At the NIST Workshop on Elliptic Curve Cryptography Standards, 2015

A random zoo:
sloth, un$\overset{i}{\downarrow}$corn, and trx

Art by Léa Kissling

Arjen K. Lenstra
and **Benjamin Wesolowski**

EPFL, Switzerland

Public randomness

There are many situations where large interests depend on random choices

Public randomness

There are many situations where large interests depend on random choices

Assemblies of citizens

Public randomness

There are many situations where large interests depend on random choices

National lotteries

Public randomness

There are many situations where large interests depend on random choices

Sports drawings of lots

Public randomness

There are many situations where large interests depend on random choices

Random sample voting

Public randomness

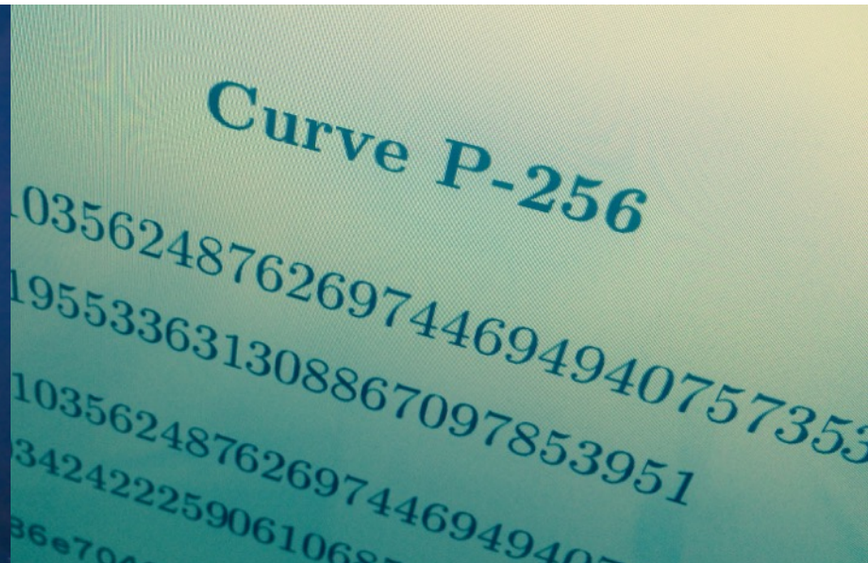
There are many situations where large interests depend on random choices

Randomness beacons

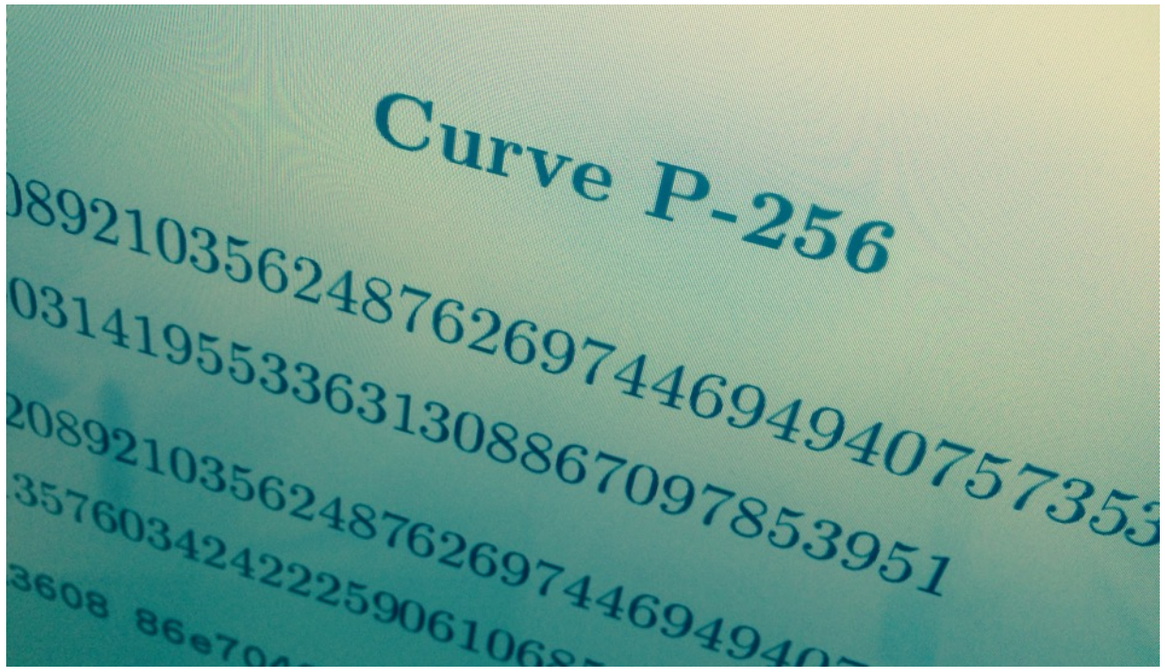
Public randomness

There are many situations where large interests depend on random choices

Cryptographic standards



Public randomness



Many secure elliptic curve parameters, and only a small set are standardized, chosen… randomly?

Convince people they can **trust** that particular curve you picked among a myriad of others

Choosing standards

How to convince people you did not cheat?

Choosing standards

How to convince people you did not cheat?

1: **do not even try**. No justification: FRP256V1 published by the ANSSI, or the OSCCA curve published with the SM2 algorithms

Choosing standards

How to convince people you did not cheat?

1: **do not even try**. No justification: FRP256V1 published by the ANSSI, or the OSCCA curve published with the SM2 algorithms

2: **verifiably hashed curves**. Publish the curve and a seed s, and it can be verified that the curve was derived from a hash of s. But no justification on the choice of s: the NIST P-curves

Choosing standards

How to convince people you did not cheat?

1: **do not even try**. No justification: FRP256V1 published by the ANSSI, or the OSCCA curve published with the SM2 algorithms

2: **verifiably hashed curves**. Publish the curve and a seed s, and it can be verified that the curve was derived from a hash of s. But no justification on the choice of s: the NIST P-curves

3: **nothing-up-my-sleeve numbers**. Publish a deterministic procedure and use digits of a natural constant as a seed (e.g., digits of π): Brainpool

Choosing standards

How to convince people you did not cheat?

1: **do not even try**.

2: **verifiably hashed curves**.

3: **nothing-up-my-sleeve numbers**.

Manipulating ~~Choosing~~ standards

How to convince people you did not cheat?

1: **do not even try**.

2: **verifiably hashed curves**.

3: **nothing-up-my-sleeve numbers**.

D. J. Bernstein, T. Chou, C. Chuengsatiansup, A. Hülsing, T. Lange, R. Niederhagen, and C. van Vredendaal

How to manipulate curve standards: a white paper for the black hat

Cryptology ePrint Archive, Report 2014/571, 2014. http://eprint.iacr.org/2014/571

Choosing standards

How to convince people you did not cheat?

4: **rigidity**? Or “no-arbitrary-choices” curves

M-221, E-222, Curve1174, Curve25519, BN(2,254), E-382, M-383, Curve383187, Curve41417, Ed448-Goldilocks, M-511, E-521

all advertised as “fully rigid”.

Choosing standards

4: **rigidity**? Or “no-arbitrary-choices” curves

… or the illusion of it?

Choosing standards

4: **rigidity**? Or “no-arbitrary-choices” curves
… or the illusion of it?

- Justifying a choice for an *x*% speed increase: rigid decision or camouflaged arbitrary choice?

trust < speed?

Choosing standards

4: **rigidity**? Or “no-arbitrary-choices” curves
… or the illusion of it?

- Justifying a choice for an *x*% speed increase: rigid decision or camouflaged arbitrary choice?

trust < speed?

- What if concerns arise about long term exposure to cryptanalysis? Standards would need to be refreshed, but rigid ⇒ predictable

