Revoke and Let Live

A Secure Key Revocation API for Cryptographic Devices

Véronique Cortier
LORIA-CNRS, Nancy (FR)

Graham Steel
INRIA, Paris (FR)

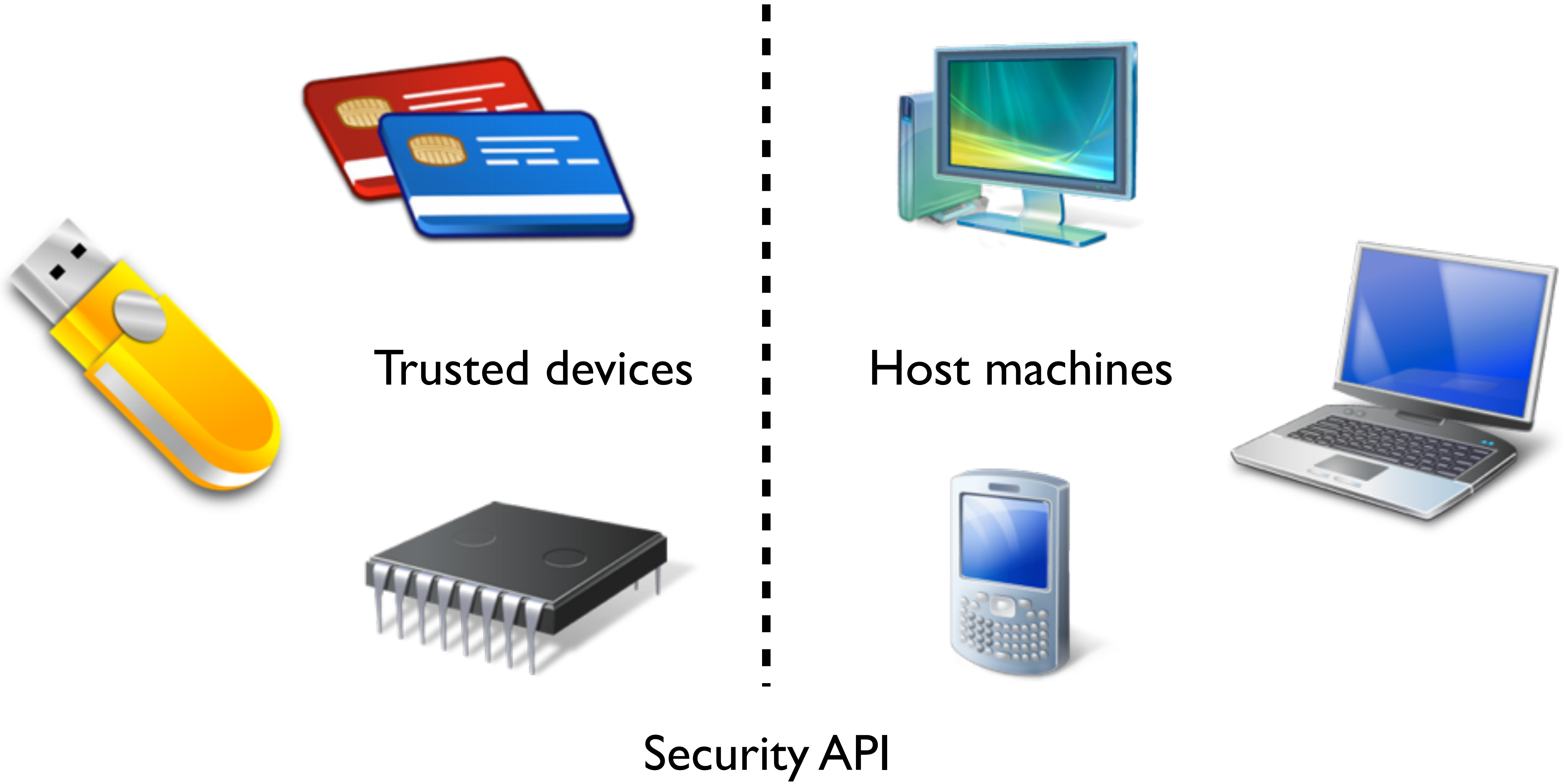
Cyrille Wiedling
LORIA-CNRS, Nancy (FR)

Séminaire Méthodes Formelles et Sécurité

Rennes, May 24th, 2013

Funded by

Security APIs



Trusted devices

Host machines

Security API

Goal : Enforce security of data stored inside the trusted device, even when connected to untrusted host machines.

Applications

- Smartphones,

- Online Banking, Asynchronous Transfer Mode,

- Electronic Ticketing Systems,

- Vehicle-to-vehicle networking.

- ...

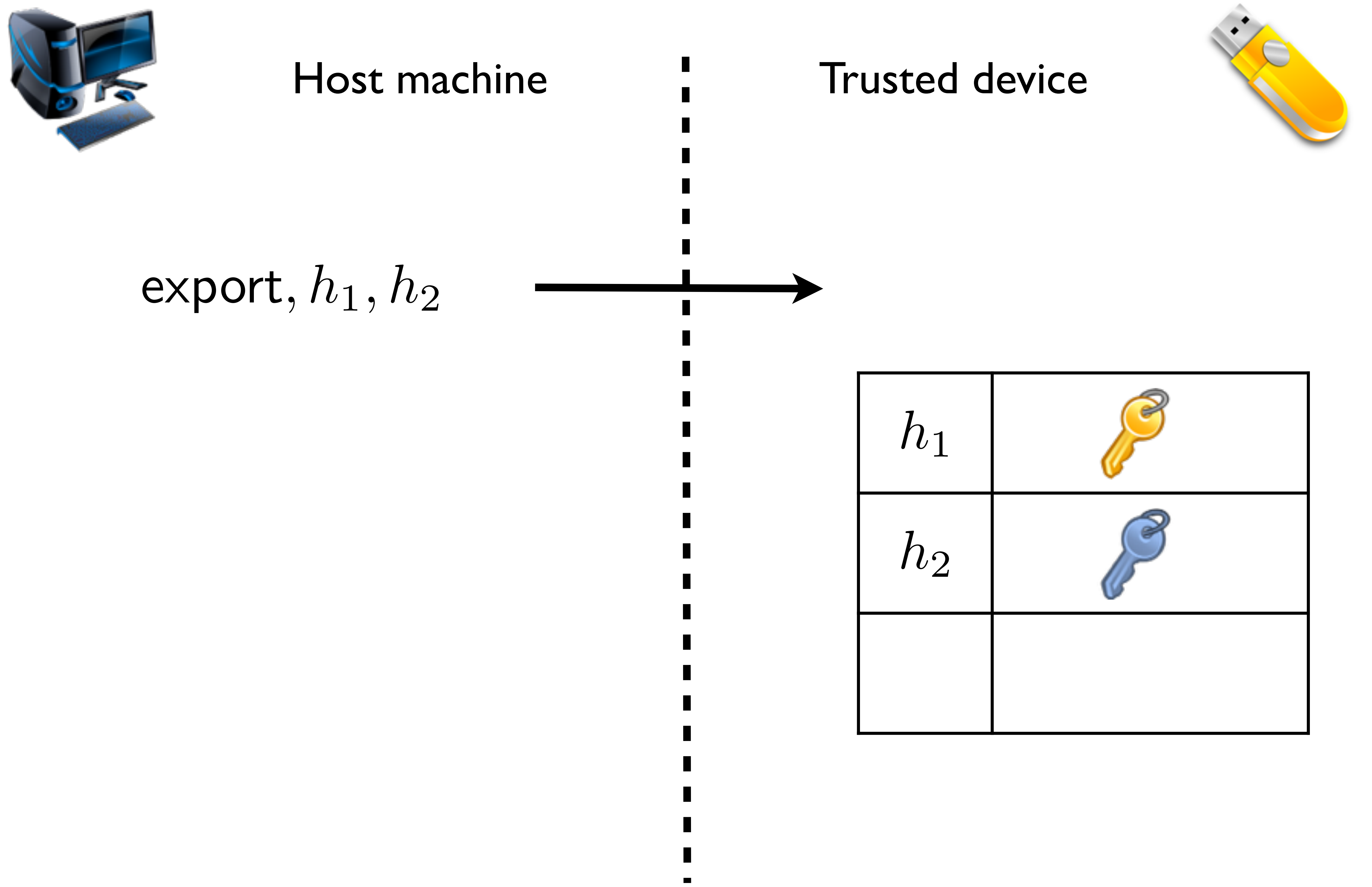
How does it work ?

Host machine

Trusted device

h_1	🔑
h_2	🔑

How does it work ?



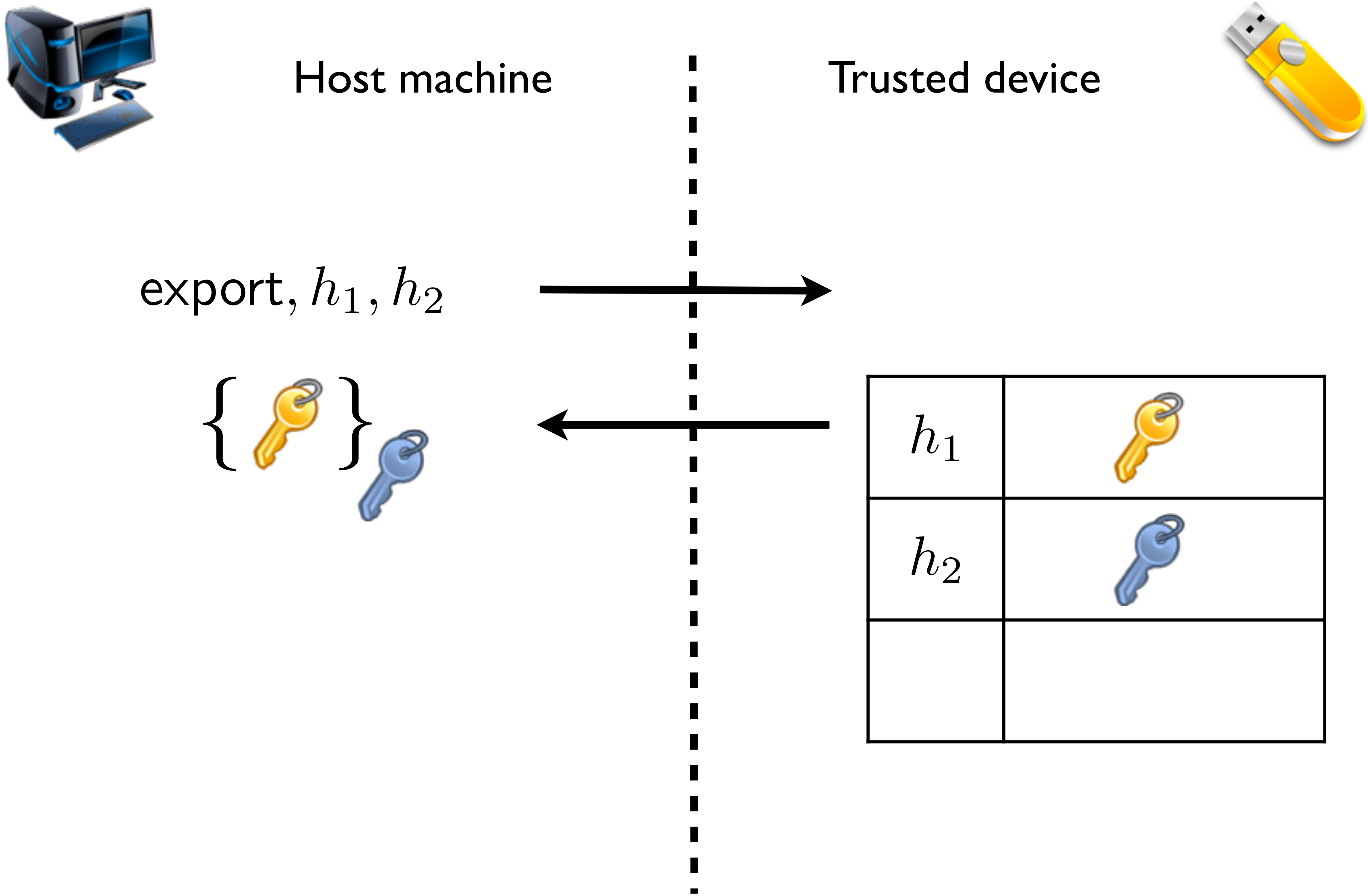
Host machine

Trusted device

export, h_1, h_2

h_1	🔑
h_2	🔑

How does it work ?



Host machine

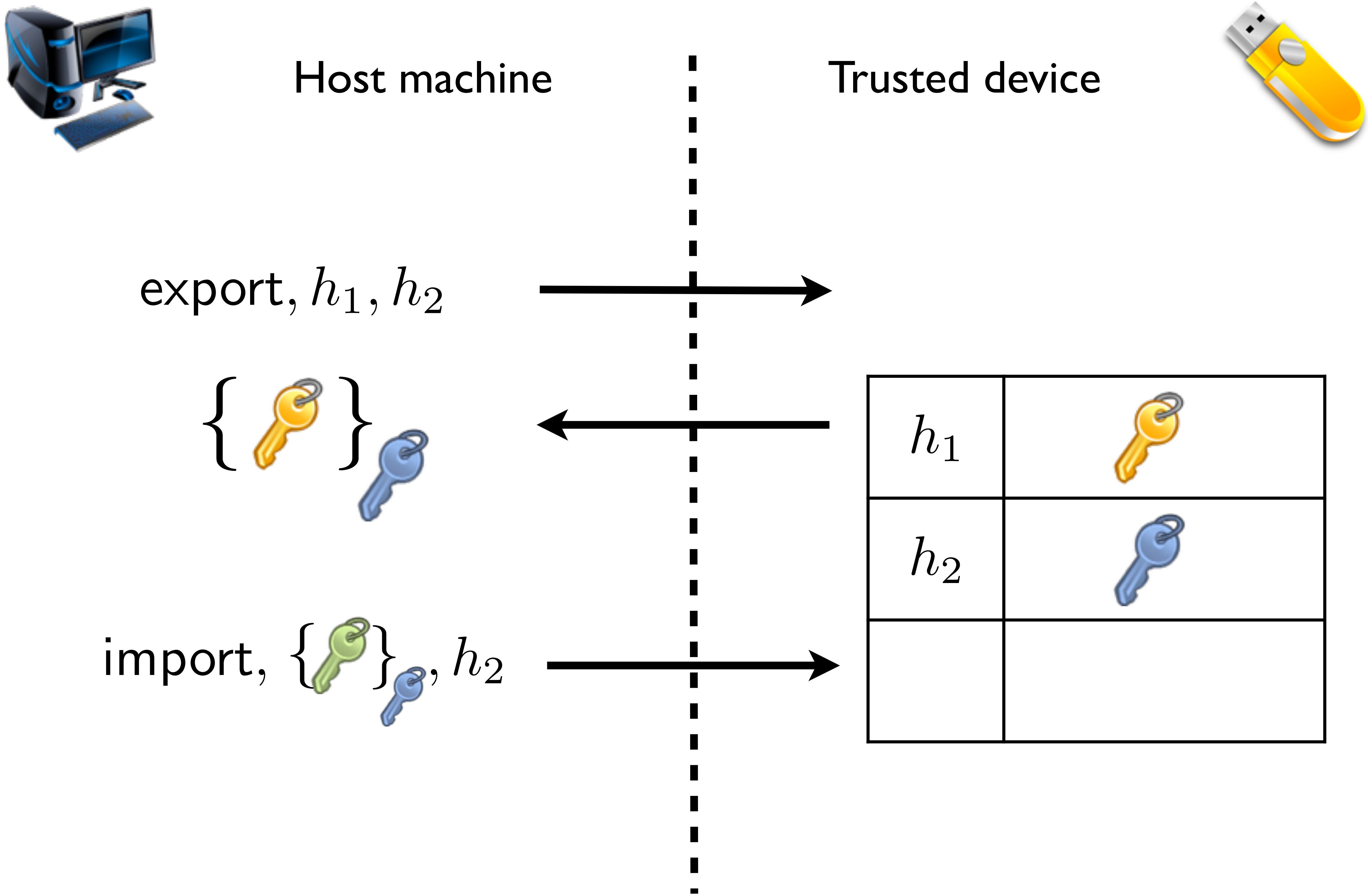
Trusted device

export, h_1, h_2

$\{$🔑$\}$🔑

h_1	🔑
h_2	🔑

How does it work ?



Host machine

Trusted device

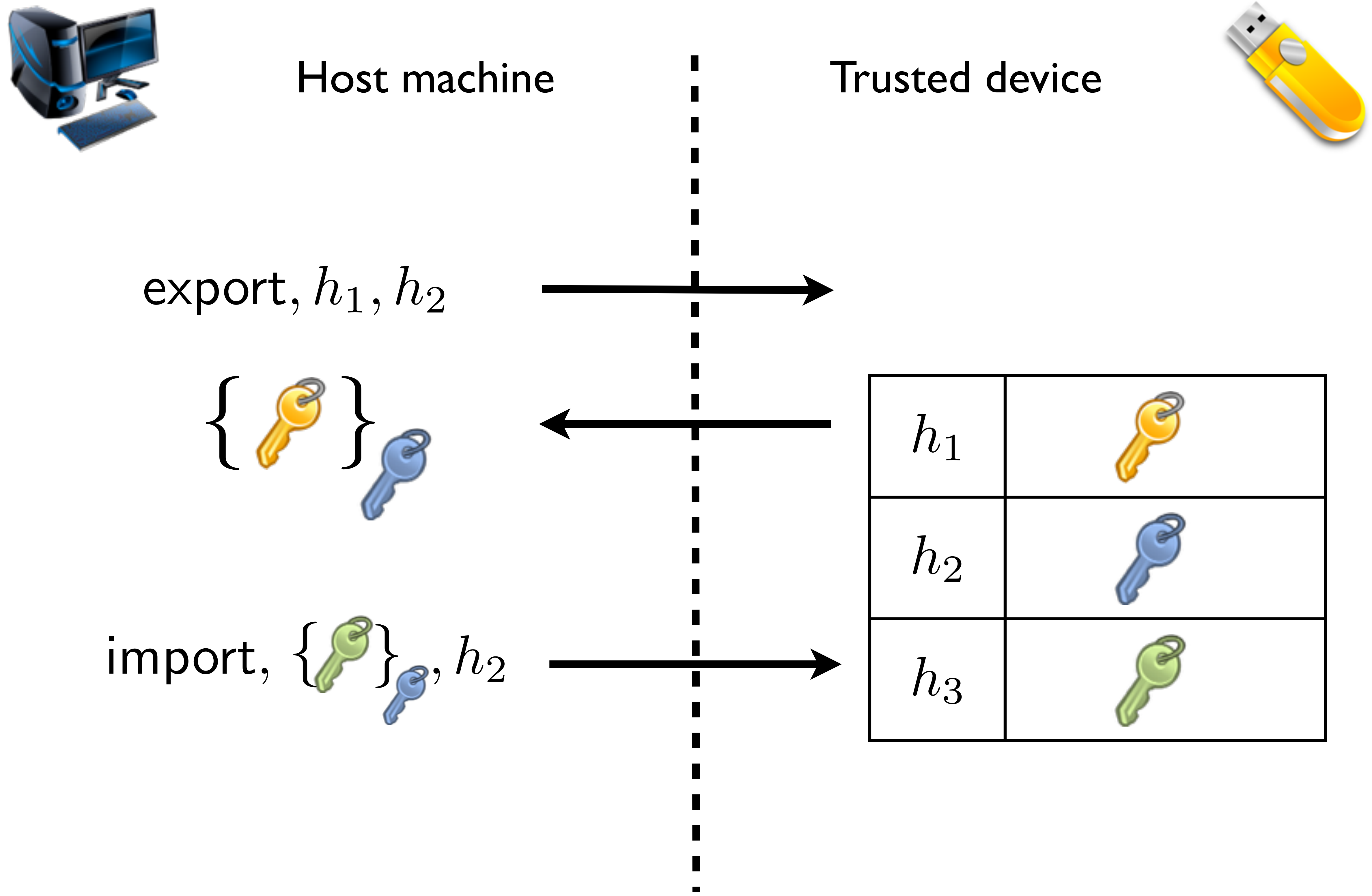
export, h_1, h_2

import, $\{\ \}$, h_2

h_1

h_2

How does it work ?



Host machine

Trusted device

export, h_1, h_2

$\{\ \}$

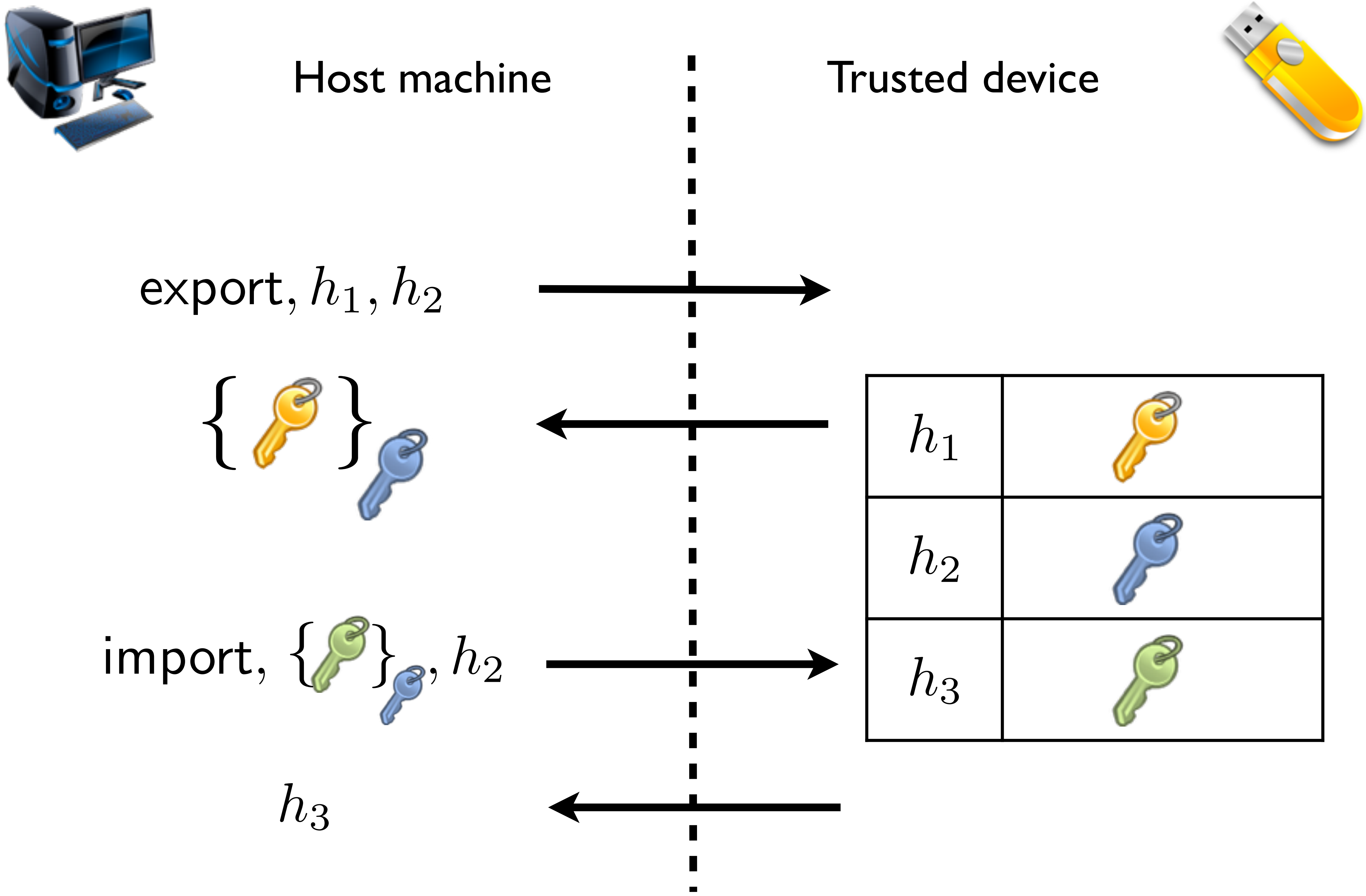
import, $\{\ \}$, h_2

h_1

h_2

h_3

How does it work ?



