Math. Struct. in Comp. Science (2013), *vol.* 23, *pp.* 608–634. © Cambridge University Press 2012
doi:10.1017/S0960129512000825 First published online 20 December 2012

Quantum attacks on pseudorandom generators[†]

ELLOÁ B. GUEDES, F. M. DE ASSIS, and BERNARDO LULA JR.

IQuanta – Institute for Studies in Quantum Computation and Information,
Federal University of Campina Grande,
Av. Aprígio Veloso, 882 – CZ.A, 58429-140,
Campina Grande – PB, Brazil
Email: elloaguedes@gmail.com; fmarcos@dee.ufcg.edu.br; lula@dsc.ufcg.edu.br

Received 19 March 2011; revised 24 May 2012

There are advantages in the use of quantum computing in the elaboration of attacks on certain pseudorandom generators when compared with analogous attacks using classical computing. This paper presents a polynomial time quantum attack on the Blum–Micali generator, which is considered secure against threats from classical computers. The proposed attack uses a Grover inspired procedure together with the quantum discrete logarithm, and is able to recover previous and future outputs of the generator under attack, thereby completely compromising its unpredictability. The attack can also be adapted to other generators, such as Blum–Micali generators with multiple hard-core predicates and generators from the Blum–Micali construction, and also to scenarios where the requirements on the bits are relaxed. Such attacks represent a threat to the security of the pseudorandom generators adopted in many real-world cryptosystems.

1. Introduction

A *random generator* is a system whose output consists of fully unpredictable numerical sequences. Such generators are composed of two elements:

(i) a non-deterministic phenomena; and
(ii) a post-processor that compresses the sequence previously produced in order to minimise statistical defects.

Several phenomena can be used to build random generators, including radioactive decay, electronic noise, chemical decay and air turbulence in hard drives (van Tilborg 2005).

Random numbers produced by random generators are used in many computational applications, such as in engineering methods and in the study and simulation of biological processes. However, digital computers are not able to generate such sequences without the addition of an external random generator. To overcome this limitation, an algorithmic alternative can be adopted in the form of a*pseudorandom generator*.

Pseudorandom generators are deterministic and recursive algorithms. They can be viewed as a function f that produces a number x_i from ℓ previous numbers:

$$x_i = f(x_{i-1}, x_{i-2}, \ldots, x_{i-\ell}),$$

[†] The authors gratefully acknowledge the financial support given by the Brazilian Funding Agencies CAPES and CNPq.

where ℓ is called the *order* of the generator, and the set of values at the start of the recursion is called the *seed*. A variety of pseudorandom generators have been proposed, including congruential generators, feedback shift register generators and generators based on cellular automata or chaotic systems (Gentle 2003).

Pseudorandom generators play an important role in cryptography. Session keys, initialisation vectors, salts to be hashed with passwords and unique parameters in digital signatures are all examples of the cryptographic application of pseudorandom generators. In fact, it is generally considered that the security of well-designed cryptographic algorithms depends crucially on the random choice of keys and bit sequences (Paar and Pelzl 2010; Goldreich 2005).

Given the importance of pseudorandom generators to cryptography, it is a crucial task to analyse their vulnerability against attacks in order to ensure security of the entire cryptosystem (Paar and Pelzl 2010; Eastlake *et al.* 2005). The security of the pseudorandom generators adopted in many cryptosystem is based on hypotheses of the hardness of certain problems for classical computers that make it impractical to elaborate efficient classical attacks against them (Sidorenko and Schoenmakers 2005a; Kelsey *et al.* 1998; Sidorenko and Schoenmakers 2005b). This calls attention to the use of other computational paradigms in the elaboration of attacks aiming to exploit their possible advantages.

Quantum computing is a computational paradigm based on quantum mechanics. Several efficient quantum computing algorithms have been proposed for problems where there is no known polynomial-time classical computing algorithm. The most remarkable results in this area are Grover's and Shor's algorithms. Grover's algorithm (Grover 1997), also called the quantum search algorithm, performs a search on an unsorted database with a quadratic speedup compared with its classical counterpart. Shor's algorithm (Shor 1997) enables polynomial-time solutions for integer factoring and discrete logarithms. Such results motivate the elaboration of attacks on pseudorandom generators with the aim of verifying their vulnerability against this computational paradigm.

In this paper, we present a quantum attack on the Blum–Micali generator, which is a cryptographically secure pseudorandom generator that has been widely adopted in cryptosystems (Blum and Micali 1984). The proposed attack is composed of three stages: the second stage is a Grover inspired procedure and the third stage uses Shor's discrete logarithm algorithm. As a result of this attack, the previous and future output of the generator become predictable, thereby completely compromising the security of the generator.

The attack described in this paper is not restricted to the Blum–Micali generator. There are generalisations that extend it to compromise multiple hard-core predicates and other generators using the Blum–Micali construction, such as the Blum–Blum–Shub (Blum *et al.* 1986) and Kaliski (Kaliski 1988) generators. These generalisations also allow attacks even when the adversary intercepts non-consecutive bits, or when there are fewer bits than required.

The results observed indicate that speedups are achieved when the proposed attack is compared with its classical counterpart. In a practical scenario, this means that the gains provided by the quantum paradigm facilitate the task of reproducing the outputs of the generator, and this can be critical for a scenario where unpredictability is an indispensable resource.

1.1. *Organisation of the paper*

We begin in Section 2 with some of the concepts underlying the Blum–Micali generator. Section 3 describes a classical attack on this generator, and the quantum computing attack is then described in Section 4, with an example given in Section 5. Section 6 discusses the computational complexity of the quantum computing attack, and some generalisations of the attack are presented in Section 7. Finally, our conclusions and suggestions for future works are discussed in Section 8.

2. The Blum–Micali generator

The *Blum–Micali generator* was the first cryptographically secure pseudorandom generator proposed in the literature. It is composed of two elements:

(i) a *one-way permutation* that implements the recursive function of the generator;
(ii) a *hard-core predicate* for the one-way permutation that produces bits that will be output.

The security of the Blum–Micali generator is founded on the hypothesis of the hardness of its one-way permutation, which is based on the discrete logarithm problem (Blum and Micali 1984; Sidorenko and Schoenmakers 2005b).

Some mathematical concepts are needed in order to understand the workings of this generator. Let p be a large prime and $n = \lceil \log p \rceil$ be the binary length of p. The set $\mathbb{Z}_p^* = \{1, 2, \ldots, p-1\}$ stands for the cyclic group under multiplication $\bmod p$. Let g be a generator of $\mathbb{Z}_p^*$ and $x_0 \in_R \mathbb{Z}_p^*$ (where $a \in_R A$ denotes the uniformly random choice of an element a from the set A).

The Blum–Micali generator is prepared with parameters (p, g, x_0), as previously described, where x_0 is the seed of the generator. This generator produces pseudorandom bits using the discrete logarithm one-way permutation over the domain $\mathbb{Z}_p^*$, and a hard-core predicate for the permutation, denoted by δ:

$$x_i = g^{x_{i-1}} \bmod p \tag{1}$$

$$b_i = \delta(x_i). \tag{2}$$

Equation (1) is called the *exponential map* and δ is a binary function defined by

$$\delta(x) = \begin{cases} 1 & \text{if } x > \frac{p-1}{2} \\ 0 & \text{otherwise.} \end{cases}$$

The *generator's internal state at time i*, denoted by $X(i)$, is an ordered set containing the seed and the values that were produced by the exponential map up to the time i, that is,

$$X(i) = \{x_i, x_{i-1}, \ldots, x_1, x_0\}.$$

The value $x_i \in X(i)$ is the *representative* of the generator's internal state at the time i. The parameters p and g are publicly available, but the generator's internal state, including the

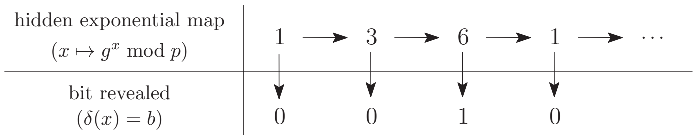

Figure 1. Diagram of the workings of a Blum–Micali generator initialised with parameters $(p = 7, g = 3, x_0 = 1)$.

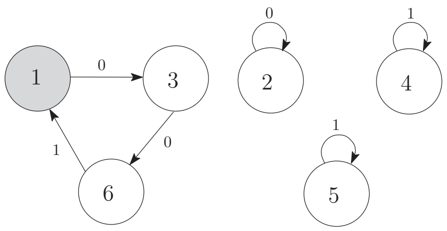

Figure 2. Functional graph of the Blum–Micali generator $(p = 7, g = 3, x = 1)$.

seed, must be kept secret. Only the bits output by the function δ are seen directly in the generator's output.

The security of this generator relies on the hardness of evaluating the inversion of the one-way function $x_{i+1} = g^{x_i} \bmod p$, that is, evaluating

$$x_i = \log_g(x_{i+1}) \bmod p.$$

This is an instance of the discrete logarithm problem, and there is no known efficient classical computing algorithm to perform this operation (Gregg 2003).

To illustrate the definition of the Blum–Micali generator, suppose it was configured with parameter values $(7, 3, 1)$ for p, g and x_0, respectively. Figure 1 shows a diagram that describes the evolution of its internal states and the bits output during this process.

A Blum–Micali generator can also be expressed in terms of a *functional graph*. A functional graph is a digraph that respects the restriction that there is only one edge directed out from each vertex (Cloutier and Holden 2010). This graphical representation allows a better understanding of how this generator works, and also of its initialisation procedures.