Choosing standards

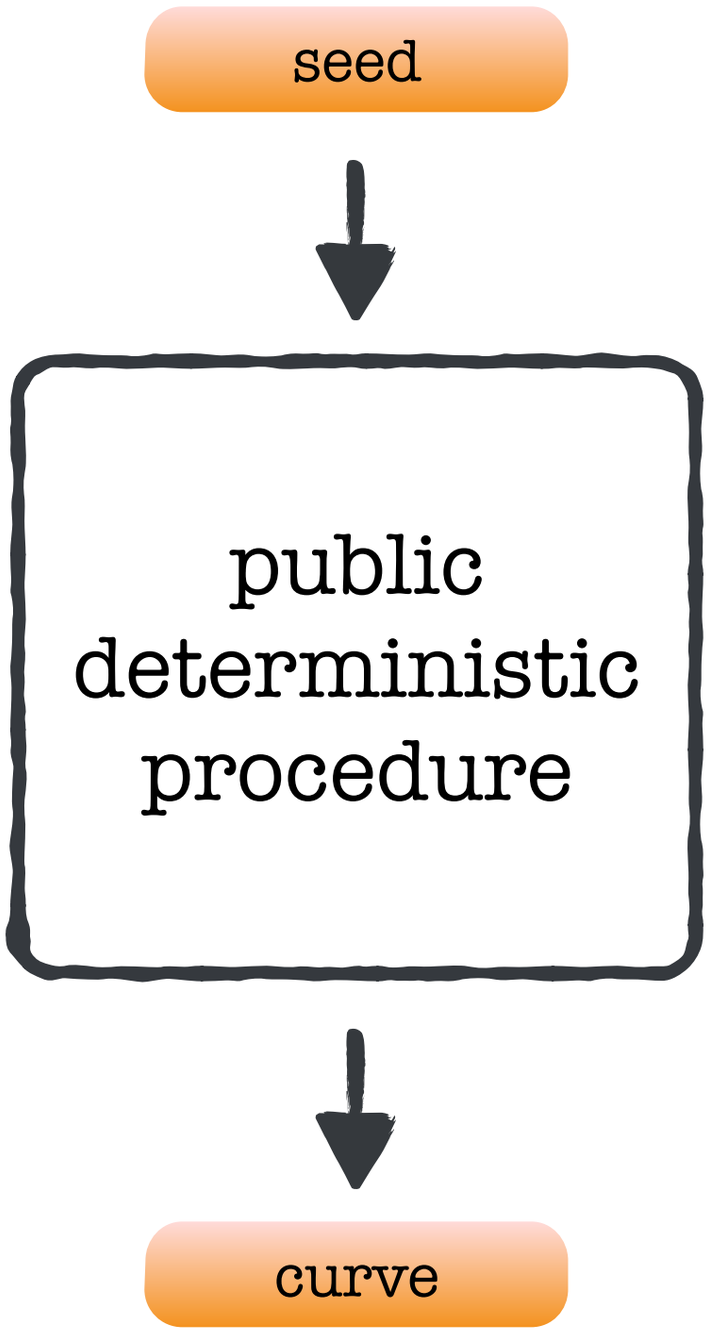
4: **rigidity**? Or “no-arbitrary-choices” curves
… or the illusion of it?

- Justifying a choice for an *x*% speed increase: rigid decision or camouflaged arbitrary choice?

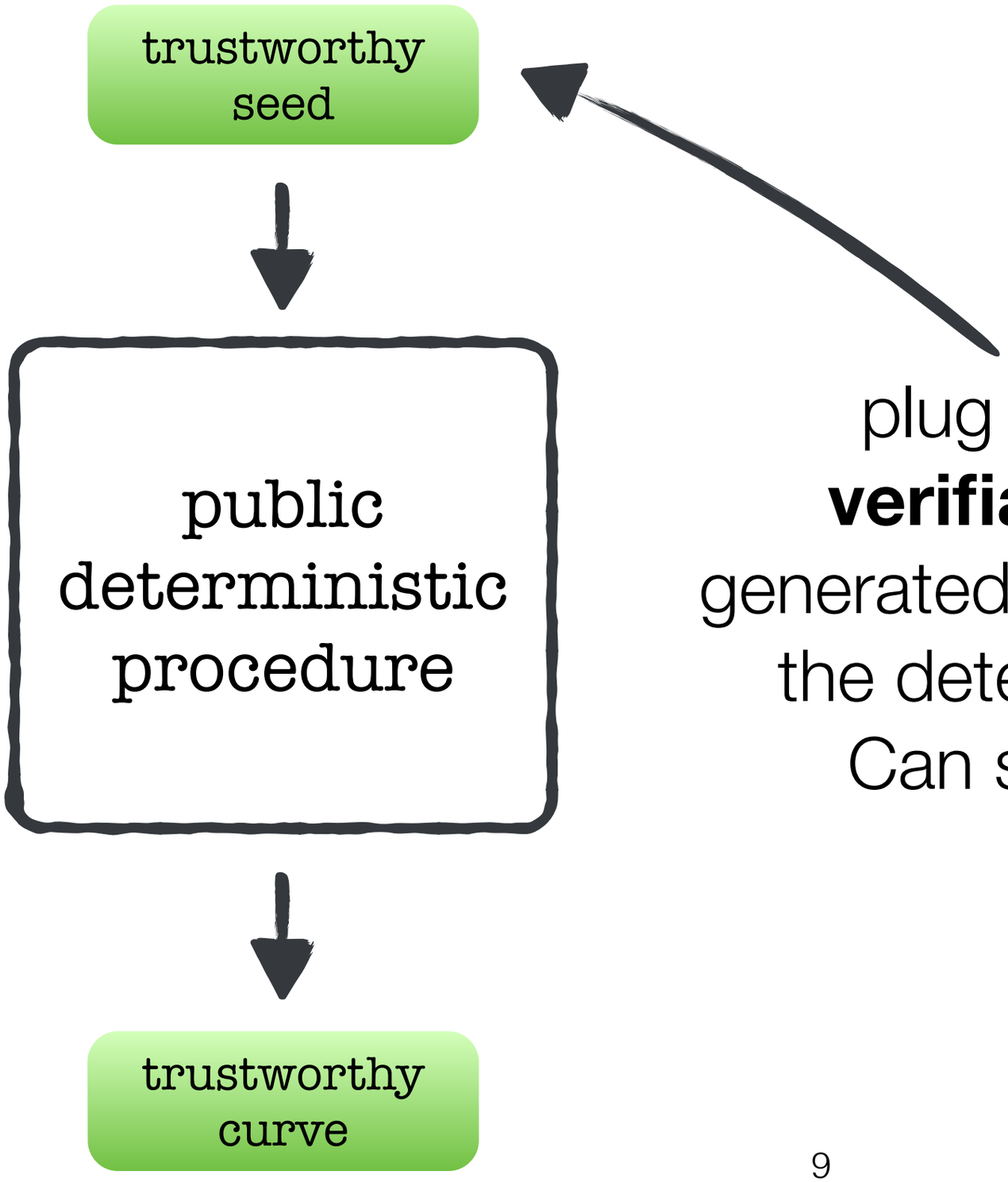
trust < speed?

- What if concerns arise about long term exposure to cryptanalysis? Standards would need to be refreshed, but rigid ⇒ predictable
- Rigid curves are not “normal”: small coefficients by construction

General framework



General framework



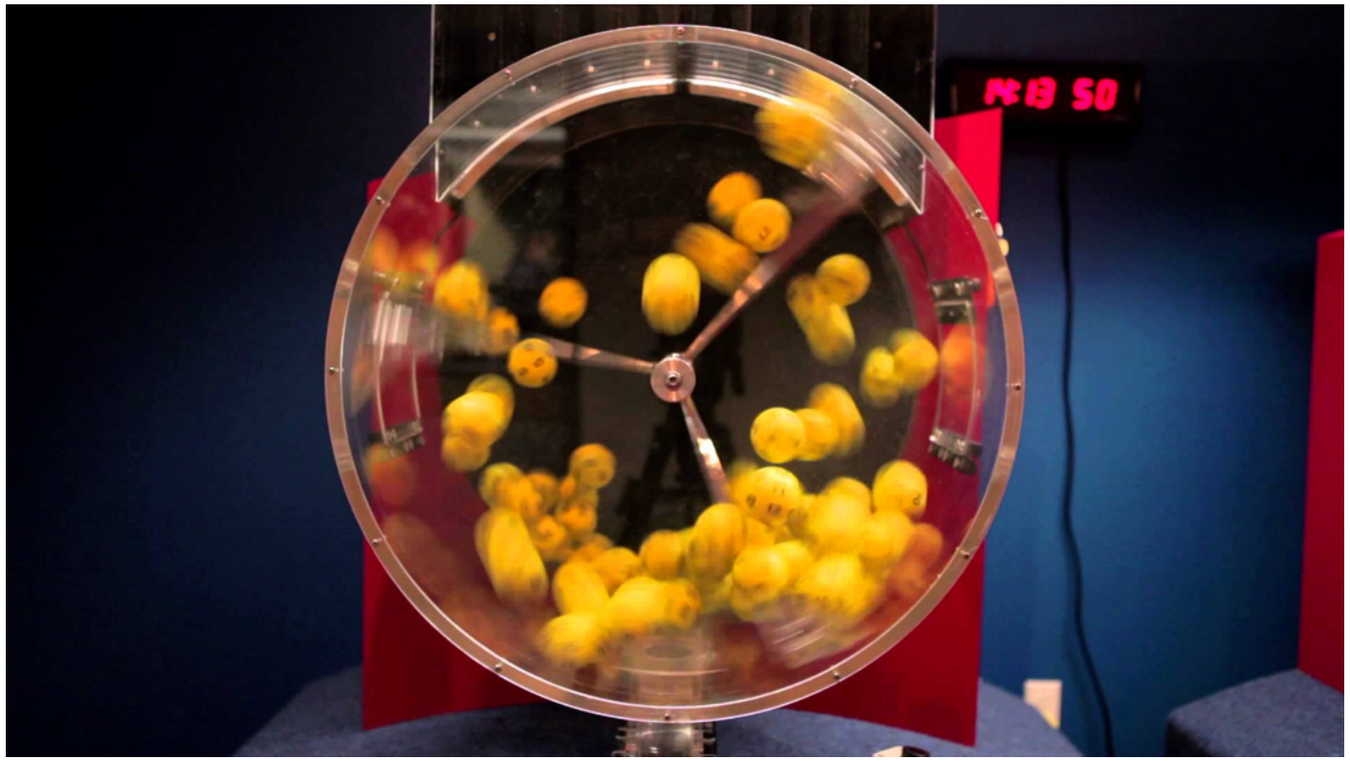
plug in a **trustworthy**, **verifiable** random seed, generated after the publication of the deterministic procedure. Can such a thing exist?