Host machine

Trusted device

export, h_1, h_2

import, h_2

h_3

h_1	🔑
h_2	🔑
h_3	🔑

Related Work

Many flaws found on **PKCS #11** security tokens.

M. Bortolozzo, M. Centenaro,
R. Focardi and G. Steel, CSF'10.

Related Work

Many flaws found on **PKCS #11** security tokens.

M. Bortolozzo, M. Centenaro,
R. Focardi and G. Steel, CSF'10.

TOOKAN

Proposals for key management APIs with security proofs.

J. Courant, J.-F. Monin, WITS'06.

C. Cachin, N. Chandran, CSF'09.

V. Cortier, G. Steel, ESORICS'09...

Related Work

Many flaws found on **PKCS #11** security tokens.

M. Bortolozzo, M. Centenaro,
R. Focardi and G. Steel, CSF'10.

Proposals for key management APIs with security proofs.

J. Courant, J.-F. Monin, WITS'06.

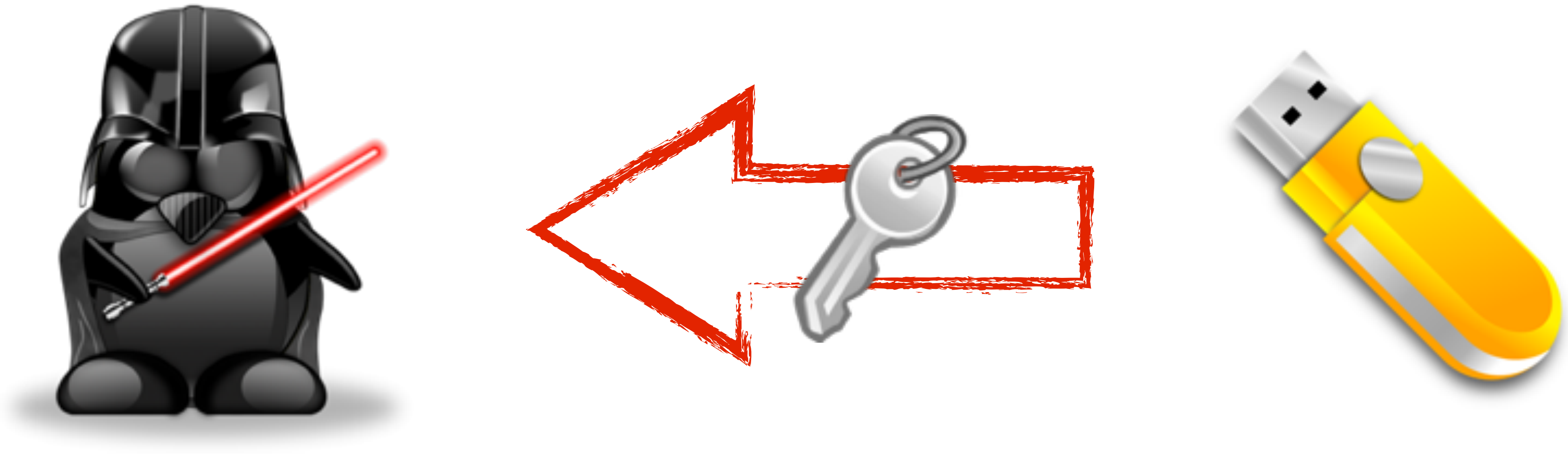
C. Cachin, N. Chandran, CSF'09.

V. Cortier, G. Steel, ESORICS'09...

Use of long-term keys implying
unrecoverable loss of devices if keys are lost

Breaking Keys in a TRD

«Because I’m bad, really really bad !»



There are ways for the attacker to **break some keys** of a Tamper-Resistant Device (TRD):

- Bruteforcing,
- Side-channel attack,
- ...

(More) Related Work

Proposals for key management APIs **with revocation**:

L. Eschenauer, V. D. Gligor, CCS'02.

(Using a control server)

X. Z. Yong Wan, B. Ramamurthy, ICC'07.

(Secret sharing scheme)

(More) Related Work

Proposals for key management APIs **with revocation**:

L. Eschenauer, V. D. Gligor, CCS’02.

(Using a control server)

X. Z. Yong Wan, B. Ramamurthy, ICC’07.

(Secret sharing scheme)

⚠ **Still use long-term keys !**

(More) Related Work

Proposals for key management APIs **with revocation**:

L. Eschenauer, V. D. Gligor, CCS'02.

(Using a control server)

X. Z. Yong Wan, B. Ramamurthy, ICC'07.

(Secret sharing scheme)

⚠ **Still use long-term keys !**

F. E. Kargl, Sevecom, 2009...

(Two root keys)

(More) Related Work

Proposals for key management APIs **with revocation**:

L. Eschenauer, V. D. Gligor, CCS'02.

(Using a control server)

X. Z. Yong Wan, B. Ramamurthy, ICC'07.

(Secret sharing scheme)

⚠ **Still use long-term keys !**

F. E. Kargl, Sevecom, 2009...

(Two root keys)

⚠ **Attacked by S. Möderschein & P. Modesti**

(solution proposed but no security proof)

Ideal Key Revocation API

Keys must remain **confidential**:

Information about key should not be recovered by the intruder.

Ideal Key Revocation API

Keys must remain **confidential**:

Information about key should not be recovered by the intruder.

Any key should be **revocable**:

The more sensitive a key is, the more an attacker will try to break it.

Ideal Key Revocation API

Keys must remain **confidential**:

Information about key should not be recovered by the intruder.

Any key should be **revocable**:

The more sensitive a key is, the more an attacker will try to break it.

The device should remain **functional**:

A revocation of a key should not prevent the user from using his/her device.

Our Contributions

- **Design** of an API satisfying previous properties with :
  - **update** functionality,
  - **revocation** functionality.

- A **formal proof of security** ensuring three properties :
  - A **key remains secret** unless it is broken (brute forced),
  - the system is able to **recover itself** from an attack,
  - a **revocation immediately secures** the device.

Description of the API

Some **assumptions** on the tamper-resistant devices:

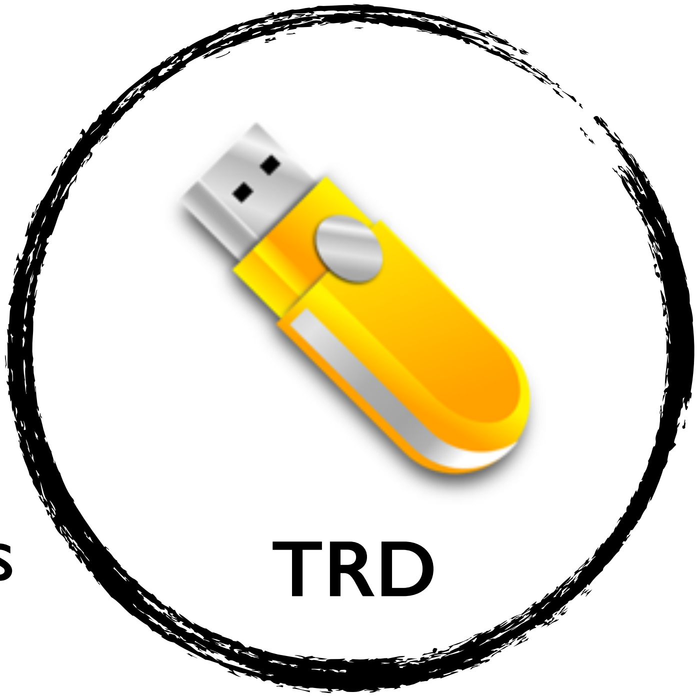
Description of the API

Some **assumptions** on the tamper-resistant devices:

A **clock** assumed synchronized with a global clock

Description of the API

Some **assumptions** on the tamper-resistant devices:



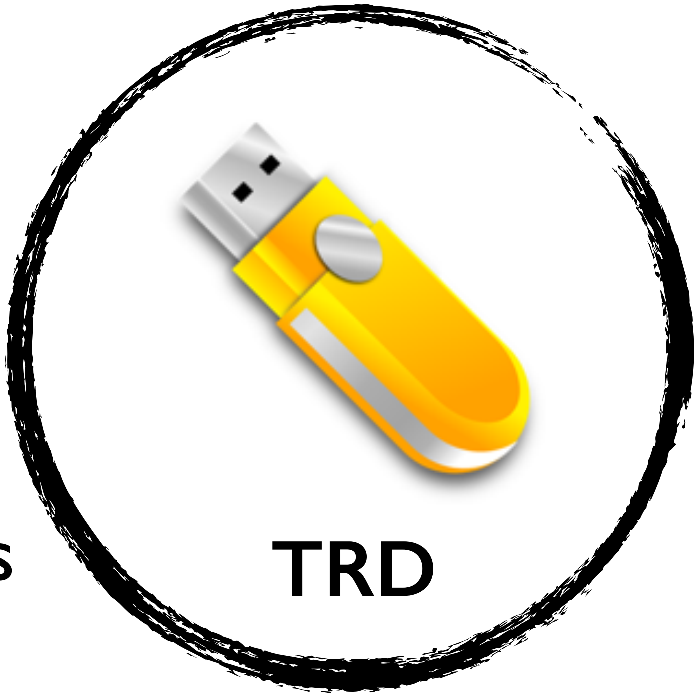
A **table** indexed by handles to store keys' information (level, validity date, value, ...)

A **clock** assumed synchronized with a global clock

Description of the API

Some **assumptions** on the tamper-resistant devices:

A **table** indexed by handles to store keys' information (level, validity date, value, ...)



TRD

A **blacklist** of elements of the form (l, t)

A **clock** assumed synchronized with a global clock

Description of the API

We also assume a **hierarchy of levels** for keys:

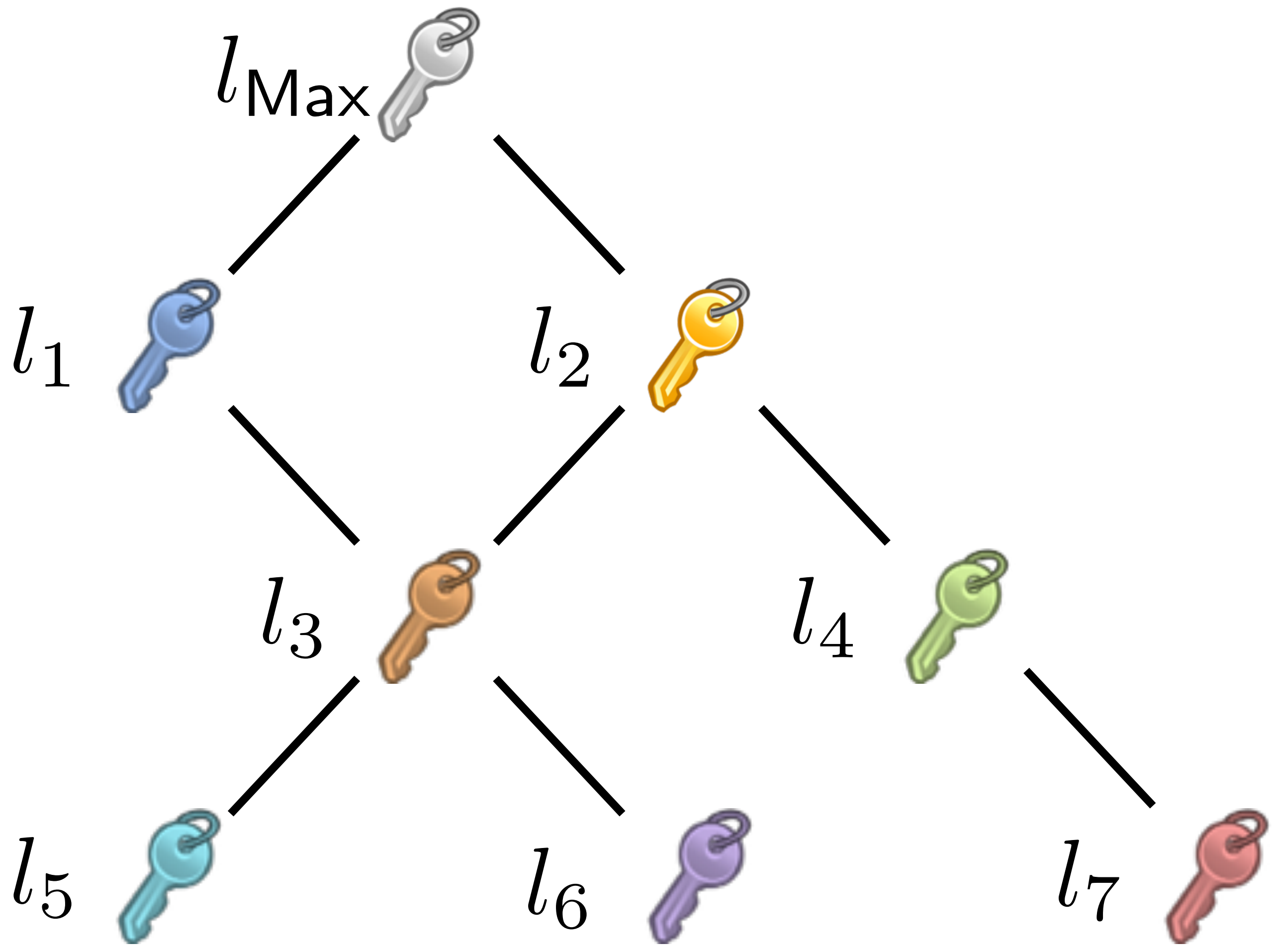
- with a (partial) order,
- with a maximal and a minimal element.

Description of the API

We also assume a **hierarchy of levels** for keys:

- with a (partial) order,
- with a maximal and a minimal element.

Example:

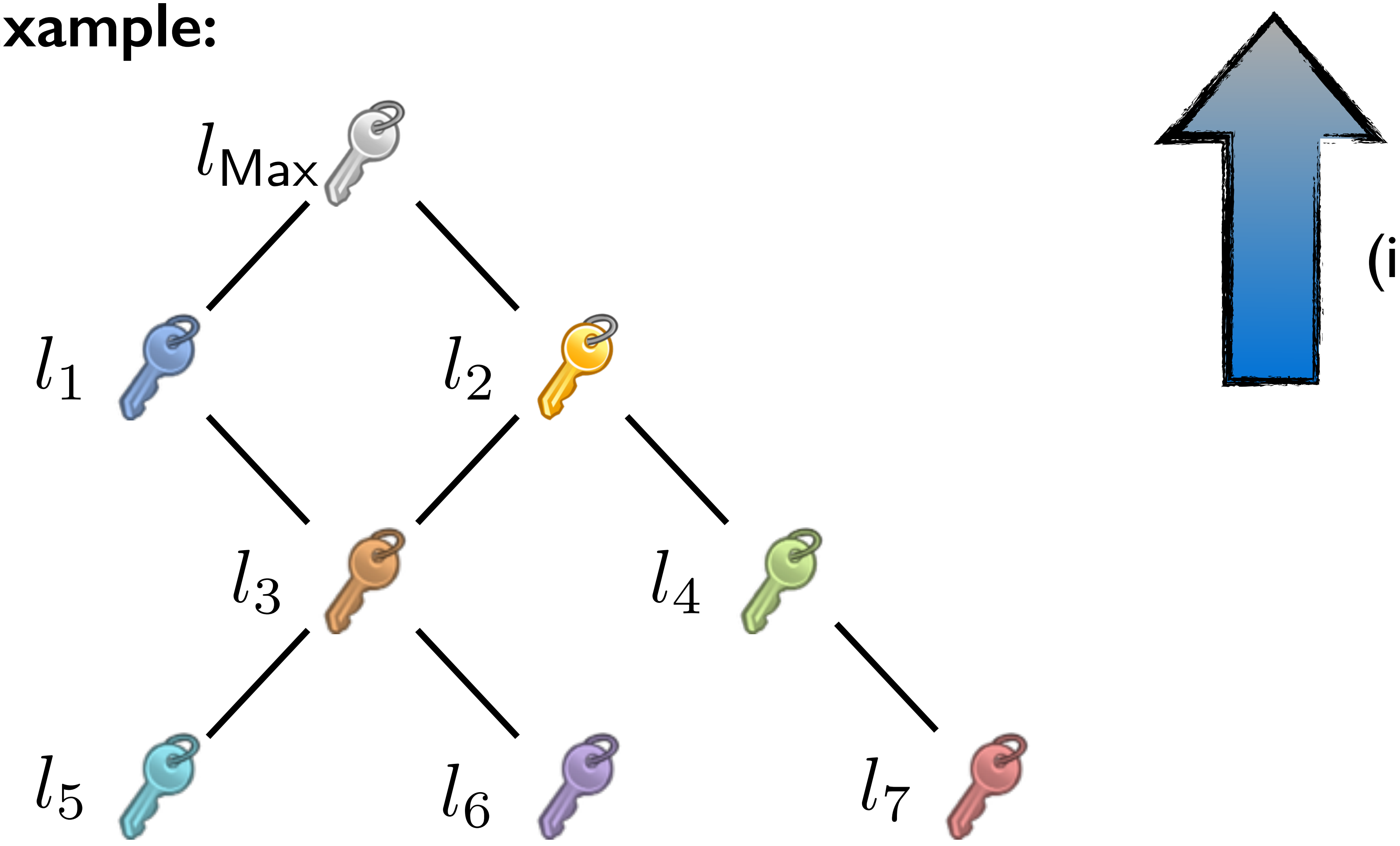


Description of the API

We also assume a **hierarchy of levels** for keys:

- with a (partial) order,
- with a maximal and a minimal element.

Example:

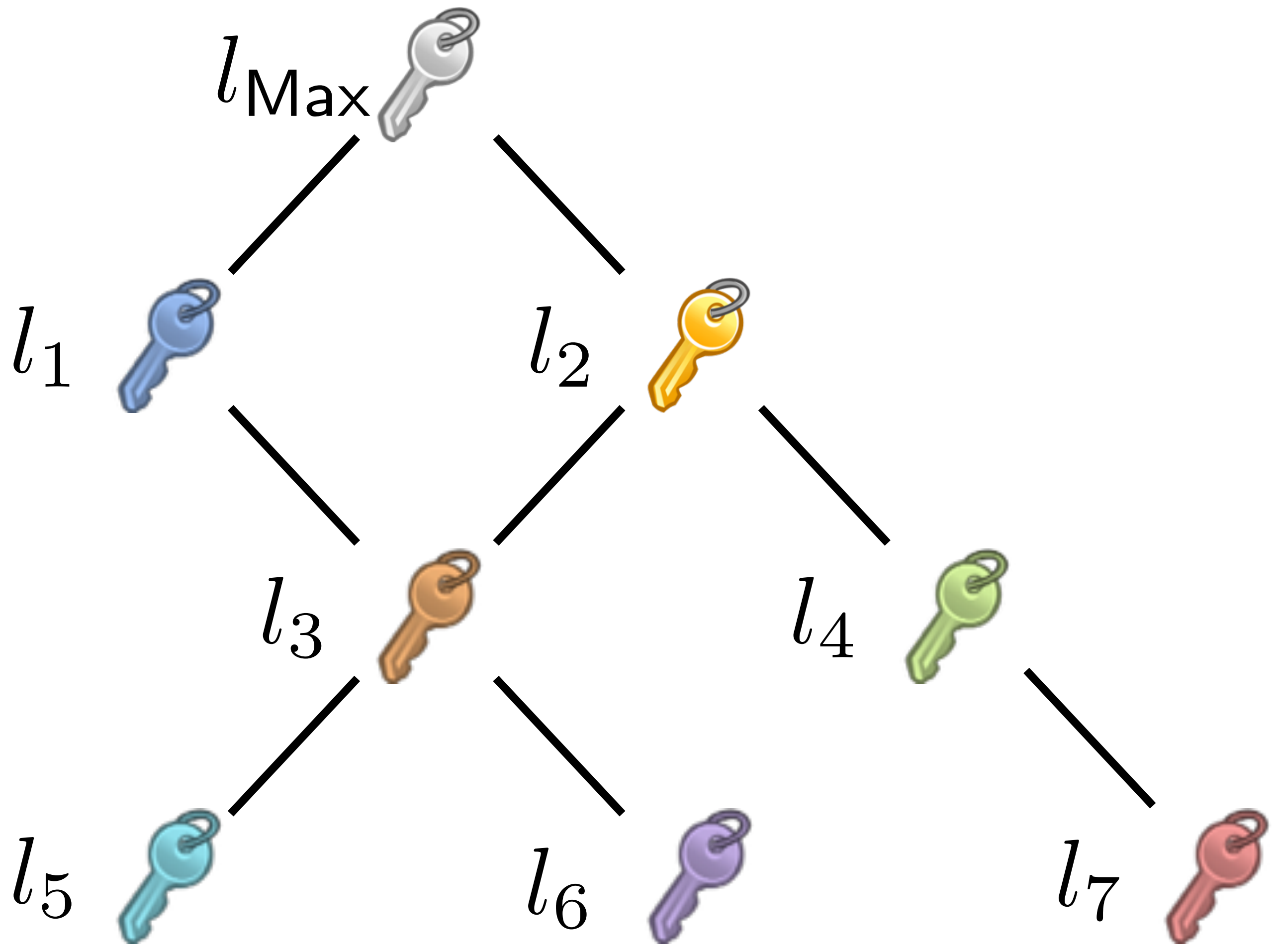


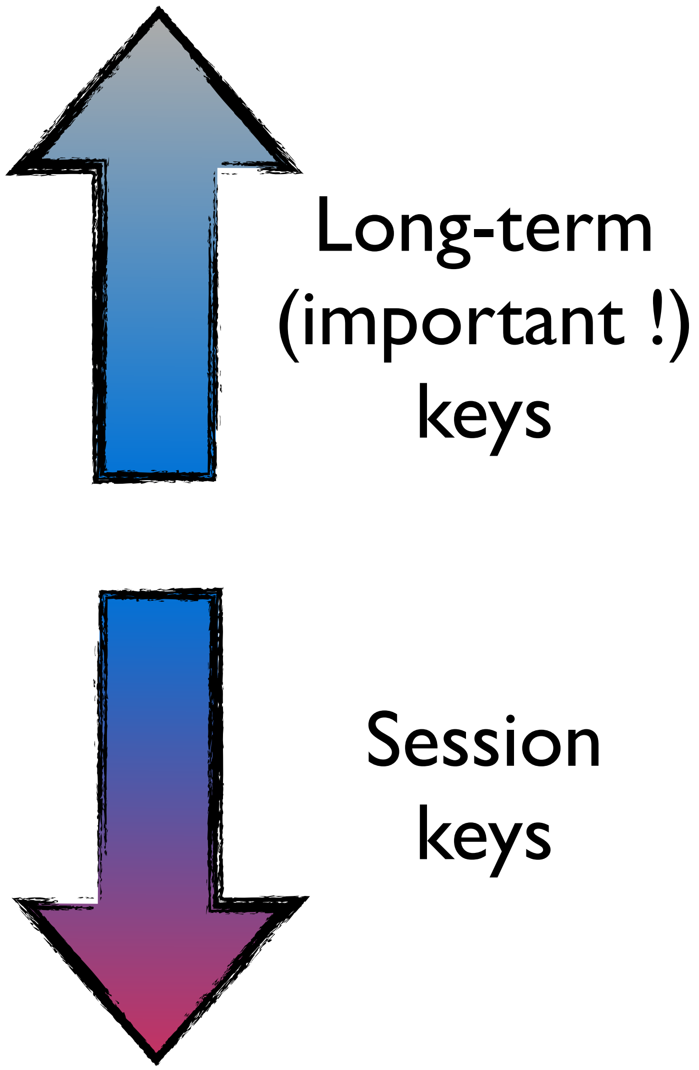
Description of the API

We also assume a **hierarchy of levels** for keys:

- with a (partial) order,
- with a maximal and a minimal element.

Example:





User’s Commands

We have a set of **basic commands**.

User’s Commands

We have a set of **basic commands**.