A functional graph of the Blum–Micali generator, denoted by $G = \langle V, E \rangle$, is a graph whose set of vertices is given by $V = \mathbb{Z}_p^*$. There is an edge $(x, y) \in E$ directed from the vertex x to the vertex y if there is an exponential mapping $x \mapsto y$ (see Equation (1)). Such an edge is labelled with the bit produced by its origin vertex, that is, with $\delta(x)$.

As an example of such a graph, consider Figure 2, which shows a functional graph of the Blum–Micali generator initialised with parameters $(p = 7, g = 3, x = 1)$. Vertex 1, related to the seed, is emphasised.

From this figure, it can be seen that there are three self loops, that is, three vertices with edges directed to themselves, and also a cycle of size 3, containing the vertices 1, 3 and 6. The vertices with self loops are called *fixed points* in the exponential map of the Blum–Micali generator. The fixed points must be avoided as seeds because they always produce the same output. Furthermore, the choice of the seed must consider the longest cycles in order to maximise the period of the generator.

Because of the widespread use of the Blum–Micali generator, the analysis of its vulnerabilities against attacks is crucially important. The next section presents a classical attack on this generator, which starts with the interception of some bits output by it. Proofs of correctness and a complexity analysis of the attack are also provided.

3. Classical attack on the Blum–Micali generator

In this section we will describe a simple classical algorithm to attack the Blum–Micali generator. To introduce this attack, suppose there is an *adversary* of the generator who knows the public parameters p and g and has also intercepted a finite sequence of j bits, denoted by $\mathbf{b}(j) = \{b_i\}$, where $1 \leqslant i \leqslant j$. This adversary aims to retrieve the generator's internal state, that is, the set $X(j)$.

To discover the generator's internal state, the adversary must use the $\mathbf{b}(j)$ bits intercepted and the information that $x_0 \in \mathbb{Z}_p^*$. With the public parameters, it is possible to reconstruct rules (1) and (2). Thus, the adversary needs to make assumptions about the elements of $X(j)$, and then use the bits in $\mathbf{b}(j)$ to confirm or refute them.

The set $\hat{X}_i$ will be referred as the *estimator set relative to* x_i, or simply the *estimator set*. To carry out the attack on the Blum–Micali generator, the adversary will start with $\hat{X}_0 = \mathbb{Z}_p^*$, since $x_0 \in \mathbb{Z}_p^*$.

Given an estimator set $\hat{X}_{i-1}$ and a bit $b_i \in \mathbf{b}(j)$, the adversary will proceed as follows to obtain $\hat{X}_i$. First compute $A_i = g^{\hat{X}_{i-1}}$, the image of $\hat{X}_{i-1}$ under the exponential map $x \mapsto g^x \bmod p$. Let $A_i = A_i^{(0)} \cup A_i^{(1)}$ be the partition of A_i due to Equation (2), that is, $a \in A_i^{(0)}$ if and only if $\delta(a) = 0$, and similarly for $A_i^{(1)}$. In this way, the estimator set $\hat{X}_i$ will be

$$\hat{X}_i = A_i^{(b_i)}. \tag{3}$$

That is, $\hat{X}_i = \delta^{-1}(b_j) \cap A_i$ where $\delta^{-1}(b_j)$ denotes the inverse image of b_j.

To verify that the described attack recovers the internal state of a Blum–Micali generator, it must be proved that:

(i) Every estimator set $\hat{X}_i$ contains $x_i \in X(i)$.

(ii) Given sufficient bits in $\mathbf{b}(j)$, either the size of the set $\hat{X}_i$ is unitary for i large enough or the adversary will easily be able to predict the generator's next output.

The proofs are given below. In order to avoid trivialities, we assume that x_0 is member of a maximal cycle in the functional graph of the exponential map (Equation (1)) since otherwise there is a high probability that the sequence produced is easily predictable.

Lemma 3.1. Every set $\hat{X}_i$ of estimators contains the corresponding representative $x_i \in X(i)$, that is, $x_i \in \hat{X}_i$.

Proof. The proof is by induction on i. Clearly, the claim is valid for $i = 0$ as $x_0 \in \hat{X}_0 = \mathbb{Z}_p^*$, so we assume $x_i \in \hat{X}_i$ as our induction hypothesis. We must show that $x_{i+1} \in \hat{X}_{i+1}$.

Indeed, from Equation (3), $y \in \hat{X}_{i+1}$ if and only if:

(1) $y = g^x \bmod p$ for some $x \in \hat{X}_i$; and

(2) $\delta(y) = b_{i+1}$.

But, by the induction hypothesis, $x_i \in \hat{X}_i$ and, according to the generator's productions,

$$\delta(x_{i+1}) = \delta(g^{x_i} \bmod p) = b_{i+1}.$$

So x_{i+1} meets both requirements (1) and (2) and the proof is complete. □

Using the proposed attack, an adversary is able to predict the next bit in two situations:

(i) $\left|\hat{X}_i\right| = 1$; or

(ii) $A^{(0)}_{i+1} = \varnothing$ or $A^{(1)}_{i+1} = \varnothing$.

In other words, the next bit is *completely predictable* if one of the described situations occurs. The first situation happens when the number of bits observed was sufficient to identify precisely the representative of the generator's internal state. In this case, there is no doubt about the next bit that will be output. The second situation also allows the adversary to predict the next bit correctly, but the adversary does not have complete knowledge about the generator's internal state. In other words, in the first situation there is no uncertainty about its internal state or about the next bit, while in the second situation, there is no uncertainty about the bit produced, but there is uncertainty about the generator's internal state.

For i large enough, and assuming that the second situation is less likely to occur given the assumption previously made, the adversary will be able to look up the bits intercepted and reduce the estimator set at every step. The convergence to a solution is synthesised in the proof below.

Lemma 3.2. The sequence of estimator sets' sizes is non-increasing, that is,

$$|\hat{X}_{i+1}| \leqslant |\hat{X}_i| \qquad i = 1, 2, \ldots \tag{4}$$

Moreover, if equality holds in Equation (4), then the bit b_{i+1} becomes completely predictable.

Note that the lemma implies that there is uncertainty about the bit output only if the size of the estimator set is decreased.

Proof. We have

$$|\hat{X}_{i+1}| \overset{(a)}{=} \left|A^{(b_{i+1})}_{i+1}\right| \overset{(b)}{\leqslant} |A_{i+1}| \overset{(c)}{=} |g^{\hat{X}_i} \pmod p)| \overset{(d)}{=} |\hat{X}_i|.$$

where:

— The equalities (a) and (c) correspond to the definitions of $\hat{X}_{i+1}$ and A_{i+1}, respectively.

— The inequality (b) is due to the fact that $A^{(b_{i+1})}_{i+1}$ is a subset of A_{i+1} and both are finite sets.

— The equality (d) follows from the fact that g^x is a bijective map.

This concludes the first part of the proof.

To see why b_{i+1} is completely predictable if equality holds in Equation (4), we assume that

$$|\hat{X}_{i+1}| = |\hat{X}_i|$$

in the sequence of Equation (3). It then follows that

$$\left|A_{i+1}^{(b_{i+1})}\right| = |A_{i+1}|.$$

But since

$$A_{i+1}^{(b_{i+1})} \subseteq A_{i+1},$$

the equality is only possible if

$$A_{i+1}^{(b_{i+1})} = A_{i+1}.$$

Hence, the adversary can guess the next bit perfectly, since A_{i+1} only contains elements x such that $\delta(x) = b_{i+1}$. □

After recovering the representative of the generator's internal state by following the previously mentioned steps, the adversary is able to discover the future output of the generator using Equations (1) and (2). But to recover the previous elements of the generator's internal state, the adversary needs to perform discrete logarithm operations. The adversary starts with x_j, p and g and obtains x_{j-1}. The same operation is then repeated, but replacing x_j by x_{j-1}, and so on, until x_0 is obtained.

The attack presented retrieves the generator's internal state, in which the seed is included. In consequence, all previous and future outputs are known by the adversary. Thus, in the taxonomy of attacks presented in Kelsey *et al.* (1998), this attack on the Blum–Micali generator is classified as a *permanent compromise attack*.

To conclude the description of the classical attack on the Blum–Micali generator, we need to define the number of bits the adversary must intercept to identify precisely the representative of the generator's internal state at time i. A naive algorithm would demand $p-1$ bits, since this is the number of elements in the initial estimator set $\hat{X}_0$. On the other hand, the procedure described takes advantage of the workings of the generator and also considers that the estimator sets reduce in size, on average, by half per bit observed. This implies that the adversary needs to intercept $\log p$ bits, on average, to perform a successful attack. This number corresponds to the best scheme of questions to identify the representative precisely. This result is in accordance with Boyar (1989) and Krawczyk (1992), and has also been confirmed by experiments (Guedes *et al.* 2010b).