Folkloric methods

How to generate incorruptible random numbers?

Folkloric methods

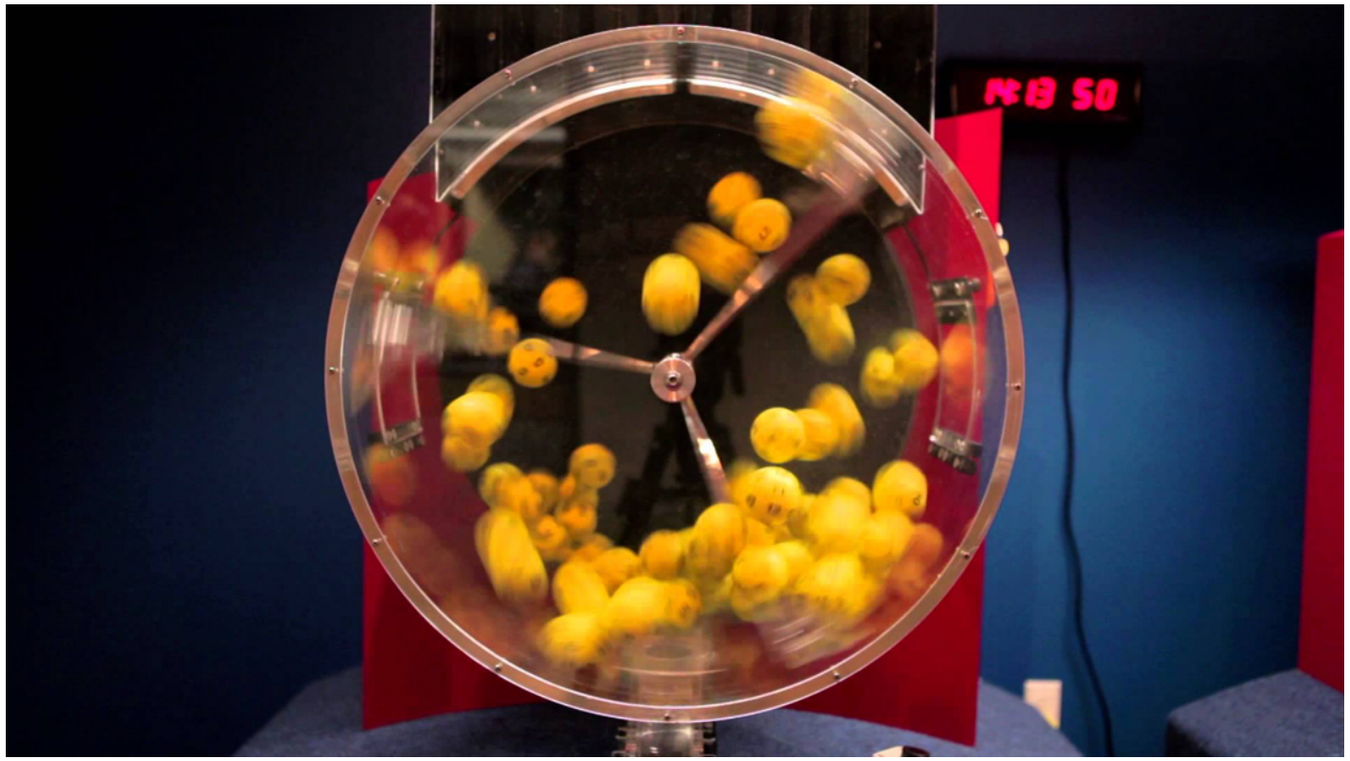
How to generate incorruptible random numbers?



“Live” broadcast of lotteries…

Folkloric methods

How to generate incorruptible random numbers?



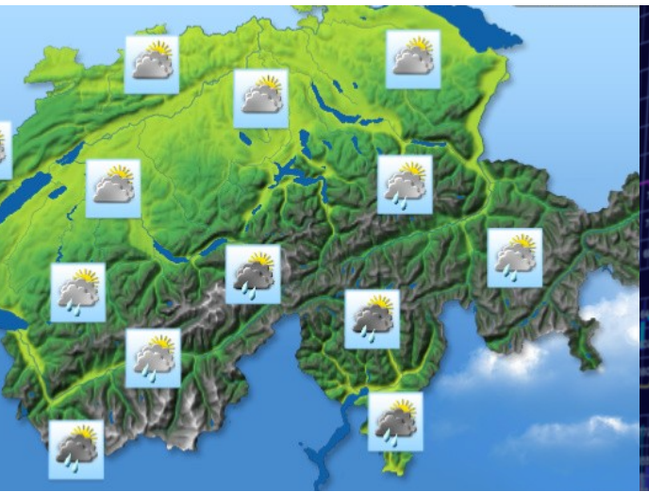
“Live” broadcast of lotteries…

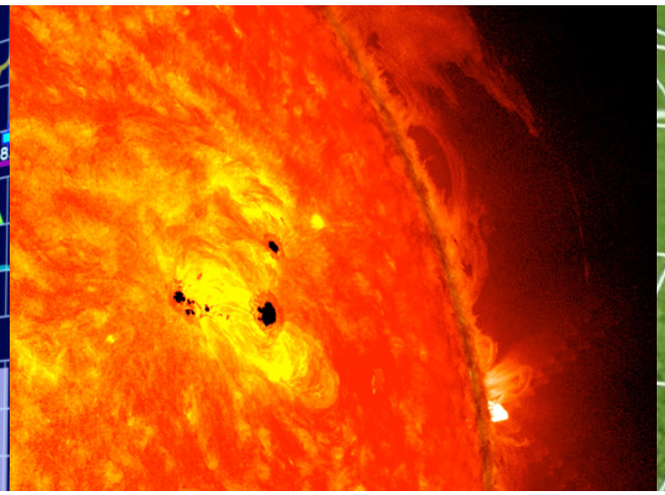
http://www.businesspundit.com/5-of-the-biggest-lottery-scandals/

Folkloric methods

How to generate incorruptible random numbers?

Use publicly available unpredictable data?

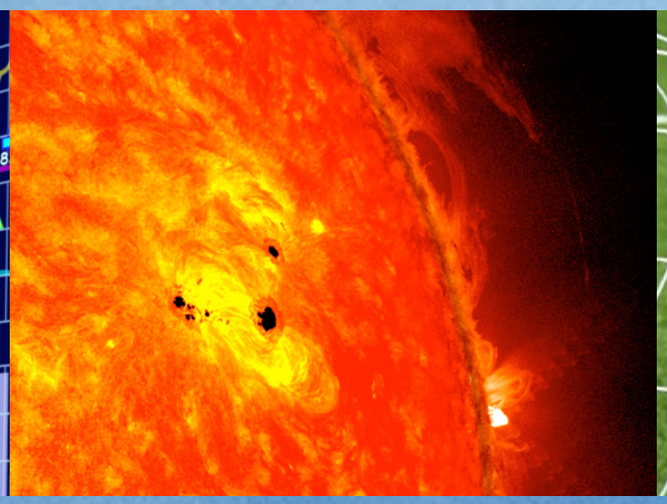




Folkloric methods

How to generate incorruptible random numbers?

Use publicly available unpredictable data?



Various kinds of ways to orchestrate, and manipulate.

The Truman show

Folkloric methods

How to generate incorruptible random numbers?

Can you trust a “random” number that you have not influenced yourself?

The Truman show (not being paranoid)

Via an online protocol

Via an online protocol

A group of people can agree on a pseudo-random number via an online protocol using **commitment schemes**:

Via an online protocol

A group of people can agree on a pseudo-random number via an online protocol using **commitment schemes**:

- everybody **picks** an s in $\{0,1\}^n$ and **publishes** a commitment $c(s)$

Via an online protocol

A group of people can agree on a pseudo-random number via an online protocol using **commitment schemes**:

- everybody **picks** an s in $\{0,1\}^n$ and **publishes** a commitment $c(s)$
- after all the commitments are received, everybody **reveals** their s

Via an online protocol

A group of people can agree on a pseudo-random number via an online protocol using **commitment schemes**:

- everybody **picks** an s in $\{0,1\}^n$ and **publishes** a commitment $c(s)$
- after all the commitments are received, everybody **reveals** their s
- the group can **agree** on the xor-sum $s = \bigoplus s$

Via an online protocol

A group of people can agree on a pseudo-random number via an online protocol using **commitment schemes**:

- everybody **picks** an s in $\{0,1\}^n$ and **publishes** a commitment $c(s)$
- after all the commitments are received, everybody **reveals** their s
- the group can **agree** on the xor-sum $s = \bigoplus s$

Does this apply here?

Via an online protocol

A group of people can agree on a pseudo-random number via an online protocol using **commitment schemes**:

- everybody **picks** an s in $\{0,1\}^n$ and **publishes** a commitment $c(s)$
- after all the commitments are received, everybody **reveals** their s
- the group can **agree** on the xor-sum $s = \bigoplus s$

Does this apply here?

