{SHA-256}(\text{SHA-256}(x))$. Required surface clock cycles per processor, as a function of the number of processors ($\log_2$ scale).

[1] The hash function being used by the protocol is $\mathrm{H}(x) := \text{SHA-256}(\text{SHA-256}(x)$.

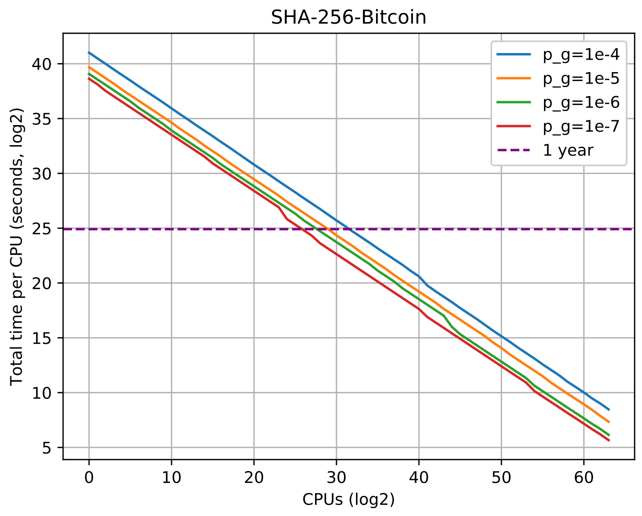

Fig. 24. Bitcoin's cryptographic hash function H(x) := SHA-256(SHA-256(x)). Required time per processor, as a function of the number of processors ($\log_2$ scale).

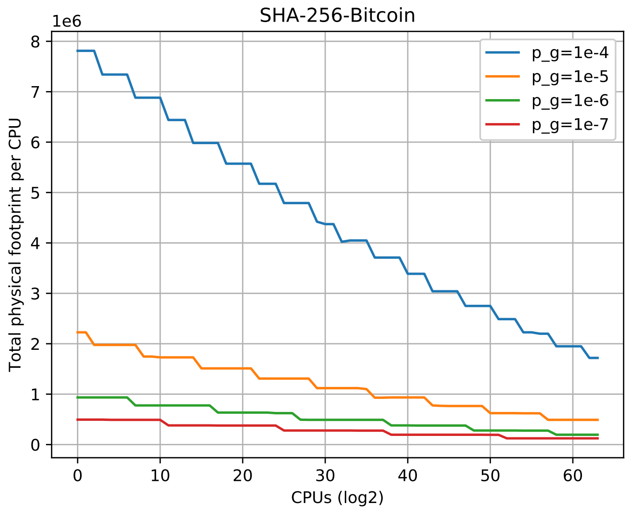

Fig. 25. Bitcoin's cryptographic hash function H(x) := SHA-256(SHA-256(x)). Physical footprint (physical qubits) per processor, as a function of the number of processors ($\log_2$ scale).

evolution

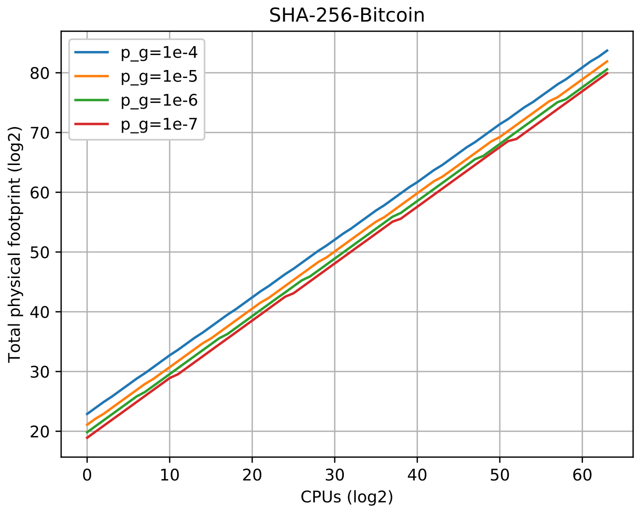

Fig. 26. Bitcoin's cryptographic hash function $\mathrm{H}(x) := \text{SHA-256}(\text{SHA-256}(x))$. Total physical footprint (physical qubits), as a function of the number of processors ($\log_2$ scale). Note that the qubits are not correlated across processors.