Running example:

Alice

h_1	🔑$, l, v, m$
h_2	
h_3	

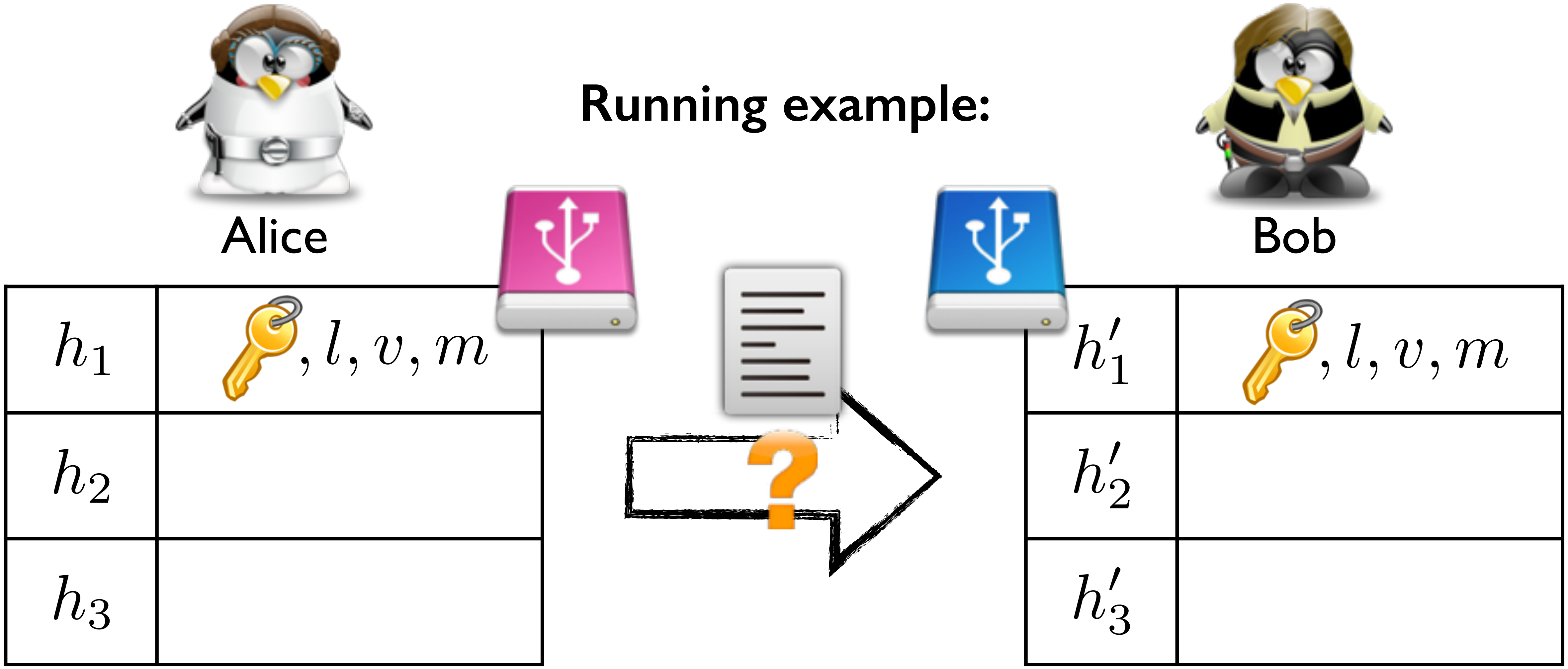
Bob

h'_1	🔑$, l, v, m$
h'_2	
h'_3	

User’s Commands

We have a set of **basic commands**.

Running example:



h_1	🔑$, l, v, m$
h_2	
h_3	

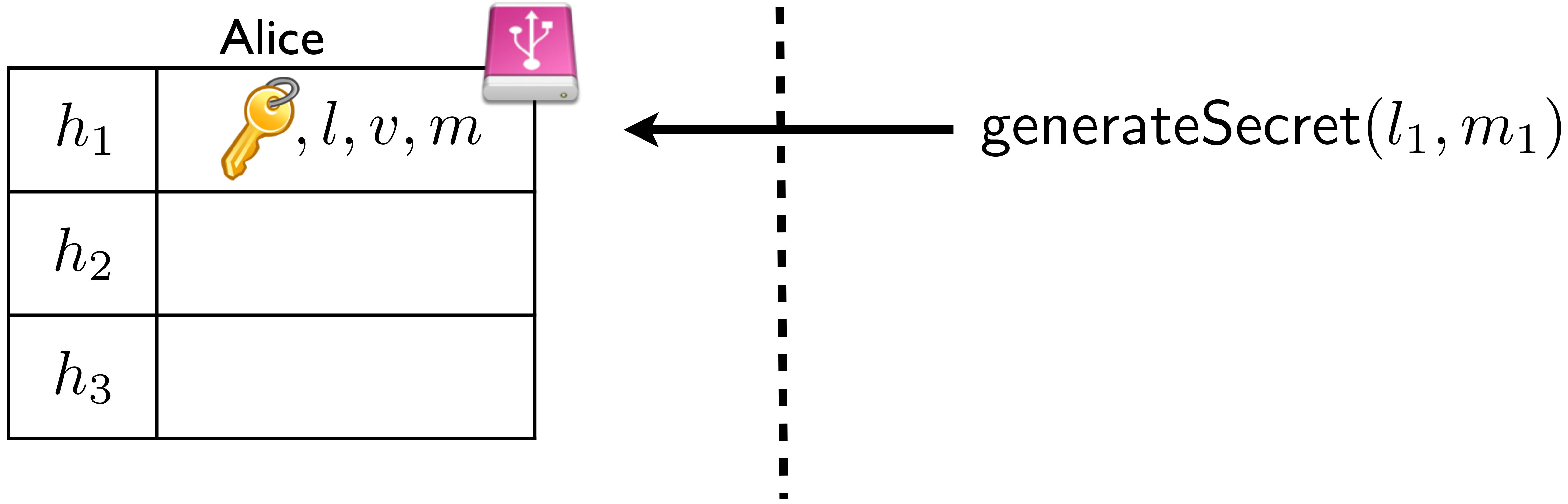
h'_1	🔑$, l, v, m$
h'_2	
h'_3	

Alice and Bob share a key and wish to securely exchange a message.

User’s Commands

	Alice
h_1	🔑$, l, v, m$
h_2	
h_3	

User's Commands

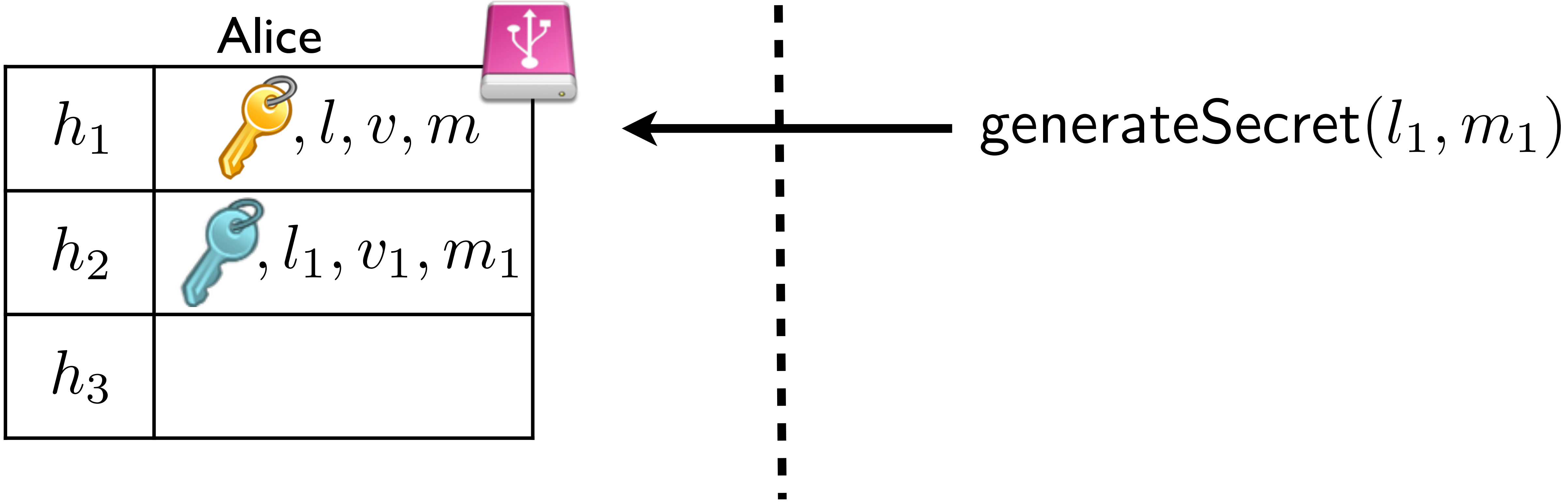


Alice

h_1	🔑$, l, v, m$
h_2	
h_3	

generateSecret(l_1, m_1)

User's Commands

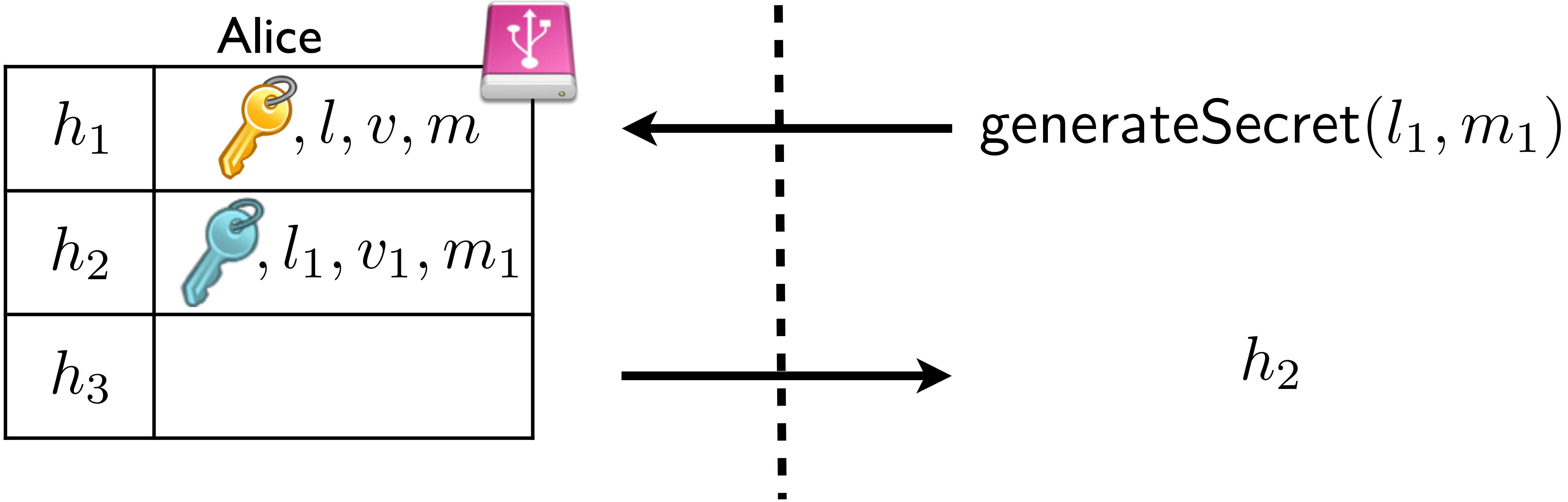


Alice

h_1	🔑, l, v, m
h_2	🔑, l_1, v_1, m_1
h_3	

generateSecret(l_1, m_1)

User's Commands



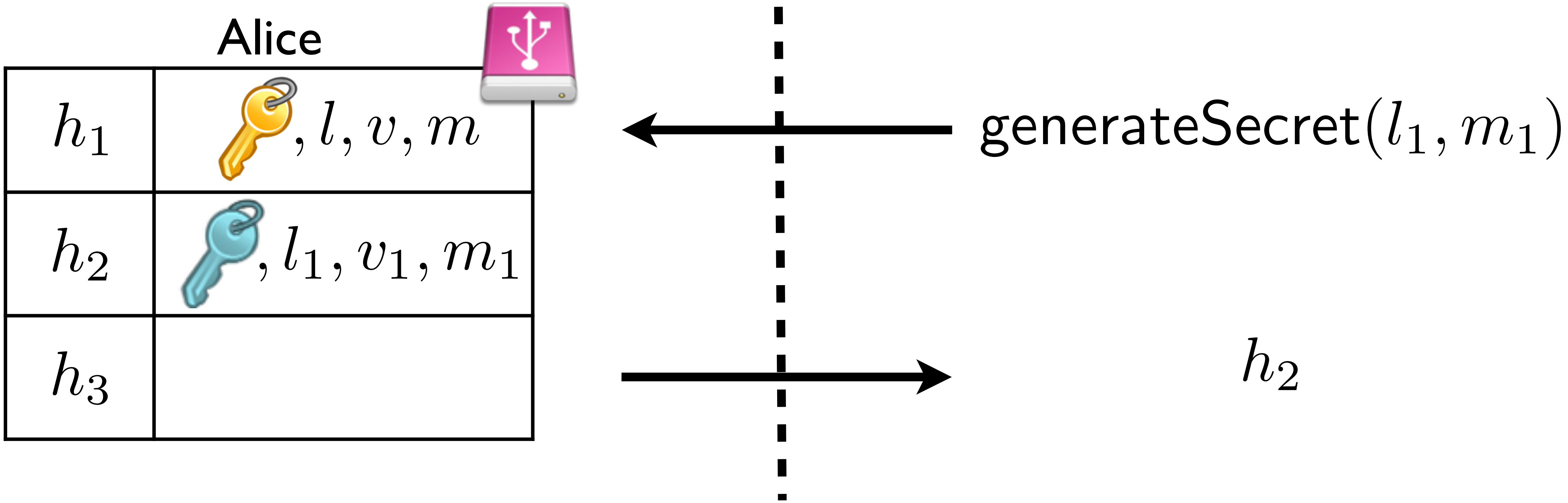
Alice

h_1	🔑$, l, v, m$
h_2	🔑$, l_1, v_1, m_1$
h_3	

generateSecret(l_1, m_1)

h_2

User’s Commands



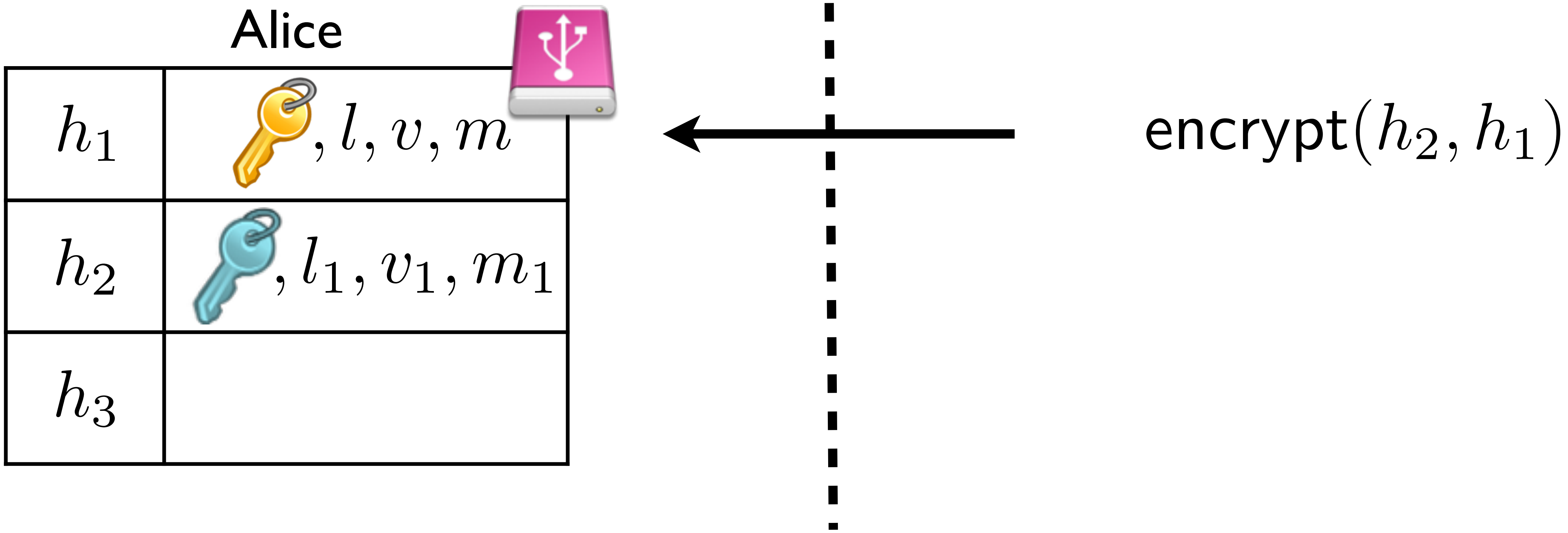
Alice

h_1	🔑$, l, v, m$
h_2	🔑$, l_1, v_1, m_1$
h_3	

generateSecret(l_1, m_1)

h_2

To share the new session key with Bob, Alice needs to « export » the new key.

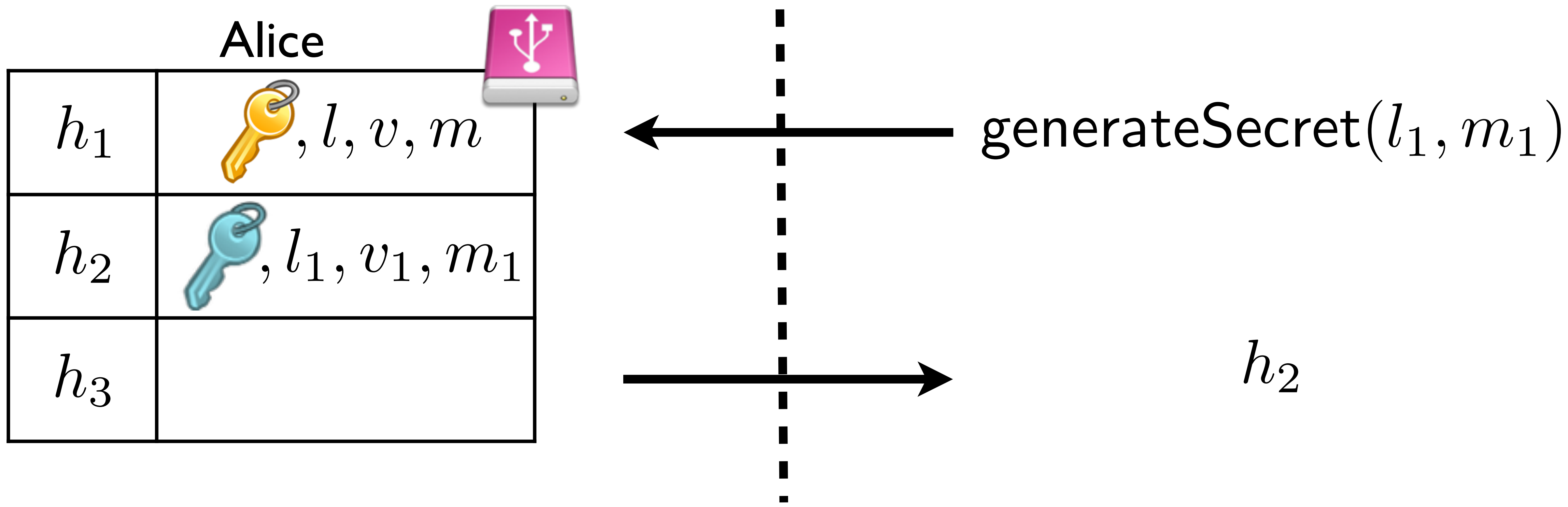


Alice

h_1	🔑$, l, v, m$
h_2	🔑$, l_1, v_1, m_1$
h_3	

encrypt(h_2, h_1)

User's Commands



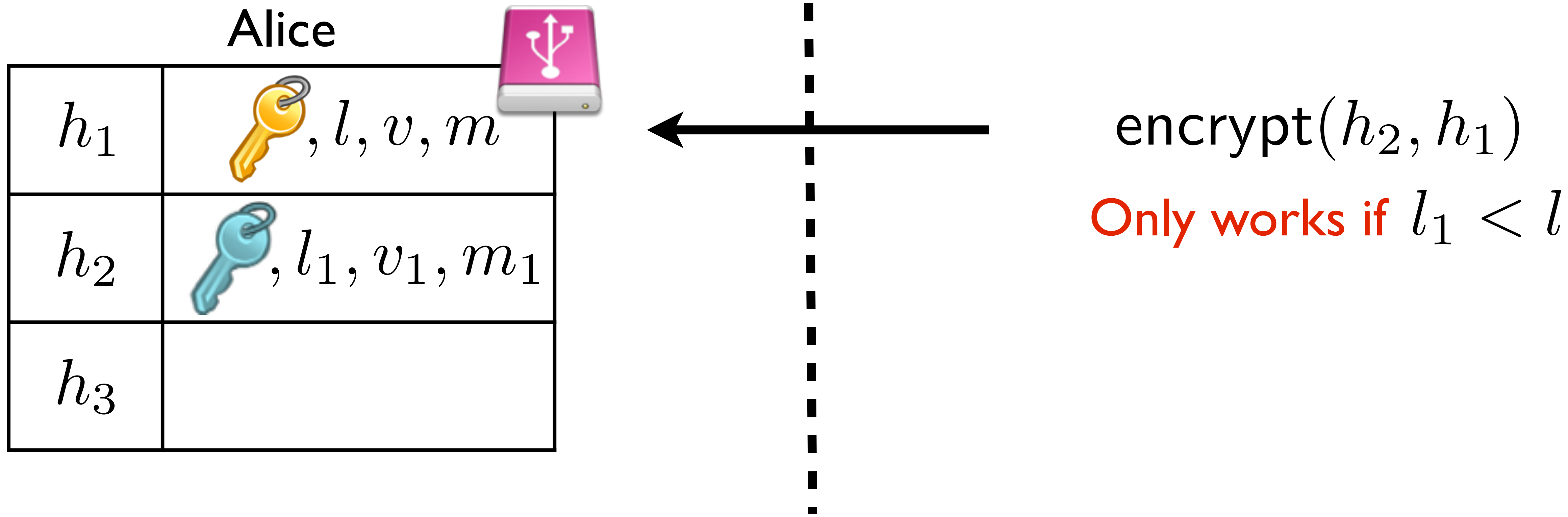
Alice

h_1	🔑$, l, v, m$
h_2	🔑$, l_1, v_1, m_1$
h_3	

generateSecret(l_1, m_1)

h_2

To share the new session key with Bob, Alice needs to « export » the new key.