As an example of the attack described, suppose the adversary eavesdropped 2 sequenced output bits, $\mathbf{b}(2) = \{10\}$, from a Blum–Micali generator with parameters $p = 7$ and $g = 3$. The adversary would start his estimators to the seed with the set $\hat{X}_0 = \{1, 2, \ldots, 6\}$. The first step is to perform a modular exponentiation over $\hat{X}_0$, resulting in A_1. Note that the sets A_1 and $\hat{X}_0$ are equal since they contain all elements from the permutation domain. With the information that the first bit observed was $b_1 = 1$, the adversary would discard the numbers lower than $(7-1)/2 = 3$ and perform the operation $g^x \bmod p$ in the remaining ones, resulting in the set of estimators to x_1, $\hat{X}_1 = \{4, 5, 6\}$. Repeating this process, but with the next observed bit $b_2 = 0$, the adversary would update from $\hat{X}_1$ his estimators to x_2, resulting in $\hat{X}_2 = \{1\}$. In this case, with only two intercepted bits, the adversary

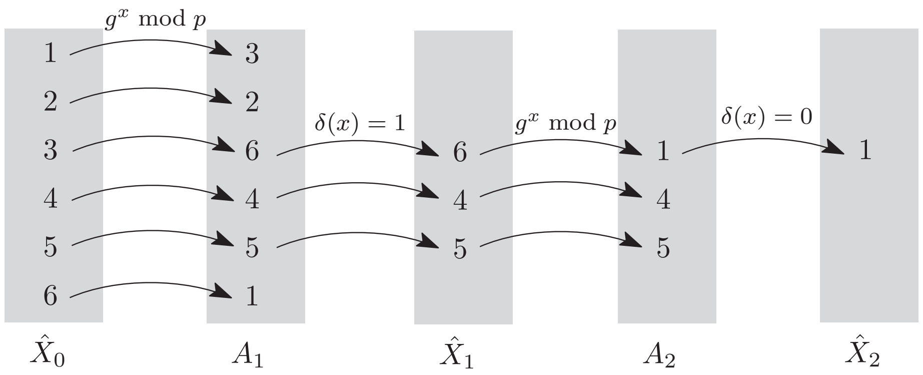

Figure 3. Graphical depiction of the attack performed on a Blum–Micali generator.

could retrieve, with 100% certainty, the representative of the generator's internal state. To recover the complete internal state of the generator, two discrete logarithm operations would result in

$$X(2) = \{x_2 = 1, x_1 = 6, x_0 = 3\}.$$

This example is presented graphically in Figure 3.

The complexity analysis of the classical permanent compromise attack on the Blum–Micali generator will be described in the next section.

3.1. *Complexity analysis of the classical attack*

Despite being able to recover the generator's internal state, this attack turns out to be inefficient. We will assume that p is a large prime. We will also assume that the modular exponentiation operation can be implemented in $O(x)$ time, where x is the exponent in $g^x \bmod p$. Since the result of the first modular exponentiation is equal to the set $\hat{X}_0$, we will define its cost as 0.

Upon observing the first bit output by the generator, we need to pass through all the elements in A_1 to verify which of them could have produced that bit. Those satisfying the criteria will remain, and the others will be discarded. On average, half of the elements will satisfy the criteria. This operation has a linear cost of $O(p)$ because all the elements in A_1 must be evaluated.

A modular exponentiation must then be performed on the remaining elements comprising the set $\hat{X}_1$, resulting in A_2. Suppose the elements greater than $(p-1)/2$ remain. The cost of performing the modular exponentiation on them will be

$$O\left(\frac{p+1}{2}\right) + \cdots + O(p-1) = O(p^2).$$

In this way, the cost of obtaining A_2 from A_1 is

$$T(p) = O(p^2) + O(p) = O(p^2).$$

As can be seen from the first bit observed and the modular exponentiation performed, some conclusions can be drawn about the cost of the next steps. The cost of the modular exponentiation in the estimators set is $O(p^2)$ in the worst case. Furthermore, upon observing a bit, on average, $p/2^t$ elements will remain, where $t = 1, 2, \ldots, j$. These results will be used to analyse the total cost of the classical attack.

The costs of obtaining A_2 from A_1, A_3 from A_2 and so on are described below, where we make use of the assumptions previously made.

$$\begin{aligned}
T(p) &= \overbrace{O(p^2)}^{\text{Modular Exponentiation}} + \overbrace{O(p)}^{\text{Observing a bit}} = O(p^2)\\
T\left(\frac{p}{2}\right) &= O(p^2) + O\left(\frac{p}{2}\right) = O(p^2)\\
&\vdots\\
T(1) &= \underbrace{O(p^2)}_{\text{Greater exponent remains}} + O(1) = O(p^2)
\end{aligned}$$

The cost of recovering the representative is the sum of the cost of the parts, that is

$$\begin{aligned}
T(p) + T\left(\frac{p}{2}\right) + T\left(\frac{p}{4}\right) + \cdots + T(1) &= O(p^2) + O(p^2) + \cdots + O(p^2)\\
&= \log p \cdot O(p^2)\\
&= O(p^2 \log p)
\end{aligned}$$

After this first part, j discrete logarithm operations must be carried out in order to recover the previous elements $x_{j-1}, \ldots, x_0$ of the generator's internal state. But, there is no known classical polynomial time algorithm to evaluate discrete logarithms efficiently. In this way, the cost of this operation is, in the worst case,

$$j \cdot O(2^p) = \log p \cdot O(2^p) = O(2^p \cdot \log p).$$

Therefore, the whole cost of the classical algorithm to attack the Blum–Micali generator is

$$O(p^2 \log p) + O(2^p \cdot \log p) = O(2^p \cdot \log p).$$

In other words, the attack presented is infeasible when using the classical computing paradigm.

Quantum computation explores the possibilities of applying quantum mechanics to computer science. If built, quantum computers would provide speedups over conventional computers for a variety of problems (Ambainis 2004; Nielsen and Chuang 2005, page 7). This statement is interesting from the point of view of attacks on cryptosystems, because the speedups achieved will improve the performance of such attacks. In this context, the next section presents a quantum version of the attack previously described. The purpose of this new version is to use quantum computing to increase efficiency.

4. Quantum attack on the Blum–Micali generator

The quantum attack on the Blum–Micali generator is based on the same idea as the classical one: the bits intercepted by the adversary are used to confirm or refute hypotheses about the generator's internal state. The main differences between the classical and quantum algorithms are due to the different computational paradigms adopted.

The attack is composed of three stages:

(1) *identification of the representative*;
(2) *amplitude amplification*; and
(3) *internal state recovery.*

The second stage is inspired by the quantum search algorithm (Grover 1997), and the third makes use of the quantum discrete logarithm algorithm proposed in Shor (1997).

The following sections describe each of these stages in detail. In presenting the quantum attack, we will assume that the adversary has intercepted enough sequential bits to identify the representative of the generator's internal state precisely – see Section 7.3 for the changes needed when this requirement is relaxed.

4.1. *Identification of the representative*

The identification of the representative stage requires the bits intercepted by the adversary $\mathbf{b}(j)$ along with public parameters p and g to define a quantum algorithm able to identify, at a quantum level, the representative $x_j \in X(j)$.

The input to this stage is composed of two registers, described as follows:

(1) **Space of the solution**:
This contains $n = \lceil \log p \rceil$ qubits that will be used to represent the elements in $\mathbb{Z}_p^*$. All qubits of this register must be initialised to $|0\rangle$.
(2) **Ancillary qubits**:
For every bit that the adversary has intercepted from the generator, there must be one ancillary qubit in this second register, therefore, this register must have, on average, $j = \log p$ qubits. All ancillary qubits must be initialised to $|0\rangle$.

Thus, according to the initialisation procedures described, the input state is

$$|\psi_0\rangle = |0\rangle^{\otimes n} |0\rangle^{\otimes j} .$$

Figure 4 shows the circuit that implements the identification of the representative stage. It is composed of gates H, δ_{b_i} and $g^{()}$, and the goal is to 'mark' the representative $x_j \in X(j)$, where 'marking' the representative means associating it with the state $|1\dots1\rangle$ in the second register.

The identification of the representative starts with an application of the Hadamard gate to the first register. This operation will result in a uniformly distributed superposition in which all elements of the domain of the generator are contained, that is, the outcome is a superposition of the states $\{|0\rangle, |1\rangle, \dots, |2^n - 1\rangle\}$ (where $n = \lceil \log p \rceil$), which contains $\{|0\rangle, \dots, |p-1\rangle\}$, whose elements correspond to $\mathbb{Z}_p^*$. The state resulting from the Hadamard

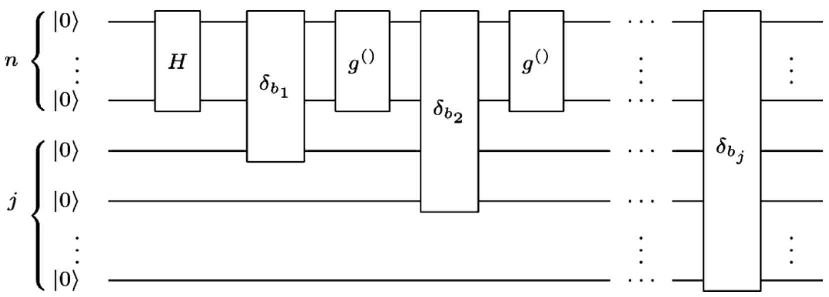

Figure 4. Quantum circuit implementing the identification of the representative stage.

operation is given by

$$\begin{aligned}|\psi_1\rangle &= H^{\otimes n} \otimes \mathbb{I}^{\otimes j} |\psi_0\rangle \\ &= \frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} |i\rangle\, |0\rangle^{\otimes j}.\end{aligned}$$

Let $|\varphi\rangle$ denote the result of applying the Hadamard gate exclusively to the first register, that is,

$$|\varphi\rangle = \frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} |i\rangle\,. \tag{5}$$

In this way, $|\psi_1\rangle$ can also be written as

$$|\psi_1\rangle = |\varphi\rangle\, |0\rangle^{\otimes j}.$$

The gates $g^{()}$ and δ_{b_i} are defined in terms of the one-way permutation and hard-core predicates of the generator under attack. They can be constructed as follows:

$$\delta_{b_i} |x\rangle\, |y\rangle_i = \begin{cases} |x\rangle\, |\overline{y}\rangle_i & \text{if } x \in \mathbb{Z}_p^* \text{ and } \delta(x) = b_i \\ |x\rangle\, |y\rangle_i & \text{otherwise} \end{cases}$$

$$g^{()} |x\rangle = \begin{cases} |g^x \bmod p\rangle & \text{if } x \in \mathbb{Z}_p^* \\ |x\rangle & \text{otherwise} \end{cases}$$

where the operation denoted by $|\overline{y}\rangle_i$ indicates the application of the Pauli-X gate[†] to the state $|y\rangle$ of the ith qubit of the second register conditioned on the value in the first register. Gates δ_{b_i} can be constructed from multiple controlled Pauli-X gates. The operation of $g^{()}$ requires the recognition of $\mathbb{Z}_p^*$, which can be performed using 'is less than or greater than' comparisons, which can be implemented efficiently on a quantum computer since they are already efficient in classical computing (Bennett 1973). We can also follow the procedures in Meter and Itoh (2005) to implement the modular exponentiations in the gate $g^{()}$.