6 Intrinsic cost of parallelized Grover's algorithm

More efficient quantum implementations of AES and SHA imply more efficient cryptanalysis. In this section, we aim to bound how much further optimized implementations of these cryptographic functions could help. We do so by assuming a trivial cost of 1 for each function evaluation.

6.1 Searching space of size 2^{56}

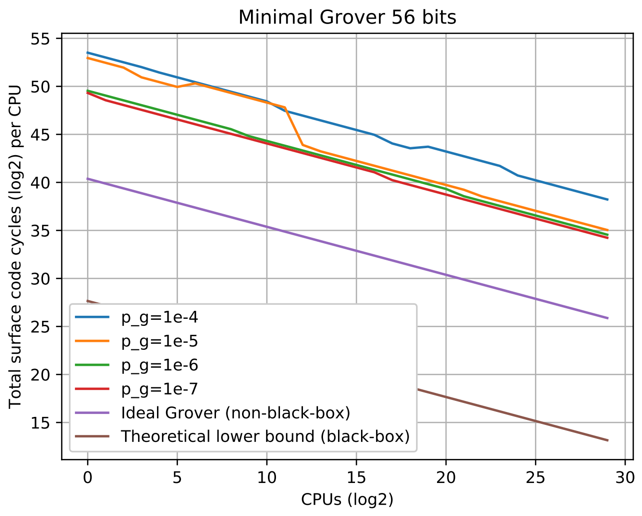

Fig. 27. Running Grover with a trivial oracle, for a searching space of size 2^{56}. Required surface clock cycles per processor, as a function of the number of processors ($\log_2$ scale).

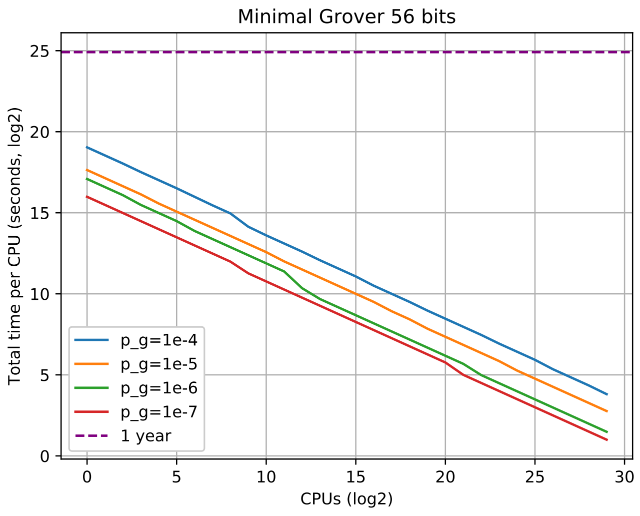

Fig. 28. Running Grover with a trivial oracle, for a searching space of size 2^{56}. Required time per processor, as a function of the number of processors ($\log_2$ scale). The dotted horizontal line indicates one year.

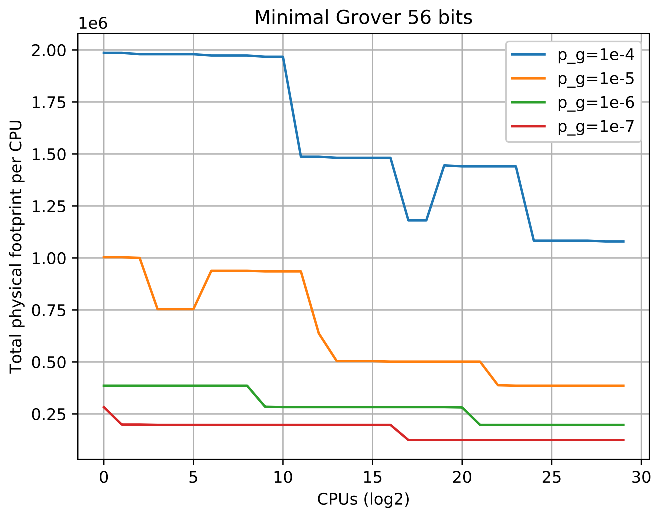

Fig. 29. Running Grover with a trivial oracle, for a searching space of size 2^{56}. Physical footprint (physical qubits) per processor, as a function of the number of processors ($\log_2$ scale).