- This protocol does not **scale**

Via an online protocol

A group of people can agree on a pseudo-random number via an online protocol using **commitment schemes**:

- everybody **picks** an s in $\{0,1\}^n$ and **publishes** a commitment $c(s)$
- after all the commitments are received, everybody **reveals** their s
- the group can **agree** on the xor-sum $s = \bigoplus s$

Does this apply here?

- This protocol does not **scale**
- Two rounds, too involved

Via an online protocol

A group of people can agree on a pseudo-random number via an online protocol using **commitment schemes**:

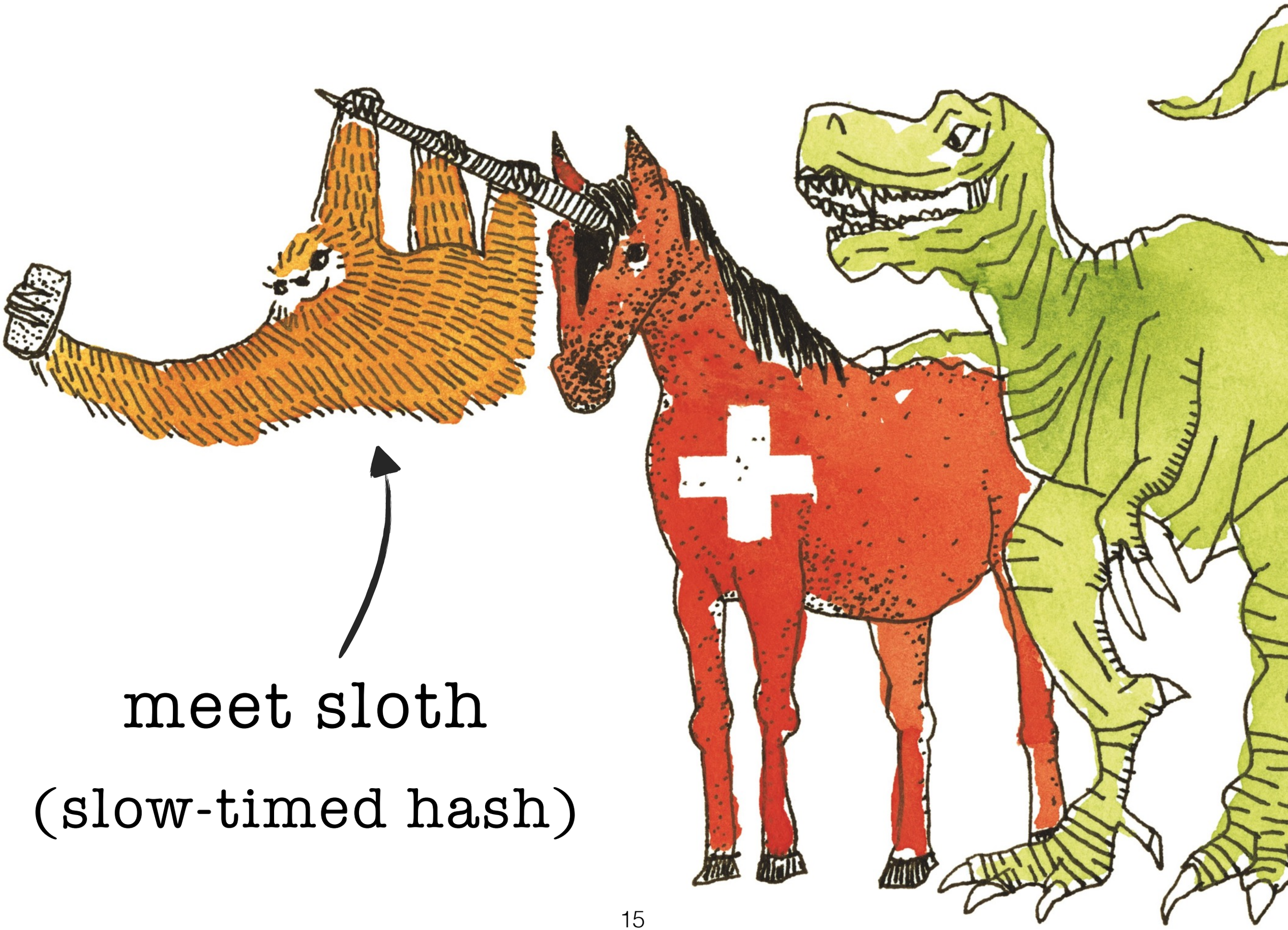
- everybody **picks** an s in $\{0,1\}^n$ and **publishes** a commitment $c(s)$
- after all the commitments are received, everybody **reveals** their s
- the group can **agree** on the xor-sum $s = \bigoplus s$

Does this apply here?

- This protocol does not **scale**
- Two rounds, too involved
- Subject to **denial of service**, or **manipulation**

What do we want?

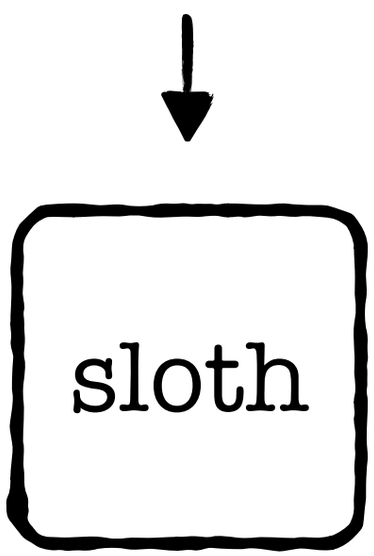
- A **simple**, one round protocol
- **Anybody** can participate, without prior notice
- **Easy** to use: as simple as a tweet
- Secure, **incorruptible**
- No time pressure: the (pseudo)random number need not be generated instantly


meet sloth
(slow-timed hash)

Sloth: slow-timed hash

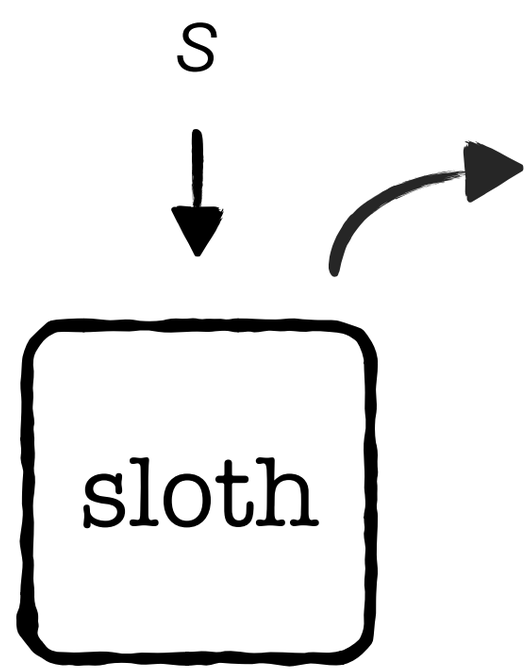
(all kinds of data as input)

s



Sloth: slow-timed hash

(all kinds of data as input)



c, a commitment to the input

Sloth: slow-timed hash

(all kinds of data as input)

s

c, a commitment to the input

sloth

≥ 10 minutes

g

(a hashed value)

Sloth: slow-timed hash

(all kinds of data as input)