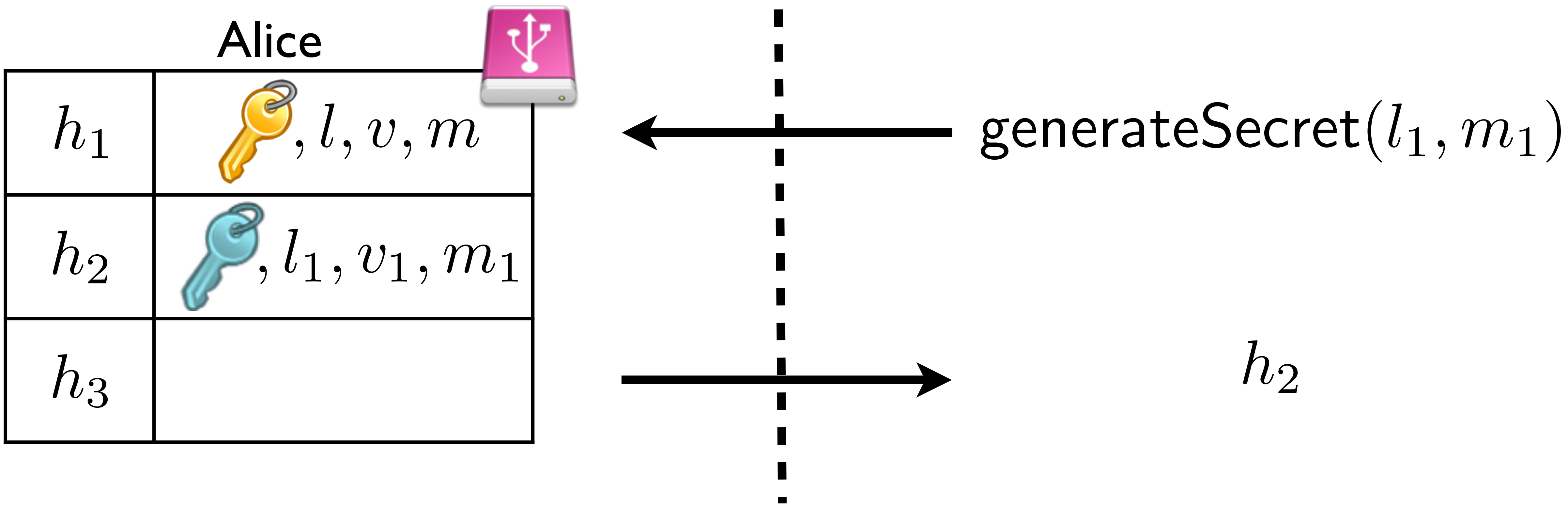
Alice

h_1	🔑$, l, v, m$
h_2	🔑$, l_1, v_1, m_1$
h_3	

encrypt(h_2, h_1)

Only works if $l_1 < l$

User's Commands



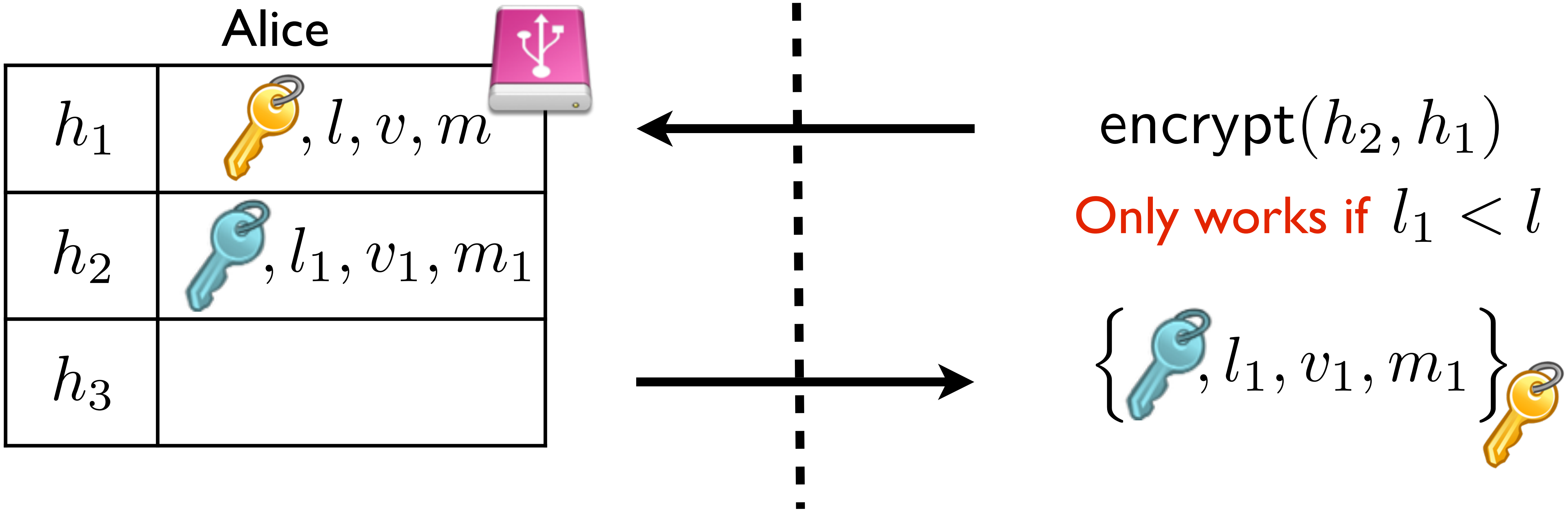
Alice

h_1	🔑, l, v, m
h_2	🔑, l_1, v_1, m_1
h_3	

generateSecret(l_1, m_1)

h_2

To share the new session key with Bob, Alice needs to « export » the new key.



Alice

h_1	🔑, l, v, m
h_2	🔑, l_1, v_1, m_1
h_3	

encrypt(h_2, h_1)

Only works if $l_1 < l$

{🔑, l_1, v_1, m_1}🔑

User’s Commands

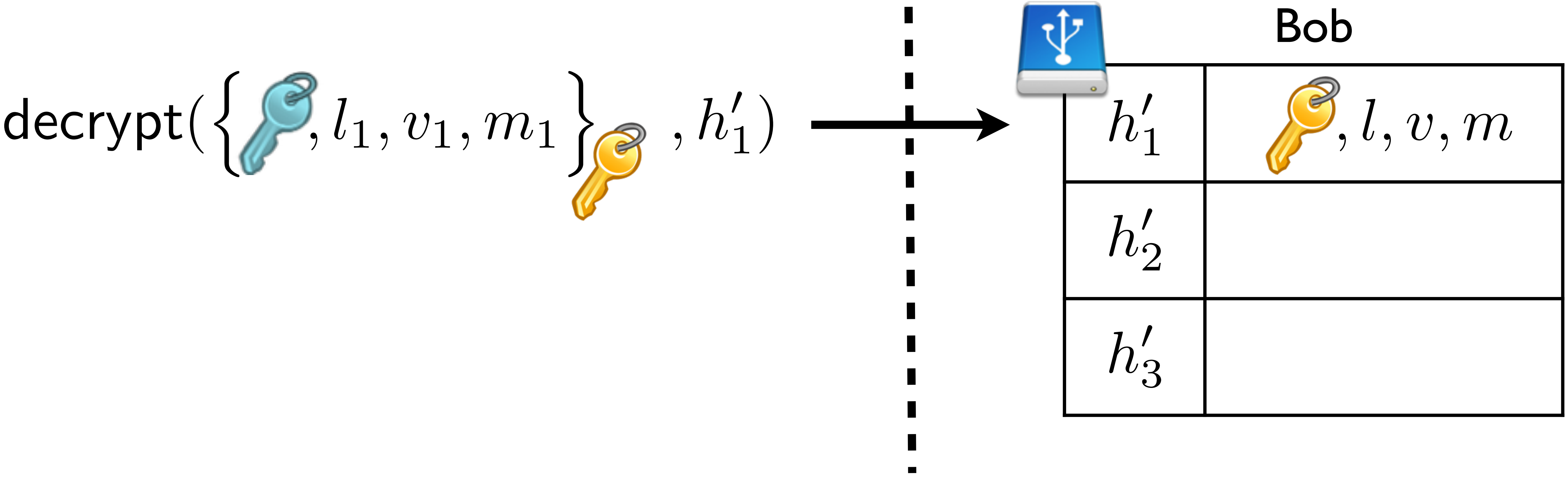
Alice sends the new session key to Bob which can « import » it in his TRD.

Bob

h'_1	🔑$, l, v, m$
h'_2	
h'_3	

User’s Commands

Alice sends the new session key to Bob which can « import » it in his TRD.



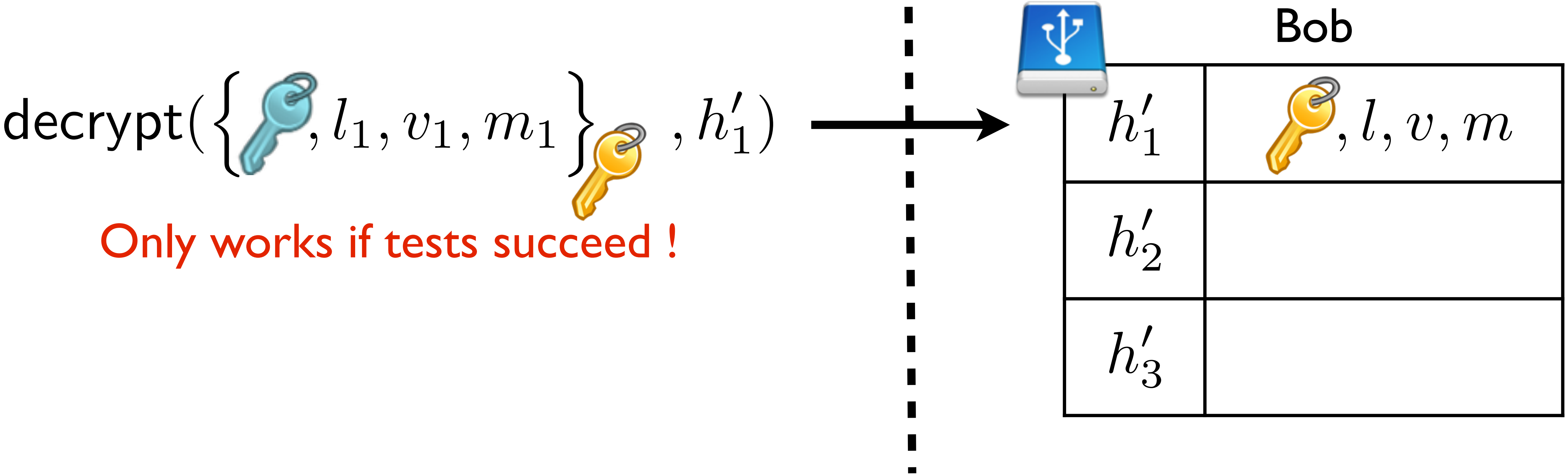
decrypt($\{$🔑$, l_1, v_1, m_1\}_{🔑}$ $, h'_1$)

Bob

h'_1	🔑$, l, v, m$
h'_2	
h'_3	

User’s Commands

Alice sends the new session key to Bob which can « import » it in his TRD.



decrypt($\{$🔑$, l_1, v_1, m_1\}_{🔑}, h'_1)$

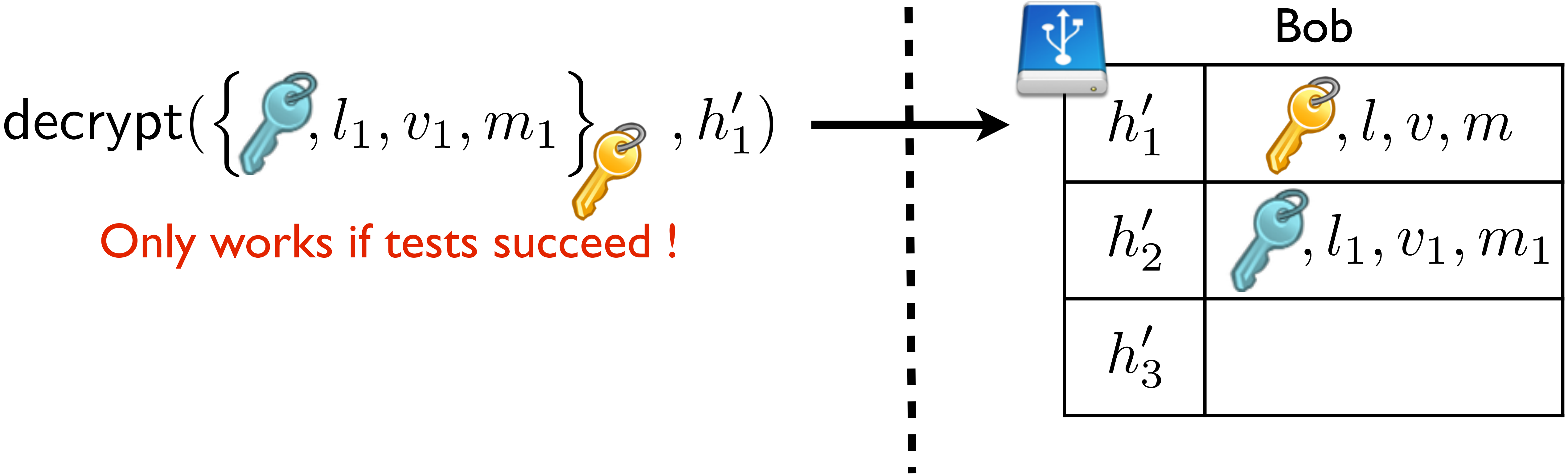
Only works if tests succeed !

Bob

h'_1	🔑$, l, v, m$
h'_2	
h'_3	

User's Commands

Alice sends the new session key to Bob which can « import » it in his TRD.



decrypt({🔑, l_1, v_1, m_1}🔑, h'_1)

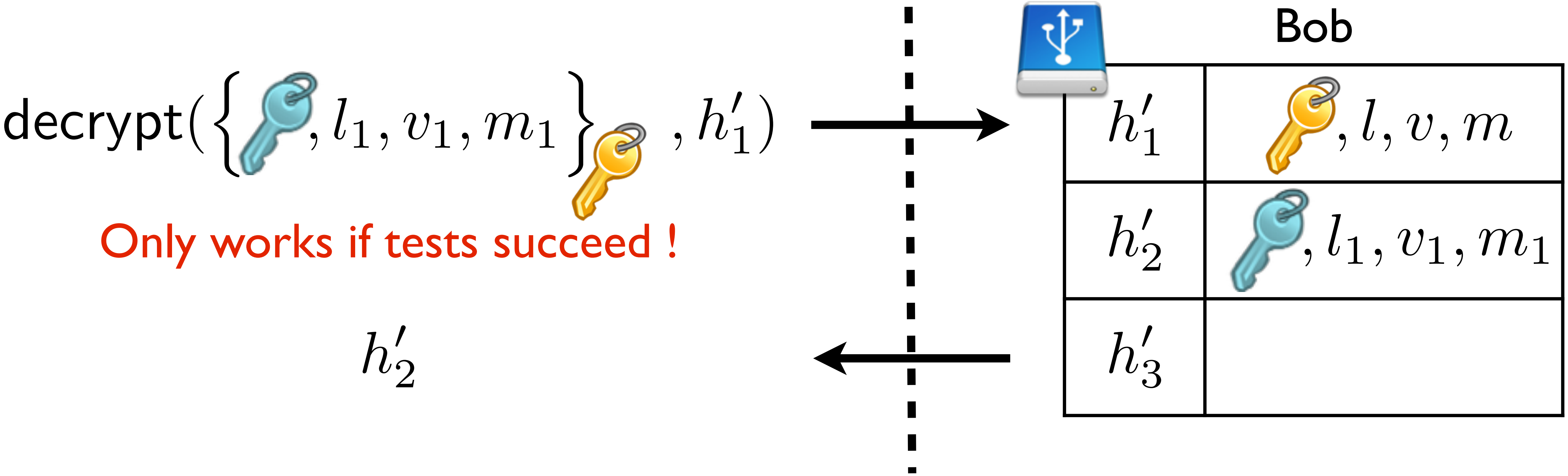
Only works if tests succeed !

Bob

h'_1	🔑, l, v, m
h'_2	🔑, l_1, v_1, m_1
h'_3	

User’s Commands

Alice sends the new session key to Bob which can « import » it in his TRD.



decrypt({🔑, $l_1, v_1, m_1\}_{🔑}, h'_1)$

Only works if tests succeed !

h'_2

Bob

h'_1	🔑, l, v, m
h'_2	🔑, l_1, v_1, m_1
h'_3	

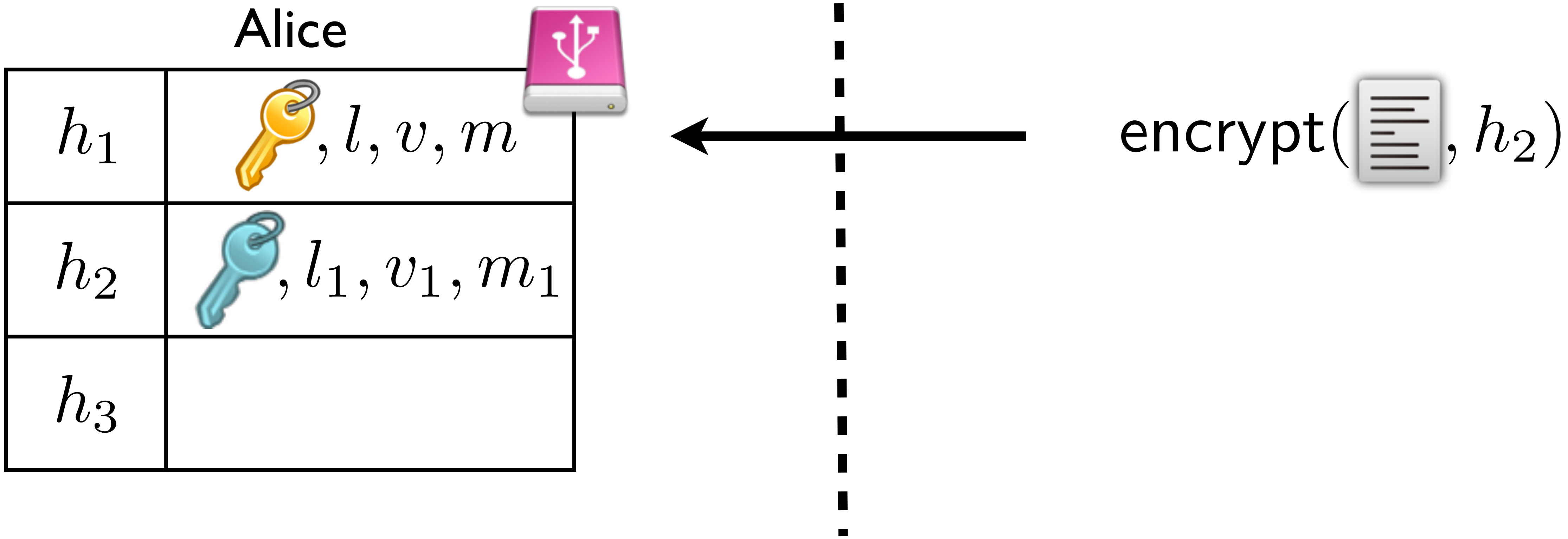
User’s Commands

Alice can now encrypt the message using the session key.

	Alice
h_1	🔑$, l, v, m$
h_2	🔑$, l_1, v_1, m_1$
h_3	

User’s Commands

Alice can now encrypt the message using the session key.



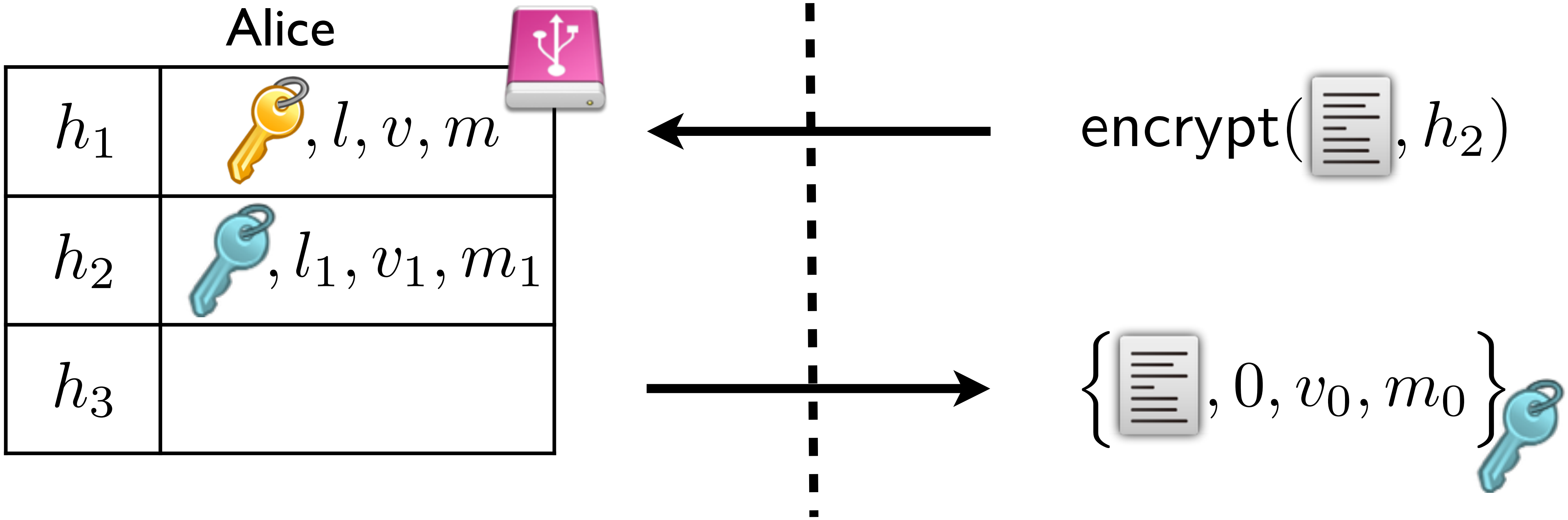
Alice

h_1	🔑$, l, v, m$
h_2	🔑$, l_1, v_1, m_1$
h_3	

encrypt(📄$, h_2)$

User’s Commands

Alice can now encrypt the message using the session key.



Alice

h_1	🔑, l, v, m
h_2	🔑, l_1, v_1, m_1
h_3	

encrypt(📄, h_2)

$\{$📄$, 0, v_0, m_0\}$🔑

User’s Commands

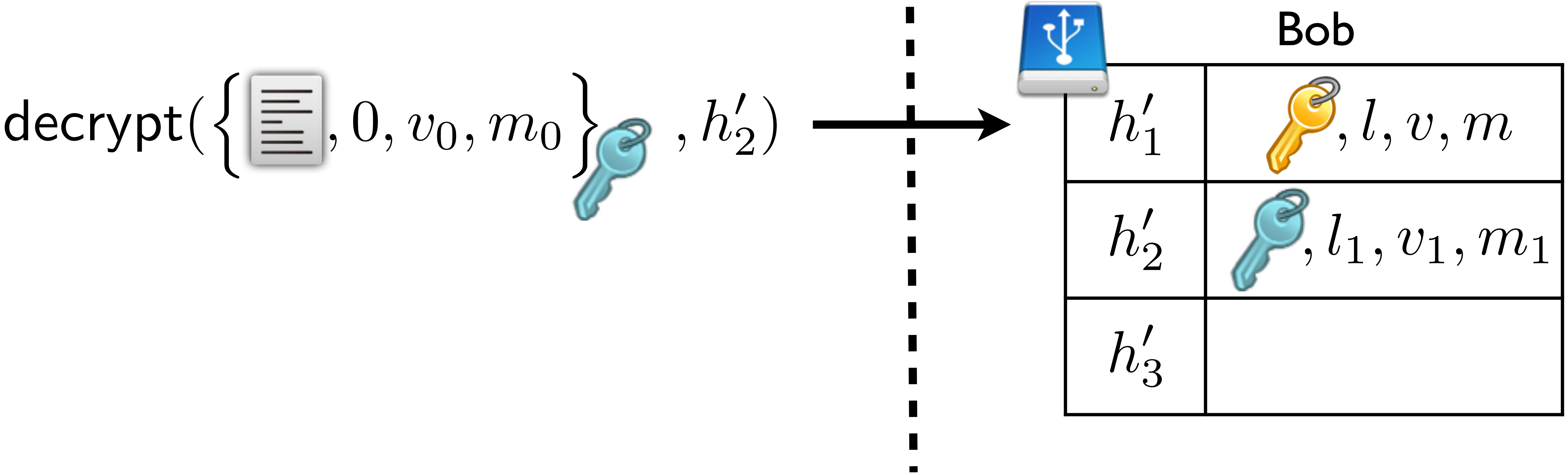
And, finally, Alice sends the encrypted message to Bob, which decrypts it.

Bob

h'_1	🔑$, l, v, m$
h'_2	🔑$, l_1, v_1, m_1$
h'_3	

User's Commands

And, finally, Alice sends the encrypted message to Bob, which decrypts it.