[†] The Pauli-X gate is also referred to as σ_X and has matrix form $X = \left[\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right]$.

It should be noted that each application of the pairs $g^{()}$ and δ_{b_i} is analogous to obtaining an estimator set of the generator's internal state at time *i*. The elements in the generator's internal state are then progressively associated with $|1\rangle^{\otimes j}$ in the second register.

After the first part of the algorithm is complete, the state can be described by

$$|\psi_2\rangle = \alpha_{x_j}|x_j\rangle|1\dots1\rangle + \sum_{k=0,k\neq x_j}^{2^n-1}\alpha_k|k\rangle|y\neq1\dots1\rangle \tag{6}$$

where

$$|\alpha_{x_j}|^2 + \sum_{k=0,k\neq x_j}^{2^n-1}|\alpha_k|^2 = 1.$$

As shown in Section 3, the representative is always in the generator's internal state, thus $\alpha_{x_j}\neq 0$. Defining $p_{x_j} = |\alpha_{x_j}|^2$, $p_{x_{j\perp}} = 1 - p_{x_j}$, so $0 < p_{x_j} < 1$, we can renormalise as follows:

$$\begin{aligned}|\psi_{x_j}\rangle &= \frac{\alpha_{x_j}}{\sqrt{p_{x_j}}}|x_j\rangle|1\dots1\rangle\\ |\psi_{x_{j\perp}}\rangle &= \sum_{k=0,k\neq x_j}^{2^n-1}\frac{\alpha_k}{\sqrt{p_{\neg x_j}}}|k\rangle|y\neq1\dots1\rangle.\end{aligned}$$

According to the conventions established, the state $|\psi_2\rangle$ in Equation (6) can be rewritten as

$$|\psi_2\rangle = \sqrt{p_{x_j}}|\psi_{x_j}\rangle + \sqrt{p_{x_{j\perp}}}|\psi_{x_{j\perp}}\rangle.$$

Using a geometric representation,

$$|\psi_2\rangle = \sin(\theta)|\psi_{x_j}\rangle + \cos(\theta)|\psi_{x_{j\perp}}\rangle. \tag{7}$$

where $\theta\in(0,\pi/2)$ satisfies $\sin^2(\theta) = p_{x_j}$.

After the conclusion of this first stage of the attack, the representative of the generator's internal state at time *j* is identified by its association with $|1\rangle^{\otimes j}$ in the second register. However, its amplitude has the same value as that of the remaining elements. This means that a measurement of the first register after this first stage would return any component of the superposition with the same probability, indicating that this marking procedure has operated exclusively at the quantum level. Therefore, to retrieve the representative after a measurement, it is necessary to amplify its amplitude. This procedure characterises the second stage of the attack and will be described in the next section.

4.2. *Amplitude amplification*

The amplitude amplification stage will increase the probability of outputting the representative when the first register is measured. This stage is composed of iterations of the gates *F* and *A* followed by a measurement in the first register, as shown in Figure 5. It is important to note that no measurement is made between the first and second stages, that is, the operations of the amplitude amplification stage are performed on the quantum state resulting from the identification of the representative stage.

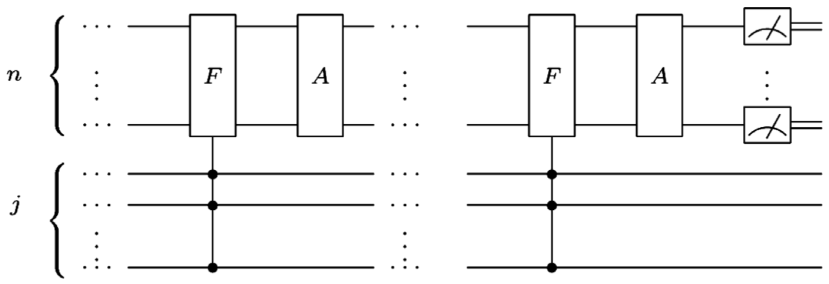

Figure 5. Quantum circuit for the amplitude amplification stage.

The gate F performs a -1 phase shift in the first register controlled by the value $|1\rangle^{\otimes j}$ in the second register. After this operation, the gate A increases the amplitude of the component whose phase is inverted. The F gates are defined by

$$F_{|y\rangle}|x\rangle = \begin{cases} -|x\rangle & \text{if } |y\rangle = |11\dots1\rangle \\ |x\rangle & \text{otherwise} \end{cases}$$

$$A = 2 \cdot |\varphi\rangle\langle\varphi| - \mathbb{I}$$

where $|\varphi\rangle$ is given in Equation (5) and $\mathbb{I}$ denotes the identity matrix. The operation of a pair of gates F and A, respectively, is called an *iteration*, denoted by G, that is, $G = A \cdot F$. The application of k successive iterations G on the input state (see Equation (7)) can be summarised by

$$G^k |\psi_2\rangle = \sin((2 \cdot k + 1) \cdot \theta)\left|\psi_{x_j}\right\rangle + \cos((2 \cdot k + 1) \cdot \theta)\left|\psi_{x_{j\perp}}\right\rangle.$$

The number k is the optimal number of iterations required to maximise the amplification of the representative's amplitude. Its determination involves the assumption previously made that the adversary intercepted enough bits to identify the generator's internal state precisely. Taking this into account, the number of iterations k is given by

$$k = \left\lfloor \frac{\pi}{4}\sqrt{p} \right\rceil,$$

where $\lfloor \cdot \rceil$ denotes the closest integer function.

At the conclusion of this second part, it is expected, with high probability, that a measurement in the first register would retrieve the representative x_j of the generator's internal state. When we are in possession of this number, the internal state recovery stage described in the next section must be carried out.

However, we will first take a closer look into the operations of the second stage to reveal similarities with the quantum search algorithm of Grover (1997). In fact, the representative (solution) has its phase flipped and amplified through iterations. However, while Grover's algorithm requires a black-box function to identify the solution to the problem, in the proposed attack, we can identify the representative by its association with $1\dots1$ in the second register because of the procedures performed in the first stage. This is the main difference between the proposed attack and the standard quantum search. The latter assumes a quantum black-box function that can return the solution in one computational

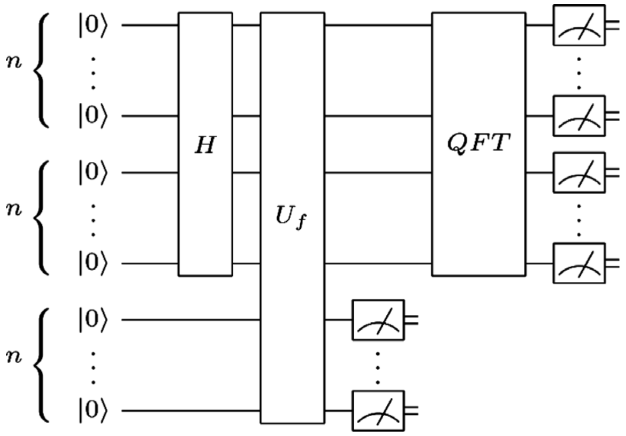

Figure 6. Quantum circuit for the discrete logarithm.

step (Hirvensalo 2001, pages 74–79), but this assumption is not required in the algorithm proposed here, so we can say that this stage is 'inspired' by the Grover algorithm, but there are considerable differences.

4.3. *Internal state recovery*

After the amplitude amplification stage, the adversary has recovered the representative x_j of the generator's internal state at time j. Using Equations (1) and 2, it is then possible to predict the next output. However, despite the fact that the adversary knows some previous outputs, the internal states associated with them still remain unknown. Thus, in order to recover the set $\{x_0, x_1, \ldots, x_{j-1}\}$, the quantum discrete logarithm will be applied.

The discrete logarithm will be performed using the quantum algorithm proposed in Shor (1997), the circuit for which is shown in Figure 6. This circuit is composed of a Hadamard gate, denoted by H, an oracle U_f, a gate that implements the quantum Fourier transform, denoted by QFT, and some measurement gates.

The first use of the quantum discrete logarithm algorithm in the attack will take as input x_j, which is obtained from the amplitude amplification stage, together with g and p, as in the Blum–Micali generator definition (see Section 2). The output is $x_{j-1} \in X(j)$. Three registers of size n, where n is the binary length of p, are initialised to $|0\rangle$. In this way, the input state of this stage is

$$|\phi_0\rangle = |0\rangle^{\otimes n} |0\rangle^{\otimes n} |0\rangle^{\otimes n}$$

The gate U_f is an oracle implementing the function

$$f(a,b) = g^a \cdot x_j^{-b} \bmod p = g^a \cdot g^{x_{j-1}-b} \bmod p = g^{a-x_{j-1} \cdot b} \bmod p$$

where x_j and g are parameters. It should be noted that for a certain pair (a_1, b_1), we have $f(a_1, b_2) = f(a_2, b_2)$ if and only if

$$(a_2, b_2) = (a_1, b_1) + \lambda \cdot (x_{j-1}, 1)$$

for any $\lambda \in \mathbb{Z}_{p-1}$. The pair $(x_{j-1}, 1)$ is called the *period* of the function f.

