Key-Evolution Schemes Resilient to Space-Bounded Leakage*

Stefan Dziembowski[1] and Tomasz Kazana[2] and Daniel Wichs[3]

[1] University of Warsaw and Sapienza University of Rome
[2] BioInfoBank Research Institute and University of Warsaw
[3] New York University

Abstract. Much recent work in cryptography attempts to build secure schemes in the presence of *side-channel leakage* or leakage caused by malicious software, like computer viruses. In this setting, the adversary may obtain some additional information (beyond the control of the scheme designer) about the internal secret state of a cryptographic scheme. Here, we consider key-evolution schemes that allow a user to evolve a secret-key K_1 via a *deterministic* function f, to get updated keys $K_2 = f(K_1), K_3 = f(K_2), \ldots$. Such a scheme is *leakage-resilient* if an adversary that can leak on the first i steps of the evolution process does not get any useful information about any future keys. For such schemes, one must assume some restriction on the *complexity* of the leakage to prevent *pre-computation attacks*, where the leakage on a key K_i simply pre-computes a future key K_{i+t} and leaks even a single bit on it.

Much of the prior work on this problem, and the restrictions made therein, can be divided into two types. Theoretical work offers rigor and provable security, but at the cost of having to make strong restrictions on the type of leakage and designing complicated schemes to make standard reduction-based proof techniques go through (an example of such an assumption is the "only computation leaks" axiom). On the other hand, practical work focuses on simple and efficient schemes, often at the cost of only achieving an intuitive notion of security without formal well-specified guarantees.

In this paper, we complement the two tracks via a middle-of-the-road approach. On one hand, we rely on the random-oracle model. On the other hand, we show that even in the random-oracle model, designing secure leakage-resilient schemes is susceptible to pitfalls. For example, just assuming that leakage "cannot evaluate the random oracle" can be misleading. Instead, we define a new model in which we assume that the "leakage" can be any arbitrary *space bounded* computation that can make random oracle calls itself. We connect the space-complexity of a computation in the random-oracle modeling to the *pebbling complexity* on graphs. Using this connection, we derive meaningful guarantees for relatively simple key-evolution constructions.

Our scheme is secure also against a large and natural class of active attacks, where an attacker can leak as well as tamper with the internals of a device. This is especially important if the key evolution is performed on a PC that can be attacked by a virus, a setting considered by prior work in the *bounded retrieval model*

* The European Research Council has provided financial support to the first two authors of this paper under the European Community's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement no CNTM-207908.

(BRM)). This paper provides the first scheme were the adversary in the BRM can also modify the data stored on the machine.

keywords: graph pebbling, leakage-resilient cryptography, bounded-retrieval model

1 Introduction

In the recent years, there has been a growing interest in the design of cryptographic schemes that are secure even if implemented on a devices that can leak information. The motivation for this research comes from the fact that, in practice, it is very hard to construct hardware that does not reveal any "extra" information about its internal data. In particular, leakage on the internals can often be obtained using *side-channel attacks*, that exploit physical phenomena such as electromagnetic radiation [36,30], timing [5], power consumption [29], acoustic emanations [38], and many others (see e.g. [33] for an overview). Another case in which the adversary may obtain leakage from cryptographic protocols is the situation when the protocols are implemented on PCs on which the adversary can install malicious software, like the computer viruses.

The first papers proposing algorithmic countermeasures against the side-channel attacks came from the practitioners' community (e.g. [7]). Later this area attracted a lot of attention also from the theoreticians, starting from the seminal papers of Ishai et al. [24], Micali and Reyzin [31], and Akavia et al. [1]. The theoretical countermeasures against the virus attacks are known under the name *bounded-retrieval model* [8,14].

In this paper we consider the following natural problem. Suppose a cryptographic key K_0 is stored on a device that leaks information. If leakage occurs continuously, then the adversary may obtain more and more information about the key, and eventually learn it entirely. A natural idea to prevent this from happening is to periodically update the key i.e. to repeatedly apply some *key evolution function* f to it, obtaining a sequence of keys $K_0, K_1, \ldots$, where each $K_{i+1} := f(K_i)$. The key evolution function should be constructed in such a way that the evolved key K_i used in period i is indistinguishable from uniform, even if the adversary can leak from the entire evolution process $K_1 \to K_2 \to \ldots \to K_{i-1}$. We will assume that the key evolution is deterministic (i.e. it does not depend on any external randomness) hence these keys K_i will be shared by synchronized devices, which evolve their keys simultaneously but independently (without communication). As we will see, the design of key-evolution functions is also intimately related to the design of leakage-resilient stream-ciphers, and pseudo-random generators. The problem of designing such primitives been studied before, both by the practitioners and by the theoreticians. Next, we look at several models of leakage-resilience and their results.

1.1 Leakage Resilient Key Evolution: Theory vs. Practice

Theoretical work on leakage-resilience usually included a formal model for reasoning about leakage. Usually, side-channel leakage is modeled as a family of functions where the attacker can choose a function from the family and learn the output of this function applied to the secret key. For example, a popular and powerful model of Akavia et al. [1], allows the adversary to compute arbitrary poly-time leakage functions of the internal secret key, subject only to the constraint that the amount of data retrieved (the output-length of the function) is bounded. Unfortunately, when it comes to key-evolution, it is clear that security *cannot* be achieved in this model, even if the adversary is restricted to leaking a *single bit*! In fact, just given the ability to leak on the initial key K_1, the adversarial leakage-function can *pre-compute* any future key K_i and output (say) the first bit of it. If the adversary can leak even 1 bit in several consecutive rounds, the adversary can eventually recover any future key K_i in full! This example (called a *key-precomputation attack* in [19]) shows that, when considering the security of the key-evolution schemes, the sole restriction that the output of the leakage function is bounded does not suffice. However, it is also easy to see that the leakage-functions used in the above counter-examples are extremely artificial, and are very unlikely to model natural side-channel attacks that occur in real life. Hence, it is natural to look for different (weaker) models for the leakage, that still cover all the *realistic* attacks, but in which key-evolution schemes may exist. We survey two such key-evolution schemes in their corresponding models of leakage. The two schemes come from two very different point views: one theoretical with an emphasis on models and proofs, and the other practical with an emphasis toward efficiency and simplicity. Therefore, it is interesting to compare their advantages and disadvantages.

The scheme of Dziembowski and Pietrzak [19]. On the theoretical side, [19] constructed a stream cipher (and, implicitly, a key evolution scheme) in a formal model called "only computation leaks information", first proposed by Micali and Reyzin [31] and refined in [19]. In this model, the internal memory of the device is separated into two or more segments, and all computation is divided into simpler sub-computations that access only some small subset of these segments. The assumption is that, during each sub-computation, the adversary can leak an arbitrary bounded-length function of only the memory segments accessed by the sub-computation. In other words, during any computational step, data can leak if and only if it is accessed. Pre-computation attacks can therefore be prevented, since the adversary can never get any global leakage of the entire state of the system needed to compute a future key.

The actual scheme of [19] (and a related scheme of [34]) uses an alternating structure with two memory segments accessed in alternating rounds. The main drawback of this model is that it relies on the highly controversial assumption that data which is not accessed cannot leak. Also, the security of solutions in this model is highly dependent on "implementation details" such what data is accessed when.