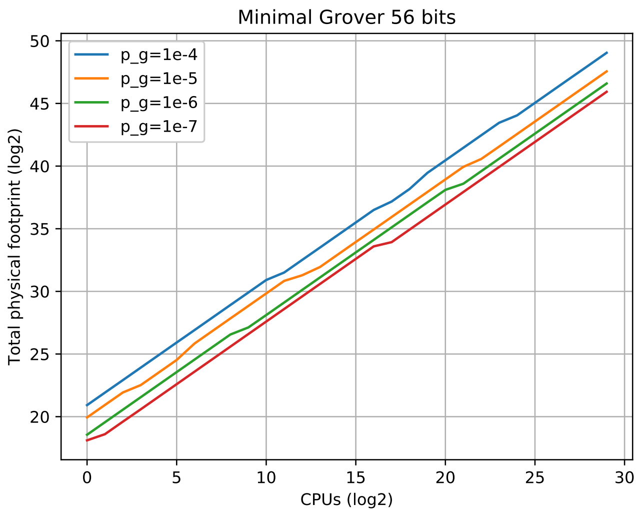

Fig. 30. Running Grover with a trivial oracle, for a searching space of size 2^{56}. Total physical footprint (physical qubits), as a function of the number of processors ($\log_2$ scale). Note that the qubits are not correlated across processors.

6.2 Searching space of size 2^{64}

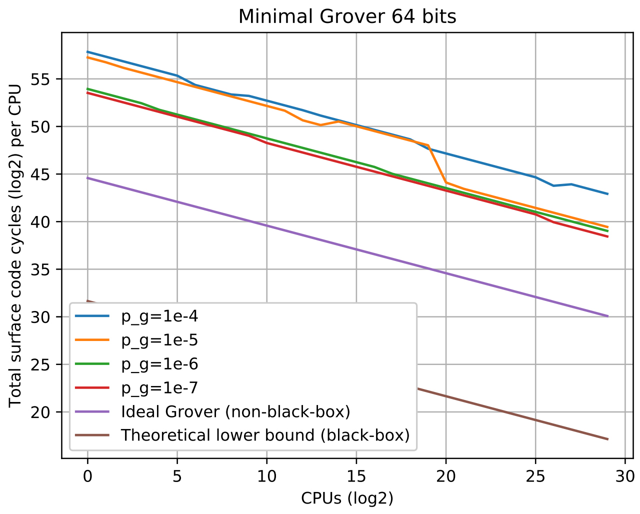

Fig. 31. Running Grover with a trivial oracle, for a searching space of size 2^{64}. Required surface clock cycles per processor, as a function of the number of processors ($\log_2$ scale).

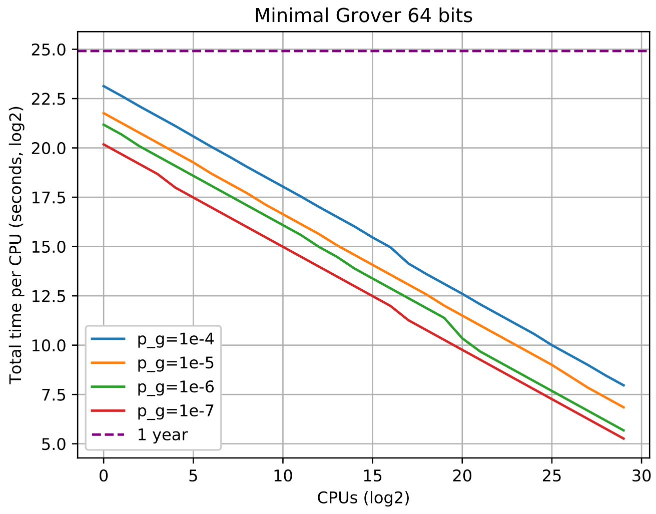

Fig. 32. Running Grover with a trivial oracle, for a searching space of size 2^{64}. Required time per processor, as a function of the number of processors ($\log_2$ scale).