A brief description of the steps performed in this part is given below (Jozsa 2001). They are heavily based on number theory and are justified in Shor (1997).

The first step of the algorithm is to apply the Hadamard gate to the first two registers to give

$$\begin{aligned} |\phi_1\rangle &= H^{\otimes 2\cdot n} |\psi_0\rangle \\ &= \frac{1}{\sqrt{2^n}} \cdot \frac{1}{\sqrt{2^n}} \sum_{a=0}^{2^n-1} |a\rangle \sum_{b=0}^{2^n-1} |b\rangle\, |0\rangle^{\otimes n} \\ &= \frac{1}{2^n} \sum_{a=0}^{2^n-1} \sum_{b=0}^{2^n-1} |a\rangle\, |b\rangle\, |0\rangle^{\otimes n} . \end{aligned}$$

In the context of the discrete logarithm, only the values of a and b that belong to $\mathbb{Z}_{p-1}$ are of interest because they belong to the domain of the function f implemented by the oracle. For those values that are not of interest, a comparison with p at the end of the process can be performed, and a new execution of the algorithm carried out if the value obtained is greater than p. If we only consider the values in the domain of f, we can denote $|\phi_1\rangle$ by

$$|\phi_1\rangle = \frac{1}{p-1} \sum_{a=0}^{p-2} \sum_{b=0}^{p-2} |a\rangle\, |b\rangle\, |0\rangle^{\otimes n} .$$

The next step of this stage is to apply the gate U_f to the input. The result of this operation is the state

$$|\phi_2\rangle = \frac{1}{p-1} \sum_{a=0}^{p-2} \sum_{b=0}^{p-2} |a\rangle\, |b\rangle\, |f(a,b)\rangle .$$

As shown in Figure 6, we now need to measure the third register. This measurement will collapse the superposition and will result in the value $\omega_0 = f(a_0, b_0)$ in the third register. Making use of the periodicity of f, the quantum system enters the periodic state $|\phi_3\rangle$:

$$|\phi_3\rangle = \frac{1}{\sqrt{p-1}} \sum_{k=0}^{p-1} |a_0 + k \cdot x_{j-1}\rangle\, |b_0 + k\rangle .$$

In order to eliminate the dependence on the values (a_0, b_0) chosen randomly after the measurement in the third register, the quantum Fourier transform modulo $p-1$ is applied to the remaining registers.

Let the quantity $\imath$ denote the unit imaginary number and let χ_{ℓ_1,ℓ_2} be a function defined by

$$\chi_{\ell_1,\ell_2}(a,b) = \exp\left[\frac{2 \cdot \imath \cdot \pi \cdot (a \cdot \ell_1 + b \cdot \ell_2)}{p-1}\right].$$

The application of the QFT gate to the first two registers will yield an equally weighted superposition of those labels (ℓ_1, ℓ_2) such that

$$\chi_{\ell_1,\ell_2}(x_{j-1}, 1) = 1,$$

that is,

$$x_{j-1} \cdot \ell_1 + \ell_2 \equiv 0 \bmod p-1,$$

so

$$\begin{aligned} \ell_2 &= -x_{j-1} \cdot \ell_1 \bmod p-1 \\ \ell_1 &= 0, \dots, p-2 \end{aligned}$$

(Jozsa 2001). Taking these considerations into account, the state of the system after the quantum Fourier transform is given by

$$|\phi_4\rangle = \frac{1}{\sqrt{p-1}} \sum_{\ell_1=0}^{p-2} \exp\left[\frac{2 \cdot \imath \cdot \pi}{p-1} \cdot \left(a_0 \cdot \ell_1 - b_0 \cdot x_{j-1} \cdot \ell_1\right)\right] |\ell_1\rangle \, |-x_{j-1}\ell_1\rangle \, .$$

The last step consists of a measurement in the first and second registers, which will return

$$(\ell_1, -x_{j-1}\ell_1 \bmod p-1).$$

Two situations must be considered:

(A) ℓ_1 and p are coprime:
In this case, we need to find ℓ_1^{-1}, the multiplicative inverse modulo $p-1$. This value must then be multiplied by the value observed in the second register to give x_{j-1}.

(B) ℓ_1 and p are not coprime:
This indicates a failure of the algorithm, so another execution with the same input must be carried out.

In order to increase confidence in the result, given the possible occurrence of failures, we need to carry out $O(\log\log p)$ repetitions, as a result of which, x_{j-1} will be obtained with high probability.

This process must now be repeated, but replacing j by $j-1$, that is, the input will now be x_{j-1}, p and g and the output x_{j-2}. The repetitions of the quantum discrete logarithm, as the index j decreases, continue until x_0 is obtained. As a result, the generator's complete internal state is recovered, which successfully concludes the quantum permanent compromise attack on the Blum–Micali generator.

In the next section, we will give an example of a quantum attack on a Blum–Micali generator, and then discuss the complexity analysis in Section 6.

5. Example of the quantum attack on the Blum–Micali generator

To illustrate the attack on the Blum–Micali generator, we will assume that the adversary knows the public parameters $p = 7$ and $g = 3$, and has discovered three sequenced bits $\mathbf{b}(3) = \{001\}$. Following the procedures indicated in Section 4, the quantum circuit to perform the first and second stages is illustrated in Figure 7.

The steps of the first and second stages will be described according to the evolution of the state $|\varphi\rangle$ from $|\varphi_0\rangle$ to $|\varphi_8\rangle$. The input of the algorithm is prepared in the following state according to the definition rules for each register:

$$|\varphi_0\rangle = |000\rangle \, |000\rangle \, .$$

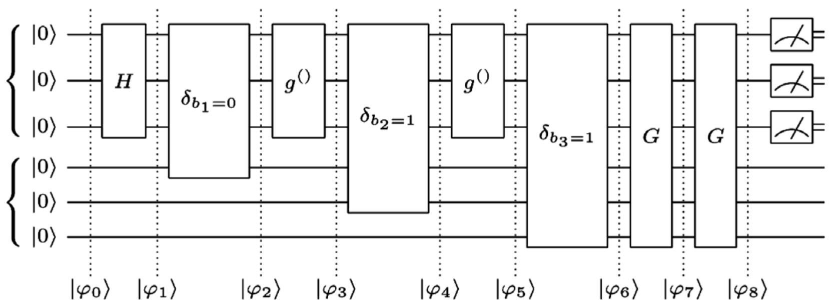

Figure 7. Quantum circuit to exemplify an attack on a Blum–Micali generator.

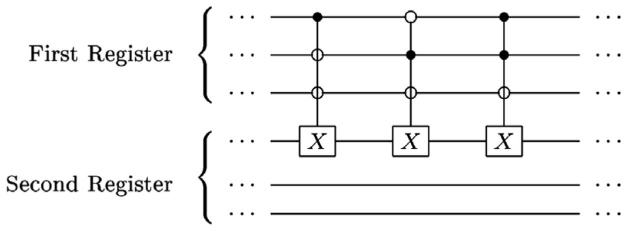

Figure 8. Decomposition of the gate $\delta_{b_1=0}$ into multiple controlled Pauli-X gates.

The next step is to apply gate H to the first register, with the result

$$|\varphi_1\rangle = \left[\frac{1}{\sqrt{8}}\left(|0\rangle + |1\rangle + \ldots + |7\rangle\right)\right]|000\rangle .$$

It can be seen that all the elements of the domain $\mathbb{Z}_7^* = \{1, 2, \ldots, 6\}$ are represented in the state $|\varphi_1\rangle$. The next step of the algorithm is to apply gate $\delta_{b_1=0}$, which, since $i = 1$, will associate with $|1\rangle$ in the first qubit of the second register all those elements in $\mathbb{Z}_7^*$ that would have produced the bit $b_1 = 0$. The result of this operation is

$$\begin{aligned} |\varphi_2\rangle \;=\; & \frac{1}{\sqrt{8}}\left(|0\rangle|000\rangle + |3\rangle|100\rangle + |2\rangle|100\rangle + |6\rangle|000\rangle\right. \\ & \left.+ |4\rangle|000\rangle + |5\rangle|000\rangle + |1\rangle|100\rangle + |7\rangle|000\rangle\right). \end{aligned}$$

The gate $\delta_{b_1=0}$ can be implemented using multiple controlled Pauli-X gates, as shown in Figure 8. This illustration considers the upper qubit on the first register as the least significant one. The $\bullet$ denotes controls that are activated by the $|1\rangle$ state, and the $\circ$ denotes controls that are activated by the $|0\rangle$ state.