The scheme of Kocher [28]. On the practical side, Kocher [28] proposes a simple and efficient solution to just use a "sufficiently complicated" cryptographic hash function for the key-evolution function f. This scheme seems intuitively secure since it's unlikely that any natural and therefore "sufficiently simple" leakage could learn anything about $K_{i+1} = f(K_i)$ from K_i (or even from the entire key-evolution process used to derive K_i), if f is "sufficiently complicated". However, [28] does not offer any meaningful model in which to analyze the above intuition. The main idea is to assume that the adversarial leakage on the ith update $K_i \rightarrow K_{i+1}$ can be an *arbitrary function* of the entire state of that update subject to the constraints: (1) the leakage function cannot make any random-oracle calls, (2) the output-length of the leakage function is bounded to be just slightly smaller than the key-length $|K|$. At first, it may seem that constraint (1) offers a meaningful way of capturing "sufficiently simple" leakage. Unfortunately, it is not clear what this constraint means in practice, and can lead to counter-intuitive consequences, described next.

Since the amount of leakage tolerated by the scheme should be close to $|K|$, it is natural to try to increase $|K|$ if one would want to achieve more leakage. Of course, that means using a key-evolution function, and therefore hash function, with sufficiently large input-size and output-size. Assume we start with a compression function $H : \{0,1\}^{2\ell} \to \{0,1\}^{\ell}$, and we want to allow key-size $|K| = t\ell$ to allow for more leakage. The standard technique for domain-extension is the Merkle-Damgård transformation [9] (and variants of it in the indifferentiability framework) which gives a function $H' : \{0,1\}^{*} \to \{0,1\}^{\ell}$. Now, to increase the output-size, we can define $\tilde{H} : \{0,1\}^{t\ell} \to \{0,1\}^{t\ell}$ by $\tilde{H}(K) = H(H'(K)||1), \ldots, H(H'(K)||t)$. But, it is clear that, if one leaks the ℓ-bit value $H'(K_1)$ used as sub-component of the key-evolution computation $K_2 = \tilde{H}(K_1)$, then one can compute all future keys K_j and thus completely break the key-evolution scheme! Therefore, the scheme is not secure with respect to even ℓ bits of leakage when instantiated with real-world hash functions. Notice that the leakage-function does not perform any complicated computation; it just leaks several consecutive bits of the internal state, corresponding to $H'(K)$, which is computed as an intermediate value during the key-evolution computation.

So we see that, although the initial scheme of Kocher provides some intuitive leakage-resilience properties, one runs into pitfalls when trying to model and quantify them. In particular, by relying on the random-oracle model in an unintended way (assuming that simple functions cannot make random oracle calls), and assuming that leakage on a computation making random-oracle calls only gets the input and output of such calls, one reaches a model that doesn't correspond to reality in a meaningful way.

The scheme of Yu Yu et al. [40] In a recent important work Yu Yu et al. [40] propose a practical scheme whose security is based on the assumptions that (1) the leakage functions cannot be chosen adaptively, and (2) the leakage function cannot evaluate the hash function (which is modeled as a random oracle). The security of the scheme that we construct in this paper does not require these assumptions. On the other hand, from the engineering point of view the assumptions made in [40] may look more attractive, since they are easier to verify empirically.

Other Models of Leakage-Resilience. We note that several other models of leakage resilience, with restricted leakage functions, have appeared in the literature. For example, [24] assumes leakage functions that leak individual wires from a circuit that performs a computation. Alternatively, Faust et al. [21] assume that leakage-function is an AC^0 circuit of the internal state. Several works consider computation that uses some small/simple leak-free components [22,21,23,26].

1.2 Our Model: Space-Bounded Leakage

In this paper we propose a method for the key evolution that combines the advantages of the Kocher's practical scheme (efficiency and simplicity of model, scheme) with some of the advantages of the theoretical scheme of [19] (provable security and scalability). On a high level, we restrict the class of leakage functions to ones that are bounded in the amount of "auxiliary work space" used during the computation of the output. In particular, this should model natural leakage which is unlikely to be sufficiently complex to require much space to compute. We will analyze a variant of the Kocher key-evolution scheme in the random-oracle model, but give all parties (including the leakage functions) the ability to compute the random oracle. In particular, we define a deterministic key-evolution function f that makes random-oracle calls to compute $K_{i+1} = f(K_i)$. On a high level, pre-computation attacks will not be possible because the space allowed to the leakage function is not sufficient to pre-compute K_{i+1} from K_i.

In greater detail, we model the leakage process on the key evolution as follows. We consider an adversary $\mathcal{A} = (\mathcal{A}_{small}, \mathcal{A}_{big})$ consisting of two parts: the adversary $\mathcal{A}_{big}$ corresponds to the external real-world attacker that tries to break the scheme, and $\mathcal{A}_{small}$ corresponds to the "space-bounded" device which is storing and evolving the secret key while leaking partial information to the external attacker $\mathcal{A}_{big}$. We do not assume anything about how the computation is implemented on the device, and thus allow the computation $\mathcal{A}_{small}$ itself to be adversarial. Initially, $\mathcal{A}_{small}$ is given the random starting key K_1. Since key evolution is deterministic, this will completely specify all future keys $K_2, K_3, \ldots$. Of course, we need somehow to "force" $\mathcal{A}_{small}$ to perform the key evolution (if $\mathcal{A}_{small}$ can completely "halt" the key evolution, then he can simply keep K_1 on $\mathcal{M}$ for a long time, and slowly retrieve it bit-by-bit). In the passive case we could simply assume that, in time period i, the key K_i is fully stored on the machine and therefore there is not enough memory on the machine to store much information about any of the prior keys from earlier time periods. However, if $\mathcal{A}_{small}$ is active then this assumption could be completely unreasonable as the attacker could just keep K_1 on the machine for arbitrarily many time periods and leak it entirely. Therefore, we will introduce a special procedure that we call Verify_i, and assume that this procedure is called in each time period i to ensure that the device is storing the full key K_i at that point in time.

During the entire key-evolution process, $\mathcal{A}_{small}$ can communicate with $\mathcal{A}_{big}$ and can perform arbitrary computation (in addition to / instead of computing K_i honestly) *including* the ability to make random oracle calls. We only make three restrictions: (1) the amount of data that $\mathcal{A}_{small}$ can send to $\mathcal{A}_{big}$ *in each period* is bounded (2) the space-complexity of $\mathcal{A}_{small}$ is bounded and not much larger then the space-complexity of the honest computation of f, (3) the number of oracle-queries made by all parties is polynomial (no other computational assumptions are made). (4) At the end of round i, the machine $\mathcal{A}_{small}$ stores the correct key K_i. We will allow unlimited communication in the other direction $\mathcal{A}_{big}$ to $\mathcal{A}_{small}$.

Let us elaborate on the restrictions in more detail. Restriction (1) models the fact that natural leakage is too simple to reveal too much data about any single computational step. Restriction (2) models that fact that the *complexity* of natural leakage functions is rather simple. In particular, the space-complexity of leaking on the internals of a computation should not be much larger then the amount of space actually used by the computation itself! This seems to be a rather conservative assumption. Lastly, restriction (3) models that all parties run in polynomial time, as is standard in cryptography, and restriction (4) models the fact that the device itself correctly computes the key K_i in round i.

Now that we have explained the model, let us give some intuition why key-evolution schemes are achievable in it. Firstly, note that, in round i, the adversary can (in principle) pre-compute any future key K_{i+t} in the space it is allotted. However, we will ensure that such computation would necessarily require the adversary to erase some data about K_i (since it cannot store all of K_i *and* compute K_{i+1} simultaneously with limited space) and hence it will be unable to satisfy the requirement that K_i is stored on the system at the end of round i.