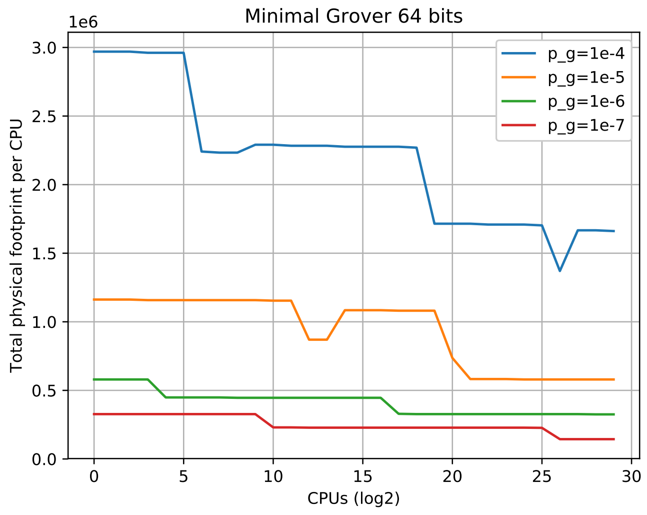

Fig. 33. Running Grover with a trivial oracle, for a searching space of size 2^{64}. Physical footprint (physical qubits) per processor, as a function of the number of processors ($\log_2$ scale).

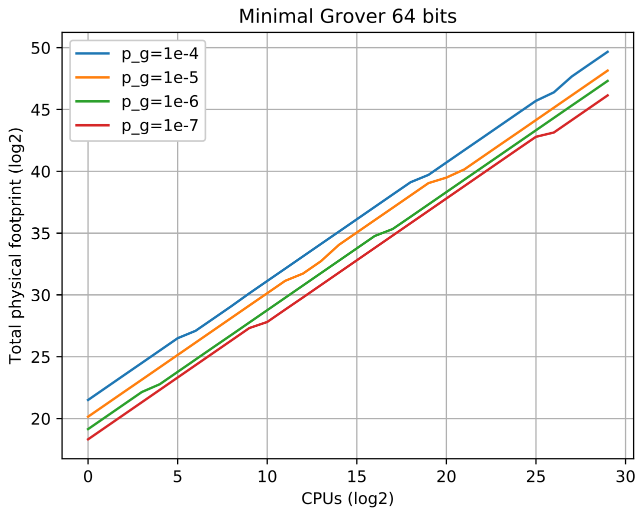

Fig. 34. Running Grover with a trivial oracle, for a searching space of size 2^{64}. Total physical footprint (physical qubits), as a function of the number of processors ($\log_2$ scale). Note that the qubits are not correlated across processors.

6.3 Searching space of size 2^{128}

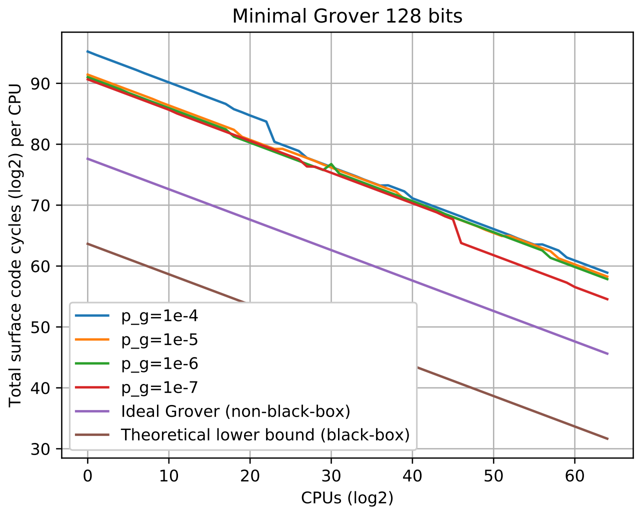

Fig. 35. Running Grover with a trivial oracle, for a searching space of size 2^{128}. Required surface clock cycles per processor, as a function of the number of processors ($\log_2$ scale).

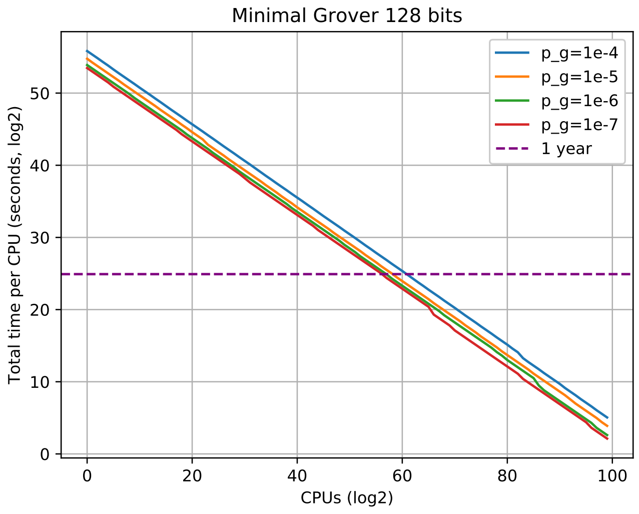

Fig. 36. Running Grover with a trivial oracle, for a searching space of size 2^{128}. Required time per processor, as a function of the number of processors ($\log_2$ scale).

