Key-Evolution Schemes Resilient to Space Bounded Leakage

Stefan Dziembowski

Tomasz Kazana

Daniel Wichs

Main contribution

We propose a secure scheme for

deterministic key-evolution

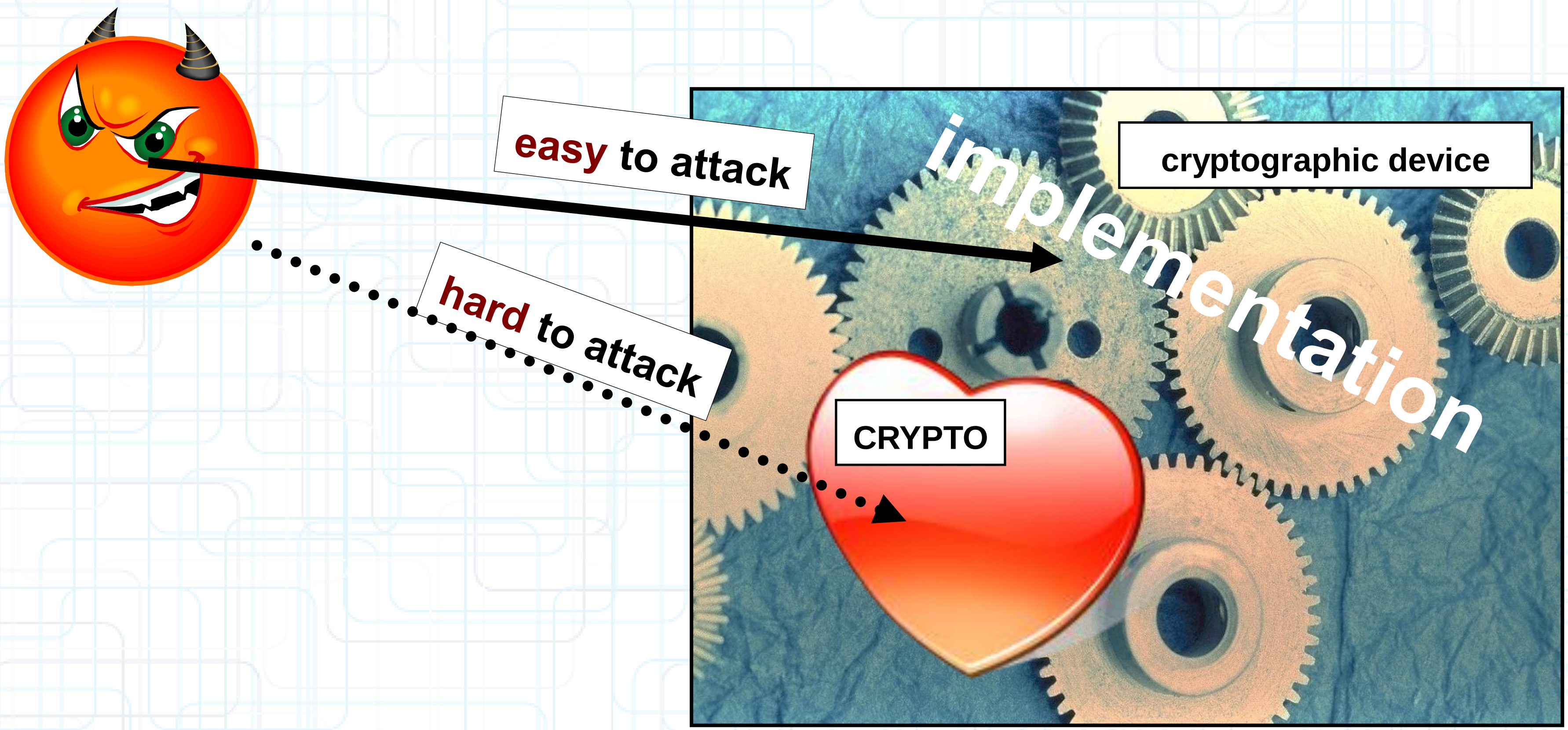
Properties:

- **leakage-resilient**
- in the **random oracle model**

Outline

1. Key-Evolution Schemes Resilient to Space Bounded **Leakage**
2. **Key-Evolution** Schemes Resilient to Space Bounded Leakage
3. Previous results in the area
4. The model
5. Random Oracle Model – remarks
6. Result and proof techniques