1.3 Our Results

We construct a key-evolution function f that is secure in the model described above. Let c be the amount of bits that the adversary can retrieve in each round, and let s be the space that the adversary can use to compute the leakage function (including the $|K|$ bits needed to store the key K). We show that our scheme is secure as long as

$$4c + s \leq 3 \cdot |K|/2 \tag{1}$$

(cf. Theorem 1). Let us mention two applications of our construction.

Security against passive leakages Firstly, suppose that the evolving key K_i is stored on some device (say, a smart-card) that may leak some information. Imagine that the device is used for (message or entity) authentication with a trusted server that has his own copy of K_i. Suppose the adversary can get a temporary access to the device, observe the process of key evolution and learn some partial information about the keys. At some later point the adversary looses access to the device. The properties of our function f will guarantee that the future keys are unknown to the adversary assuming that the leakage is bounded in the way described above. Since in this case the adversary is only passive, there is no need to perform the procedure Verify_i. It may look like the model described above is stronger than what we need for this application, since it seems too pessimistic to assume that the adversary fully controls how the keys K_i are computed on the device. It may seem tempting to consider a weaker model, where the computation is done is some honest way, and the adversary can apply the leakage functions during the evolution process. We believe that such a restriction would not make the proof simpler (while it would make the model more complicated). Moreover, going to the extreme and allowing the adversary to control the computation has the advantage that it protects us (to a certain extent) against implementation errors.

Security against active attacks in the BRM The second application of our construction concerns the bounded-retrieval model (BRM) [8,15]. In this model one constructs schemes where the cryptographic key K is very large. The idea is that K can be stored on a PC that can be infected by viruses, and, as long as the virus does not retrieve a large portion of K, the scheme should remain secure. So far, all the work in the BRM considered only the passive attacks, where the virus was not allowed to modify the data on the machine. Now, consider the following problem: suppose we are using the BRM scheme for the session-key agreement [15,6] (where a pair of users share a secret key K), and we want to evolve the secret key K stored on the machine, so that in total over a long period of time we can tolerate a leakage of more than $|K|$ bits from the machine. We show that f can be used as such a key-evolution function. The details of the model are as follows. Suppose we store the keys K_i on the machine $\mathcal{M}$, and assume that the size of the local memory on $\mathcal{M}$ is s, and the amount of bits that can be retrieved in each key-evolution round is c, and c and s are such that (1) holds. Suppose $\mathcal{M}$ wants to authenticate to a trusted server that has his own copy of K_i. If there is a virus on $\mathcal{M}$ then we can even allow him to modify the data stored on $\mathcal{M}$. The restrictions that we impose are as follows. First, we assume that any computation that the virus performs has to be done within $\mathcal{M}$'s memory (of size s). Second, we somehow need to guarantee that the verification procedure Verify_i can be performed. We do it by assuming that $\mathcal{M}$ is equipped with a small tamper-free component $\mathcal{D}$ that can periodically check if the contents of the memory is "correct". For example, $\mathcal{D}$ could store the values of some hash function of $K_1, K_2, \ldots,$, and the Verify_i procedure would just consist of hashing the contents of the memory where K_i is supposed to be stored, and comparing the result with $H(K_i)$.

1.4 Some implementation details

In this paper we do not define formally the security of the concrete schemes (like the message authentication), since we are more interested in considering the key-evolution as an abstract procedure. Every key K_i will consist of $N-1$ blocks:
$K_i = (K_i^0, \ldots, K_i^{N-1})$, each of the blocks being an output of a hash function modeled as a random oracle. Hence, we can simply say that K_i is secret if none of its blocks has every been calculated by the random oracle (in our model calculating such a block will correspond to "labeling" a vertex in some graph).

Of course, the key evolution scheme would be useless, if we could not use the evolved key in some other application. In other words, after each round i, the key evolution function should output some key κ_i, and in the formal model this κ_i should be given to $\mathcal{A}_{big}$ "for free". In our case we simply assume that κ_i is equal to one of the blocks of K_i. Note, that it does not require any modification of the model, since we can as well assume that κ_i is sent to $\mathcal{A}_{big}$ by $\mathcal{A}_{small}$.

The Verify_i procedure (that verifies the knowledge of K_i) can be implemented as follows: (1) the verifier (that knows K_i) sends a random value c to the device, and (2) the device replies with $v = \mathsf{MAC}(c, K_i)$ (where MAC is a tagging function of some message authentication code scheme, c is treated as a key for the MAC, and K_i is treated as a message), (3) the verifier checks if $v = \mathsf{MAC}(c, K_i)$, and if not then he aborts. Note, that we cannot hope for more than only verifying the correctness, since in the worst case an active adversary can anyway completely destroy the contents of the device. In our model we will not assume anything about how $\mathsf{MAC}(c, K_i)$ is computed by $\mathcal{A}_{small}$. For example, it will be possible that he partially "pre-computes" it before learning c. The only property of MAC that we use is that $\mathcal{A}_{small}$ can compute the value of $\mathsf{MAC}(c, K_i)$ only if each of the blocks of K_i were computed by him at some point earlier.

1.5 Organization

Our key-evolution function is defined using a special type of a graph, that we call a *tower graph*. The method of translating graphs into functions is described in Section 2. The tower-graphs and the function f are defined in Section 3. The main theorem is stated in Section 4, which also introduces most of the tools needed for the proof. The proof itself appears in Section 5.

1.6 Related Work

The theoretical countermeasures against the side-channel attacks were considered in [19,1,35,27,32,10,39,11,12,20,4,3]. The schemes in the Bounded Retrieval Model were constructed in [8,15,14,6,18,25,2]. We will use the technique called "graph pebbling" (cf. e.g. [37]), that was already used in cryptography in [13]. Some of our techniques (esp. those used in Sections 4.1 and 4.5) were introduced in [13] and recently extended in [17]. We note that, although our techniques are quite similar, the application is completely different (the main application of [17] is a construction of a scheme for the password-protected local storage). Unfortunately, it is impossible to use the theorems from [13,17] in a black-box way, and therefore we needed to non-trivially extend them. On a technical level, the main difference comes from the fact that in [17] the total amount of leakage was bounded globally, and in our paper we need to consider continual leakage over an unbounded number of round.

2 Random-Oracle Labeling of a Graph.

Let $G = (V, E)$ be a directed acyclic graph (DAG). A vertex v is a *child* of a vertex v' if there is an edge from v to v'. Let V_0 be the set of its *input vertices*, i.e. the vertices without children. A *labeling* of G is a function $\texttt{label}(\cdot)$, which assigns values $\texttt{label}(v) \in \{0,1\}^w$ to vertices $v \in V$. We call w the *label-length*. For any function $\mathcal{H} : \{0,1\}^* \to \{0,1\}^w$ and input-labels $K = (K_1, \ldots, K_N)$ with $K_i \in \{0,1\}^w$, we define the $(\mathcal{H}, K)$-labeling of G as follows:

- The labels of the N distinct input vertices $v_1, v_2, \ldots, v_N$ are given by $\texttt{label}(v_i) \overset{\text{def}}{=} K_i$.
- The label of every other vertex v is defined recursively by

$$\texttt{label}(v) \overset{\text{def}}{=} \mathcal{H}(\texttt{label}(v_1), \cdots, \texttt{label}(v_j), v)$$

  where $v_1, \cdots, v_j$ are the children of v.