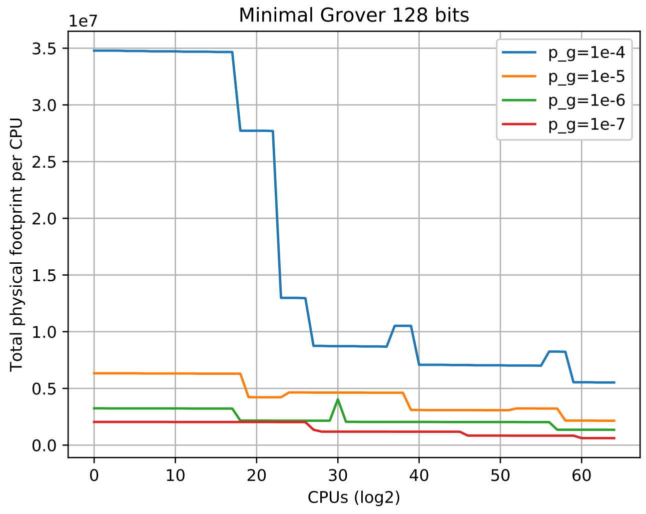

Fig. 37. Running Grover with a trivial oracle, for a searching space of size 2^{128}. Physical footprint (physical qubits) per processor, as a function of the number of processors ($\log_2$ scale).

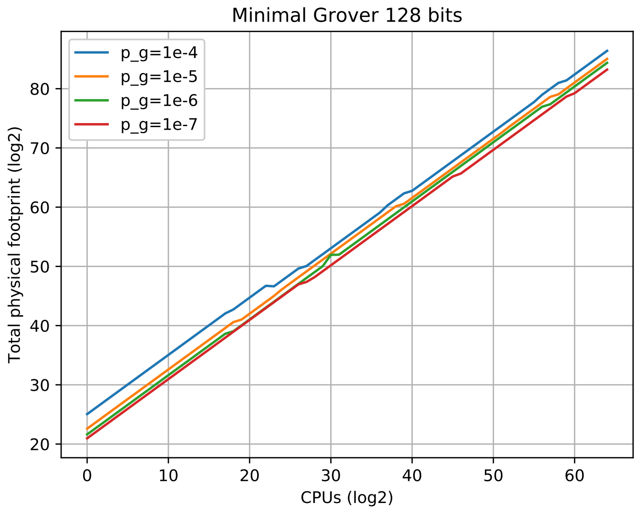

Fig. 38. Running Grover with a trivial oracle, for a searching space of size 2^{128}. Total physical footprint (physical qubits), as a function of the number of processors ($\log_2$ scale). Note that the qubits are not correlated across processors.

6.4 Searching space of size 2^{256}

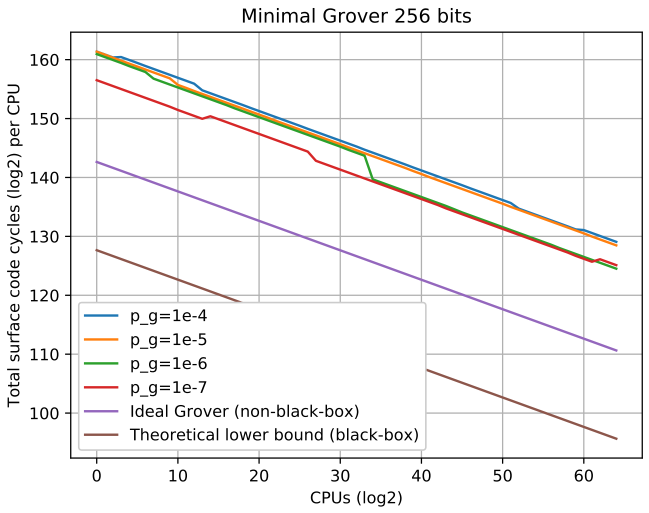

Fig. 39. Running Grover with a trivial oracle, for a searching space of size 2^{256}. Required surface clock cycles per processor, as a function of the number of processors ($\log_2$ scale).