s

c, a commitment to the input

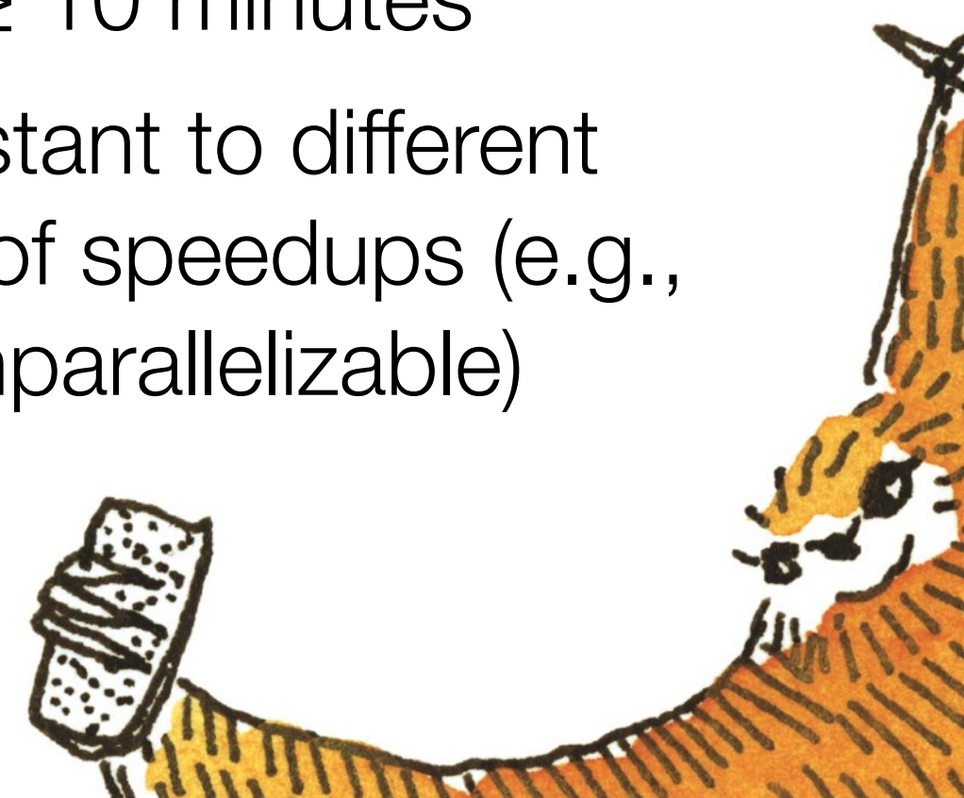
sloth

g

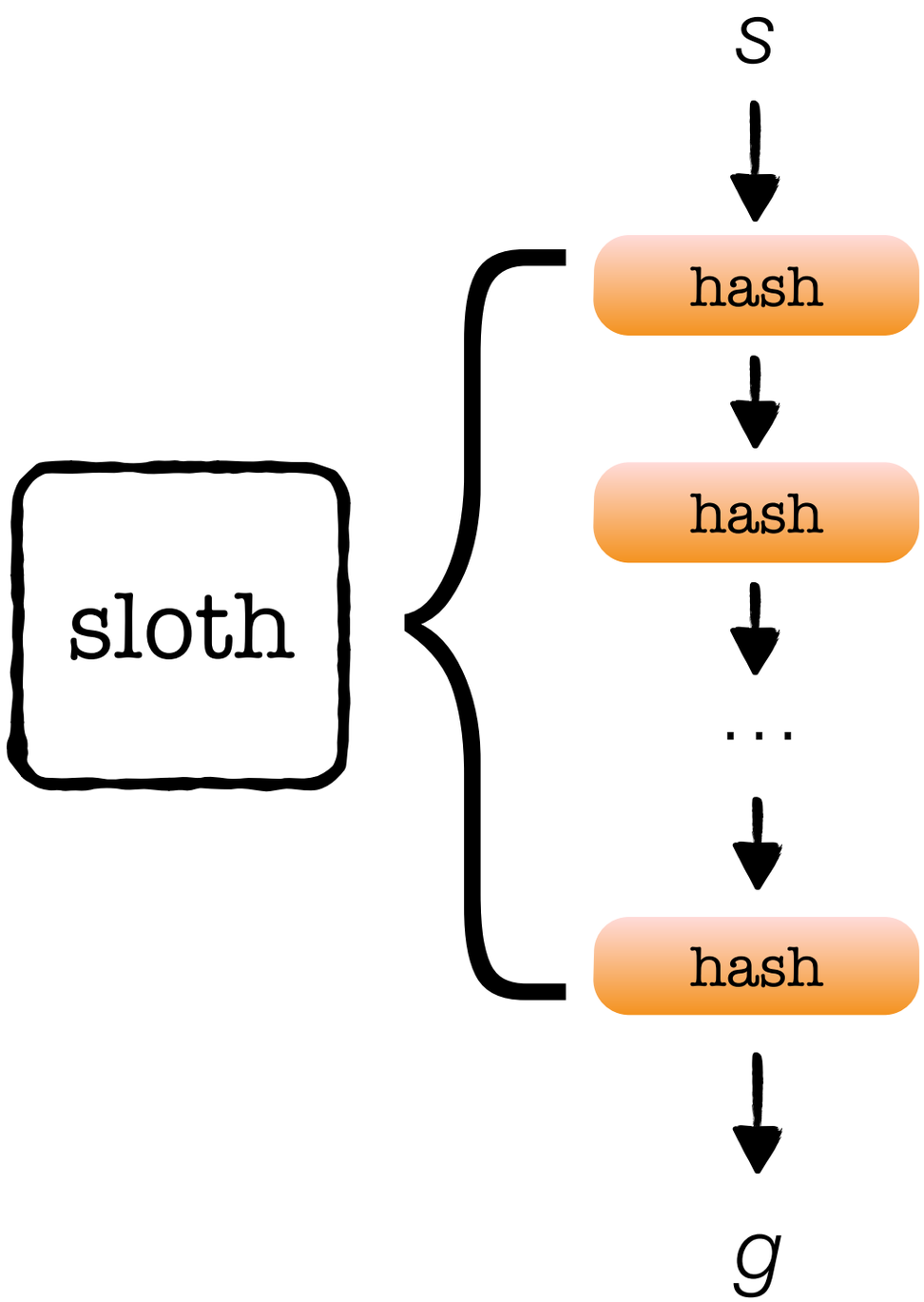
(a hashed value)

≥ 10 minutes

resistant to different kinds of speedups (e.g., unparallelizable)