A *random oracle labeling* of G is an $(\mathcal{H}, K)$-labeling of G where $\mathcal{H}$ is a random-function and K is chosen uniformly at random. For convenience, we also define $\texttt{preLabel}(v) \overset{\text{def}}{=} (\texttt{label}(v_1), \ldots, \texttt{label}(v_j), v)$, where $v_1, \ldots, v_j$ are the children of v, so that $\texttt{label}(v) = \mathcal{H}(\texttt{preLabel}(v))$.

3 Our Key-Evolution Scheme

In this section we define our key-evolution function f. We start with defining a special type of graphs, that we call the "tower graphs". A graph $G = (V, E)$ is called an (N, M)-*tower graph* if $V = \{0, \dots, M-1\} \times \{0, \dots, N-1\}$ and $E = \{((i, j), (i+1, j) : i \in \{0, 1, \dots, \}, j \in \{0, \dots, N-1\}\} \cup \{((i, j), (i+1, (j-1) \bmod N) : i \in \{0, 1, \dots\}, j \in \{0, \dots, N-1\}\}$ (cf. Figure in the appendix of extended version [16]). For $i = 0, \dots, t$ the set $V_i = \{(i, 0), \dots, (i, N-1)\}$ is called the *ith line of* G. Let $V_{\geq i}$ denote the set $V_i \cup V_{i+1} \cup \cdots$. Note that the set of the input vertices of G is equal to V_0. We will say that an (infinite) graph G is an N-*tower graph* if it is an (N, ∞)-tower graph.

We are now ready to define f. If we fix a hash function $\mathcal{H}$ and label length w then the (N, M)-tower graph G defines a function $f : \{0,1\}^{Nw} \to \{0,1\}^{Nw}$ in the following way. On an input K the function f computes the $(\mathcal{H}, K)$-labeling of G and it outputs $(K'_1, \dots, K'_N)$, where each K_i is the label of $(M-1, i)$. The procedure for computing $f(K_0, \dots, K_{N-1})$ simply computes the labels bottom-up row-by-row in the following way:

- Set $(K_0^0, \dots, K_{N-1}^0) := (K_0, \dots, K_{N-1})$.
- For $j = 1, \dots, M-1$ do
    - For $i = 0, \dots, N-1$ do $K_i^j := \mathcal{H}(K_i^{j-1}, K_{i+1 \bmod N}^{j-1}, (i, j))$

Observe that the time needed to compute f is roughly equal to $N \cdot M$ times the time needed to compute $\mathcal{H}$, and the space needed to compute f is only slightly larger than the space needed to store K, since we can overwrite each $(K_0^j, \dots, K_{N-1}^j)$ with $(K_0^{j+1}, \dots, K_{N-1}^{j+1})$ and hence re-use the space.

It is also easy to see that iterating the computation of f on the same input a times, i.e. computing $K' = f^a(K)$ can be seen as computing the labeling of an (N, aM)-tower graph, and in particular, if we want to evolve the key K^0 using the procedure $K^{i+1} = f(K^i)$ (for $i = 0, 1, \dots$) then we can look at it as a labeling the tower infinite N-tower graph, where the keys $K^1, K^2, \dots$ appear as labels of the lines $V_{1 \cdot M}, V_{2 \cdot M}, \dots$. We will call such lines the *round-switching lines*.

4 Games on Tower Graphs

We will show a connection between an adversary computing a "random oracle graph" and a pebbling game for the corresponding graph. A similar connection appears in [13] (and in [17], see Section 1.6 for more on relation between this work and [17]).

4.1 Model of Computation

Our main goal is to show that computing the labeling of a tower graph G requires a large amount of resources in the random-oracle model, and is therefore difficult. To do so, we must fix a model of computation in which we can make statements of the above form precise. Recall that we will usually consider an adversary that consists of two parts: a "space-bounded" component which gets access to the internals of an attacked device and has "bounded communication" to an external, and otherwise unrestricted, adversary.

We model such a adversary $\mathcal{A} = (\mathcal{A}_{big}, \mathcal{A}_{small})$ as a pair of interactive algorithms[4] with oracle-access to a random-oracle $\mathcal{H}(\cdot)$. Let M be some natural number that we will call the *round length*. While executing the algorithms the time is divided into rounds. Initially the computation is in a round 1. The adversary $\mathcal{A}_{small}$ is responsible for switching to next round. Namely: the round is changed to k when $\mathcal{A}_{small}$ calls special function $\mathtt{nextRound}_k(\mathtt{label}(a_1) \ldots \mathtt{label}(a_n))$, where $\{a_1, \ldots, a_n\}$ is the kth *round-switching line* $V_{k \cdot M}$. A round k can be switched only to round $k+1$, in other words the order of the round-changing calls has to be $\mathtt{nextRound}_1, \mathtt{nextRound}_2, \ldots$. The period between the calls $\mathtt{nextRound}_i$ and $\mathtt{nextRound}_{i+1}$ will be called *the ith round*. The algorithm $\mathcal{A}_{big}$ will only be restricted in the number of oracle calls made. On the other hand, we impose the following additional restrictions on $\mathcal{A}_{small}$:

- s-bounded space: The total amount of space used by $\mathcal{A}_{small}$ is bounded by s. That is, we can accurately describe the entire configuration of $\mathcal{A}_{small}$ at any point in time using s bits.[5]
- c-bounded communication: The total number of outgoing bits communicated by $\mathcal{A}_{small}$ in each round is bounded by c. [6]

Note that these restrictions imply that he total number of outgoing bits communicated by $\mathcal{A}_{small}$ in every round is bounded by c and there is no global bound for communication. We use the notation $\mathcal{A}^{\mathcal{H}(\cdot)}(K) = \left(\mathcal{A}_{big}^{\mathcal{H}(\cdot)}() \leftrightarrows \mathcal{A}_{small}^{\mathcal{H}(\cdot)}(K)\right)$ to denote the interactive execution of $\mathcal{A}_{big}$ and $\mathcal{A}_{small}$, where $\mathcal{A}_{small}$ gets input K and both machines have access to the oracle $\mathcal{H}(\cdot)$. In particular, we will usually (only) care about the list of random-oracle calls made by $\mathcal{A}_{big}$ and $\mathcal{A}_{small}$ during such an execution. We say that an execution $\mathcal{A}^{H(\cdot)}(K)$ *labels* a vertex v, if a random-oracle call to $\mathtt{preLabel}(v)$ is made by either $\mathcal{A}_{big}$ or $\mathcal{A}_{small}$. We are now ready to state our main theorem.

Theorem 1. *Let G be a N-tower graph and $\lambda > 0$. Suppose c, s and q are such that $\frac{4c+s+\lambda}{w-\log(q)} \leq N + N/2$, and let T be an arbitrary natural number. Let $\mathcal{A} = (\mathcal{A}_{big}, \mathcal{A}_{small})$ be an adversary with c-bounded communication and s-bounded storage that makes at most q queries to H. The probability p (taken over the choice of $(\mathcal{H}, K)$) that there exists $i = 1, \ldots, T-1$ such that $\mathcal{A}$ labels the line $V_{(i+1) \cdot M}$ of G in round i is at most*

$$q \cdot 2^{-w} + T \cdot 2^{1-\lambda} \tag{2}$$

The proof appears in Section 5. The necessary machinery is introduced in the next sections.

[4] Say ITMs, interactive RAMs, ... The exact model will not matter.

[5] This is somewhat different than standard space-complexity considered in complexity theory, even when we restrict the discussion to ITMs. Firstly, the configuration of $\mathcal{A}_{small}$ includes the value of *all* tapes, including the input tape. Secondly, it includes the current state that the machine is in and the position of all the tape heads.