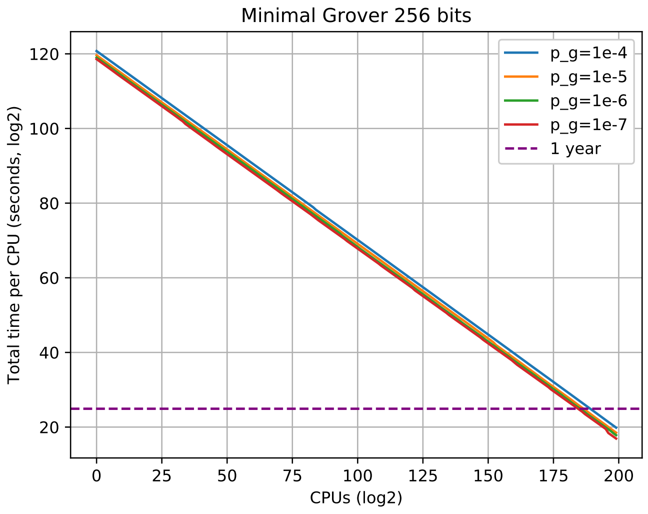

Fig. 40. Running Grover with a trivial oracle, for a searching space of size 2^{256}. Required time per processor, as a function of the number of processors ($\log_2$ scale).

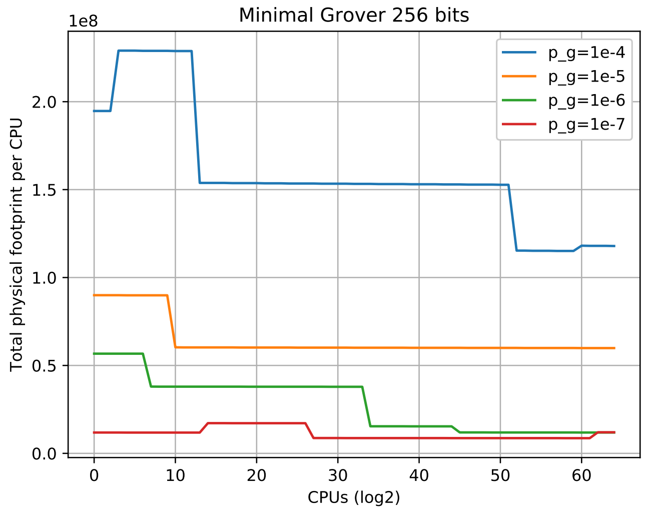

Fig. 41. Running Grover with a trivial oracle, for a searching space of size 2^{256}. Physical footprint (physical qubits) per processor, as a function of the number of processors ($\log_2$ scale).

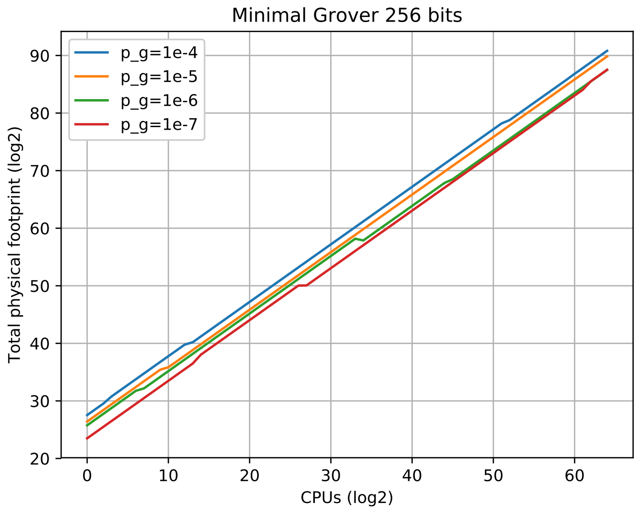

Fig. 42. Running Grover with a trivial oracle, for a searching space of size 2^{256}. Total physical footprint (physical qubits), as a function of the number of processors ($\log_2$ scale). Note that the qubits are not correlated across processors.