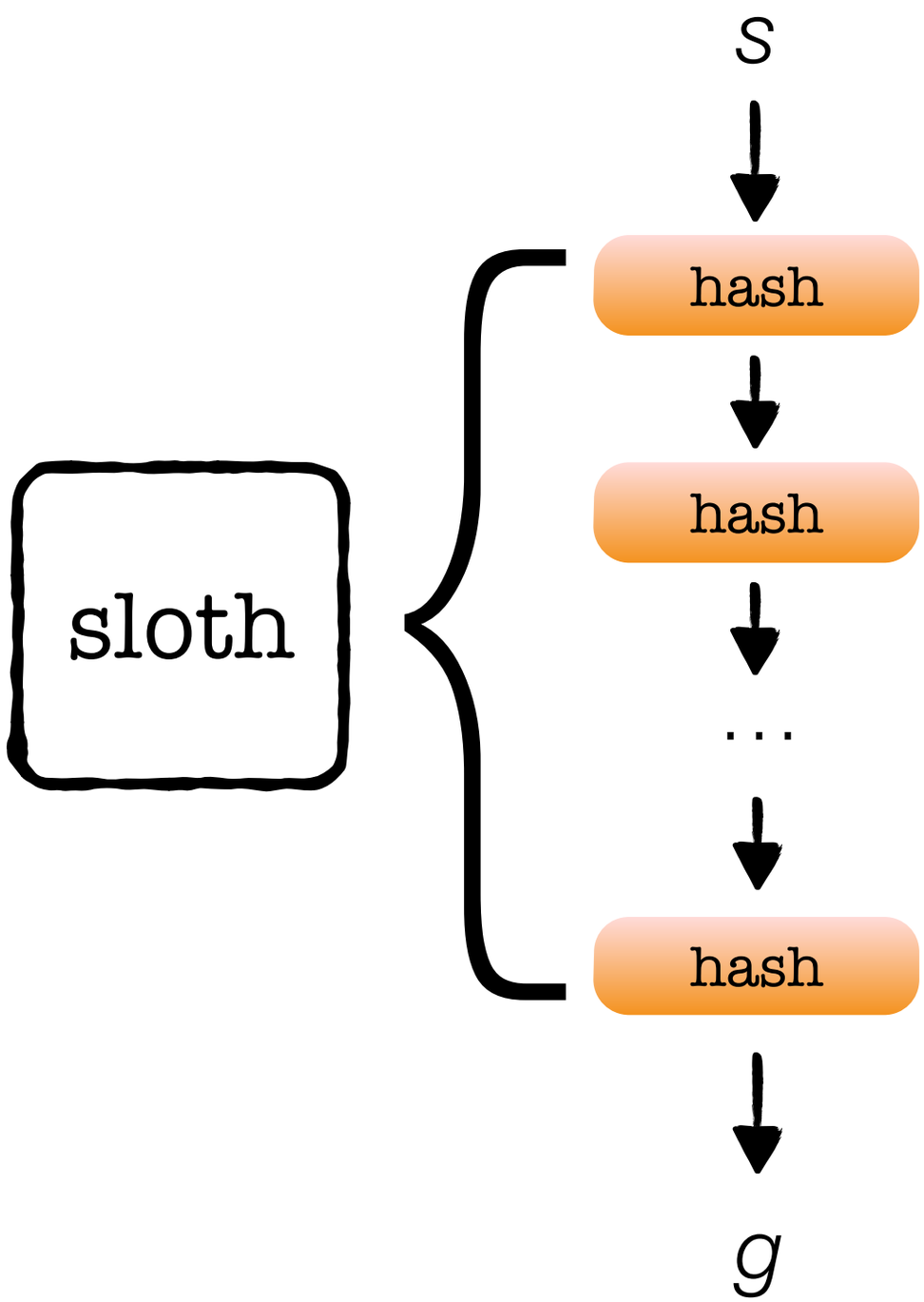


Sloth: slow-timed hash



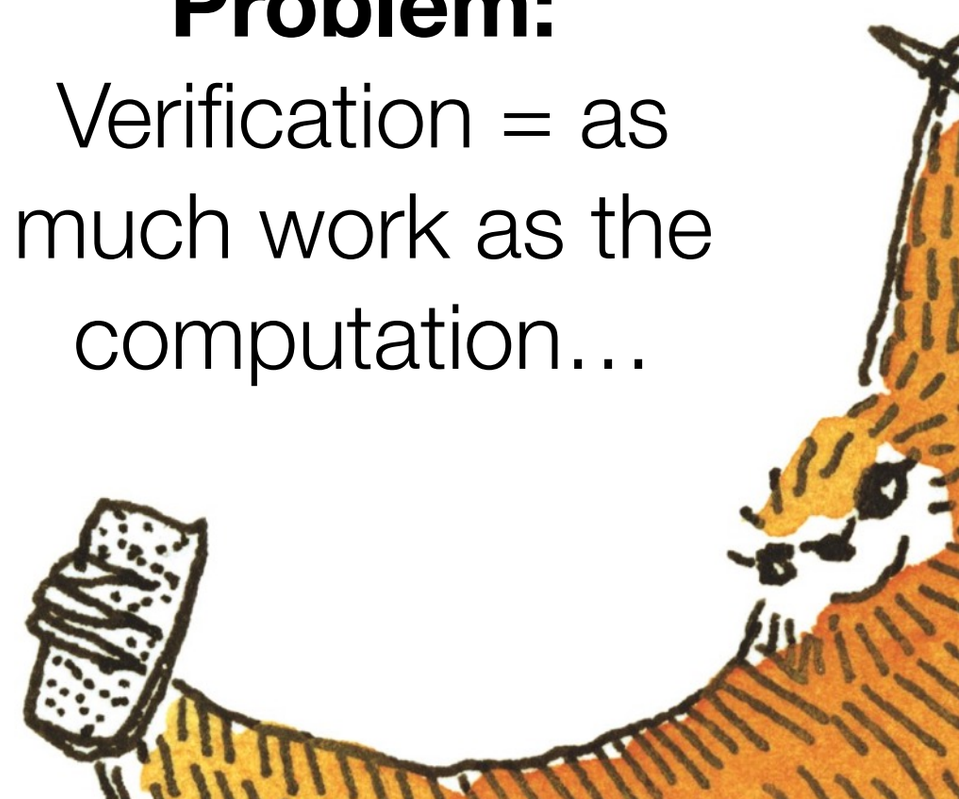
Enough iterations to make the computation last ≥ 10 minutes

Sloth: slow-timed hash



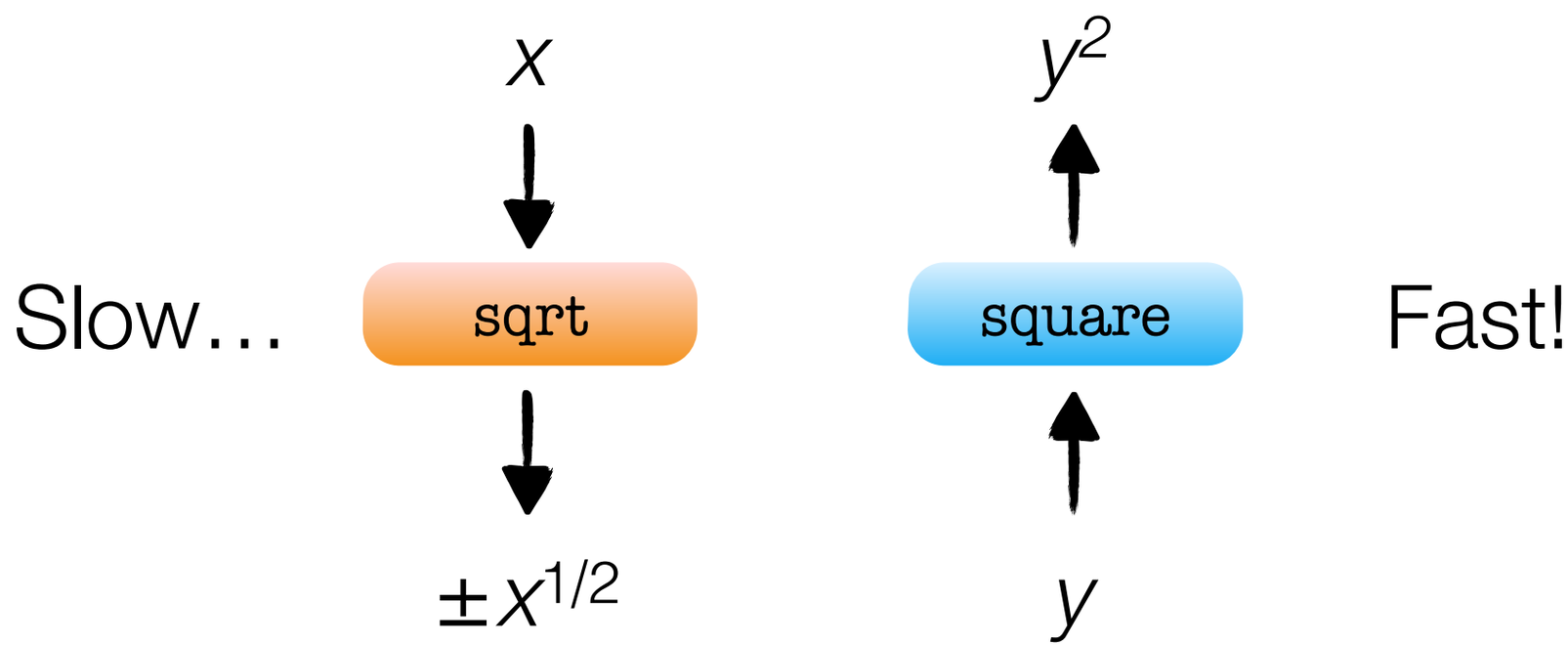
Enough iterations to make the computation last ≥ 10 minutes