Key-Evolution Schemes Resilient to Space Bounded **Leakage**

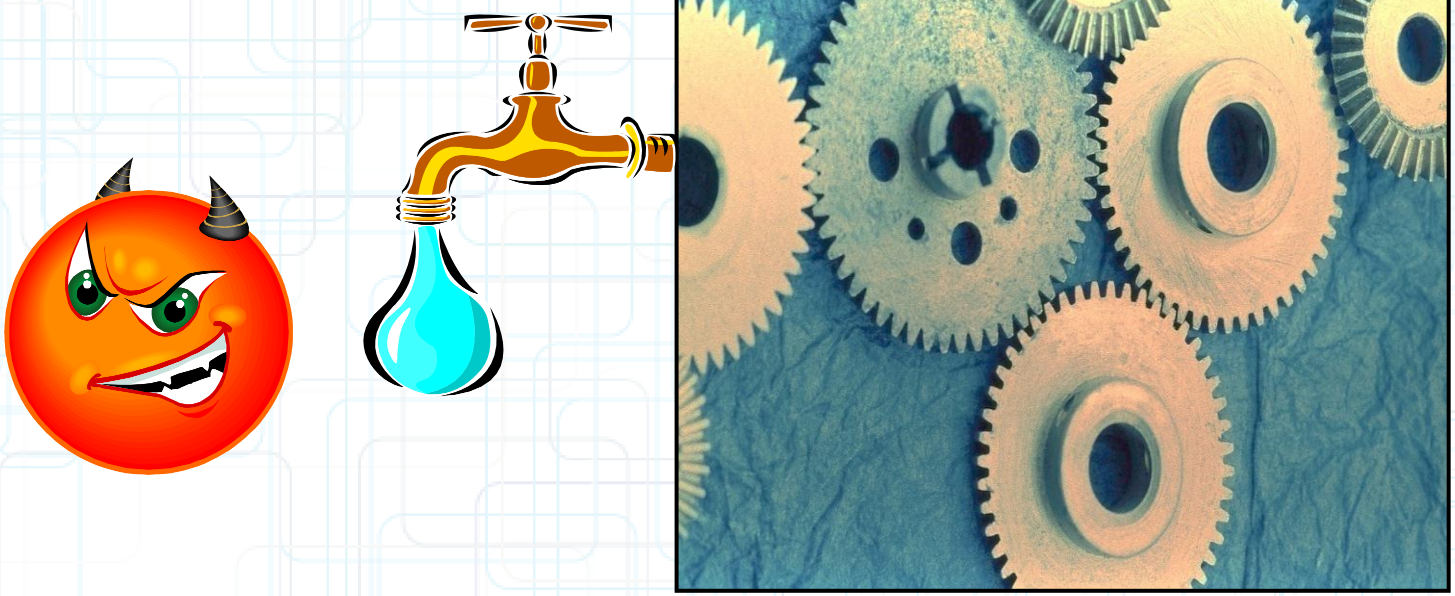


Key-Evolution Schemes Resilient to Space Bounded **Leakage**

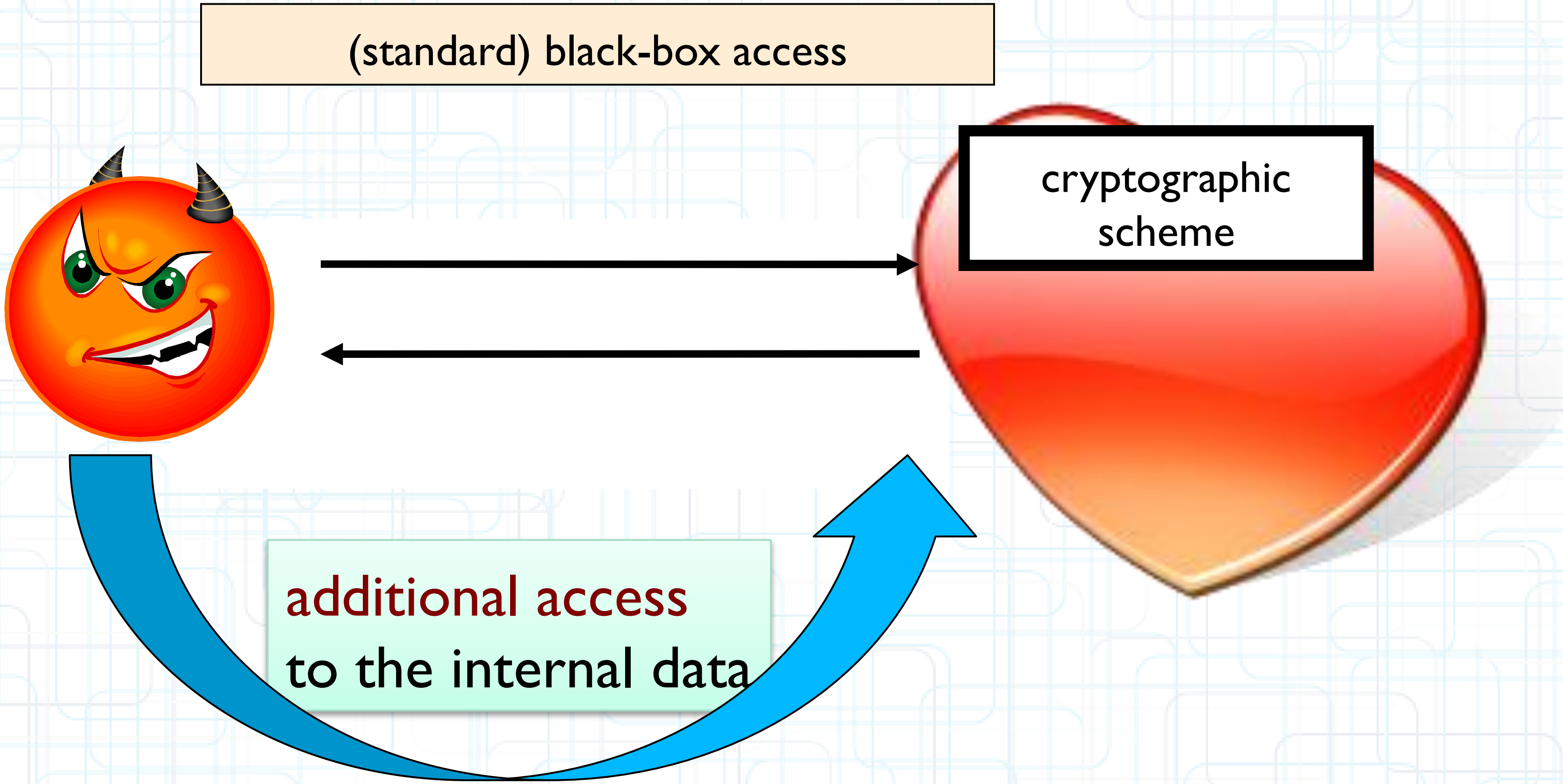
Side channel information:

- power consumption,
- electromagnetic leaks,
- timing information,

etc.