Proceeding with the attack, the next step is to apply the $g^{()}$ to the input state. This gate is defined by

$$g^{()} = |0\rangle\langle 0| + |3\rangle\langle 1| + |2\rangle\langle 2| + |6\rangle\langle 3| + |4\rangle\langle 4| + |5\rangle\langle 5| + |1\rangle\langle 6| + |7\rangle\langle 7| .$$

The state after applying $g^{()}$ is given by

$$\begin{aligned} |\varphi_3\rangle \;=\; & \frac{1}{\sqrt{8}}\left(|0\rangle|000\rangle + |3\rangle|100\rangle + |2\rangle|100\rangle + |6\rangle|100\rangle\right. \\ & \left.+ |4\rangle|000\rangle + |5\rangle|000\rangle + |1\rangle|000\rangle + |7\rangle|000\rangle\right). \end{aligned}$$

The first stage continues in a similar fashion, but using the relevant definitions of $\delta_{b_2=1}$ and $\delta_{b_3=1}$. Performing the next three steps, the resulting state after the identification of the representative stage of the quantum attack is

$$\begin{aligned}|\varphi_6\rangle \;=\; & \frac{1}{\sqrt{8}}\big(|0\rangle\,|000\rangle + |1\rangle\,|100\rangle + |2\rangle\,|110\rangle + |3\rangle\,|010\rangle \\ & + |4\rangle\,|001\rangle + |5\rangle\,|001\rangle + |6\rangle\,|111\rangle + |7\rangle\,|000\rangle\big)\,.\end{aligned}$$

Note that the only state associated with 111 in the second register is the $|6\rangle$. It is also important to note that all amplitudes are equal, that is, a measurement of the first register would return any number from 0 to 7 with the same probability.

Rewriting the state $|\varphi_6\rangle$ as a partition, we get

$$\begin{aligned}|\varphi_7\rangle \;=\; & \frac{1}{\sqrt{8}}\,|6\rangle\,|111\rangle \;+\; \frac{1}{\sqrt{8}}\big(|0\rangle\,|000\rangle + |1\rangle\,|100\rangle\,|2\rangle\,|110\rangle \\ & + |3\rangle\,|010\rangle + |4\rangle\,|001\rangle + |5\rangle\,|001\rangle + |7\rangle\,|000\rangle\big) \\ =\; & \frac{1}{\sqrt{8}}\,|x_j\rangle\,|111\rangle + \sqrt{\frac{7}{8}}\,|x_{j\perp}\rangle\,|y\rangle \\ =\; & \sin(\theta)\,\big|\psi_{x_j}\big\rangle + \cos(\theta)\,\big|\psi_{x_{j\perp}}\big\rangle\end{aligned}$$

where:

— $j = 3$;
— $|x_j\rangle = |6\rangle$;
— $|x_{j\perp}\rangle$ denotes the subspace ortogonal to $|x_j\rangle$;
— $y \neq 111$;
— $\big|\psi_{x_j}\big\rangle = |6\rangle\,|111\rangle$;
— $\big|\psi_{x_j\perp}\big\rangle = |x\rangle\,|y\rangle$; and
— $\theta \in \big(0, \pi/2\big)$ satisfies $\theta = \arcsin\big(1/\sqrt{8}\big) = 0.36$ radians.

The next step of the algorithm is to perform the amplitude amplification stage, but before we can do this, we need to determine how many Grover iterations are required. Taking into account that $n = 3$, we get that $k = \big\lfloor(\pi/4)\sqrt{7})\big\rceil = 2$ iterations are required.

Therefore, $k = 2$ Grover iterations on $|\varphi_6\rangle$ will result in

$$\begin{aligned}|\varphi_8\rangle \;=\; & G^2\,|\varphi_6\rangle \\ =\; & \sin((2\cdot 2+1)\cdot\theta)\,\big|\psi_{x_j}\big\rangle + \cos((2\cdot 2+1)\cdot\theta)\,\big|\psi_{x_{j\perp}}\big\rangle \\ =\; & \sin(5\theta)\,\big|\psi_{x_j}\big\rangle + \cos(5\theta)\,\big|\psi_{x_{j\perp}}\big\rangle \\ =\; & \sin(1.8)\,\big|\psi_{x_j}\big\rangle + \cos(1.8)\,\big|\psi_{x_{j\perp}}\big\rangle\,.\end{aligned}$$

A measurement in $|\varphi_8\rangle$ will result in 6 with probability $|\sin(1.8)|^2 \approx 94.83\%$. Assuming the adversary retrieved the generator's internal state 6 after a measurement, the second stage has now been successfully concluded.

With the generator's internal state $x_3 = 6$, the adversary is able to predict the next output of the generator under attack. But the adversary does not know the previous elements of the generator's internal state. To recover them, we need to perform the third stage of the quantum attack – the generator's internal state recovery.

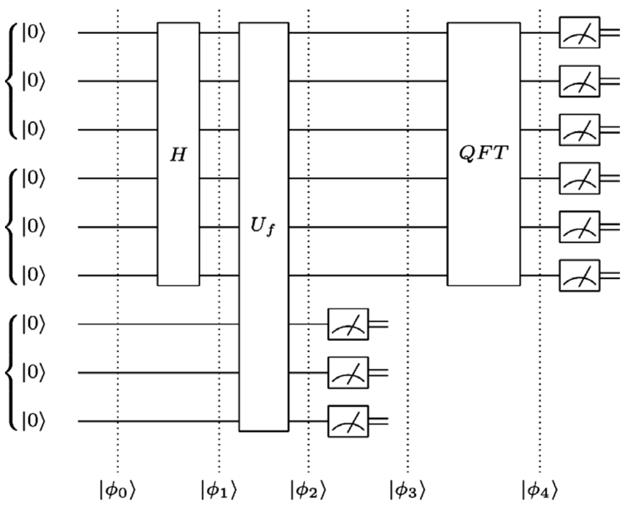

Figure 9. Quantum circuit for the internal state recovery stage to exemplify an attack on a Blum–Micali generator.

The third stage starts with the input $x_3 = 6$, $p = 7$ and $g = 3$. The oracle will implement the function

$$f(a,b) = 3^a \cdot 6^{-b} \bmod 7 = 3^a \cdot (3^{x_2})^{-b} \bmod 7 = 3^{a-x_2 \cdot b} \bmod 7.$$

The circuit implementing the third stage is shown in Figure 9. In order to illustrate how the next steps of this example are carried out, we will make some assumptions as we go along about measurement outcomes and about how the oracle works.

The initial state $|\phi_0\rangle$ is a tensor product of the three registers initialised to $|0\rangle$:

$$|\phi_0\rangle = |000\rangle\,|000\rangle\,|000\rangle\,.$$

The first step uses the Hadamard gate to put the first two registers in superposition:

$$|\phi_1\rangle = \frac{1}{8}\sum_{a=0}^{7}\sum_{b=0}^{7}|a\rangle\,|b\rangle\,|000\rangle\,.$$

Restricting the sum to the values in the domain of f, we get $|\phi_1\rangle$ is given by

$$|\phi_1\rangle = \frac{1}{6}\sum_{a=0}^{5}\sum_{b=0}^{5}|a\rangle\,|b\rangle\,|000\rangle\,.$$

Applying the oracle U_f to the state $|\phi_1\rangle$ gives the state

$$|\phi_2\rangle = \frac{1}{6}\sum_{a=0}^{5}\sum_{b=0}^{5}|a\rangle\,|b\rangle\,|f(a,b)\rangle\,.$$

The next step is a measurement of the third register. Note that the possible values of (a,b) are equiprobable as a consequence of the superposition created by the Hadamard gate.

For example, let us assume that the system collapsed to $a_0 = 4$ and $b_0 = 1$, and that, in consequence, the adversary obtained the value $\omega_0 = f(4, 1) = 3^4 \cdot 6^{-1} \bmod 7 = 3$ from the measurement of the third register. Note that despite our assumption about the values assumed by a_0 and b_0 (which we will continue to make in illustrating the further steps of the third stage), they are not known by the adversary, who only sees $\omega_0 = 3$.

Taking the periodicity of f into account, the state of the system after the measurement is

$$|\phi_3\rangle = \frac{1}{\sqrt{6}} \sum_{k=0}^{5} |4 + k \cdot x_2\rangle\, |1 + k\rangle\,.$$

Removing the dependency on the values $a_0 = 4$ and $b_0 = 1$ using the quantum Fourier transform, the state $|\phi_4\rangle$ is:

$$|\phi_4\rangle = \frac{1}{\sqrt{6}} \sum_{\ell_1=0}^{5} \exp\left[\frac{2 \cdot \imath \cdot \pi}{6}(4 \cdot \ell_1 - x_2 \cdot \ell_1)\right] |\ell_1\rangle\, |{-x_2} \cdot \ell_1\rangle\,. \tag{8}$$

A measurement must now be made in the first two registers. Note that from the 6 different values that ℓ_1 can assume in Equation (8), only zero is not coprime with $p = 7$, and this is the only value that would require a new execution of the algorithm. Again, for example, let us assume that the superposition of the values of ℓ_1 collapsed to $\ell_1 = 1$. Since

$$4 \cdot 1 - x_2 \cdot 1 \equiv 0 \bmod 6,$$

it turns out that $x_2 = 3$ (but the adversary does not know this yet because it is encoded in the phase of the quantum system). Upon observing the value of the second register, the adversary gets $-x_2 \cdot \ell_1 = 3$. Note that $\ell_1 = 1$ and $p = 7$ are coprime, so we are in situation (A) of the two situations listed at the end of Section 4.3 and we can proceed. The multiplicative inverse of ℓ_1, denoted by ℓ_1^{-1}, is equal to 1, so the adversary finally obtains $x_2 = 3$, which turns out to be an element of the generator's internal state.

Once x_2 is known, the adversary must apply the quantum discrete logarithm two more times: first, taking as input $p = 7$, $g = 3$ and $x_2 = 3$ to recover x_1; and then, with $p = 7$, $g = 3$ and x_1, to recover the seed x_0. We will not work through these steps here, but they will lead the adversary to obtain $X(3) = \{x_3 = 6, x_2 = 3, x_1 = 1, x_0 = 6\}$. With this information, the adversary has successfully concluded the permanent compromise attack on the Blum–Micali generator, thereby completely compromising its unpredictability.