Problem:
Verification = as much work as the computation…



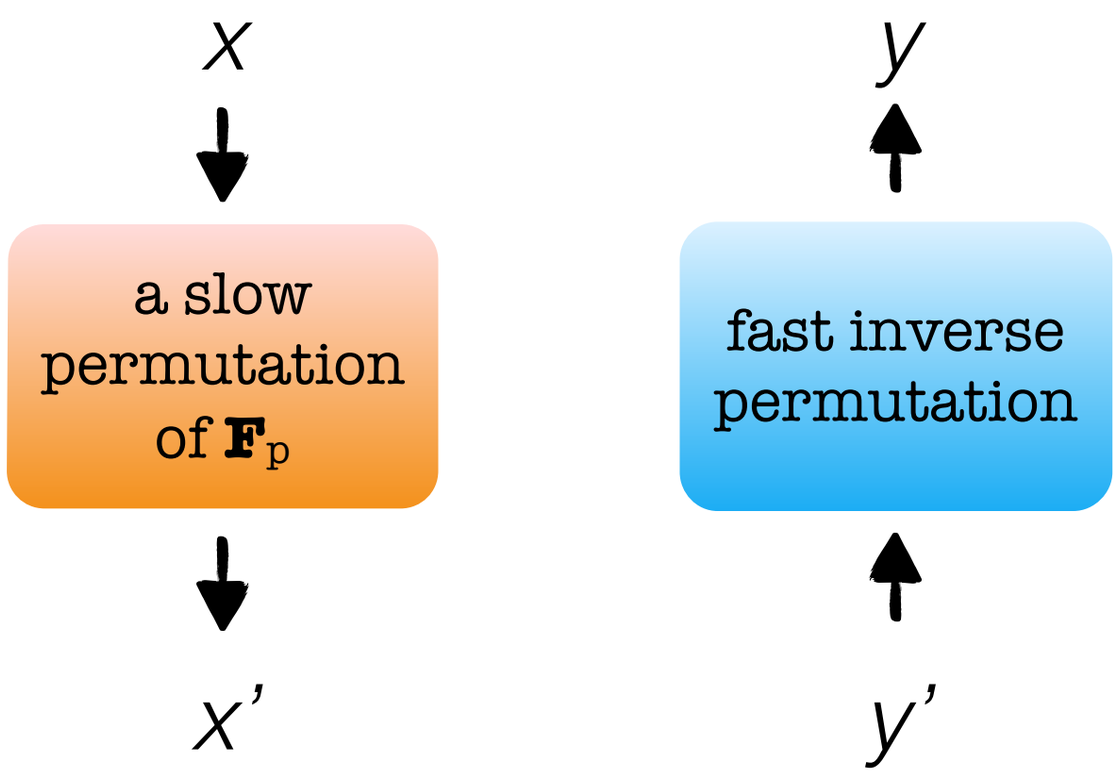
Sloth: slow-timed hash

Using square roots in $\mathbf{F}_p$



Sloth: slow-timed hash

Use $x \mapsto \pm x^{1/2}$ to design a permutation of $\mathbf{F}_p$



Sloth: slow-timed hash

s → hash → $x \in \mathbf{F}_p$

the seed

↓

permutation of $\mathbf{F}_p$

This permutation takes time

↓

$w \in \mathbf{F}_p$ a published witness

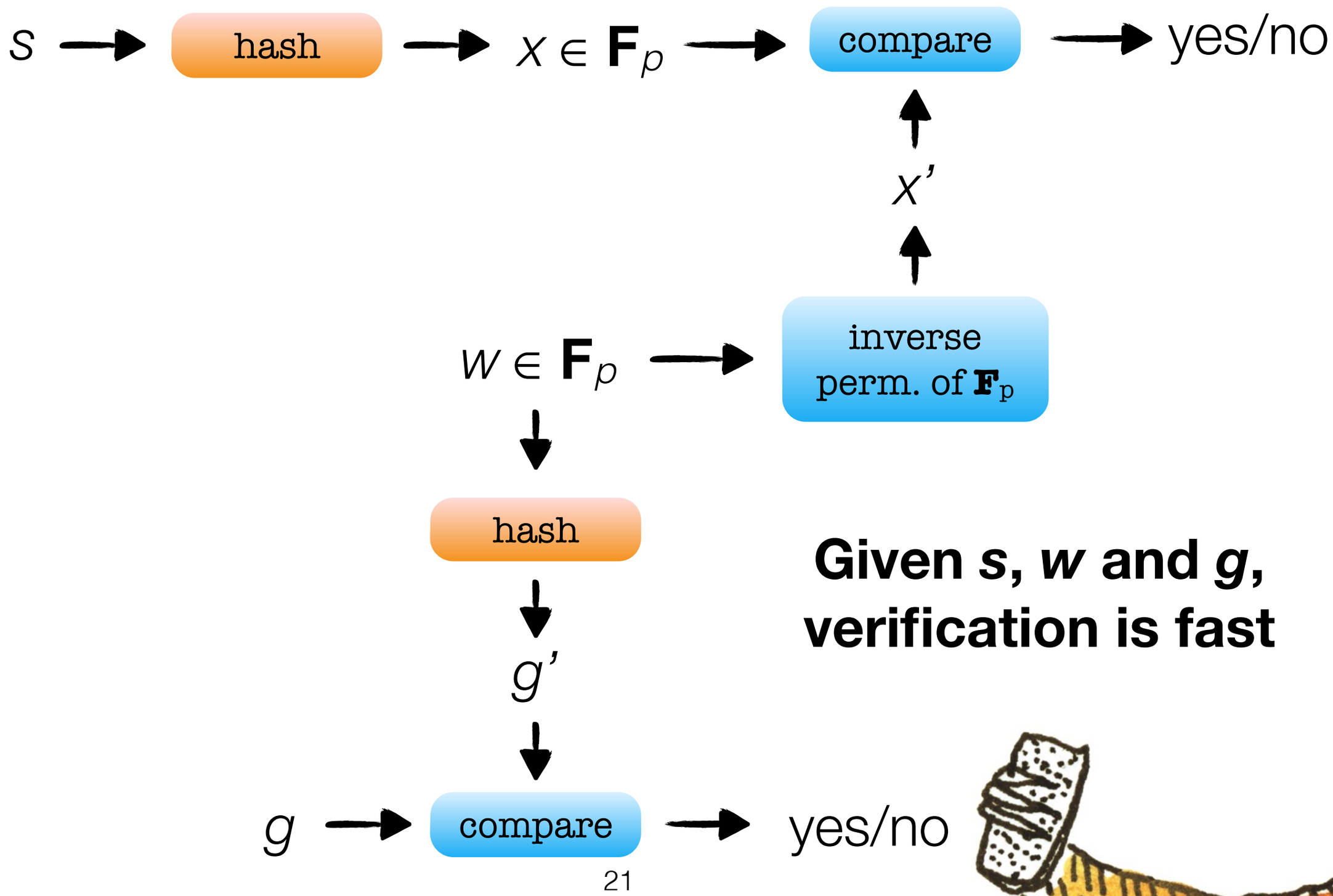
↓

hash

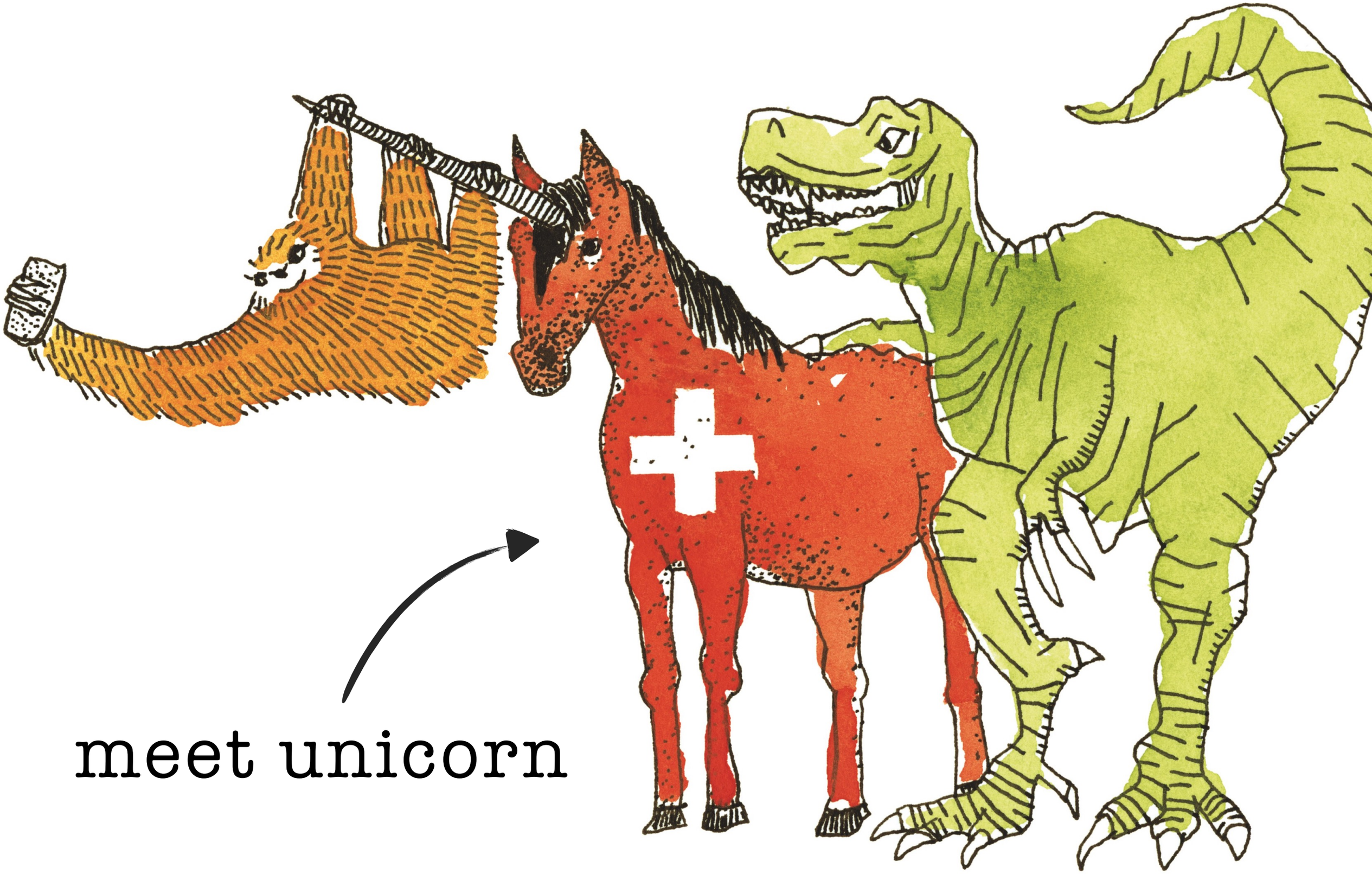
↓

the resulting hash g

Sloth: slow-timed hash



Given *s*, *w* and *g*, verification is fast



meet unicorn

(uncontestable random number)

Unicorn

Unicorn

- First, **announce** that public data gathering will take place during time interval $[t_{-1}, t_0)$

Unicorn

- First, **announce** that public data gathering will take place during time interval $[t_{-1}, t_0)$
- t_{-1}: start **receiving** all the data, concatenated in order of arrival to form the public seed s_0

Unicorn

- First, **announce** that public data gathering will take place during time interval $[t_{-1}, t_0)$
- t_{-1}: start **receiving** all the data, concatenated in order of arrival to form the public seed s_0
- t_0: choose a private seed s_1, let $s = s_1 \| s_0$. Start **computing** `sloth`(s) and immediately publish the commitment to the input s.

Unicorn

- First, **announce** that public data gathering will take place during time interval $[t_{-1}, t_0)$
- t_{-1}: start **receiving** all the data, concatenated in order of arrival to form the public seed s_0
- t_0: choose a private seed s_1, let $s = s_1 \| s_0$. Start **computing** `sloth`(s) and immediately publish the commitment to the input s.
- t_1: **publish** s_1 and the output of `sloth`: the witness w and the hash g

Unicorn

How to gather public data, such that anyone can contribute?

- Using twitter: announce a hashtag at time t_{-1}, and collect all the tweets containing it

Unicorn

t_{-1} ..... Tweets

t_0 ..... salt

$t_0+\varepsilon$ ..... sloth → publish the tweets, and a commitment to the salt

↓ ≥ 10 minutes

t_1 ..... the unicorn (uncontestable random number)

Also publish the salt and the sloth witness

Unicorn

How to check the validity of the output?

- Check that your tweet appears in the input of `sloth`
- Run the fast `sloth` verification

Unicorn

Why is it secure?

Unicorn

Why is it secure?

- cheating = targeting a property for the output

Unicorn

Why is it secure?

- cheating = targeting a property for the output
- no information is known about the output until `sloth` is completed

Unicorn

Why is it secure?

- cheating = targeting a property for the output
- no information is known about the output until `sloth` is completed
- `sloth` cannot be started before the last honest contribution to the input (tweets)

Unicorn

Why is it secure?

- cheating = targeting a property for the output
- no information is known about the output until `sloth` is completed
- `sloth` cannot be started before the last honest contribution to the input (tweets)
- `sloth` takes, say, more than 10 minutes to complete

Unicorn

Why is it secure?