Key-Evolution Schemes Resilient to Space Bounded **Leakage**



Key-Evolution Schemes Resilient to Space Bounded **Leakage**

Generally speaking we model:

- Side-channel leakage
- Leakage caused by malicious software (viruses etc.)

Key-Evolution Schemes Resilient to Space Bounded Leakage

In each round the secret key **K** gets refreshed.

Assumptions:

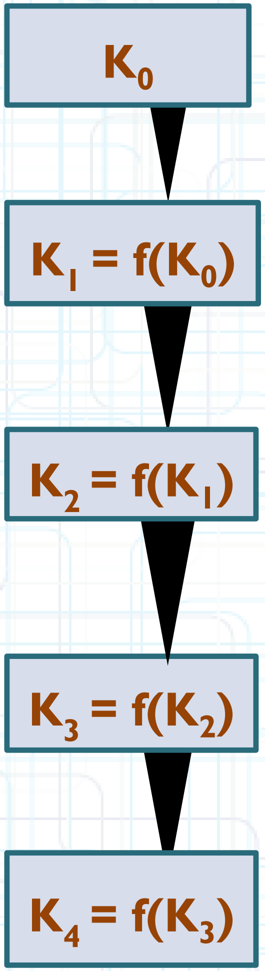
key evolution function **f** has to be **deterministic**

$$\mathbf{K_{i+1} = f(K_i)}$$

(no refreshing with external randomness)

also the refreshing procedure may cause leakage

New leakage in every round



Previous work on leakage-resilient key-evolution

Kocher:

- Leakage function cannot make any random oracle calls
- Output length is bounded slightly smaller then **|k|**

Previous work on leakage-resilient key-evolution

Yu Yu et al.:

- Leakage not adaptive
- Leakage function cannot evaluate hash function (modeled as usual by random oracle model)

Previous work on leakage-resilient key-evolution

Dziembowski and Pietrzak:

“only computation leaks information” model, so data can leak if and and only if it is accessed

Our approach
middle-of-the-road approach

Most prior "practical" papers

- **Simple and efficient**
- Intuitive notion of security **without formal guarantees**

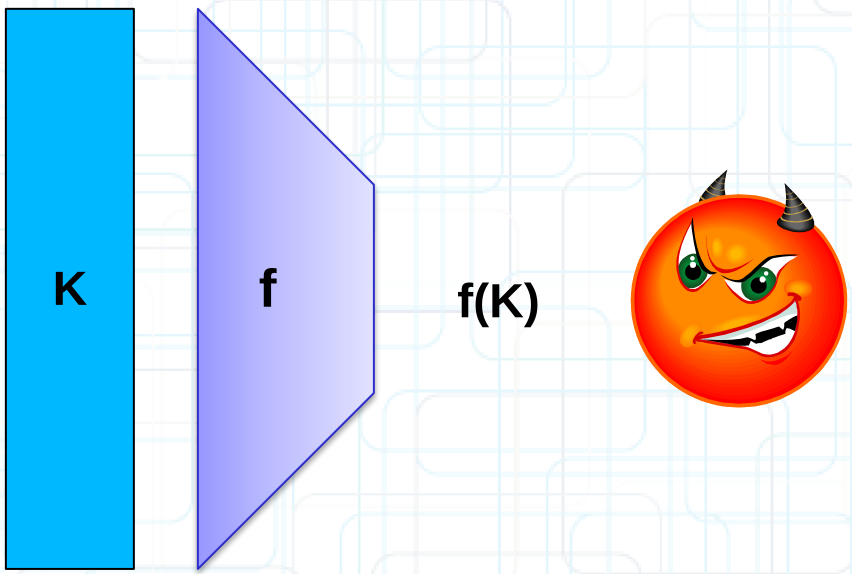
Most prior "theoretical" papers

- **Rigor and provable security**
- **Strong restrictions**, eg.
  - Only data actually used in computation can leak