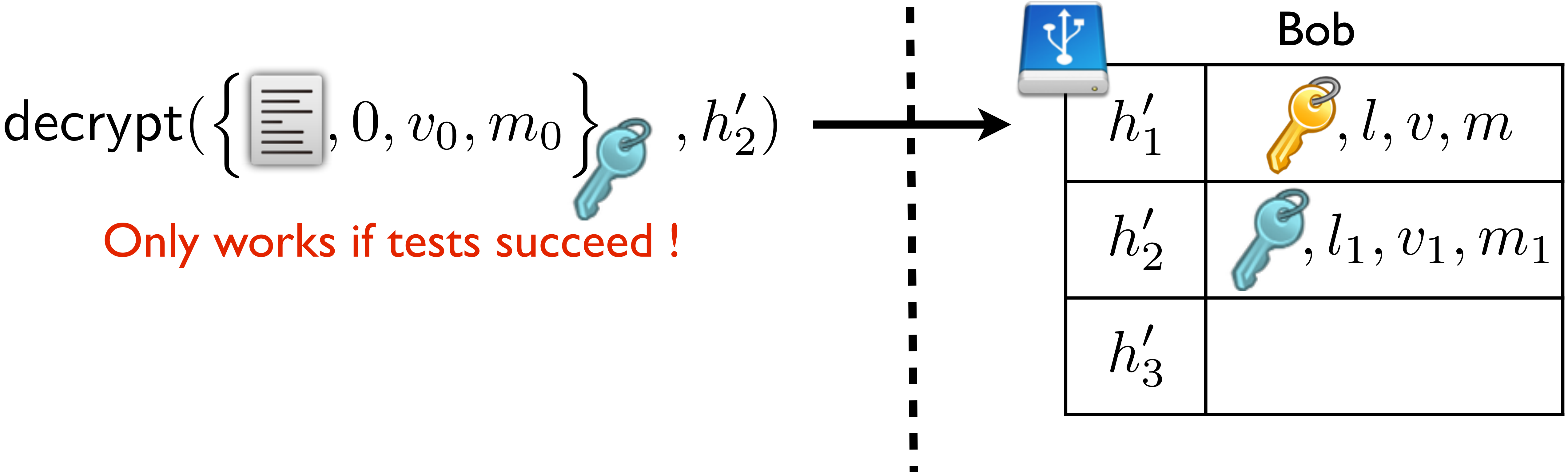
decrypt($\{$📄$, 0, v_0, m_0\}$🔑$, h'_2)$

Bob

h'_1	🔑$, l, v, m$
h'_2	🔑$, l_1, v_1, m_1$
h'_3	

User’s Commands

And, finally, Alice sends the encrypted message to Bob, which decrypts it.



decrypt($\{$ 📄 $, 0, v_0, m_0\}$ 🔑 $, h'_2)$

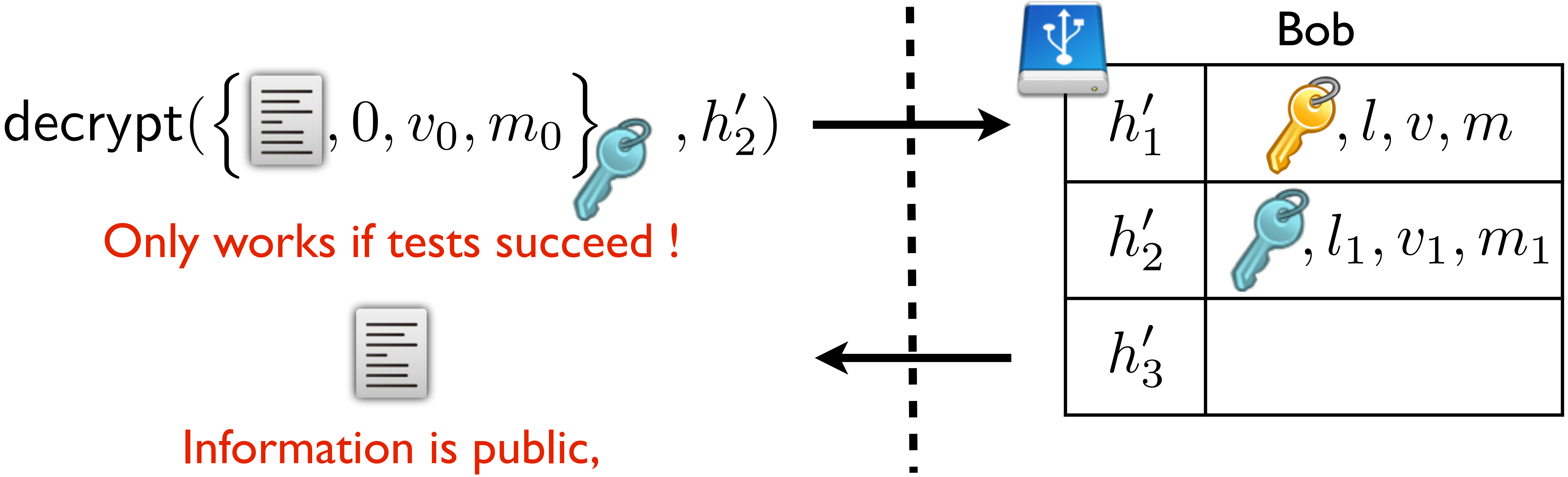
Only works if tests succeed !

Bob

h'_1	🔑 $, l, v, m$
h'_2	🔑 $, l_1, v_1, m_1$
h'_3	

User’s Commands

And, finally, Alice sends the encrypted message to Bob, which decrypts it.



decrypt({📄, $0, v_0, m_0$}🔑, h'_2)

Only works if tests succeed !

	Bob
h'_1	🔑$, l, v, m$
h'_2	🔑$, l_1, v_1, m_1$
h'_3	

📄

Information is public,
no need for handles.

User’s Commands

A set of **basic commands** (summary):

generatePublic(m)

generateSecret(l, m)

Generate a nonce or a key, and store under a handle the information.

decrypt(C, h)

Decrypt C with the key stored under h and return a message or a handle.

encrypt$(\langle X_1, \ldots, X_n \rangle, h)$

Encrypt the input under the key stored in handle h.

Lower Level Keys Management

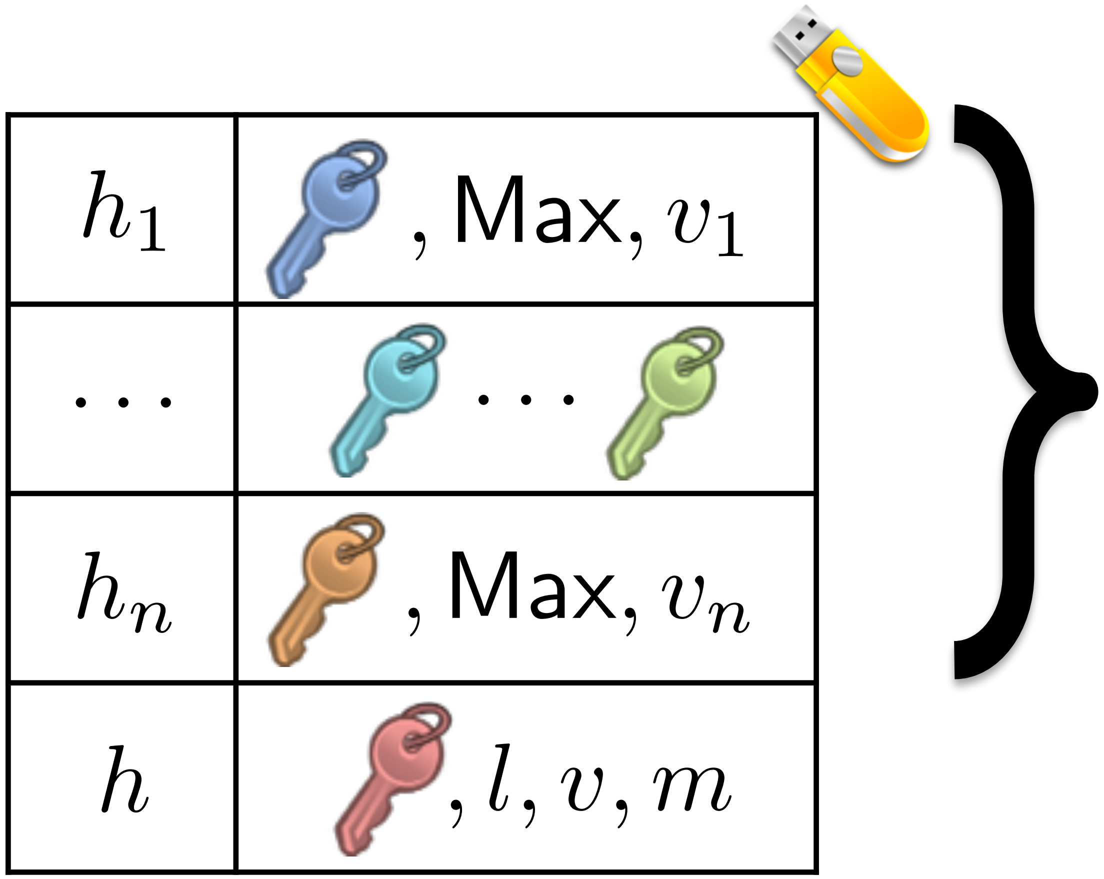
We also have **admin commands**:

- Allow to **administrate lower level keys** (i.e. level < Max).
- Need **revocation keys,** i.e. keys of level Max.
- Each device has its own set of admin keys.

Lower Level Keys Management

We also have **admin commands**:

- Allow to **administrate lower level keys** (i.e. level < Max).
- Need **revocation keys**, i.e. keys of level Max.
- Each device has its own set of admin keys.



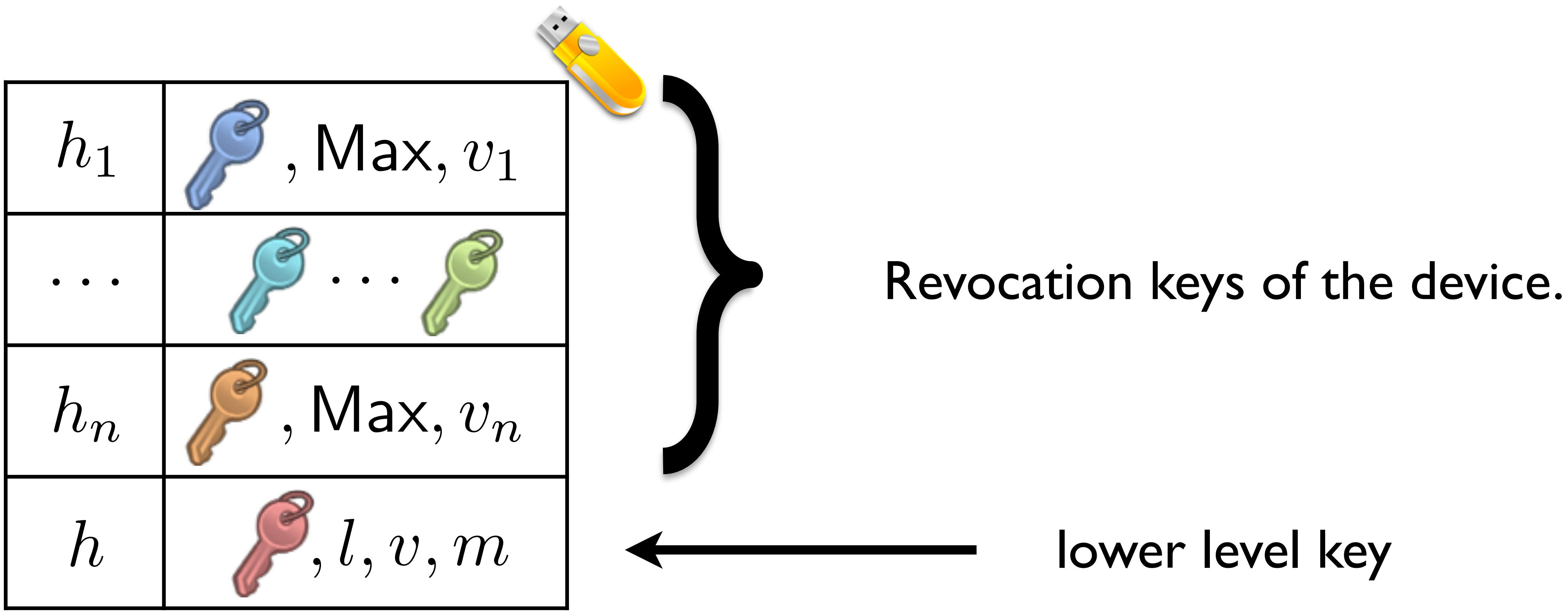
h_1	🔑 $, \mathrm{Max}, v_1$
$\ldots$	🔑 $\ldots$ 🔑
h_n	🔑 $, \mathrm{Max}, v_n$
h	🔑 $, l, v, m$

Revocation keys of the device.

Lower Level Keys Management

We also have **admin commands**:

- Allow to **administrate lower level keys** (i.e. level < Max).
- Need **revocation keys,** i.e. keys of level Max.
- Each device has its own set of admin keys.



h_1	🔑, Max, v_1
$\dots$	🔑 $\dots$ 🔑
h_n	🔑, Max, v_n
h	🔑, l, v, m

Revocation keys of the device.

lower level key

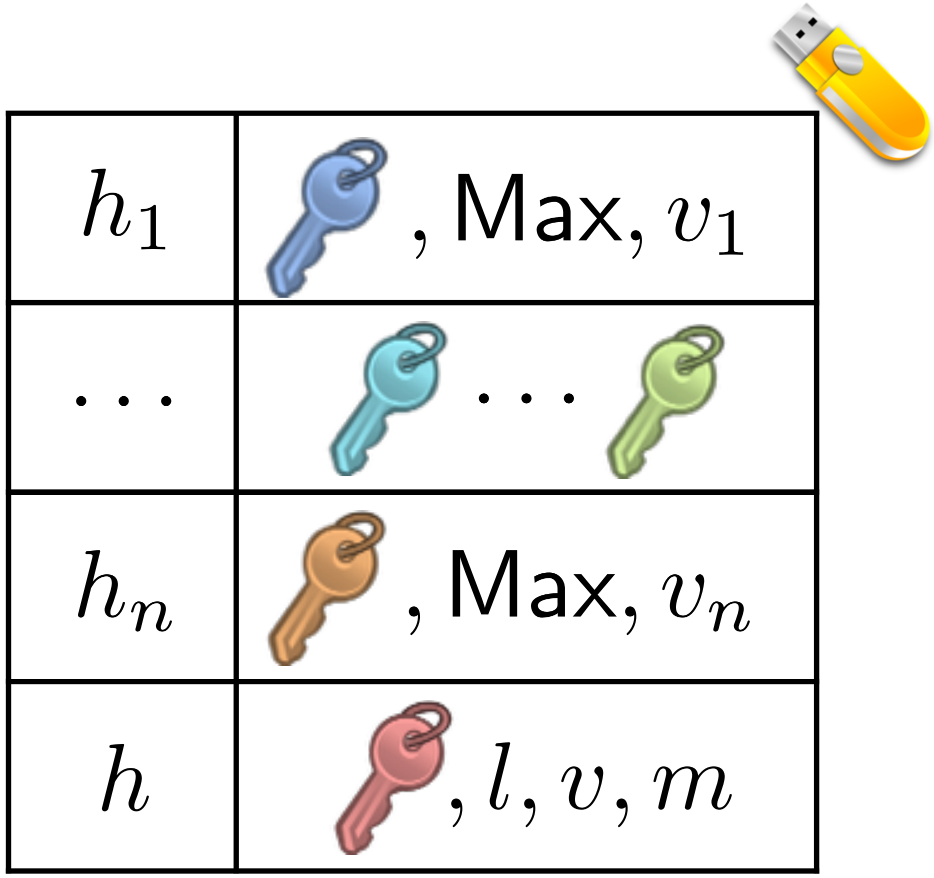
Lower Level Keys Management

h_1	🔑, Max, v_1
$\dots$	🔑 $\dots$ 🔑
h_n	🔑, Max, v_n
h	🔑, l, v, m

update$(C, h_1, \dots, h_n)$

Update value and attributes of keys that are not admin (level Max) keys.

Lower Level Keys Management



h_1	🔑, Max, v_1
$\dots$	🔑 $\dots$ 🔑
h_n	🔑, Max, v_n
h	🔑, l, v, m

update$(C, h_1, \dots, h_n)$

Update value and attributes of keys that are not admin (level Max) keys.

$$C = \left\{\text{update}, 🔑, 🔑, l', v', m'\right\}_{🔑 \dots 🔑}$$

Lower Level Keys Management

h_1	🔑, Max, v_1
$\dots$	🔑 $\dots$ 🔑
h_n	🔑, Max, v_n
h	🔑, l, v, m

update$(C, h_1, \dots, h_n)$

Update value and attributes of keys that are not admin (level Max) keys.

$$C = \left\{\text{update}, 🔑, 🔑, l', v', m'\right\}_{🔑 \dots 🔑}$$

How does it work ?

Lower Level Keys Management

h_1	🔑, Max, v_1
$\dots$	🔑 $\dots$ 🔑
h_n	🔑, Max, v_n
h	🔑, l, v, m

update$(C, h_1, \dots, h_n)$

Update value and attributes of keys that are not admin (level Max) keys.

$C = \{$update, 🔑, 🔑, $l', v', m'\}$ 🔑 $\dots$ 🔑

How does it work ?

1. Tests on keys stored under $h_1, \dots, h_n$.

   > Are they **level Max** and **valid keys** ?

Lower Level Keys Management

h_1	🔑, Max, v_1
$\ldots$	🔑 $\ldots$ 🔑
h_n	🔑, Max, v_n
h	🔑, l, v, m

update$(C, h_1, \ldots, h_n)$

Update value and attributes of keys that are not admin (level Max) keys.

$$C = \left\{\text{update}, 🔑, 🔑, l', v', m'\right\}_{🔑 \ldots 🔑}$$

How does it work ?

1. Tests on keys stored under $h_1, \ldots, h_n$.

   > Are they **level Max** and **valid keys** ?

2. Decryption of C.

   > Obtaining old/new value and new attributes.

Lower Level Keys Management

How does it work ?

3. Verify that the old key (🔑) is in the device.

h_1	🔑, Max, v_1
$\dots$	🔑 $\dots$ 🔑
h_n	🔑, Max, v_n
h	🔑, l, v, m

Lower Level Keys Management

How does it work ?

3. Verify that the old key (🔑) is in the device.

4. Tests on the new attributes l', v' of new key (🔑).

> Are the new level and validity date correct ?

h_1	🔑, Max, v_1
$\dots$	🔑 $\dots$ 🔑
h_n	🔑, Max, v_n
h	🔑, l, v, m

Lower Level Keys Management

How does it work ?

3. Verify that the old key (🔑) is in the device.

4. Tests on the new attributes l', v' of new key (🔑).

   > Are the new level and validity date correct ?

5. Table update with the new values.

h_1	🔑, Max, v_1
$\ldots$	🔑 $\ldots$ 🔑
h_n	🔑, Max, v_n
h	🔑 $, l, v, m$

Lower Level Keys Management

How does it work ?

3. Verify that the old key (🔑) is in the device.

4. Tests on the new attributes l', v' of new key (🔑).

   > Are the new level and validity date correct ?

5. Table update with the new values.

h_1	🔑, Max, v_1
$\dots$	🔑 $\dots$ 🔑
h_n	🔑, Max, v_n
h	🔑, l', v', m'

Revocation Keys Management

The same scheme applies for **revoking revocation keys**.

$\text{updateMax}(C, h_1, \ldots, h_n)$

> Require a number N_{Max} of valid revocation keys.

Revocation Keys Management

The same scheme applies for **revoking revocation keys**.

updateMax$(C, h_1, \ldots, h_n)$

> Require a number N_{Max} of valid revocation keys.

h_1	🔑, Max, v_1
h_2	🔑, Max, v_2
h_3	🔑, Max, v_3

$N_{\text{Max}} = 2$

Revocation Keys Management

The same scheme applies for **revoking revocation keys**.

updateMax$(C, h_1, \ldots, h_n)$

> Require a number N_{Max} of valid revocation keys.

$\{$UpdateMax, 🔑, 🔑, $v_1'\}_{🔑🔑}$

h_1	🔑, Max, v_1
h_2	🔑, Max, v_2
h_3	🔑, Max, v_3

$N_{\text{Max}} = 2$

Revocation Keys Management

The same scheme applies for **revoking revocation keys**.

updateMax$(C, h_1, \ldots, h_n)$

> Require a number N_{Max} of valid revocation keys.

{UpdateMax, 🔑, 🔑, v_1'}🔑🔑

h_1	🔑, Max, v_1'
h_2	🔑, Max, v_2
h_3	🔑, Max, v_3

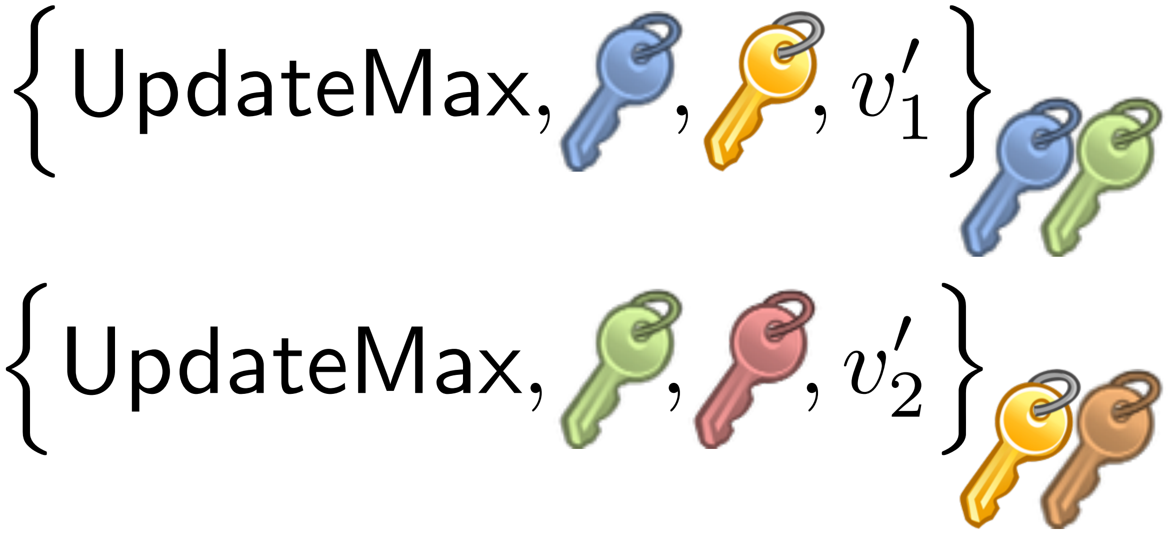
$N_{\text{Max}} = 2$

Revocation Keys Management

The same scheme applies for **revoking revocation keys**.

updateMax$(C, h_1, \dots, h_n)$

> Require a number N_{Max} of valid revocation keys.



{UpdateMax, 🔑, 🔑, v'_1}🔑🔑

{UpdateMax, 🔑, 🔑, v'_2}🔑🔑

h_1	🔑, Max, v'_1
h_2	🔑, Max, v_2
h_3	🔑, Max, v_3

$N_{\text{Max}} = 2$

Revocation Keys Management

The same scheme applies for **revoking revocation keys**.

updateMax$(C, h_1, \ldots, h_n)$

> Require a number N_{Max} of valid revocation keys.