[6] To be precise, we assume that we can completely describe the patters of outgoing communication of $\mathcal{A}_{small}$ using c bits. That is, $\mathcal{A}_{small}$ cannot convey additional information in when it sends these bits, how many bits are sent at a given time and so on...

4.2 Pebbling Games on Tower Graphs

We will consider a variant of the pebble game that we call the "red-black" pebble game over an N-tower graph $G = (V, E)$. Each vertex of the graph G can either be empty, contain a red pebble, contain a black pebble, or contain both types of pebbles. More precisely, if G is a tower graph, then a *pebbling configuration on* G is a function $\gamma : V \to \mathcal{P}(\{\texttt{red}, \texttt{black}\})$. Define $\mathit{Red}(\gamma) := \{v : \texttt{red} \in \gamma(v)\}$, and $\mathit{Black}(\gamma) := \{v : \texttt{black} \in \gamma(v)\}$. If $V' \subseteq V$ then define $\mathit{proj}(V') := (|V' \cap V_1|, \ldots, |V' \cap V_t|)$.

For a set $V' \subseteq V$ denote by $[V']$ the *closure of* V defined recursively as follows:

- if $v \in V'$ then $v \in [V']$,
- if all the children of v' are in $[V']$ then $v' \in [V']$.

An initial configuration γ_1 consists of (only) a black pebble placed on each input vertex of G. The game proceeds in steps where, in the ith step, the configuration γ_i is transformed into γ_{i+1} using one of the following four actions:

1. A red pebble can be placed on any vertex already containing a black pebble.
2. If both children of a vertex v have a red pebble on them, a red pebble can be placed on v.
3. If both children of v have *some* pebble on them (red or black), a black pebble can be placed on v.
4. A black pebble can be removed from any vertex.

A *pebbling game* is a sequence $\gamma_1 \to \gamma_2 \to \cdots \to_\ell$ of configurations. The game is — similarly to real computational model — divided into rounds. One starts a game in round 1. A round may be switched to u in a configuration γ_i if all vertices from a line $V_{u \cdot M}$ in γ_i are pebbled by some pebble (technically, a round is switched to u by issuing a request $\texttt{nextRound}_u$). This switch is not obligatory when the specific row is pebbled. However, we require that the order of the request is $\texttt{nextRound}_0, \texttt{nextRound}_1, \ldots,$.

We define the *black-pebble complexity* of a round k of a pebbling game to be the maximum number of *black pebbles* on vertices in $V_{\geq k \cdot M}$ in use at any time of round k. More precisely if $\gamma_i \to \cdots \to \gamma_j$ are the configurations in round k. Then the black pebble complexity of k is equal to $\max_{\ell=i}^{j} |\mathit{Black}(\gamma_\ell) \cap V_{\geq k \cdot M}|$. If $\gamma_i, \ldots, \gamma_j$ are as above then the red pebble complexity of k is equal to the number of times in round k in which Step 1 was applied. For a parameter X a pebbling game is *X-bounded* if for every round k we have $2R_k + B_k < X$, where R_k and B_k denote the red- and the black-pebble complexities (resp.) of round k.

4.3 Auxiliary lemmata

We need some auxiliary definitions and lemmas. For $(a_0, \ldots, a_t) \in \{0, \ldots, N\}^{t+1}$ define the *optimistic width of* $(a_0, \ldots, a_t)$ as: $\mathit{OptWidth}(a_0, \ldots, a_t) := (b_0, \ldots, b_t)$, where

- $b_0 := a_0$, and

- for every $i = 1, \dots, t$ we set

$$b_i := \begin{cases} N & \text{if } b_{i-1} = N \\ \min(N, b_{i-1} - 1 + a_i) & \text{otherwise} \end{cases}$$

Intuitively, the idea is that if $(b_1, \dots, b_t) := \mathit{OptWidth}(a_0, \dots, a_t)$ then b_i's give an upper bound on the number of pebbles in the ith line of $[V']$ (for any $V' \subseteq V$), assuming that a_i is the number of pebbles in the ith line of V'. Formally, this is shown in the following lemma.

Lemma 1. *Take any set $V' \subseteq V$ and let $(a_0, \dots, a_t) := \mathit{proj}([V'])$ and $(b_0, \dots, b_t) := \mathit{OptWidth}(\mathit{proj}(V'))$. For every i we have that $a_i \leq b_i$.*

Proof. The proof goes by induction on $i = 0, \dots, t$. Case $i = 0$ follows immediately from the fact that the closure operation does not change the configuration of the pebbles on the bottom row (V_0), and hence $a_0 = b_0$.

Now suppose the lemma holds for some i. The set of pebbles in the $(i+1)$st line of $[V']$ is equal to the sum of V'_{i+1} and the pebbles P that were derived (using the closure operation) from the pebbles in the ith line of $[V']$. By the induction hypothesis we get that the number of pebbles in the ith line of $[V']$ is at most b_{i-1}. Now, consider the case when $b_{i-1} \neq N$. From the definition of the closure operation it follows that $|P| \leq b_{i-1} - 1$. Therefore $|V'_{i+1} \cup P| \leq |V'_{i+1}| + |P| \leq a_i + b_{i-1} - 1$. Since the maximal value of $a_i + b_{i-1} - 1$ cannot be greater than N we get $|V'_{i+1} \cup P| \leq \min(N, a_i + b_{i-1} - 1)$.

The second case ($b_{i-1} = N$) follows easily from the fact that in this case $b_i = N$.

A sequence $(a_0, \dots, a_t)$ is called *wide* if for some i we have $a_i = N$. A set $V' \subseteq V$ will be called *wide* if $\mathit{proj}([V'])$ is wide. We have the following simple observation.

Lemma 2. *For $U, W \subseteq V$ such that $U \cup W$ is not wide define $(a_0, \dots, a_t) := \mathit{OptWidth}(\mathit{proj}([U \cup W]))$ and $(b_0, \dots, b_t) := \mathit{OptWidth}(\mathit{proj}([W]))$. Then, for every i we have $b_i \leq a_i - |W|$. In other words: adding $|W|$ elements to U cannot increase the values on the coordinates in $\mathit{OptWidth}(\mathit{proj}([U]))$ by more than $|W|$ (as long as the resulting set $U \cup W$ is not wide).*

Proof (sketch). Suppose we add the elements of W to U one-by-one. From the definition of the closure operation it easily follows that adding one element cannot increase $\mathit{OptWidth}(\mathit{proj}([W]))$ by more than on each coordinate (as long as the resulting set is not wide). Hence the statement of the lemma follows.

A subgraph G' of a tower graph is a *pyramid graph* (cf. Fig. in Appendix in extended version [16]) if it is induced by the set of vertices: $\{(i + x \bmod N - 1, j + y \bmod N) : 0 \leq x + y \leq N - 1\}$ for some i and j. The vertex $(i + N, j)$ will be call the *root of* G'. We now have the following lemma whose proof appears in Appendix in extended version [16].

Lemma 3. *Consider a pebbling game for initially empty pyramid graph. If the root vertex is pebbled at the end of the game then there exist a configuration γ of the considered game with sum of red pebbles in the first row and the black pebbles is at least N.*

4.4 The impossibility of pebbling

Our goal is to show that — with some restrictions on red and black pebble complexity — it is impossible to pebble any vertex in $V_{\geq (u+2)\cdot M}$ in round u. Intuitively, it means that we cannot get any information about pebbles from any line $V_{\geq (u+2)\cdot M}$, before switching to round $u+1$. More precisely, the following theorem holds:

Theorem 2. *Let N, T and X be arbitrary natural numbers such that*

$$X < \frac{3N}{2}.$$

Set $M := \frac{3N}{2}$. Suppose G is an N-tower graph. Then, for any X-bounded pebbling game for G with round length M and any configuration γ that belongs to the uth round, we have that in γ there are no pebbles on $V_{\geq (u+2)\cdot M}$.