Modelling the leakage

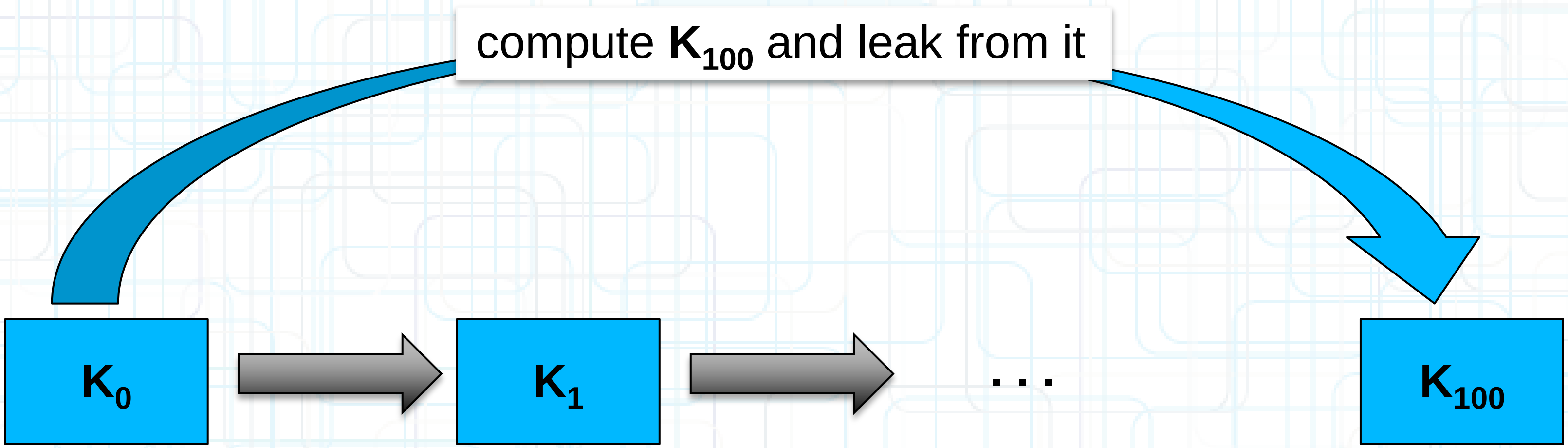
"Memory attacks", "Bounded-Retrieval Model":

The adversary is allowed to learn any input-shrinking function **f** of the secret:



Problem

The function **f** can compute the “future keys”:

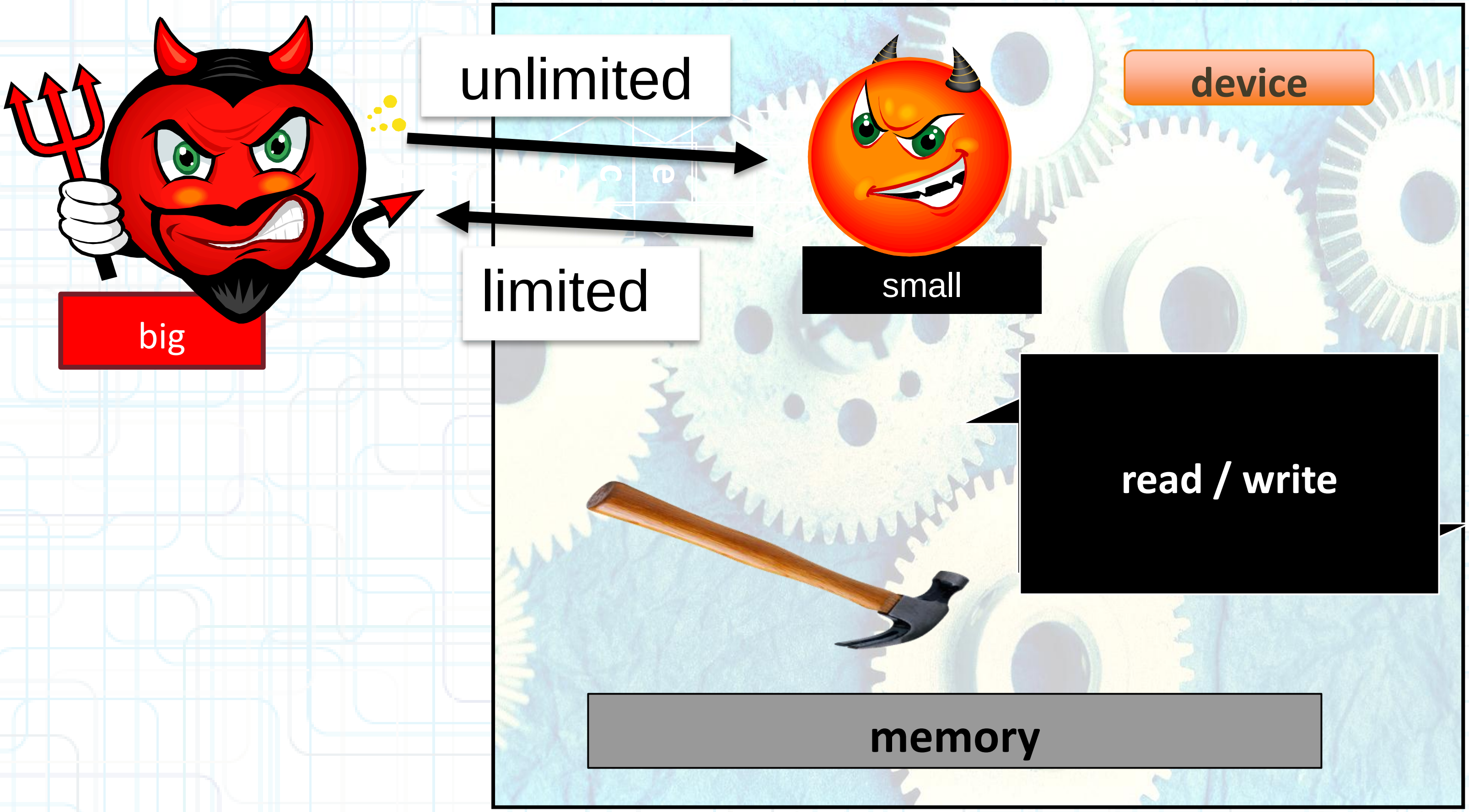


Moral: **f** has to be from a restricted class.