7 Conclusions and Discussion

We analyzed the security of several widely used symmetric ciphers and hash functions against parallelized quantum adversaries. We computed the security parameter, wall-time and physical footprint for each cryptographic primitive. Our attack model was based on a brute force searching via a parallelized version of Grover's algorithm, assuming a surface-code fault-tolerant architecture.

It is worth noting that throughout we are assuming that brute-force search where we treat the cryptographic function as a black-box is essentially the optimal attack against SHA and AES, which is currently believed to be the case.

Some symmetric key algorithms are susceptible in a model that permits "superposition attacks" [12]. In most realistic instances, these attacks are not practical, however they do shed light on the limitations of certain security proof methods in a quantum context, and remind us that we shouldn't take for granted that non-trivial attacks on symmetric key cryptography may be possible. For example, very recently, there have been several cryptanalysis results [13] and [14] that attempt to reduce breaking some symmetric algorithms to solving a system of non-linear equations. Solving these non-linear equations is then attacked using a modified version of the quantum linear equation solver algorithm [15]. The results are heavily dependent on the condition number of the non-linear system, which turns to be hard to compute (it is not known for most ciphers and hash functions such as AES or SHA). Provided the condition number is relatively small, then one may get an advantage compared to brute-force Grover search. However at this time it is not clear whether this is indeed the case, and we do not have large-scale quantum computers to experiment with.

References

1. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM Journal on Computing 26(5), 1484–1509 (1997), http://link.aip.org/link/?SMJ/26/1484/1
2. Grover, L.K.: Quantum mechanics helps in searching for a needle in a haystack. Phys. Rev. Lett. 79, 325–328 (Jul 1997), http://link.aps.org/doi/10.1103/PhysRevLett.79.325
3. Gheorghiu, V., Mosca, M.: A resource estimation framework for quantum attacks against cryptographic functions (2017), Global Risk Institute quantum risk assessment report, Sep. 2016 - Feb. 2017
4. Zalka, C.: Grover's quantum searching algorithm is optimal, e-print arXiv:quant-ph/9711070
5. Fowler, A.G., Mariantoni, M., Martinis, J.M., Cleland, A.N.: Surface codes: Towards practical large-scale quantum computation. Phys. Rev. A 86, 032324 (Sep 2012), http://link.aps.org/doi/10.1103/PhysRevA.86.032324
6. Fowler, A.G., Devitt, S.J., Jones, C.: Surface code implementation of block code state distillation. Scientific Reports 3, 1939 EP – (06 2013), http://dx.doi.org/10.1038/srep01939
7. Grassl, M., Langenberg, B., Roetteler, M., Steinwandt, R.: Applying grover's algorithm to aes: Quantum resource estimates. In: Takagi, T. (ed.) Post-Quantum Cryptography. pp. 29–43. Springer International Publishing, Cham (2016)

8. Amy, M., Di Matteo, O., Gheorghiu, V., Mosca, M., Parent, A., Schanck, J.: Estimating the cost of generic quantum pre-image attacks on sha-2 and sha-3. In: Avanzi, R., Heys, H. (eds.) Selected Areas in Cryptography – SAC 2016. pp. 317–337. Springer International Publishing, Cham (2017)
9. NIST: Federal information processing standards publication 180-2 (2002), see also the Wikipedia entry http://en.wikipedia.org/wiki/SHA-2
10. NIST: Federal information processing standards publication 202 (2015), see also the Wikipedia entry http://en.wikipedia.org/wiki/SHA-3
11. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2009), https://bitcoin.org/bitcoin.pdf
12. Kaplan, M., Leurent, G., Leverrier, A., Naya-Plasencia, M.: Breaking symmetric cryptosystems using quantum period finding, e-print arXiv:1602.05973 [quant-ph]
13. Chen, Y.A., Gao, X.S.: Quantum Algorithms for Boolean Equation Solving and Quantum Algebraic Attack on Cryptosystems (2017), arXiv:1712.06239 [quant-ph]
14. Chen, Y.A., Gao, X.S., Yuan, C.M.: Quantum Algorithms for Optimization and Polynomial Systems Solving over Finite Fields (2018), arXiv:1802.03856 [quant-ph]
15. Harrow, A.W., Hassidim, A., Lloyd, S.: Quantum algorithm for linear systems of equations. Phys. Rev. Lett. 103, 150502 (Oct 2009), http://link.aps.org/doi/10.1103/PhysRevLett.103.150502