- cheating = targeting a property for the output
- no information is known about the output until `sloth` is completed
- `sloth` cannot be started before the last honest contribution to the input (tweets)
- `sloth` takes, say, more than 10 minutes to complete
- so the input of `sloth` is committed to before anything is known by anyone about the output

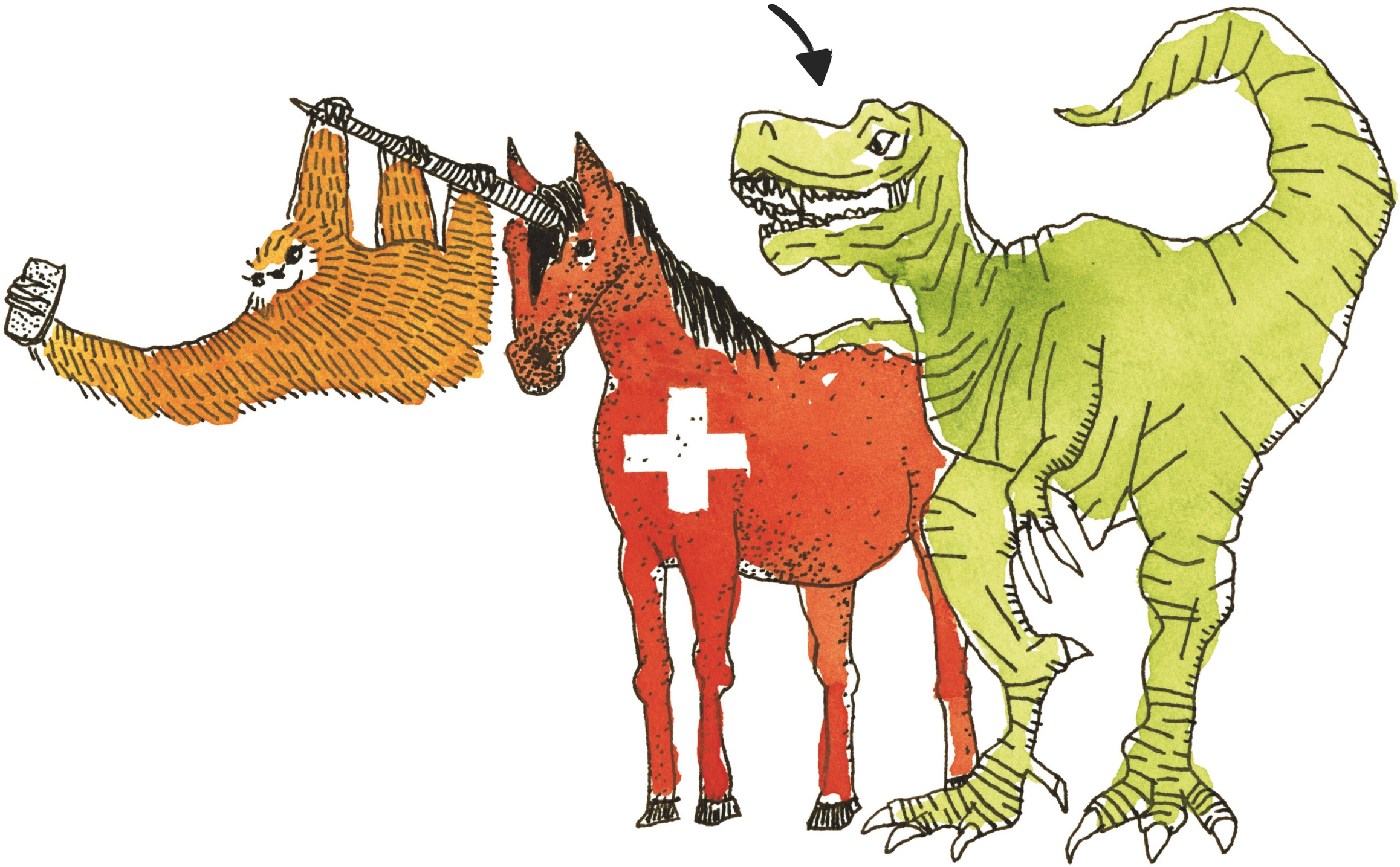
Unicorn

Why is it secure?

- cheating = targeting a property for the output
- no information is known about the output until `sloth` is completed
- `sloth` cannot be started before the last honest contribution to the input (tweets)
- `sloth` takes, say, more than 10 minutes to complete
- so the input of `sloth` is committed to before anything is known by anyone about the output

Security proof in the paper, under a formalisation of the slowness of square root extraction

meet trx



(trustworthy random elliptic curve service)

Trx

Trx

- an online service

Trx

- an online service
- every day a set of parameters is announced (e.g., security level)

Trx

- an online service
- every day a set of parameters is announced (e.g., security level)
- and a fresh unicorn is fed to the trx

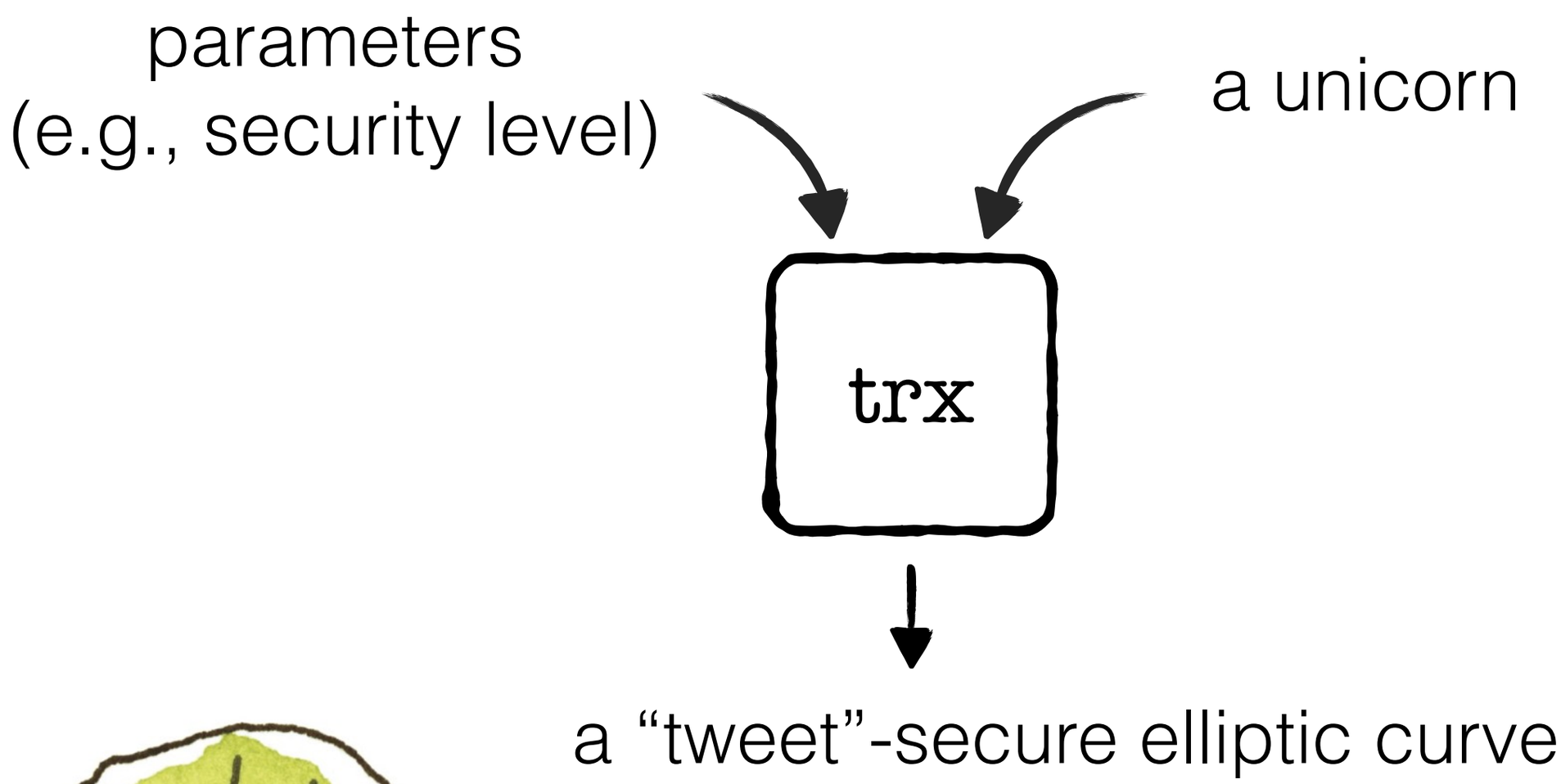
Trx

- an online service
- every day a set of parameters is announced (e.g., security level)
- and a fresh unicorn is fed to the trx
- a public, deterministic procedure derives an elliptic curve from the unicorn, satisfying the security parameters

Trx

- an online service
- every day a set of parameters is announced (e.g., security level)
- and a fresh unicorn is fed to the trx
- a public, deterministic procedure derives an elliptic curve from the unicorn, satisfying the security parameters
    - the elliptic curve, and the data for its verification, are published

Trx


parameters
(e.g., security level)
a unicorn
trx
a “tweet”-secure elliptic curve

A random zoo:

sloth, unicorn, and trx

art by Léa Kissling

Thank you!