Our solution: limit **f** computationally.

we will assume that **f** is **space bounded**

The Model

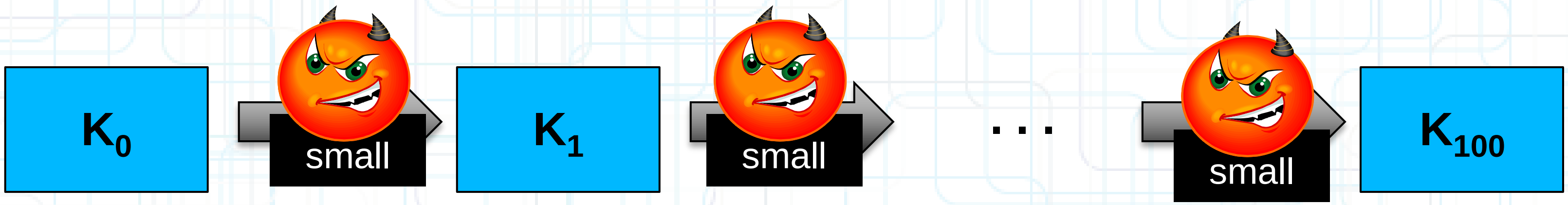


the “small” adversary can observe the key evolution and even partially control it

Security requirement: the “future keys” should remain secret.

The adversary can “partially control“ the key-evolution

The only thing that we require is that the key gets really evolved.



Adversary can use his own algorithm for evolving keys

Adversary can’t keep K_0 in the memory and leak it bit by bit because he is forced to evolve

The model remarks

- Random oracle model
  - theoretical shortcoming
- The leakage function that can make random oracle calls itself
- We **DO NOT** rely on the assumption that only data used in the computation can leak

The model remarks

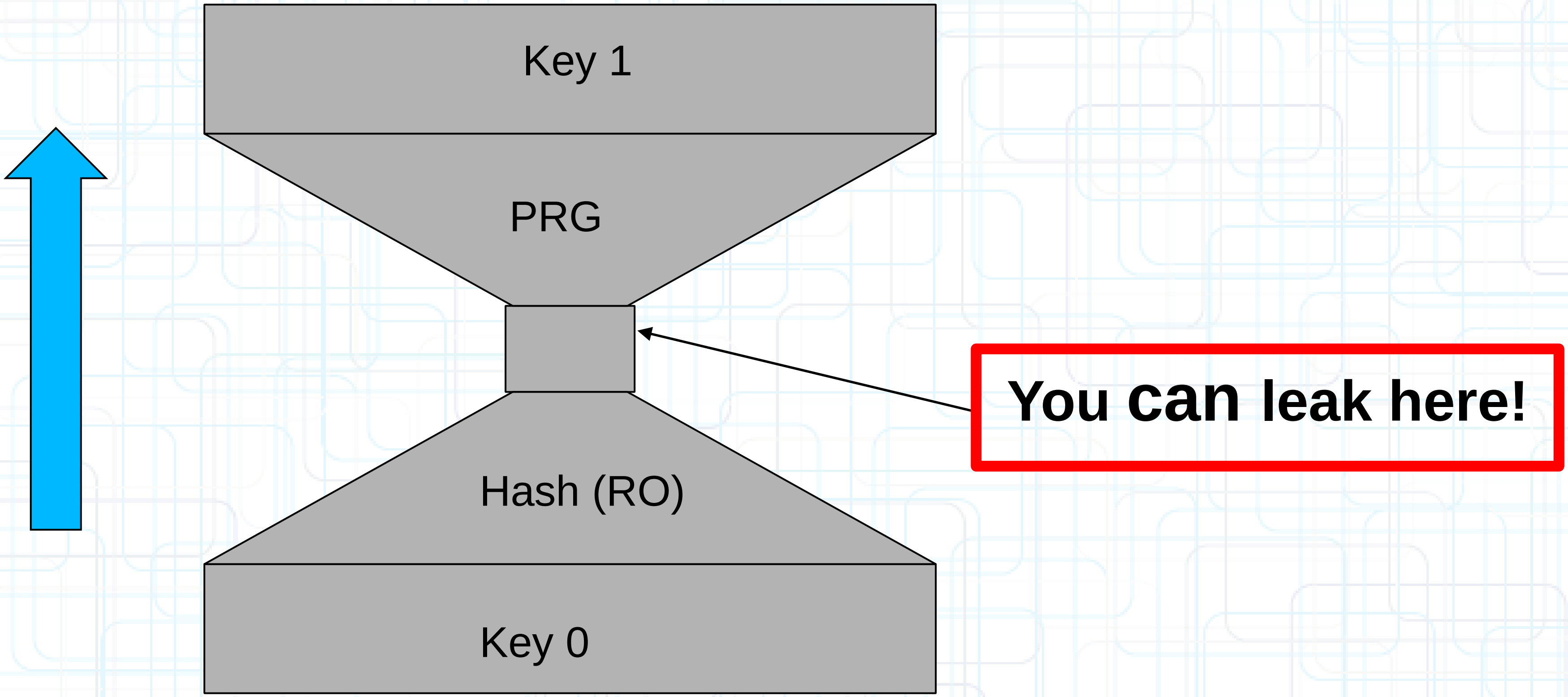
Secure against even against restricted active attacks

- Model seems to be too strong in this case.