6. Complexity analysis of the quantum attack on the Blum–Micali generator

To perform the complexity analysis of the described attack, all three stages must be considered.

In the first stage, the input has size $n + j$ qubits and the output is a quantum state of $n + j$ qubits. The number of gates needed to perform the initial superposition, the modular exponentiations and the function δ is equal to $2 \cdot j$, since two gates of the type δ_{b_i} are separated by gates $g^{()}$. Each gate acts in a single step on the input, so the cost of this first stage is

$$(2 \cdot j) \cdot O(1) = O(2 \cdot \log p) = O(\log p),$$

since $j = \log p$.

The second stage has an input size of $n + j$ qubits and an output size of n bits. The amplitude amplification performs $\sqrt{p}$ operations, each of unitary cost, so the resulting complexity of the second stage is $O(\sqrt{p})$. Note that this complexity is a result of our adopting the Grover inspired procedure.

The final stage is composed of applications of the quantum discrete logarithm procedure. The input has size of $3 \cdot n$ qubits and an output of $3 \cdot n$ bits. The complexity of each execution is $O(n^3)$ or, similarly, $O(\log^3 p)$. Thus, considering that $j = \log p$ executions are required with $O(\log \log p)$ repetitions each, the cost is

$$O(\log \log p) \cdot \log p \cdot O(\log^3 p) = O(\log \log p \cdot \log^4 p).$$

The overall cost of the quantum attack on the Blum–Micali generator is the sum of the costs of its stages:

$$O(\log p) + O(\sqrt{p}) + O(\log \log p \cdot \log^4 p) = O(\log \log p \cdot \log^4 p).$$

Therefore, the resulting complexity of the quantum algorithm is in sharp contrast with its classical counterpart complexity, which is , as shown in Section 3.1, $O(2^p \cdot \log p)$, that is, we have achieved a superpolynomial speedup, and we can conclude that the Blum–Micali pseudorandom generator is not secure against quantum computing attacks.

7. Generalising the quantum attack

The quantum attack described in Section 4 is aimed at the Blum–Micali pseudorandom generator. However, with some modifications, it can also be used as a framework for quantum attacks on certain other pseudorandom generators, or in some situations where the requirements regarding the bits intercepted are not completely satisfied. Such generalisations are characterised and presented in this section.

7.1. *Multiple hard-core predicates*

The Blum–Micali generator requires a considerable computational effort to produce bits. According to its original definition, just a single bit is extracted per iteration, that is, per modular exponentiation performed (Blum and Micali 1984).

In an attempt to improve the production of bits without compromising the security of this generator, some authors have explored the possibility of extracting more bits per iteration. Long, Wigderson and Peralta showed that $O(\log \log p)$ bits could be extracted by a single iteration of the Blum–Micali generator (Long and Wigderson 1988; Peralta 1986). Håstad *et al.* also showed that if we consider a discrete logarithm modulo a composite integer, then nearly $n/2$ bits can be extracted per modular exponentiation (Håstad *et al.* 1993). Other authors have shown that similar extractions are possible even when the exponent is small (Patel and Sundaram 1998; Gennaro 2005). In summary, this line of work tries to improve the Blum–Micali generator by the addition of multiple hard-core predicates.

For each hard-core predicate γ of the Blum–Micali generator, there is an associated quantum gate. Such a gate, say $\mathscr{Y}$, is capable of determining if a certain control register x produced a certain bit b_i ($\gamma(x) = b_i$) by registering this information in a target qubit $|y\rangle$ according to the following procedure:

$$\mathscr{Y}_{b_i} |x\rangle |y\rangle_i = \begin{cases} |x\rangle |\overline{y}\rangle_i & \text{if } x \in \mathbb{Z}_p^* \text{ and } \gamma(x) = b \\ |x\rangle |y\rangle_i & \text{otherwise} \end{cases} \tag{9}$$

where the operation denoted by $|\overline{y}\rangle_i$ indicates the application of the Pauli-X gate to the ith qubit of the second register.

If the adversary intends to attack one version of the Blum–Micali generator with multiple hard-core predicates, he would register in $\mathbf{b}$ the tuples of bits intercepted per iteration, per hard-core predicate. For example, suppose a Blum–Micali generator has three hard-core predicates. By eavesdropping on this generator, the adversary has discovered that it produced the bits 111, 011 and 100 per hard-core predicate, respectively. According to the specified procedure, he would register in $\mathbf{b}$ the following tuples: $\{(1, 0, 1), (1, 1, 0), (1, 1, 0)\}$.

In the construction of the quantum circuit to attack the generator, the adversary must consider one extra register per hard-core predicate. Each extra register must have dimension and initialisation procedures similar to the ancillary qubits register described in Section 4.1. The gates related to the hard-core predicate must be inserted between the two successive $g^{()}$ gates that implement the modular exponentiation operation.

This shows that the quantum attack can be adapted to these more general situations. This is an important consideration taking into account that, due to the efficiency in the production of bits, multiple hard-core predicates in the Blum–Micali generator are more likely to be used in real-world cryptosystems. In addition, it is possible to attack the generator more effectively since more information about the generator's internal state is captured per iteration.

7.2. *Blum–Micali construction*

The Blum–Micali construction is a family of cryptographically secure pseudorandom number generators. They are defined by a one-way permutation f over a domain $\mathscr{D}$ and by a hard-core predicate ϕ of the one-way permutation. The seed x_0 is obtained through a random choice of an element in the domain ($x_0 \in_R \mathscr{D}$). The productions (bits) from these generators are obtained as follows:

$$\begin{aligned} x_i &= f(x_{i-1}), \\ b_i &= \phi(x_i). \end{aligned}$$

There are three commonly used generators in this family: the *Blum–Micali generator*, presented in Section 2; the *Blum–Blum–Shub generator*; and the *Kaliski generator*.

The Blum–Blum–Shub generator's one-way permutation is the *Rabin function* $f : \mathbb{Z}_M \to \mathbb{Z}_M$, such that $f(x) = x^2 \bmod M$, where M is the product of two primes both congruent to 3 mod 4. The domain of this generator is $QR_M = (\mathbb{Z}_M^*)^2$. Its hard-core predicate returns the jth bit from the given parameter, where j is previously fixed. The values of M and

j are publicly available. The Blum–Blum–Shub generator is one of the most efficient pseudorandom number generators known that is provably secure under the assumption that factoring large composites is intractable (Blum *et al.* 1986).

The Kaliski generator is based on the elliptic curve discrete logarithm. Let p be a prime with $p \equiv 2 \bmod 3$, and consider a curve $E(\mathbb{F}_p)$ that consists of points $(x, y) \in \mathbb{F}_p \times \mathbb{F}_p$ such that $y^2 = x^3 + c$, where $c \in \mathbb{F}_p$. The points of $E(\mathbb{F}_p)$ together with a point at infinity, denoted by $\mathcal{O}$, form a cyclic additive group of order $p + 1$, which is the domain of the generator. Let Q be a generator of this group and φ be a function defined by

$$\varphi(P) = \begin{cases} y & \text{if } P \in E(\mathbb{F}_p) \\ p & \text{if } P = \mathcal{O}. \end{cases}$$

The seed of the Kaliski generator is a random point on the curve. The one-way permutation of this generator is a function f such that $f(P) = \varphi(P) \cdot Q$. The hard-core predicate of this function returns 1 if $\varphi(P) \geqslant (p + 1)/2$, and 0 otherwise. The security of the Kaliski generator is based on the the assumption that the elliptic curve discrete logarithm is intractable (Kaliski 1988).

As a one-way permutation defines a bijection over its domain, it is possible to build quantum gates to implement such functions (Williams 2011). Let f be a one-way permutation over the domain $\mathcal{D}$, and $\mathcal{Q}$ be a quantum gate that implements f. The gate $\mathcal{Q}$ performs the following transformations:

$$\mathcal{Q}\,|x\rangle = \begin{cases} |f(x)\rangle & \text{if } x \in \mathcal{D} \\ |x\rangle & \text{otherwise.} \end{cases}$$

Moreover, as shown in Equation (9), there is a quantum gate associated with every hard-core predicate. It turns out that it is possible to construct attacks on every generator of the Blum–Micali construction analogous to the attack shown in Section 4. The main modification is the identification of the representative stage in which the gates $g^{()}$ and δ_{b_i} must be replaced by the equivalent gates of the generator of the Blum–Micali construction under attack.

The gain achieved when generators of the Blum–Micali construction are attacked with quantum computing instead of classical computing is very likely to be the same as for the Blum–Micali generator since there are algorithms to perform factoring and elliptic curve discrete logarithms in polynomial time (Shor 1997; Proos and Zalka 2004).

It is important to note that it is also possible to attack the Blum–Blum–Shub and Kaliski generators with multiple hard-core predicates by following the procedures shown in Section 7.1. Furthermore, detailed examples of attacks on the Blum–Micali construction generators can be found in Guedes *et al.* (2010a).

7.3. *Attacks with non-consecutive bits or with fewer bits than required*

A very realistic situation that must be considered is the case in which the adversary fails to intercept adequate sequences of bits to carry out the attack. In this scenario, it may be that only non-consecutive bits or fewer bits than required are available. Such situations still gather information that can favour the adversary and must be considered in the design of an attack.

Consider first the retrieval of non-consecutive bits. In this case the adversary must keep the bits and record in which iteration they were intercepted. For instance, if an adversary retrieved the first three bits of a generator, the fifth and the eighth, he could store such bits in the following way: $\mathbf{b} = \{b_1 = 1, b_2 = 1, b_3 = 0, b_5 = 1, b_8 = 0\}$.