{UpdateMax, 🔑, 🔑, v_1'}🔑🔑

{UpdateMax, 🔑, 🔑, v_2'}🔑🔑

h_1	🔑, Max, v_1'
h_2	🔑, Max, v_2'
h_3	🔑, Max, v_3

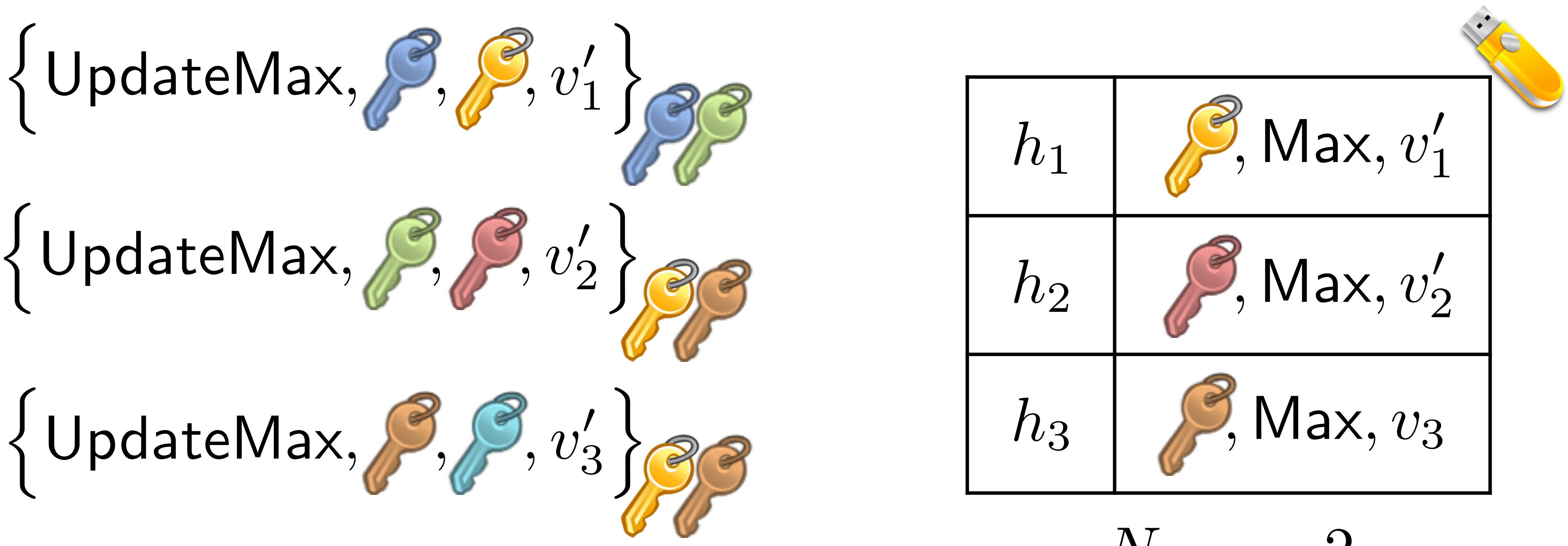
$N_{\text{Max}} = 2$

Revocation Keys Management

The same scheme applies for **revoking revocation keys**.

updateMax$(C, h_1, \ldots, h_n)$

> Require a number N_{Max} of valid revocation keys.



{UpdateMax, 🔑, 🔑, v'_1}🔑🔑

{UpdateMax, 🔑, 🔑, v'_2}🔑🔑

{UpdateMax, 🔑, 🔑, v'_3}🔑🔑

h_1	🔑, Max, v'_1
h_2	🔑, Max, v'_2
h_3	🔑, Max, v_3

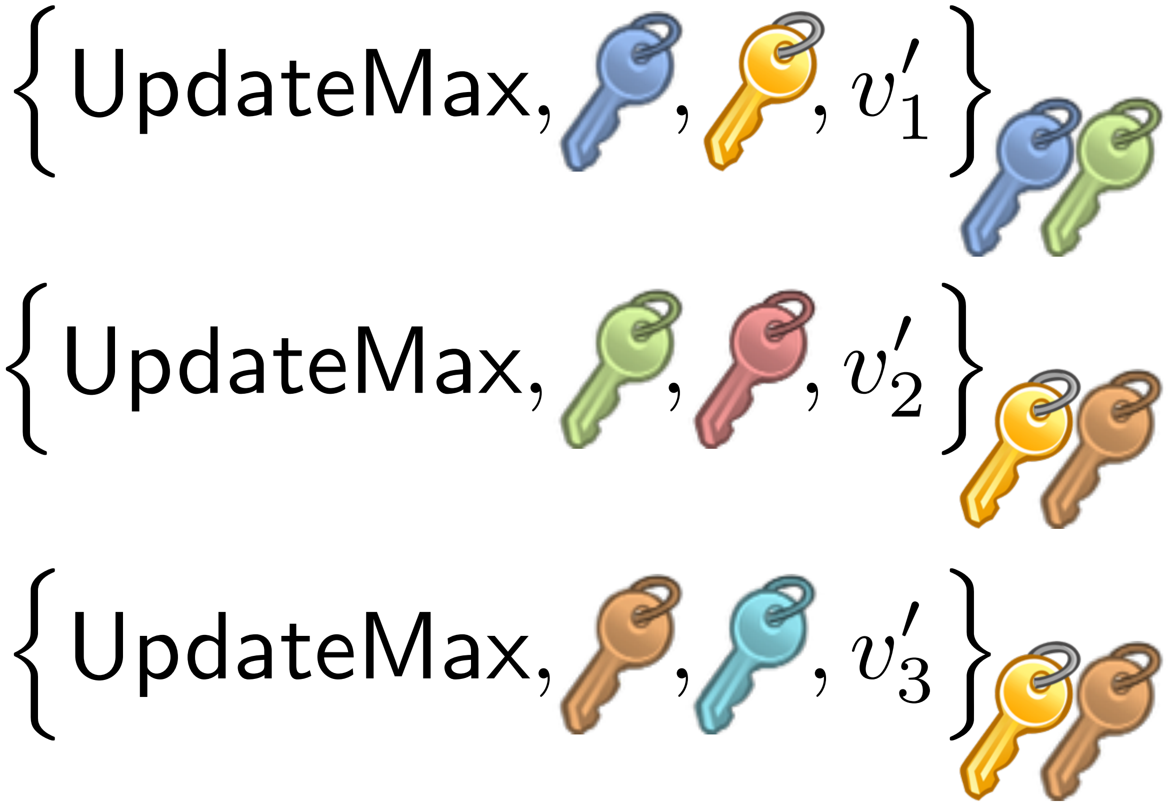
$N_{\text{Max}} = 2$

Revocation Keys Management

The same scheme applies for **revoking revocation keys**.

updateMax$(C, h_1, \ldots, h_n)$

> Require a number N_{Max} of valid revocation keys.



$\{$UpdateMax, 🔑, 🔑, $v'_1\}_{🔑🔑}$

$\{$UpdateMax, 🔑, 🔑, $v'_2\}_{🔑🔑}$

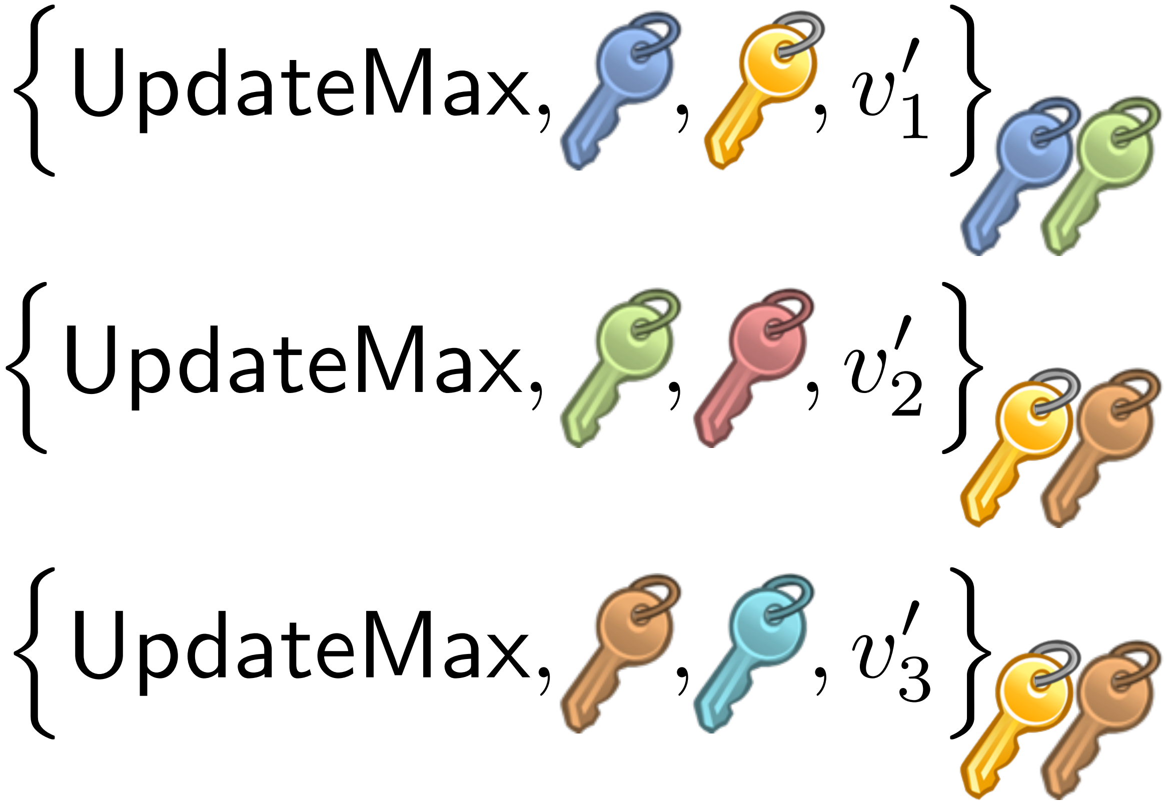
$\{$UpdateMax, 🔑, 🔑, $v'_3\}_{🔑🔑}$

h_1	🔑, Max, v'_1
h_2	🔑, Max, v'_2
h_3	🔑, Max, v'_3

$N_{\text{Max}} = 2$

Revocation Keys Management

What if (old) revocation keys can be lost and if revocation messages are public ?



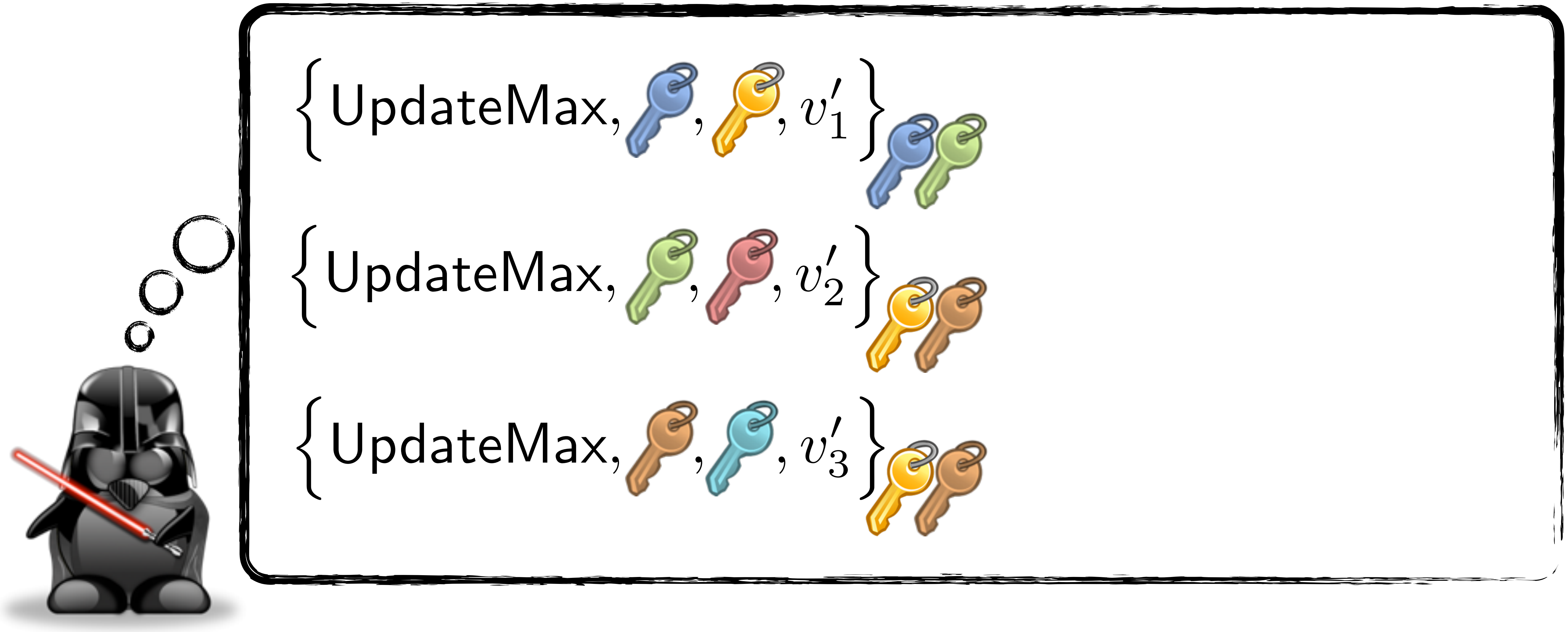
$$\left\{\text{UpdateMax}, 🔑, 🔑, v'_1\right\}_{🔑🔑}$$

$$\left\{\text{UpdateMax}, 🔑, 🔑, v'_2\right\}_{🔑🔑}$$

$$\left\{\text{UpdateMax}, 🔑, 🔑, v'_3\right\}_{🔑🔑}$$

Revocation Keys Management

What if (old) revocation keys can be lost and if revocation messages are public ?



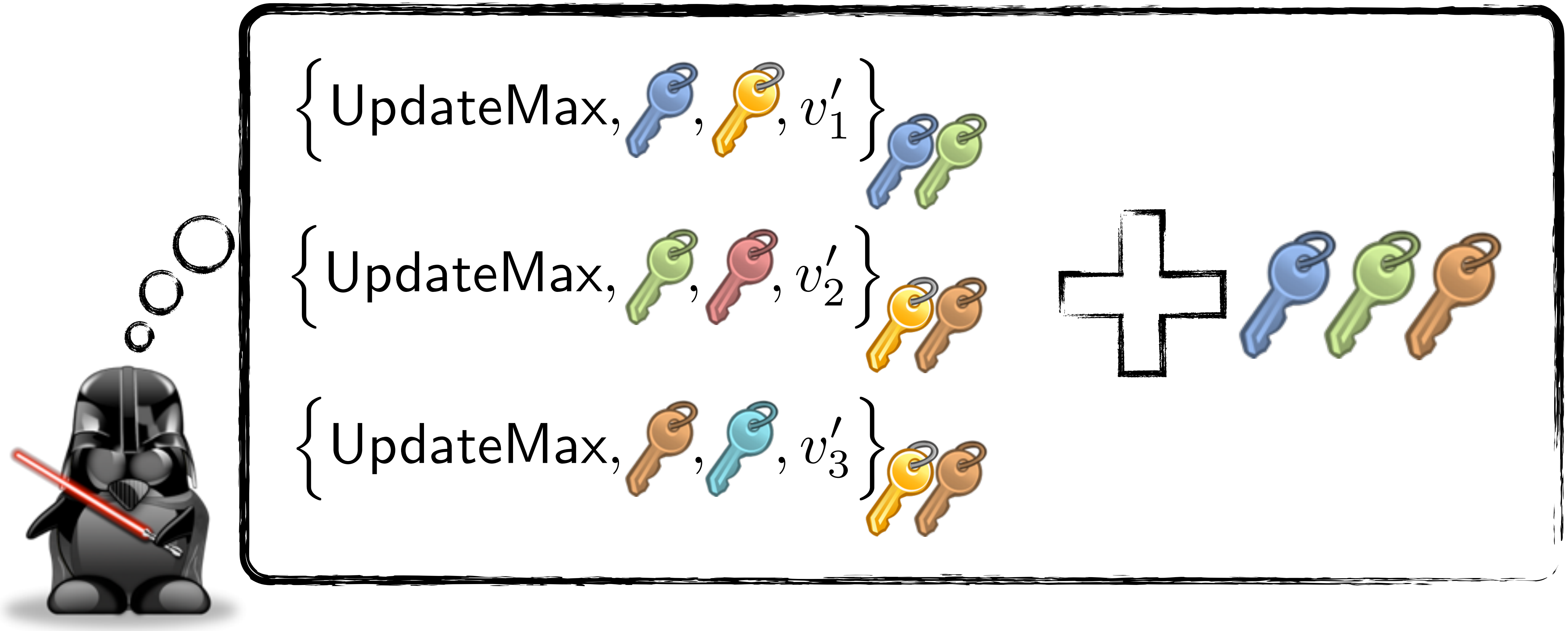
{UpdateMax, 🔑, 🔑, v'_1} 🔑🔑

{UpdateMax, 🔑, 🔑, v'_2} 🔑🔑

{UpdateMax, 🔑, 🔑, v'_3} 🔑🔑

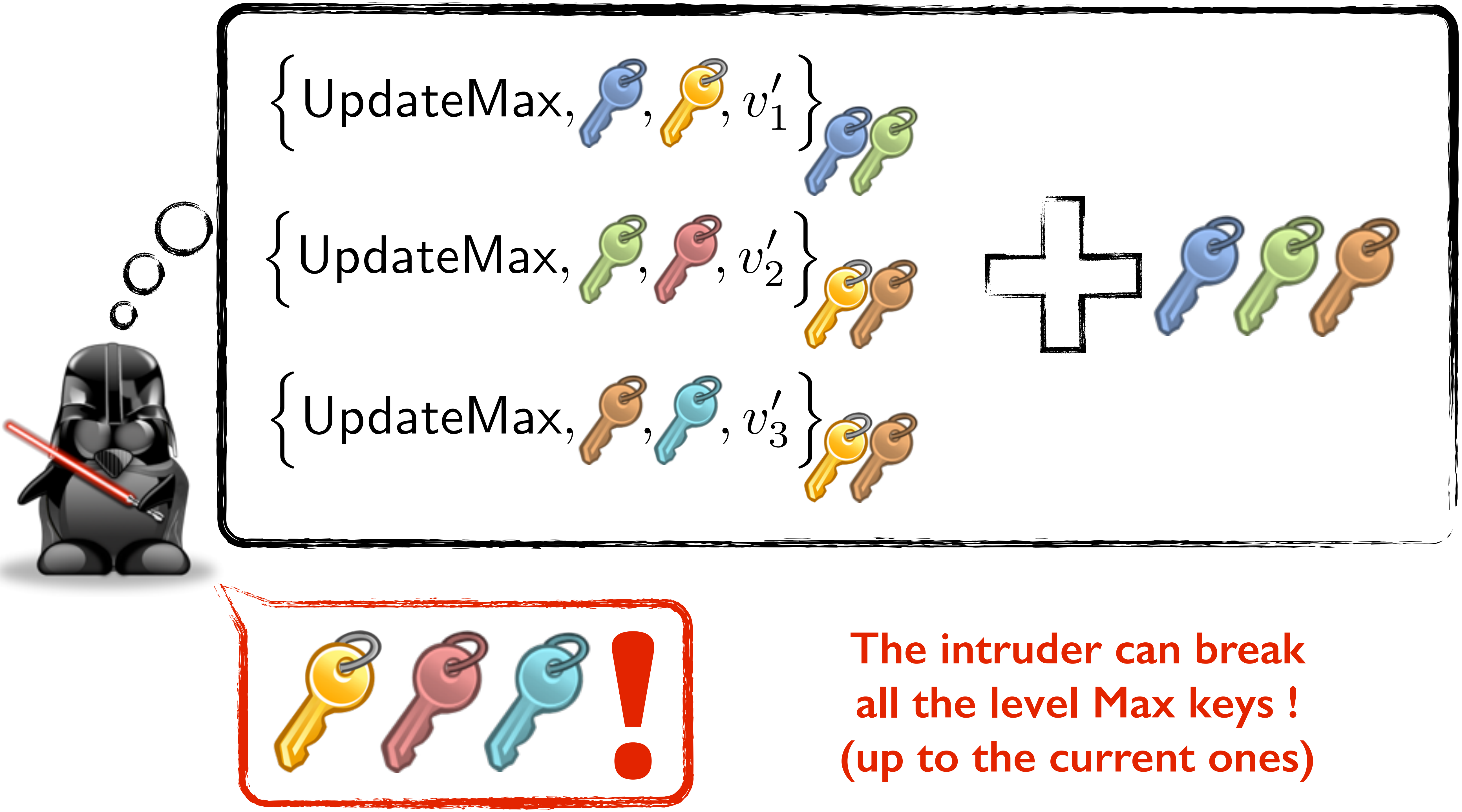
Revocation Keys Management

What if (old) revocation keys can be lost and if revocation messages are public ?


{UpdateMax, 🔑, 🔑, v'_1}🔑🔑
{UpdateMax, 🔑, 🔑, v'_2}🔑🔑
{UpdateMax, 🔑, 🔑, v'_3}🔑🔑
+ 🔑🔑🔑

Revocation Keys Management

What if (old) revocation keys can be lost and if revocation messages are public ?


$\{\text{UpdateMax}, , , v'_1\}$
$\{\text{UpdateMax}, , , v'_2\}$
$\{\text{UpdateMax}, , , v'_3\}$
!

The intruder can break all the level Max keys ! (up to the current ones)

Revocation Keys Management

Hypothesis :

Level Max commands are sent over a secure channel.

Revocation Keys Management

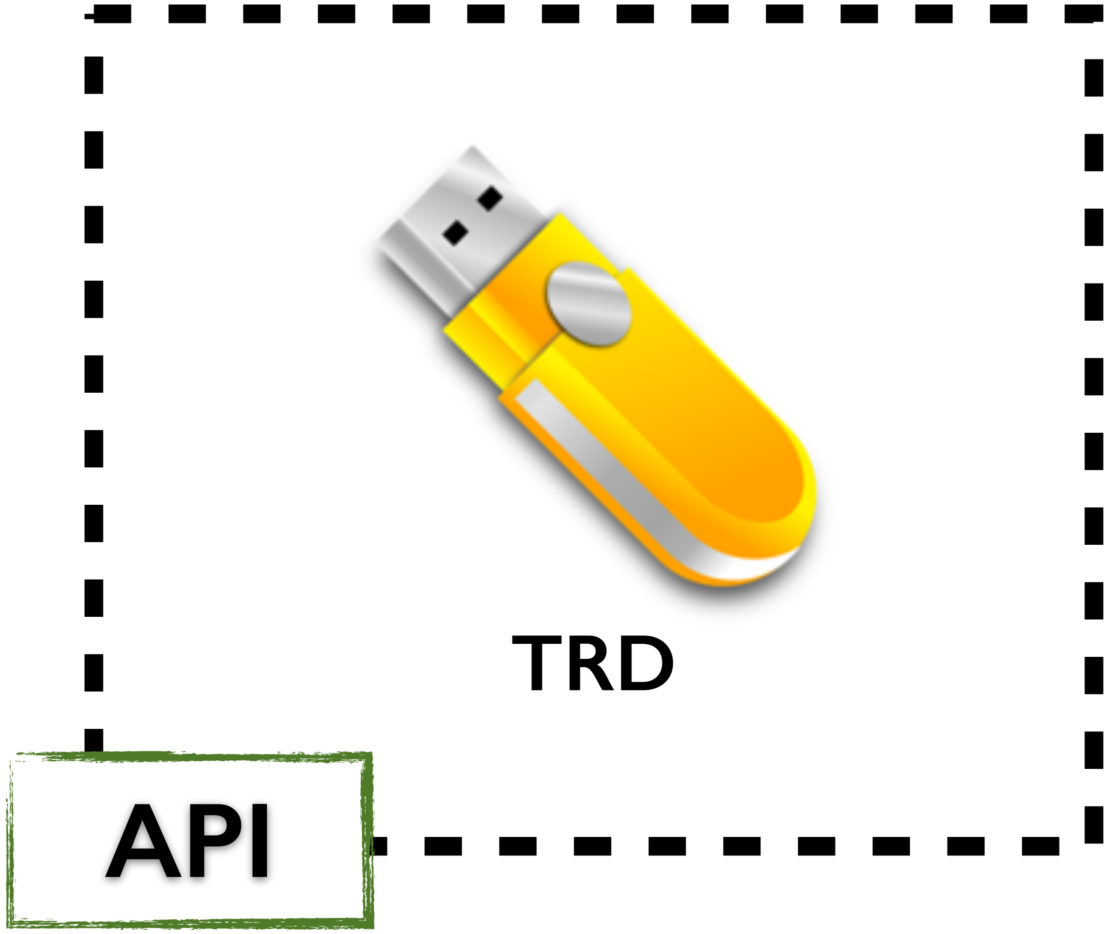
Hypothesis :

Level Max commands are sent over a secure channel.