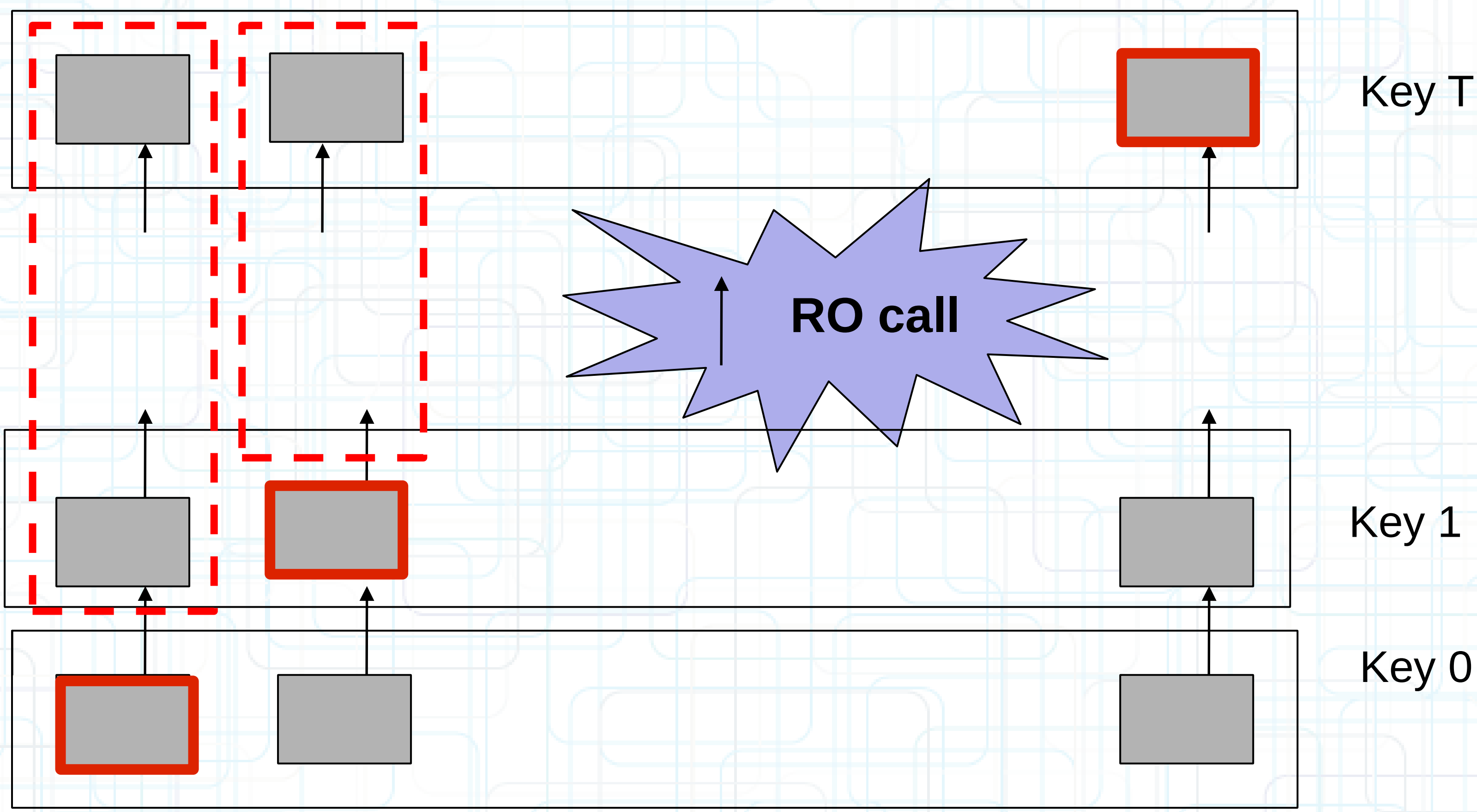
  However now it protects also against implementation errors.

We work in Random Oracle Model
Why isn't it obvious?!

Consider a following hash function:



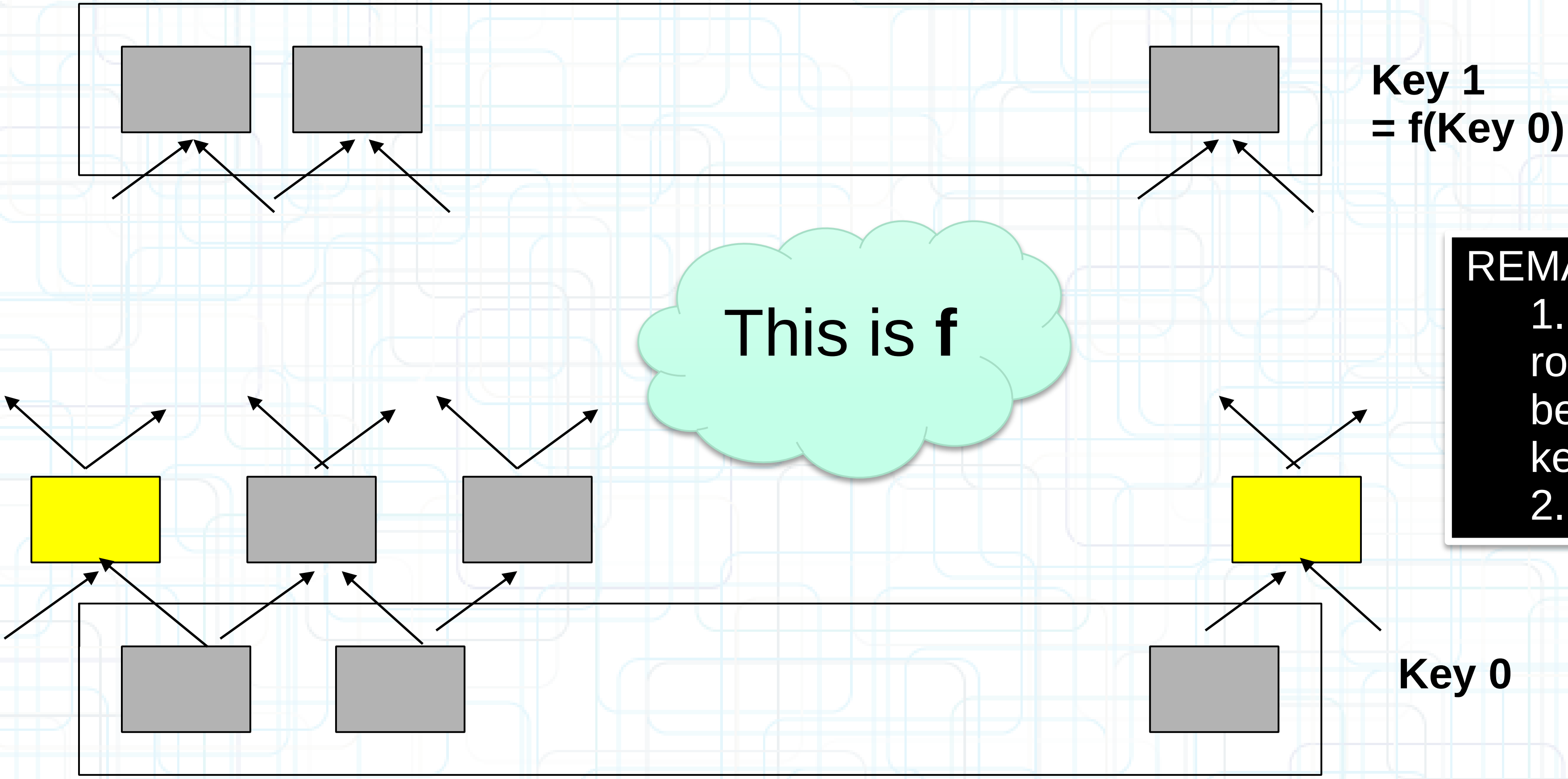
Another wrong idea



Each key is divided into blocks that evaluates independently using RO

Our solution

Only the compression function is modelled as a random oracle.



REMARKS:
1. Additional rows between keys
2. It is cyclic

Note: this requires almost no additional space.

Our result

- We show that **f** described above is secure key-evolution scheme in our model
- **c** - amount of bits that the adversary can retrieve in each round
- **s** – space that adversary can use (includes **K**)
- We need:

$$4c + s \leq 3\,|K|\,/\,2$$

A pinch of the proof

We define some specific game to be played on acyclic graph with black and red pebbles

Forget the model. For a moment we play a game.

How do I play?

DETAILS IN THE PAPER

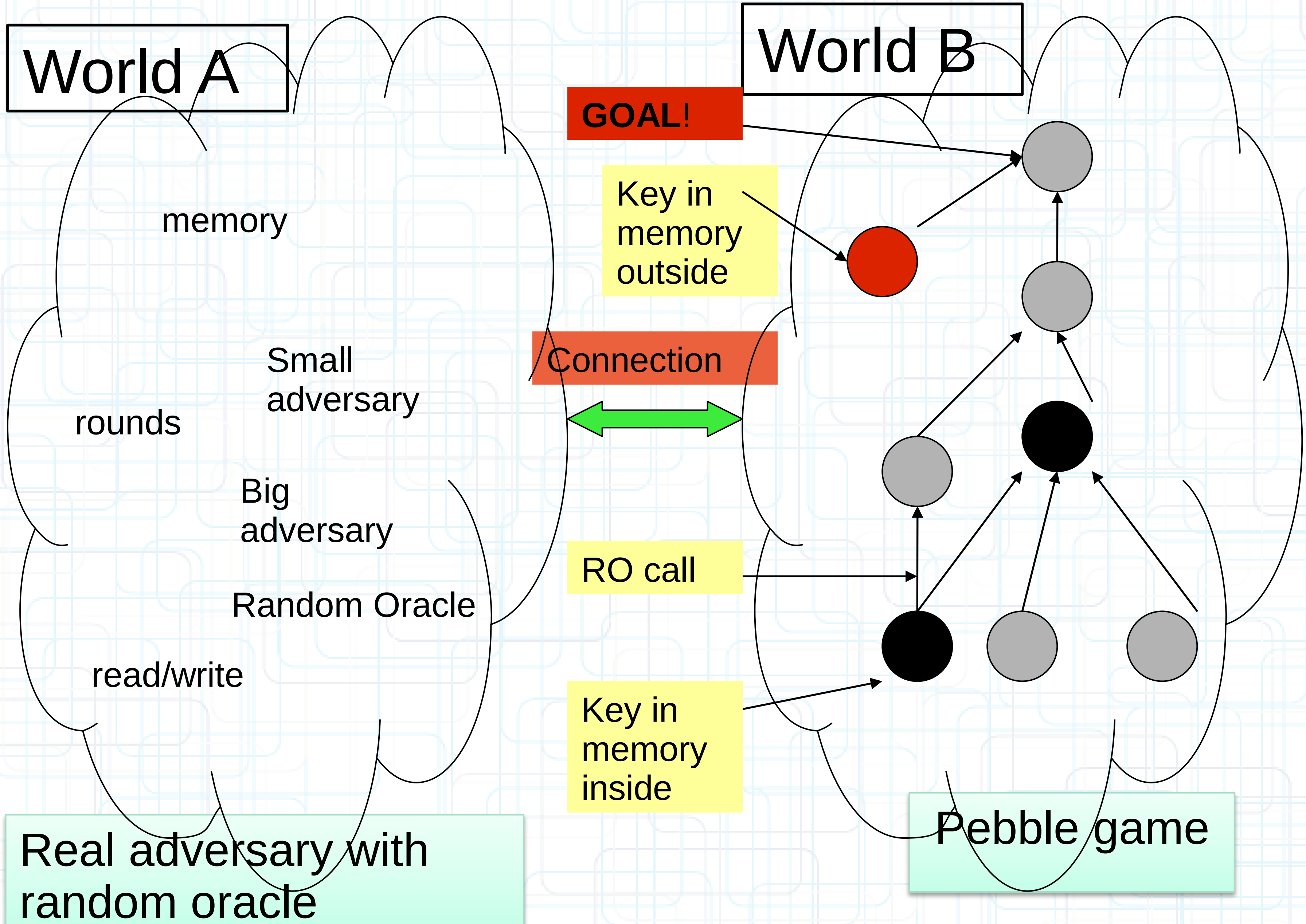
Some rules describing when it is legal to move a pebble or to put new pebble on the graph