In order to build the circuit for the quantum attack when the bits intercepted are non-consecutive, we must make some changes in the identification of the representative part: if two bits, say b_{b_x} and b_{b_y}, are consecutive, one gate $g^{()}$ must be placed between the gates δ_{b_x} and δ_{b_y}, as specified in Section 4.1; otherwise $|x - y|$ gates of type $g^{()}$ must be inserted between the gates δ_{b_x} and δ_{b_y}.

The addition of extra $g^{()}$ gates reproduces the iterations performed by the generator in which the adversary skipped the bits output. Despite allowing the attack, this configuration does require more gates and thus increased computational effort in comparison with the scenario where the bits intercepted are consecutive.

The second situation to consider is where the adversary has intercepted fewer bits than required to identify precisely the representative of the generator's internal state, say x bits where $x < \lceil \log p \rceil$. This requires two modifications to the attack. The first consists of a subspace of possible representatives instead of the single element shown previously in Equation (6). The second is a modification in the number of iterations required, which is now given by

$$k = \left\lfloor \frac{\pi}{4} \sqrt{\frac{p}{2^{(\lceil \log p \rceil - x)}}} \right\rceil .$$

That is, multiple solutions in the amplitude amplification stage must be considered. This is a consequence of the fact that the number of bits intercepted by the adversary does not allow a precise identification of the representative of the generator but of a set of possible candidates, all of them equally like to be the representative.

8. Final remarks

This paper presented a quantum attack against the Blum–Micali pseudorandom generator. In this attack, an adversary intercepts some bits output by the generator and, using a quantum computer and the procedures described, is able to retrieve the generator's internal state. In consequence, all previous and future outputs are known by the adversary, thereby compromising the unpredictability of the generator.

The quantum permanent compromise attack on the Blum–Micali generator is a three stage procedure. In the first stage, the representative of the generator's internal state is identified at a quantum level. In the second stage, the amplitude of the representative is amplified to increase its probability of being measured. Finally, quantum discrete logarithm operations are performed to recover the generator's entire internal state. This quantum attack makes use of a Grover inspired procedure and Shor's discrete logarithm algorithm, that is, it combines two of the most famous quantum algorithms.

The proposed attack provides a superpolynomial speedup when compared with its classical counterpart. This result means that the security of the Blum–Micali generator in the face of the quantum computing paradigm no longer holds. In addition to the characterisation of the attack, the number of bits to perform this quantum attack successfully is also defined to be $\log p$ bits on average.

One might think that since the security of the Blum–Micali generator is based on the hypothesis of intractability of the discrete logarithm problem, we could just use the algorithm proposed in Shor (1997) in the design of an attack on the generator. However, this is not possible because of the input requirements of this algorithm. To see why this is the case, recall that the quantum discrete logarithm procedure demands a pair g and x_j to build an oracle, but that x_j is still hidden in the generator's internal state. Therefore, an attack based exclusively on Shor's discrete logarithm algorithm is impossible.

In addition to the description of a quantum permanent compromise attack on the Blum–Micali generator, we have also suggested generalisations that allow modifications to the algorithm to enable attacks on Blum–Micali generators with multiple hard-core predicates and on generators belonging to the Blum–Micali construction family, as well as in situations where the requirements on the intercepted bits are relaxed.

The contributions of this work can be divided into two areas: contributions to cryptanalysis and contributions to quantum computing. The contributions of this paper in the area of cryptanalysis are:

(i) improvements in the knowledge of the possible attacks against pseudorandom generators that have been adopted in many real-world cryptosystems;
(ii) the definition of algorithms to attack pseudorandom generators; and
(iii) an analysis of the security of certain pseudorandom generators against threats from quantum computing.

The contributions of this paper in the area of quantum computing are:

(i) the development of new quantum computing algorithms;
(ii) the description of threats against classical cryptosystems; and
(iii) speedups over equivalent classical algorithms.

Some suggestions for future work are:

(i) an estimate of bounds on the number of bits required to attack the Blum–Blum–Shub and Kaliski generators; and
(ii) an expansion of the quantum attack to a wider range of cryptographically secure pseudorandom generators, such as those defined by one-way functions (Håstad *et al.* 1999).

Contributions in these areas would increase the applicability of the proposed attack.

Acknowledgements

We are grateful for the valuable comments provided by the anonymous reviewers and A. F. de Lima, G. O. Santos, A. P. Mendonca and M. F. Fragoso.

References

Ambainis, A. (2004) Quantum search algorithms. *SIGACT News* **35** 22–35.

Bennett, C. (1973) Logical Reversibility of Computation. *IBM Journal of Research and Development* **17** 525–532.

Blum, L., Blum, M. and Shub, M. (1986) A Simple Unpredictable Pseudo-Random Number Generator. *SIAM Journal on Computing* **15** 364–383.

Blum, M. and Micali, S. (1984) How to generate cryptographically strong sequences of pseudo-random bits. *SIAM Journal on Computing* **13** 850–864.

Boyar, J. (1989) Inferring Sequences Produced by Pseudo-Random Number Generators. *Journal of the Association for Computing Machinery* **36** 139–141.

Cloutier, D. R. and Holden, J. (2010) Mapping the discrete logarithm. *Involve* **3** 197–213.

Eastlake, D., Schiller, J. and Crocker, S. (2005) Randomness Requirements for Security. Technical report, Network Working Group – Request for Comments 4086.

Gennaro, R. (2005) An improved pseudo-random generator based on the discrete logarithm problem. *Journal of Cryptology* **18** (2) 91–110.

Gentle, J. E. (2003) *Random Number Generation and Monte Carlo Methods*, Springer-Verlag.

Goldreich, O. (2005) *Foundations of Cryptography – A Primer*, now Publishers Inc.

Gregg, J. A. (2003) On factoring integers and evaluating discrete logarithms, Master's thesis, Harvard College.

Grover, L. K. (1997) Quantum mechanics helps in searching for a needle in a haystack. *Physical Review Letters* **79** 325–328.

Guedes, E. B., de Assis, F. M. and Lula, B. Jr. (2010a) Examples of the Generalized Quantum Permanent Compromise Attack on the Blum–Micali Construction. (Available at `http://arxiv.org/abs/1012.1776`.)

Guedes, E. B., de Assis, F. M. and Lula, B. Jr. (2010b) A Generalized Quantum Permanent Compromise Attack on the Blum–Micali Construction. In: III Workshop-School of Quantum Computation and Information (WECIQ).

Hirvensalo, M. (2001) *Quantum Computing*, Springer-Verlag.

Håstad, J., Impagliazzoy, R., Levinz, L. A. and Luby, M. (1999) A pseudorandom generator from any one-way function. *SIAM Journal on Computing* **28** (4) 1364–1396.

Håstad, J., Schrift, A. and Shamir, A. (1993) The discrete logarithm modulo a composite hides $o(n)$ bits. *Journal of Computer and System Sciences* **47** 376–404.

Jozsa, R. (2001) Quantum factoring, discrete logarithms, and the hidden subgroup problem. *Computing in Science and Engineering* **3** (2) 34–43.

Kaliski, B. S. (1988) *Elliptic Curves and Cryptography: A Pseudorandom Bit Generator and Other Tools*, Ph.D. thesis, MIT.

Kelsey, J., Schneider, B., Wagner, D. and Hall, C. (1998) Cryptanalytic attacks on pseudorandom number generators. *Springer-Verlag Lecture Notes in Computer Science* **1372** 168–188.

Krawczyk, H. (1992) How to Predict Congruential Generators. *Journal of Algorithms* **13** 527–545.

Long, D. and Wigderson, A. (1988) The discrete log hides $o(logn)$ bits. *SIAM Journal on Computing* **17** 363–372.

Meter, R. V. and Itoh, K. M. (2005) Fast Quantum Modular Exponentiation. *Physical Review A* **71** (5) 052320.

Nielsen, M. A. and Chuang, I. L. (2005) *Quantum Computation and Information*, Bookman.

Paar, C. and Pelzl, J. (2010) *Understanding Cryptography*, Springer-Verlag.

Patel, S. and Sundaram, G. (1998) An efficient discrete log pseudo random generator. In: Krawczyk, H. (ed.) Advances in Cryptology – CRYPTO'98. Proceedings 18th Annual

International Cryptology Conference. *Springer-Verlag Lecture Notes in Computer Science* **1462** 304–317.

Peralta, R. (1986) Simultaneous security of bits in the discrete log. In: Pichler, F. (ed.) Advances in Cryptology – EUROCRYPT'85. *Springer-Verlag Lecture Notes in Computer Science* **219** 62–72.

Proos, J. and Zalka, C. (2004) Shor's discrete logarithm quantum algorithm for elliptic curves. *Quantum Information and Computation* **3** (4) 317–344.

Shor, P. (1997) Polynomial-time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM Journal on Computing* **26** 1484–1509.

Sidorenko, A. and Schoenmakers, B. (2005a) Concrete security of the Blum–Blum–Shub pseudorandom generator. In: Smart, N. P. (ed.) Proceedings Cryptography and Coding: 10th IMA International Conference. *Springer-Verlag Lecture Notes in Computer Science* **3796** 355–375.

Sidorenko, A. and Schoenmakers, B. (2005b) State recovery attacks on pseudorandom generators. In: Wolf, C., Lucks, S. and Yau, P.-W. (eds.) Proceedings Western European Workshop on Research in Cryptology. *GI Lecture Notes in Informatics* **74** 53–63.

van Tilborg, H. C. (2005) *Encyclopedia of Cryptography and Security*, Springer-Verlag.

Williams, C. P. (2011) *Explorations in Quantum Computing*, second edition, Springer-Verlag.