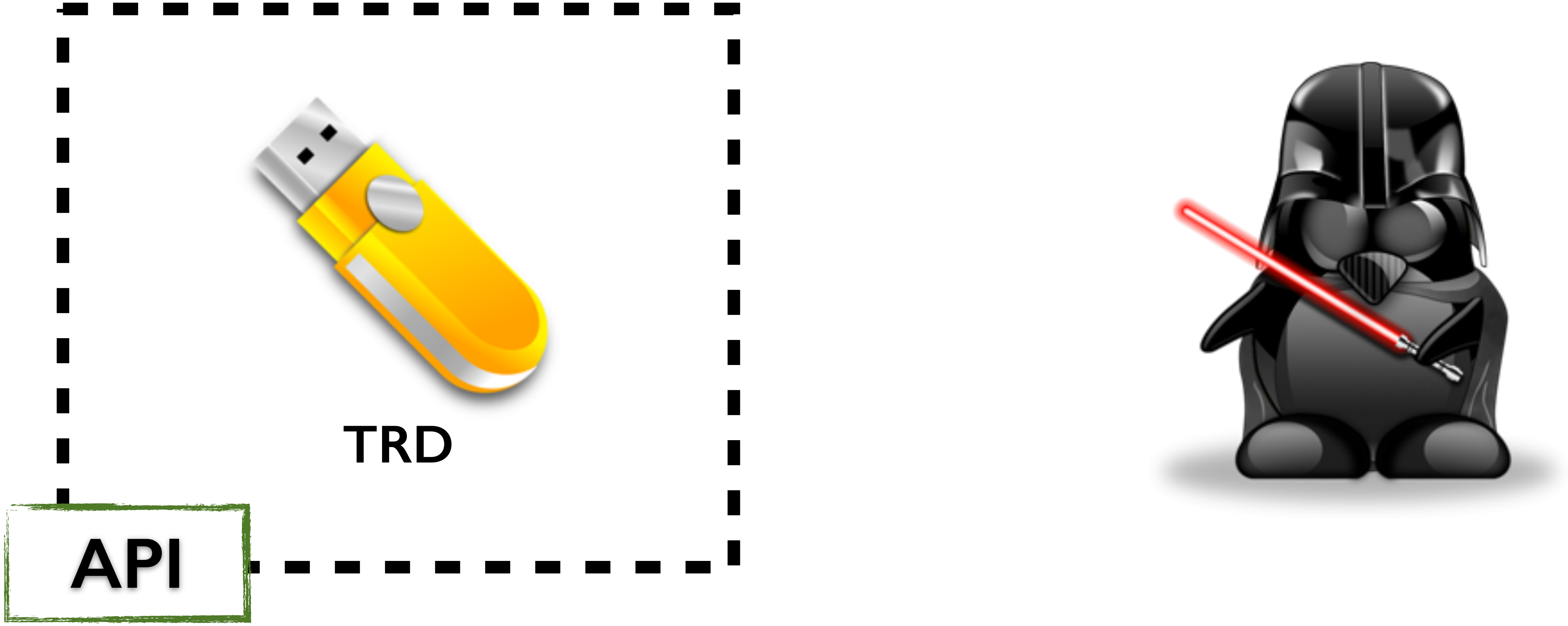
This can be achieved by several means :

- The administrator has a physical access to the TRD that needs to be updated,
- The user would connect his/her TRD to a trusted machine, on which a secure channel (e.g. via TLS) is established with the key administrator.

And now, what about Security ?



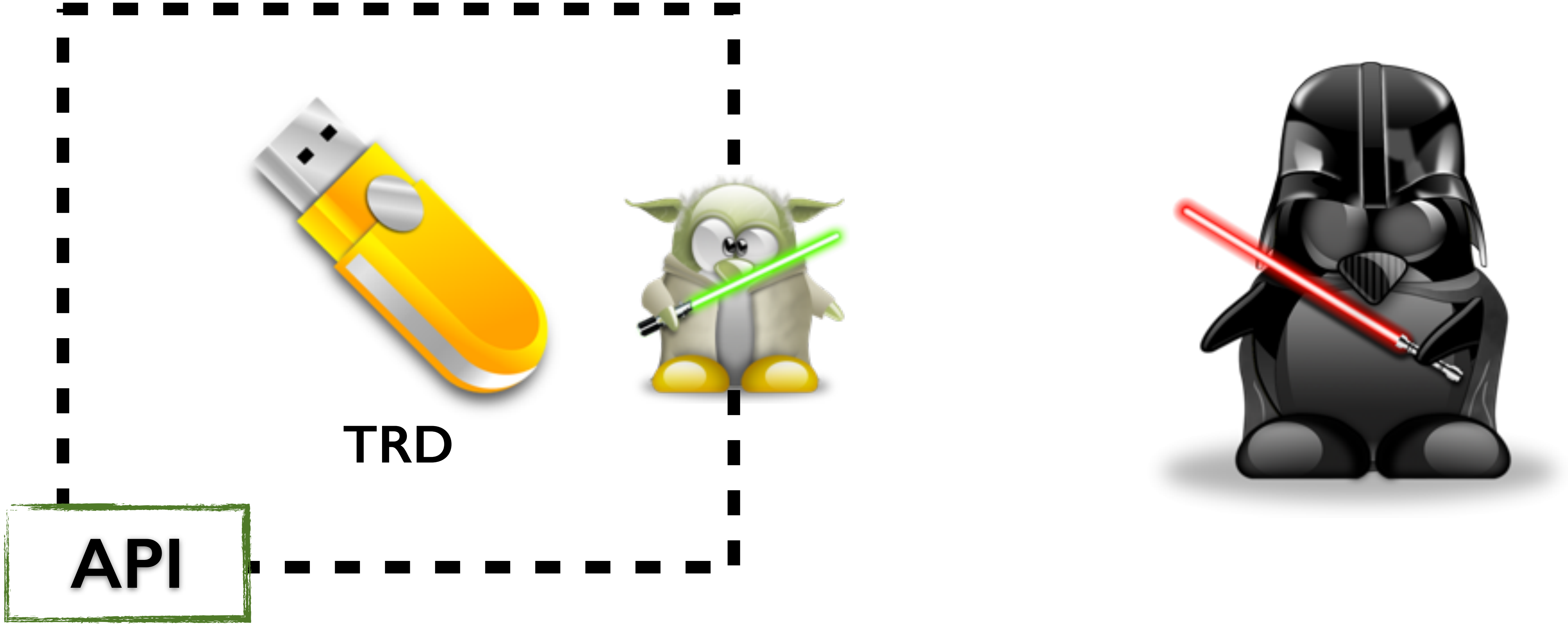
And now, what about Security ?



TRD

API

And now, what about Security ?

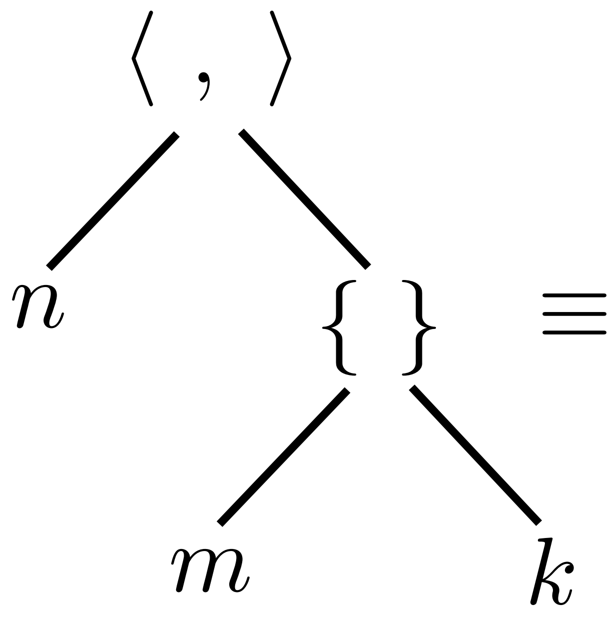


Abstraction

Messages are represented by **terms**

Nonces, keys :

$$n, m, \ldots, k_1, k_2, \ldots$$



$$\equiv \langle n, \{m\}_k \rangle$$

Primitives :

$$\{m\}_k, \langle m_1, m_2 \rangle$$

Modeling deduction rules :

$$\frac{x \quad y}{\langle x, y \rangle} \qquad \frac{\langle x, y \rangle}{x} \qquad \frac{\langle x, y \rangle}{y} \qquad \frac{x \quad y}{\{x\}_y} \qquad \frac{\{x\}_y \quad y}{x}$$

Formal Model

We model the system using **global states**:

$$(\mathcal{P}, \mathcal{I}, \mathfrak{M}, N, K, t)$$

Formal Model

We model the system using **global states**:

$$(\mathcal{P}, \mathcal{I}, \mathfrak{M}, N, K, t)$$

$\mathcal{P}$, the set of **TRDs** in use in the system.

Formal Model

We model the system using **global states**:

$$(\mathcal{P}, \mathcal{I}, \mathfrak{M}, N, K, t)$$

$\mathcal{P}$, the set of **TRDs** in use in the system.

$\mathfrak{M}$, the set of **messages** that have been **sent on the network**.
(Represents also the **knowledge of the intruder.**)

Formal Model

We model the system using **global states**:

$$(\mathcal{P}, \mathcal{I}, \mathfrak{M}, N, K, t)$$

$\mathcal{P}$, the set of **TRDs** in use in the system.

$\mathfrak{M}$, the set of **messages** that have been **sent on the network**.
(Represents also the **knowledge of the intruder**.)

N, the set of **nonces** currently in used in the system.

Formal Model

We model the system using **global states**:

$$(\mathcal{P}, \mathcal{I}, \mathfrak{M}, N, K, t)$$

$\mathcal{P}$, the set of **TRDs** in use in the system.

$\mathfrak{M}$, the set of **messages** that have been **sent on the network**.
(Represents also the **knowledge of the intruder.**)

N, the set of **nonces** currently in used in the system.

K, the set of **keys** currently in used in the system.

Formal Model

We model the system using **global states**:

$$(\mathcal{P}, \mathcal{I}, \mathfrak{M}, N, K, t)$$

$\mathcal{P}$, the set of **TRDs** in use in the system.

$\mathfrak{M}$, the set of **messages** that have been **sent on the network**.
(Represents also the **knowledge of the intruder**.)

N, the set of **nonces** currently in used in the system.

K, the set of **keys** currently in used in the system.

t, represents the **current time**.

Formal Model

We model the system using **global states**:

$$(\mathcal{P}, \mathcal{I}, \mathfrak{M}, N, K, t)$$

$$\mathcal{I} : a \mapsto (\Theta_a, H_a, \mathfrak{B}_a, t, N_a, K_a)$$

is a **function** describing the **local state** of TRD . a

Formal Model

We model the system using **global states**:

$$(\mathcal{P}, \mathcal{I}, \mathfrak{M}, N, K, t)$$

$\mathcal{I} : a \mapsto (\Theta_a, H_a, \mathfrak{B}_a, t, N_a, K_a)$

is a **function** describing the **local state** of TRD . a

$\mathfrak{B}_a$, the set of **blacklisted levels**.

Formal Model

We model the system using **global states**:

$$(\mathcal{P}, \mathcal{I}, \mathfrak{M}, N, K, t)$$

$$\mathcal{I} : a \mapsto (\Theta_a, H_a, \mathfrak{B}_a, t, N_a, K_a)$$

is a **function** describing the **local state** of TRD . a

$\mathfrak{B}_a$, the set of **blacklisted levels**.

Θ_a , a function representing the **memory** of the TRD.

Handle	Value	Level	Validity	Misc.
h_1	🔑	l_1	v_1	m_1
h_2	🔑	l_2	v_2	-
...	...	...	...	...

Formal Model

Semantics

consists in several **transitions** modifying the global state.

Formal Model

Semantics

consists in several **transitions** modifying the global state.

$$(\mathrm{TIM}) \quad (\mathcal{P}, \mathcal{I}, \mathfrak{M}, N, K, t) \longrightarrow (\mathcal{P}, \mathcal{I}, \mathfrak{M}, N, K, t') \qquad (t' > t)$$

models the **time passing**...

Formal Model

Semantics

consists in several **transitions** modifying the global state.

$$(\mathrm{TIM}) \quad (\mathcal{P}, \mathcal{I}, \mathfrak{M}, N, K, t) \longrightarrow (\mathcal{P}, \mathcal{I}, \mathfrak{M}, N, K, t') \qquad (t' > t)$$

models the **time passing**...

$$(\mathrm{DED}) \quad (\mathcal{P}, \mathcal{I}, \mathfrak{M}, N, K, t) \longrightarrow (\mathcal{P}, \mathcal{I}, \mathfrak{M} \cup \{m\}, N, K, t)$$

models the **deduction abilities** of the intruder. $(\mathfrak{M} \vdash m)$

Formal Model

Semantics

consists in several transitions modifying the global state.

$$\text{(TIM)} \quad (\mathcal{P}, \mathcal{I}, \mathfrak{M}, N, K, t) \longrightarrow (\mathcal{P}, \mathcal{I}, \mathfrak{M}, N, K, t') \qquad (t' > t)$$

models the time passing...

$$\text{(DED)} \quad (\mathcal{P}, \mathcal{I}, \mathfrak{M}, N, K, t) \longrightarrow (\mathcal{P}, \mathcal{I}, \mathfrak{M} \cup \{m\}, N, K, t)$$

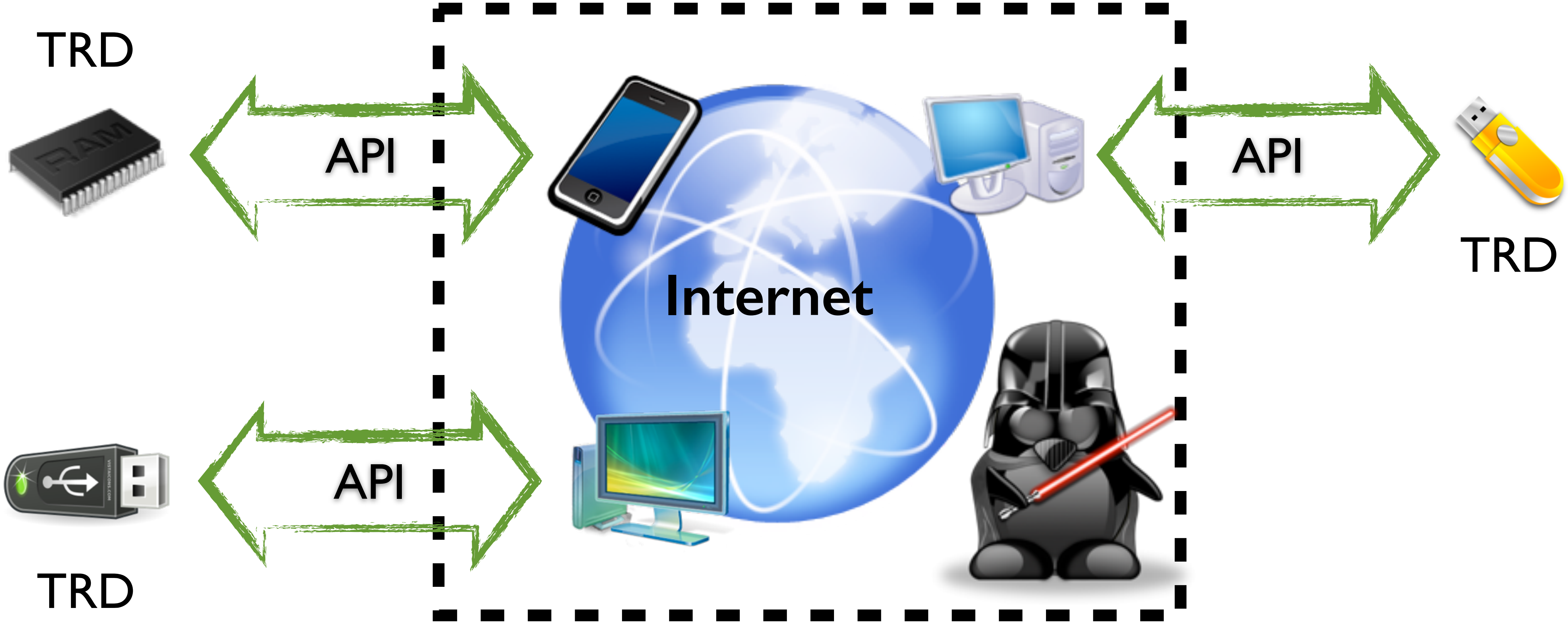
models the deduction abilities of the intruder. $(\mathfrak{M} \vdash m)$

$$\text{(UPD)} \quad (\mathcal{P}, \mathcal{I}, \mathfrak{M}, N, K, t) \longrightarrow (\mathcal{P}, \mathcal{I}', \mathfrak{M} \cup \{m\}, N', K', t)$$

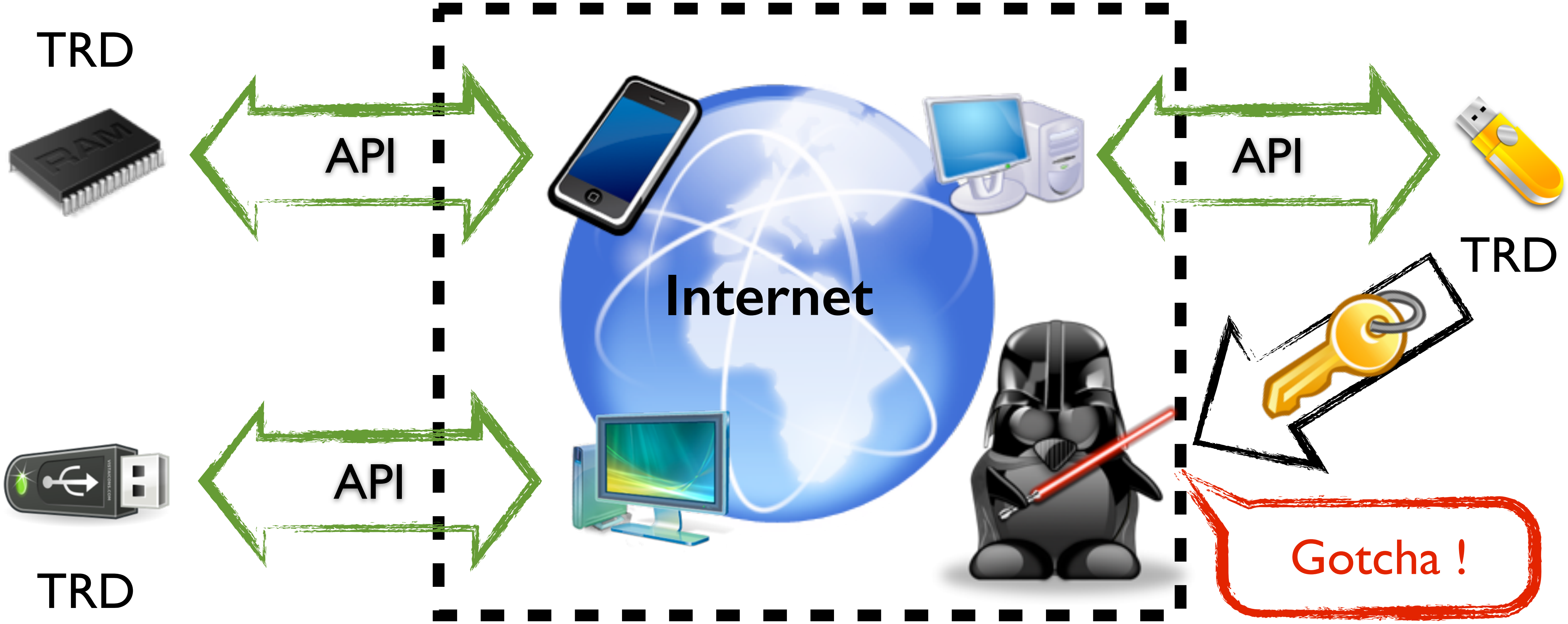
models changes when an update command is performed.

$$m = \Big\{\mathsf{update}, k, k', l', v', m'\Big\}_{k_1 \cdots k_n}$$

Knowledge of the Intruder

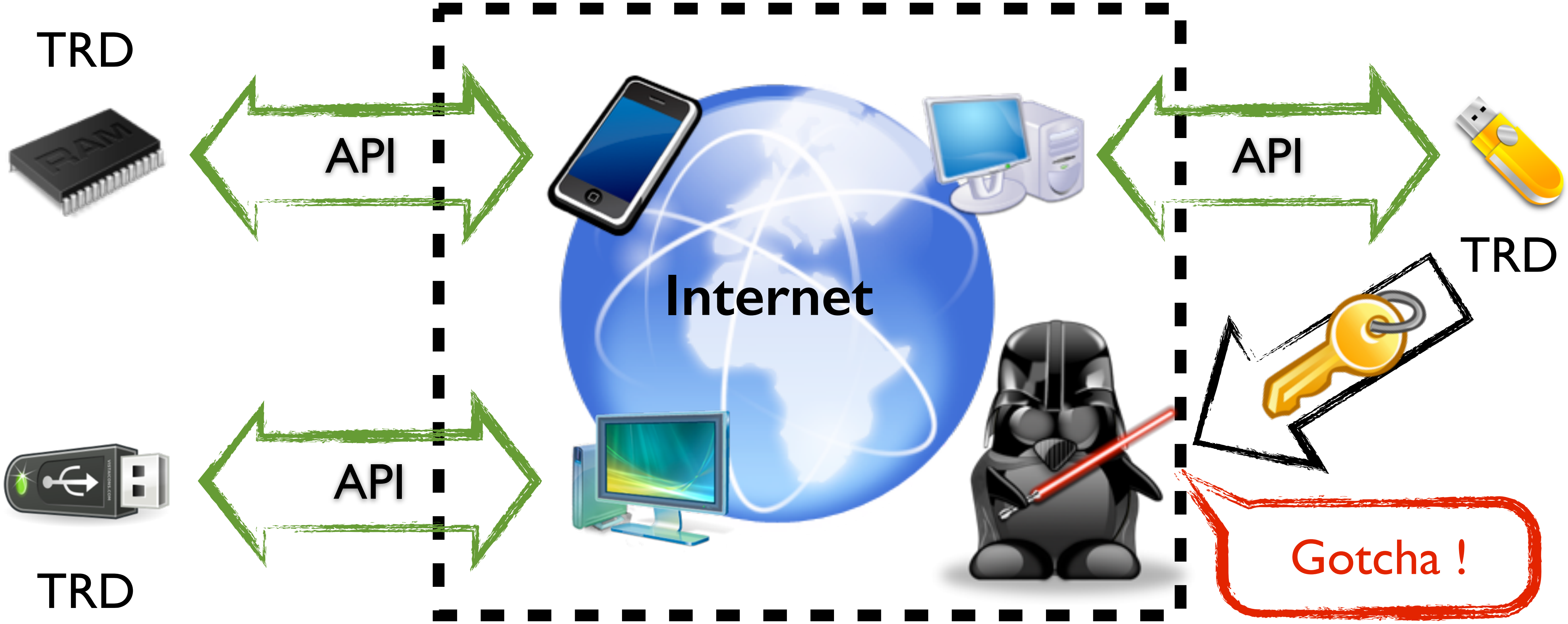


Knowledge of the Intruder



A key in a TRD may be lost and known by the intruder

Knowledge of the Intruder

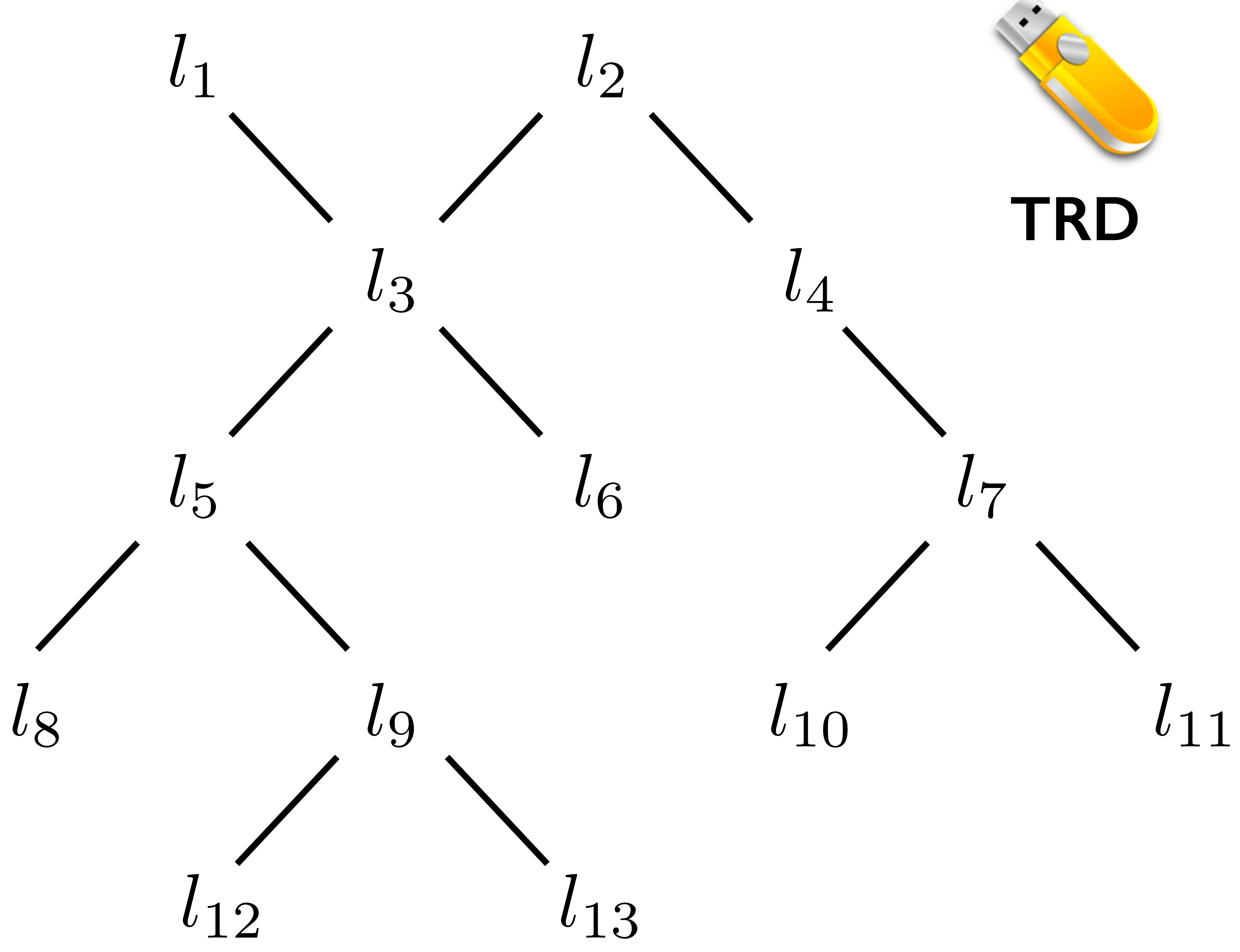


A key in a TRD may be lost and known by the intruder

Hypothesis :