Goal: put a pebble on some specific vertices

Number of pebbles you can use is limited

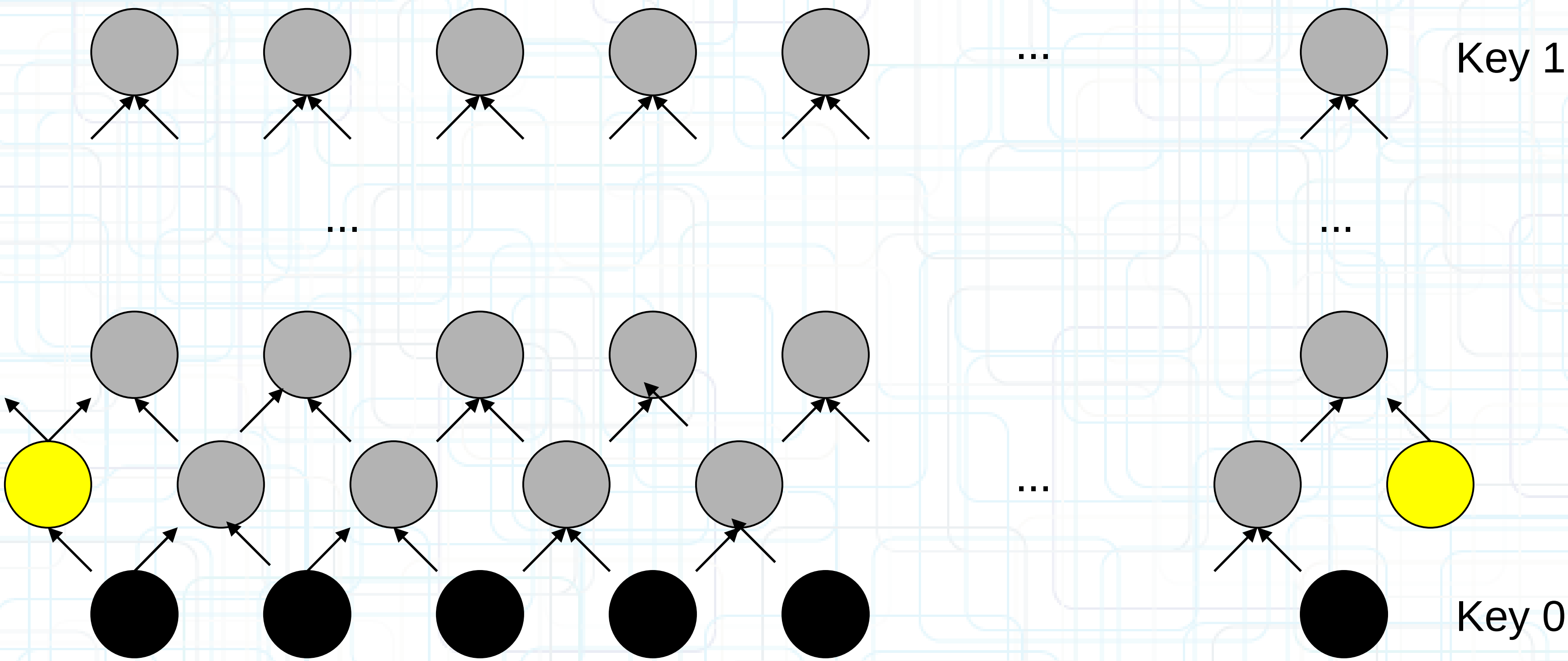
When you achieve some intermediate goal vertex – you get some new pebbles ≈ new round operation

A pinch of the proof



A pinch of the proof

Pebbling game corresponding to our construction **f:**



Intuition: It is hard to pebble top row with limited number of pebbles

You saw this graph before. But – it used to be a graph of the order of calling RO. Now it is a graph for a game.

A pinch of the proof

Remark: Connection is not trivial!

Intermediate keys are **not** atoms

For example an adversary may delete just few last bits of each key and “guess” those when needed (so in fact adversary may put just a *part of pebble* on a vertex)

The proof should somehow include above possibility

Thank you!