Proof. In an execution of a pebbling game a pebble will be called *heavy* if it is a black pebble, or a red pebble placed on the graph using rule 1 (cf. Page 10). For a round u let γ_{i_u} be the last configuration of this round. We claim that in γ_{i_u} there is no pebble on $V_{\geq (u+2)\cdot M}$. Let A_u be the set of all pebbled vertices in the configuration γ_{i_u} and let Q_u be the set of all pebbles in A_u except of the black pebbles lying on the line $u \cdot M$. More precisely: $Q_u = A_u \setminus (A_u \cap V_{u\cdot M} \cap \mathit{Black}(\gamma))$. Set $Y_u := V_{\geq M\cdot u} \setminus V_{\geq M\cdot (u+1)}$

Lemma 4. *For every u we have:*

1. $A_u \cap V_{\geq (u+2)\cdot M} = \emptyset$
2. $\mathit{OptWidth}(\mathit{proj}([Q_u])) = (a_0, \ldots, a_{T\cdot M}) < (\underbrace{N, \ldots, N}_{u\cdot M}, \underbrace{N/2, \ldots, N/2}_{M}, 1, \ldots, 1)$,

   in particular $[Q_u]$ is not wide.

After showing this we will be done with the proof since Point 1 of Lemma 4 clearly implies that Theorem 2 holds.

Proof (Proof of Lemma 4). Induction on $u = 1, 2, \ldots$. The base of the induction holds trivially since in the initial configuration only the bottom line is pebbled. Let us now assume the statement holds for some u. Now we prove the following claims for next round $u+1$:

Claim. During the entire round $u+1$ there must be at least $N/2$ heavy pebbles in the subgraph Y_u (i.e. the lines $u \cdot M, \ldots, (u+1)\cdot M - 1$).

Proof. Let us consider any configuration γ from this round and denote the set of heavy pebbles in X in γ by P. At the end of the round every vertex on the $(u+1)$st round-switching line $V_{(u+1)\cdot M}$ will contain a pebble. Therefore the closure $[P]$ of heavy pebbles P from the current configuration and the pebbles from the previous rounds Q_u need to contain whole line $V_{(u+1)\cdot M}$. This is because otherwise one would never be able to pebble $V_{(u+1)\cdot M}$ in the future (this follows easily from the definition of closure and the pebbling game). Hence $[P \cup Q_u]$ has to be wide and therefore (from Lemma 1) $\mathit{OptWidth}(P \cup Q_u$ also has to be wide. On the other hand, by the induction hypothesis we know that every coordinate of $\mathit{OptWidth}(Q_u)$ on positions $u \cdot M, \ldots$ is smaller than $N/2$. Now, by Lemma 2 adding to Q_u a set of cardinality $|P|$ cannot increase any of this coordinates by more than $|P|$. Hence $|P| \geq N/2$.

Claim. Through the whole round $u+1$ no vertex in $V_{\geq (u+2)\cdot M}$ is pebbled.

Proof. For the sake of contradiction assume that the claim is not true. So, we have a configuration γ from round $u+1$ with a pebble on some vertex v from set $V_{\geq (u+2)\cdot M}$. Denote by $(V', E')'$ the subgraph forming the pyramid graph with root in vertex v. From Lemma 3 we have that before γ there was a configuration γ' that: had b black pebbles on V' and had r red pebbles in bottom line of V' (which is the line $V_{\geq (u+2)\cdot M - N+1}$ of the tower graph) and $b+s \geq N$. However from Claim 4.4 there are at least $N/2$ heavy pebbles on Y_u, and Y_u is disjoint with the pyramid (V', E'). The number of all heavy pebbles is $A := B + R < (N-R) + (N/2)$. Therefore in the set $V_{\geq M\cdot(u+1)} \supset V'$ there are at most $(N-R)$ heavy pebbles. Since $b+s$ is bounded by the number of heavy pebbles so we have a contradiction with the fact that $b+s \geq N$.

Claim. $\mathit{OptWidth}(\mathit{proj}([Q_{u+1}])) = (a_0, \ldots, a_{T\cdot M}) < (\underbrace{N, \ldots, N}_{u+1\cdot M}, N/2, \ldots, N/2)$

Proof. Denote the configuration at the end of this round by γ and the set of heavy pebbles in γ by P. Let P' denote P without black-pebbled vertices from $(u+1)$th finishing line. From definition, we have $[Q_{u+1}] = [Q_u \cup P']$. In γ the $(u+1)$th finishing line is pebbled. There at most R red pebbles, so at least $N-R$ black pebbles are on this line. So $|P'|$ is at most $A - (N-R) = B + 2R - N < N/2$. Similarly as at the end of proof of the Claim 4.4 adding to Q_u a set of cardinality $|P'| < N/2$ cannot increase any coordinate of $\mathit{OptWidth}$ by more then $|P'|$. This finishes the proof.

Claims 4.4 and 4.4 prove inductive hypothesis for $u+1$. Hence we are done.

4.5 Connection Between RO Labeling and the Pebbling Game

We now connect the random-oracle labeling of a tower graph G in our model of computation to the red-black pebbling game on G (a similar connection appeared recently in [17] and it is an extention of the technique from [13]). The idea is to show that from any execution of $\mathcal{A}$ (with space bounded by some s, and communication bounded by some c) we can construct some pebbling game for pebble game described before. Then, we show that (with high probability) this game respects the rules of the game and is (B, R)-bounded (for some B, R that will depend on c and s). We will then combine it with Theorem 2 to conclude that some specific oracle calls are impossible.

The main fact about the connection is that — in every round — the black-pebble complexity of the pebbling will correspond to the space-complexity of $\mathcal{A}_{small}$ and the red-pebble complexity corresponds to the communication-complexity of $\mathcal{A}_{small}$.

First, let us strictly define the method of translating an execution of $\mathcal{A}$ into a pebbling game. Let $\mathcal{H} : \{0,1\}^* \to \{0,1\}^w$ be a random oracle, and let $K = (K_1, \ldots, K_N)$ be a labeling of the input-vertices of G. For any algorithms $\mathcal{A} = (\mathcal{A}_{big}, \mathcal{A}_{small})$ we can use the execution $\mathcal{A}^{\mathcal{H}(\cdot)}(K) = \left(\mathcal{A}_{big}^{\mathcal{H}(\cdot)} \leftrightarrows \mathcal{A}_{small}^{\mathcal{H}(\cdot)}(K)\right)$ to construct a red-black pebbling of the graph G. In particular, we get a transcript listing all oracle calls made during its entire execution, and whether they were made by $\mathcal{A}_{small}$ or $\mathcal{A}_{big}$ and all `nextRound` calls made by $\mathcal{A}_{small}$.

We fix some terminology about the transcript. Given $(\mathcal{H}, K)$, we say that an oracle call of the form $\mathcal{H}(\mathtt{label}_1, \mathtt{label}_2, v)$ is *correct* if $(\mathtt{label}_1, \mathtt{label}_2, v) = \mathtt{preLabel}(v)$. We call the children v_1, v_2 of v the input-vertices of the oracle call, and v is the output-vertex of the oracle call.

Using the transcript (along with the description of $\mathcal{H}, K$) we define the *ex-post-facto* pebbling of the graph G. We do so by processing the random-oracle calls and `nextRound` calls in the transcript one-by-one starting with the earliest one, and, for each call, we take the following steps:

Change round: If $\mathcal{A}_{small}$ calls `nextRound`, change round in the pebble game.

Place all necessary red pebbles: A vertex v is *red-necessary* if, looking at the *entire transcript* of all oracle calls, there exists some correct oracle call made by $\mathcal{A}_{big}$ with v as an input-vertex, which *precedes* all correct oracle calls made by $\mathcal{A}_{big}$ with v as an output-vertex. If the call is taken in kth round and $v \in V_{\geq k \cdot M}$ then we say that v is *k-red-necessary*.

Go through all red-necessary vertices v one-by-one and, for each one check if that has a black pebble,but no red pebble. If so, put red pebble on v.[7]

Delete all unnecessary black pebbles: A vertex v is *black-necessary* if it is *not* red-necessary and, *in the remainder of the transcript* of oracle-calls that have not yet been processed (including the current call), there exists some correct oracle call made by $\mathcal{A}_{small}$ with v as an input-vertex such that:

- In the *remainder of the transcript*, there is no *earlier* correct oracle call made by $\mathcal{A}_{small}$ with v as an output-vertex.
- In the *entire transcript*, there is no *earlier* correct oracle call made by $\mathcal{A}_{big}$ with v as an output-vertex.

Go through all vertices v which are *not* black-necessary but have a black pebble on them, one-by-one, and remove the black pebble.[8]

Process oracle call: If the current oracle call is correct and made by $\mathcal{A}_{small}$ (respectively $\mathcal{A}_{big}$) with output vertex v, we put a black (respectively red) pebble on v.

We notice that every vertex that is labeled by the execution of $\mathcal{A}^{\mathcal{H}(\cdot)}(K)$ gets a (red or black) pebble placed on it in the corresponding ex-post-facto pebbling (although, of course, this pebble may have been removed at some later point). Moreover, the order in which vertices get red/black pebbles corresponds to the order in which the oracle calls are made by $\mathcal{A}$.

As mentioned before, we now show that, for any adversary $\mathcal{A} = (\mathcal{A}_{small}, \mathcal{A}_{big})$ which is space/communication bounded, and which makes a bounded number of oracle

[7] Note that the set of red-necessary vertices does not change throughout the process. Intuitively, these are the vertices whose labels must be communicated by $\mathcal{A}_{small}$ to $\mathcal{A}_{big}$ at some point in time, and correspondingly for which we need to take pebbling-action 1 to place a red pebble on them. We choose to take this action as early as legally possible, since it might allow us to remove related black pebbles early.

[8] Note that the set of black-necessary vertices can be different at different points in the process. Intuitively, at any point in time, a black-necessary vertex is one whose label must be stored in the memory of $\mathcal{A}_{small}$ since it will not be re-computed by $\mathcal{A}_{small}$ via oracle calls, it was never communicated to $\mathcal{A}_{big}$, nor will it be computed by $\mathcal{A}_{big}$ in time.

calls, the ex-post-facto pebbling is legal and has small space/communication complexity.

Theorem 3. *Let G be an N-tower graph. Let $\mathcal{A} = (\mathcal{A}_{big}, \mathcal{A}_{small})$ be any adversarial labeling game in our restricted model of computation. Let $(\mathcal{H}, K)$ define a random-oracle labeling of the graph G, with label-length w. Assume that $\mathcal{A}$ makes at most q random-oracle queries during the execution. Then, the ex-post-facto pebbling of G corresponding to an execution of $\mathcal{A}^{\mathcal{H}(\cdot)}(K)$ has the following properties (for any k):*

1. *It is a legal pebbling (i.e. follows the rules of the red-black pebbling game and changes round only when appropriate condition is hold) with probability $1 - \frac{q}{2^w}$ over the choice of $(\mathcal{H}, K)$.*
2. *Assuming that $\mathcal{A}_{small}$ has c-bounded communication and that in rounds $0, \dots, k-2$ no vertices from $V_{\geq k \cdot M}$ were pebbled in the* ex-post-facto *game then, for any $\lambda \geq 0$ the red-pebble complexity of the round k is at most $\frac{2c+\lambda}{w-\log(q)}$ with probability $1 - 2^{-\lambda}$ over the choice of $(\mathcal{H}, K)$.*
3. *Assuming that $\mathcal{A}_{small}$ has s-bounded storage and c-bounded communication and that in rounds $0, \dots, k-2$ no vertices from $V_{\geq k \cdot M}$ were pebbled then, for any $\lambda > 0$, the* sum *of the red-pebble complexity* and *the black-pebble complexity of the round k is at most $\frac{2c+s+\lambda}{w-\log(q)}$ with probability $1 - 2^{-\lambda}$ over the choice of $(\mathcal{H}, K)$.*

The proof appears in Appendix in extended version [16].

5 Proof of Theorem 1

Consider an execution of $\mathcal{A}$ and a corresponding ex-post-facto pebbling game $\mathcal{G}$. Let $\mathcal{X}$ denote an event that $\mathcal{G}$ is legal. For $i = 1, \dots, T$ let B_i and R_i denote the respective black- and red-pebble complexities of round i in $\mathcal{G}$. Moreover, let $\mathcal{Y}_i$ denote the event that in the round i we have that (1) $B_i + 2R_i < N + N/2$, and (2) no vertex in $V_{(i+2)\cdot M}$ has been pebbled. Recall that every vertex that is labeled gets also pebbled. Therefore we have

$$\begin{aligned} 1 - p &\geq P(\mathcal{Y}_1 \wedge \dots \wedge \mathcal{Y}_{T-2}) \\ &\geq P(\mathcal{X} \wedge \mathcal{Y}_1 \wedge \dots \wedge \mathcal{Y}_{T-2}) \\ &= P(\mathcal{X}) \cdot P(\mathcal{Y}_1 | \mathcal{X}) \cdot P(\mathcal{Y}_2 | \mathcal{Y}_1 \wedge \mathcal{X}) \cdots P(\mathcal{Y}_{T-2} | \mathcal{Y}_1 \wedge \dots \wedge \mathcal{Y}_{T-3} \wedge \mathcal{X}) \end{aligned} \quad (3)$$

Let us look at a term $P(\mathcal{Y}_i | \mathcal{Y}_1 \wedge \dots \wedge \mathcal{Y}_{i-1} \wedge \mathcal{X})$. Suppose $\mathcal{Y}_1 \wedge \dots \wedge \mathcal{Y}_{i-1} \wedge \mathcal{X}$ occurred. The events $\mathcal{Y}_1, \dots \mathcal{Y}_{i-2}$ together imply that until round $i-2$ no pebble has been placed on any vertex in $V_{\geq i \cdot M}$. Hence:

- $R_i \leq \frac{2c+\lambda}{w-\log(q)}$ with probability at least $1 - 2^{-\lambda}$ (from Part 2 of Theorem 3), and
- $B_i + R_i \leq \frac{2c+s+\lambda}{w-\log(q)}$ with probability at least $1 - 2^{-\lambda}$ (from Part 3 of Theorem 3).

Therefore we get that $B_i + 2R_i \leq \frac{4c+s+\lambda}{w \log(q)} \leq N + N/2$ with probability at least $1 - 2^{1-\lambda}$. Since the events $\mathcal{Y}_1, \dots \mathcal{Y}_{i-1}$ also imply that $B_j + 2R_j \leq N + N/2$ holds for every $j < i$, therefore we can apply Theorem 2 and get that with probability $1 - 2^{1-\lambda}$

no pebble is put on any vertex in $V_{\geq(i+1)\cdot M}$ in round i. Since we also know that the pebbling is legal (because we assumed that $\mathcal{X}$ occurred), a vertex can be labeled by $\mathcal{A}$ only if it is pebbled. Hence $\mathcal{Y}_i$ holds (with probability at least $1 - 2^{1-\lambda}$). From Part 1 of Theorem 3 we have that $P(\mathcal{X}) \geq 1 - \frac{q}{2^w}$. Putting things together we get $P(\mathcal{Y}_i|\mathcal{Y}_1 \wedge \cdots \wedge \mathcal{Y}_{i-1} \wedge \mathcal{X}) \geq 1 - 2^{1-\lambda}$. Hence (3) is at least equal to

$$\begin{aligned} &\left(1 - \frac{q}{2^w}\right) \cdot \left(1 - 2^{1-\lambda}\right)^{T-2} \\ \geq &\left(1 - q \cdot 2^{-w}\right) \cdot \left(1 - T \cdot 2^{1-\lambda}\right) \\ \geq &\, 1 - q \cdot 2^{-w} - T \cdot 2^{1-\lambda} \end{aligned}$$

Therefore p is at most (2).

References

1. Adi Akavia, Shafi Goldwasser, and Vinod Vaikuntanathan. Simultaneous hardcore bits and cryptography against memory attacks. In *TCC*, 2009.
2. Joel Alwen, Yevgeniy Dodis, and Daniel Wichs. Leakage-resilient public-key cryptography in the bounded-retrieval model, 2009. `http://eprint.iacr.org/`.
3. Zvika Brakerski and Shafi Goldwasser. Circular and leakage resilient public-key encryption under subgroup indistinguishability (or: Quadratic residuosity strikes back), 2010.
4. Zvika Brakerski, Yael Tauman Kalai, Jonathan Katz, and Vinod Vaikuntanathan. Cryptography resilient to continual memory leakage, 2010.
5. David Brumley and Dan Boneh. Remote timing attacks are practical. *Comput. Netw.*, 2005.
6. David Cash, Yan Zong Ding, Yevgeniy Dodis, Wenke Lee, Richard J. Lipton, and Shabsi Walfish. Intrusion-resilient key exchange in the bounded retrieval model. In *TCC*, 2007.
7. Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract ower-analysis attacks. In *CRYPTO*, 1999.
8. Giovanni Di Crescenzo, Richard J. Lipton, and Shabsi Walfish. Perfectly secure password protocols in the bounded retrieval model. In *TCC*, 2006.
9. Ivan Damgård. A design principle for hash functions. In *CRYPTO*, 1989.
10. Francesco Davì, Stefan Dziembowski, and Daniele Venturi. Leakage-resilient storage, 2009. `http://eprint.iacr.org/`, accepted to SCN'10.
11. Yevgeniy Dodis, Shafi Goldwasser, Yael Tauman Kalai, Chris Peikert, and Vinod Vaikuntanathan. Public-key encryption schemes with auxiliary inputs. In *TCC*, 2010.
12. Yevgeniy Dodis, Kristiyan Haralambiev, Adriana Lopez-Alt, and Daniel Wichs. Cryptography against continuous memory attacks, 2010.
13. Cynthia Dwork, Moni Naor, and Hoeteck Wee. Pebbling and proofs of work. In *CRYPTO*, 2005.
14. Stefan Dziembowski. Intrusion-resilience via the bounded-storage model. In *TCC*, 2006.
15. Stefan Dziembowski. On forward-secure storage. In *CRYPTO*, 2006.
16. Stefan Dziembowski, Tomasz Kazana, and Daniel Wichs. Key-evolution schemes resilient to space-bounded leakage, 2011. `http://eprint.iacr.org/`.
17. Stefan Dziembowski, Tomasz Kazana, and Daniel Wichs. One-time computable self-erasing functions. In *TCC*, 2011.
18. Stefan Dziembowski and Krzysztof Pietrzak. Intrusion-resilient secret sharing. In *FOCS*, 2007.
19. Stefan Dziembowski and Krzysztof Pietrzak. Leakage-resilient cryptography. In *FOCS*, 2008.

20. Sebastian Faust, Eike Kiltz, Krzysztof Pietrzak, and Guy N. Rothblum. Leakage-resilient signatures. In *TCC*, 2010.
21. Sebastian Faust, Tal Rabin, Leonid Reyzin, Eran Tromer, and Vinod Vaikuntanathan. Protecting circuits from leakage: the computationally-bounded and noisy cases. to appear in proc. Eurocrypt 2010.
22. Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. One-time programs. In *CRYPTO*, 2008.
23. Shafi Goldwasser and Guy N. Rothblum. Securing computation against continuous leakage. In *CRYPTO*, 2010.
24. Yuval Ishai, Amit Sahai, and David Wagner. Private Circuits: Securing Hardware against Probing Attacks. In *CRYPTO*, 2003.
25. Yevgeniy Dodis Joel Alwen and Daniel Wichs. Leakage resilient public-key cryptography in the bounded retrieval model. In *Advances in Cryptology - CRYPTO*, 2009.
26. Ali Juma and Yevgeniy Vahlis. Protecting cryptographic keys against continual leakage. In *CRYPTO*, 2010.
27. Jonathan Katz and Vinod Vaikuntanathan. Signature schemes with bounded leakage resilience. In *ASIACRYPT*, 2009.
28. Paul Kocher. Design and validation strategies for obtaining assurance in countermeasures to power analysis and related attacks. *NIST Physical Security Testing Workshop*, 2005.
29. Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *CRYPTO*, 1999.
30. Markus G. Kuhn. Compromising emanations: eavesdropping risks of computer displays, 2003. Technical Report UCAM-CL-TR-577.
31. Silvio Micali and Leonid Reyzin. Physically observable cryptography (extended abstract). In *TCC*, 2004.
32. Moni Naor and Gil Segev. Public-key cryptosystems resilient to key leakage. In *Advances in Cryptology - CRYPTO*, August 2009.
33. European Network of Excellence (ECRYPT). The side channel cryptanalysis lounge. http://www.crypto.ruhr-uni-bochum.de/en_sclounge.html. retrieved on 7.04.2010.
34. Krzysztof Pietrzak. A leakage-resilient mode of operation. In *EUROCRYPT*, 2009.
35. Krzysztof Pietrzak. A leakage-resilient mode of operation. In *EUROCRYPT*, 2009.
36. Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (ema): Measures and counter-measures for smart cards. In *E-smart*, 2001.
37. John E. Savage. *Models of Computation: Exploring the Power of Computing*. Addison Wesley, 1997.
38. Adi Shamir and Eran Tromer. Acoustic cryptanalysis. on nosy people and noisy machines. A webpage: http://people.csail.mit.edu/tromer/acoustic/ accessed on 27.05.2009.
39. François-Xavier Standaert, Tal Malkin, and Moti Yung. A unified framework for the analysis of side-channel key recovery attacks. In *EUROCRYPT*, 2009.
40. Yu Yu, Francois-Xavier Standaert, Olivier Pereira, and Moti Yung. Practical leakage-resilient pseudorandom generators. In *CCS: ACM Conference on Computer and Communications Security.*, 2010. To Appear.