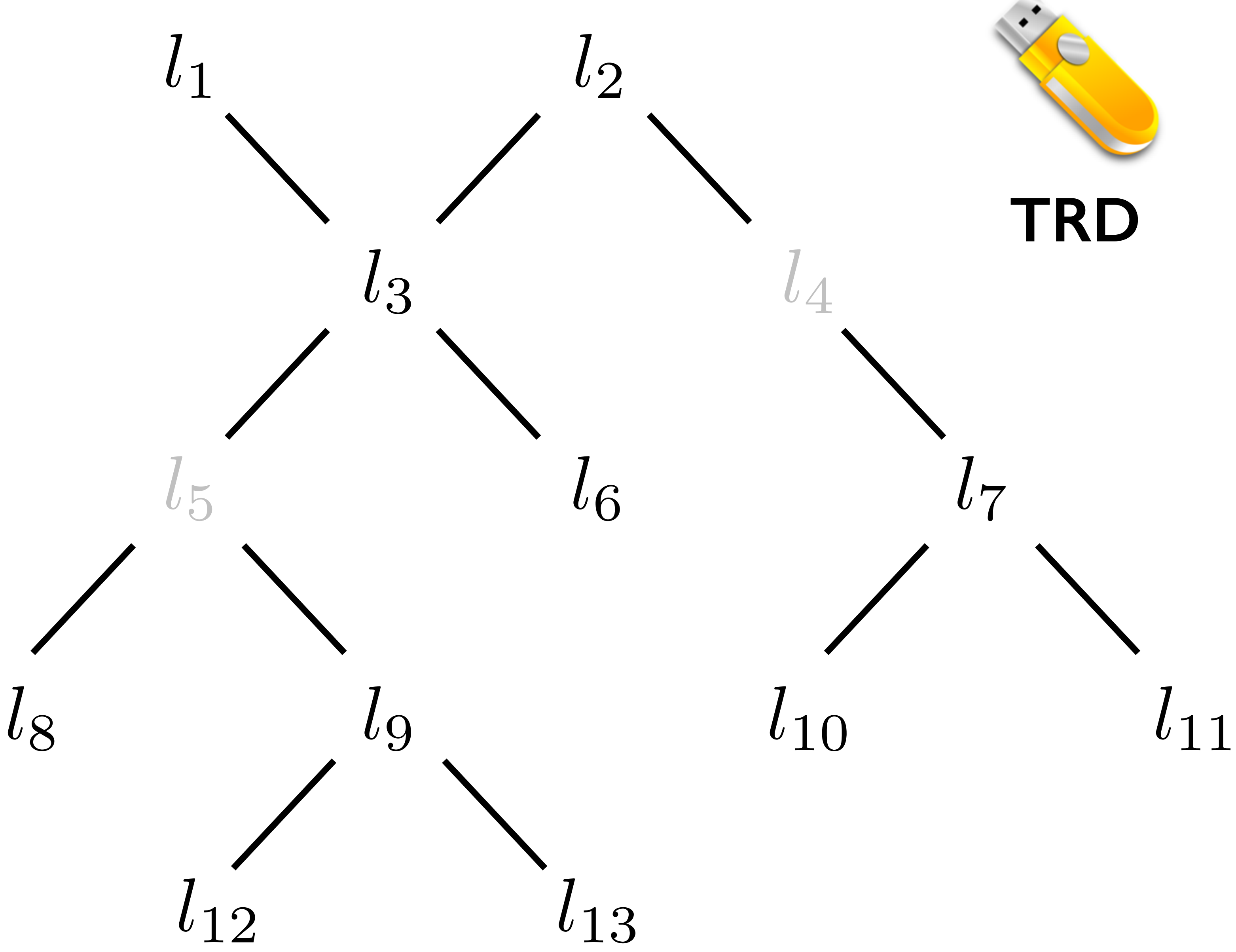
At most a total of $N_{\text{Max}} - 1$ different « current » level Max keys for one TRD can be lost.

What about lost keys ?



TRD

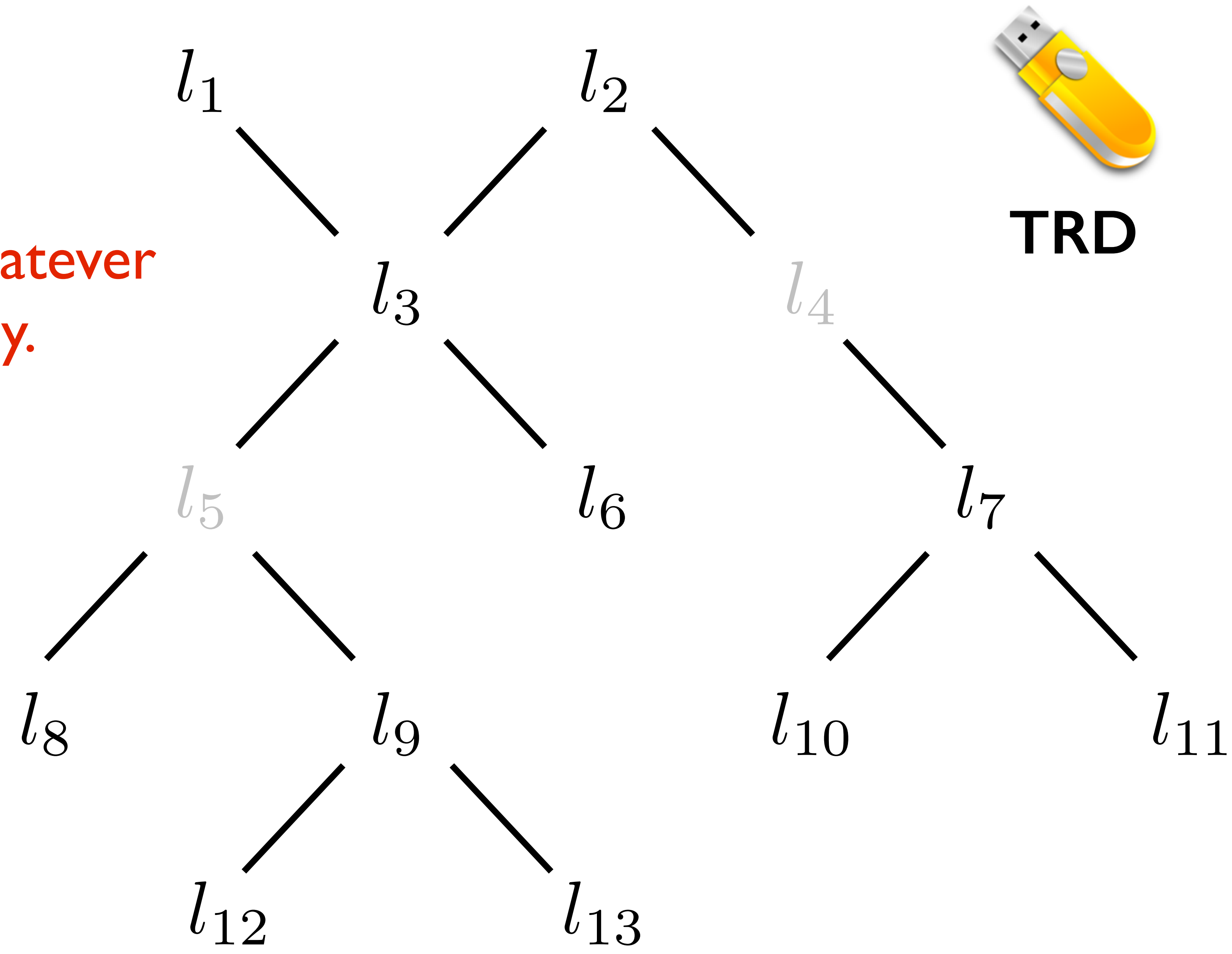
What about lost keys ?



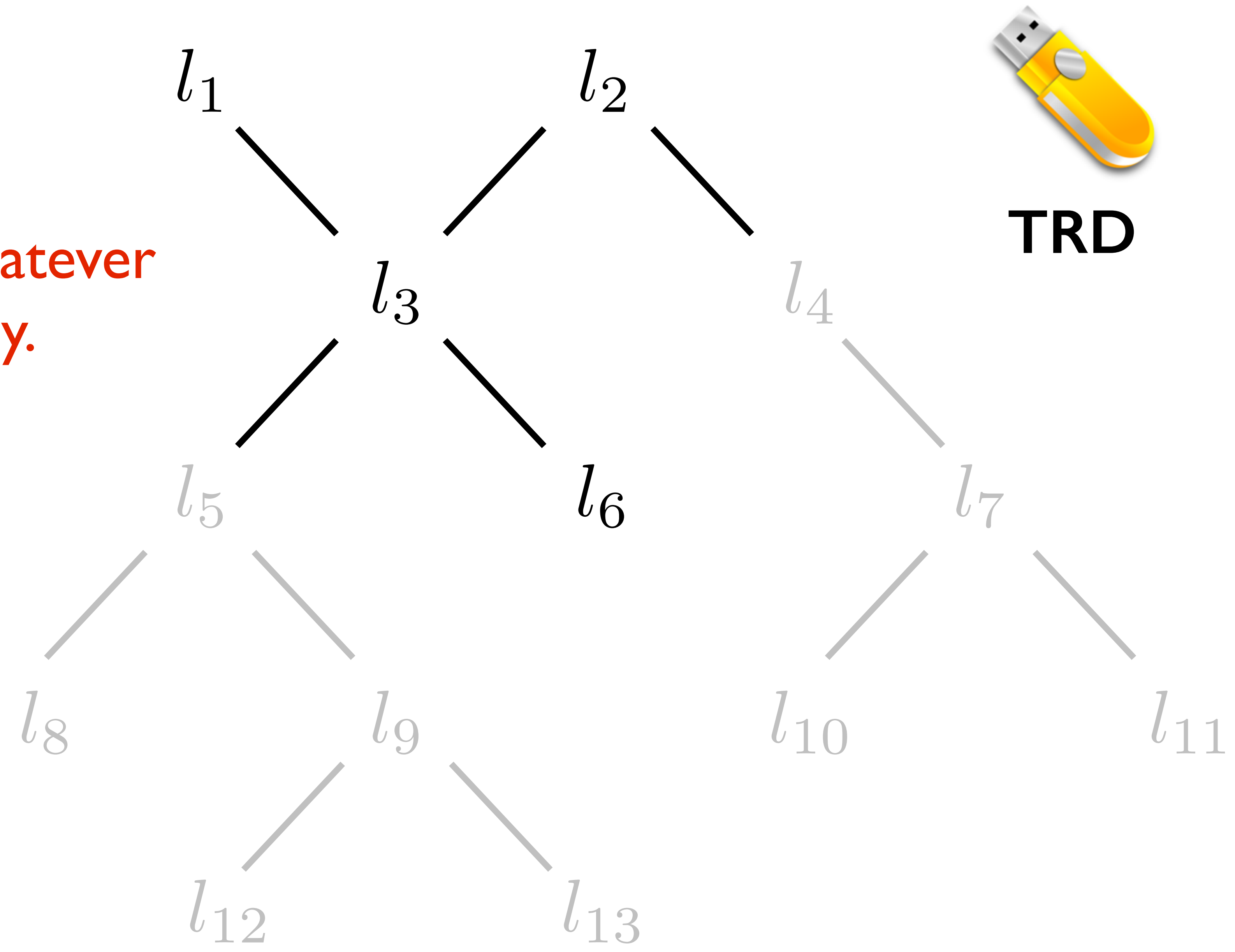
TRD

What about lost keys ?

The intruder has control over whatever is under a level with a lost key.



What about lost keys ?



The intruder has control over whatever is under a level with a lost key.

She may use an encrypt command to **get a key** with a lower level in a TRD containing a lost key.

Ex : Receive $\{\langle$ 🔑 $, l_9, v, m\rangle\}_{🔑}$ with 🔑 lost and of level l_5 .

Secrecy Result

«I keep my secrets secret !»

Even if the **intruder** may :

- **control the network** and host machines,
- **break some keys** (but not too many revocation keys),

Secrecy Result

«I keep my secrets secret !»

Even if the intruder may :

- control the network and host machines,
- break some keys (but not too many revocation keys),

We have :

Theorem 1

Keys remain **secret** (not deducible) provided :

A valid expiration date & not « under a lost »

Secrecy Result

«I keep my secrets secret !»

Formally speaking...

Theorem 1

Let $E = (\mathcal{P}, \mathcal{I}, \mathfrak{M}, N, K, t)$ be a global state, $\mathrm{L_V}$ a set of (broken) levels and $k \in K$.

$$\forall k \text{ s.t. } \mathsf{Level}(k) \not\leq \mathrm{L_V}, \quad \mathfrak{M} \not\vdash k$$

Secrecy Result

«I keep my secrets secret !»

Formally speaking...

Theorem 1

Let $E = (\mathcal{P}, \mathcal{I}, \mathfrak{M}, N, K, t)$ be a global state, $\mathrm{L_V}$ a set of (broken) levels and $k \in K$.

$$\forall k \text{ s.t. } \mathsf{Level}(k) \not\leq \mathrm{L_V}, \quad \mathfrak{M} \not\vdash k$$

Proof (sketch of):

> **Find invariant properties** of the system.

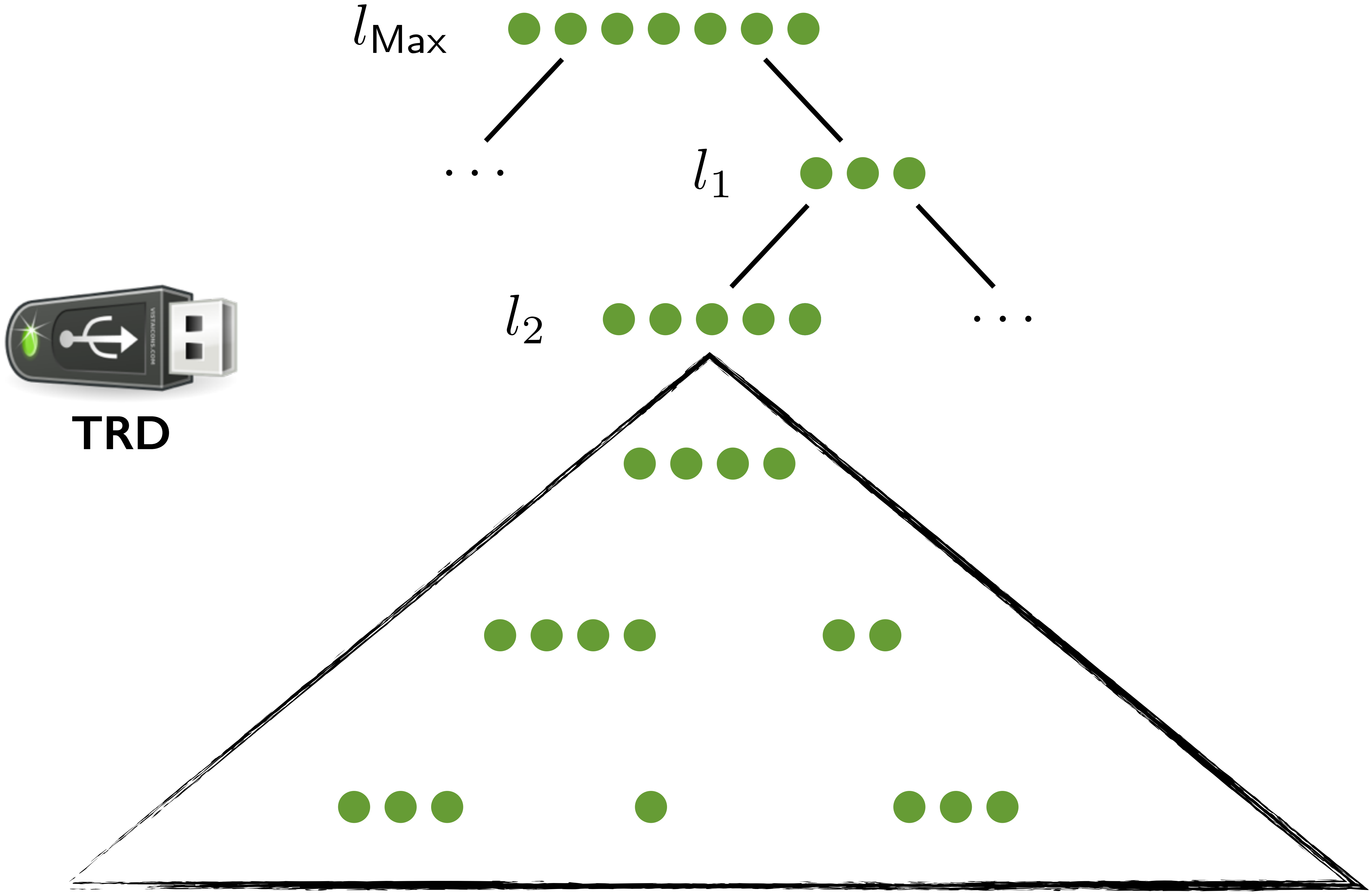
> **Prove** them !

Self Repair Property

«It’s just a flesh wound !»

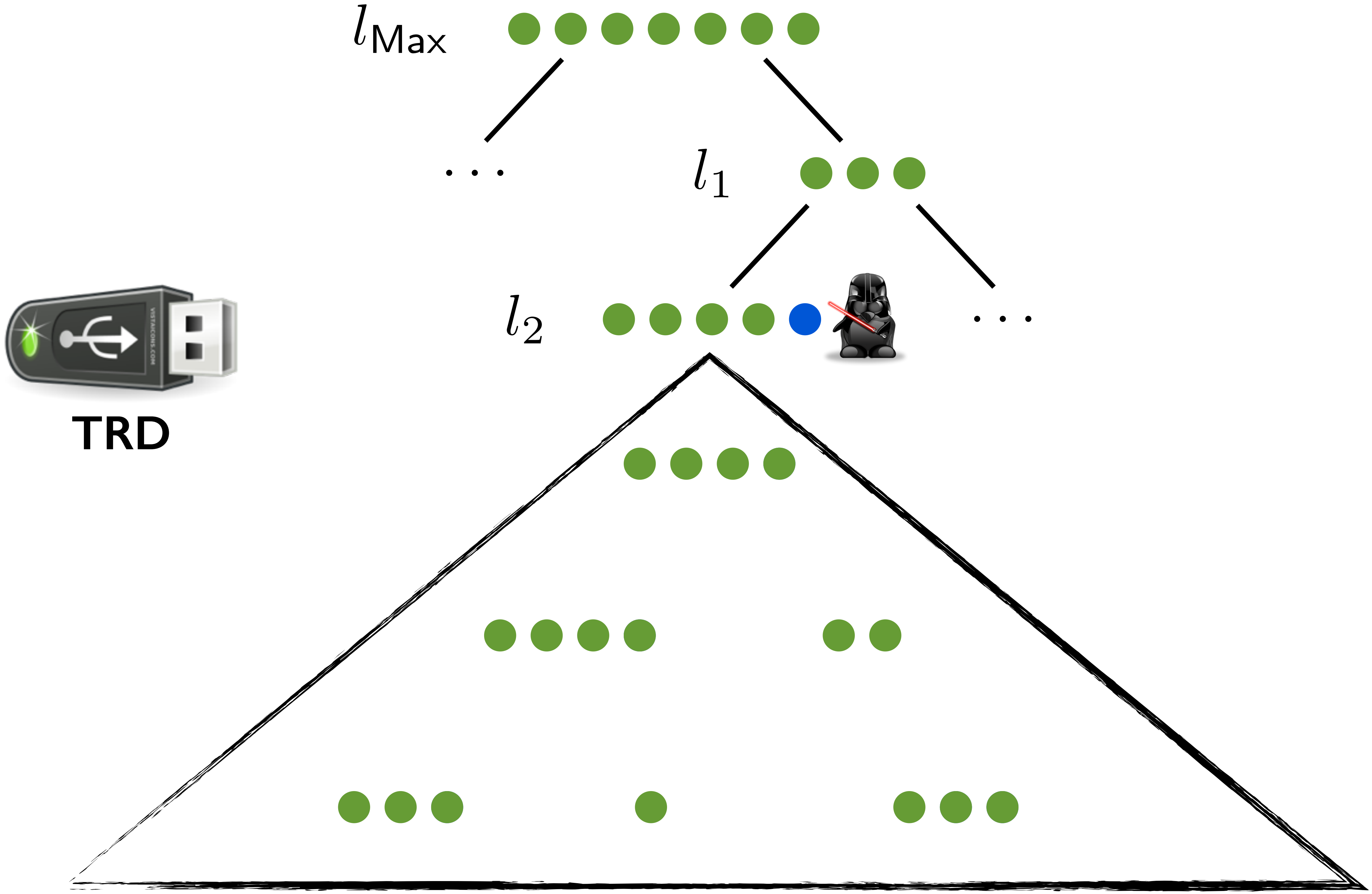
Self Repair Property

«It's just a flesh wound !»



Self Repair Property

«It's just a flesh wound !»



l_{Max}

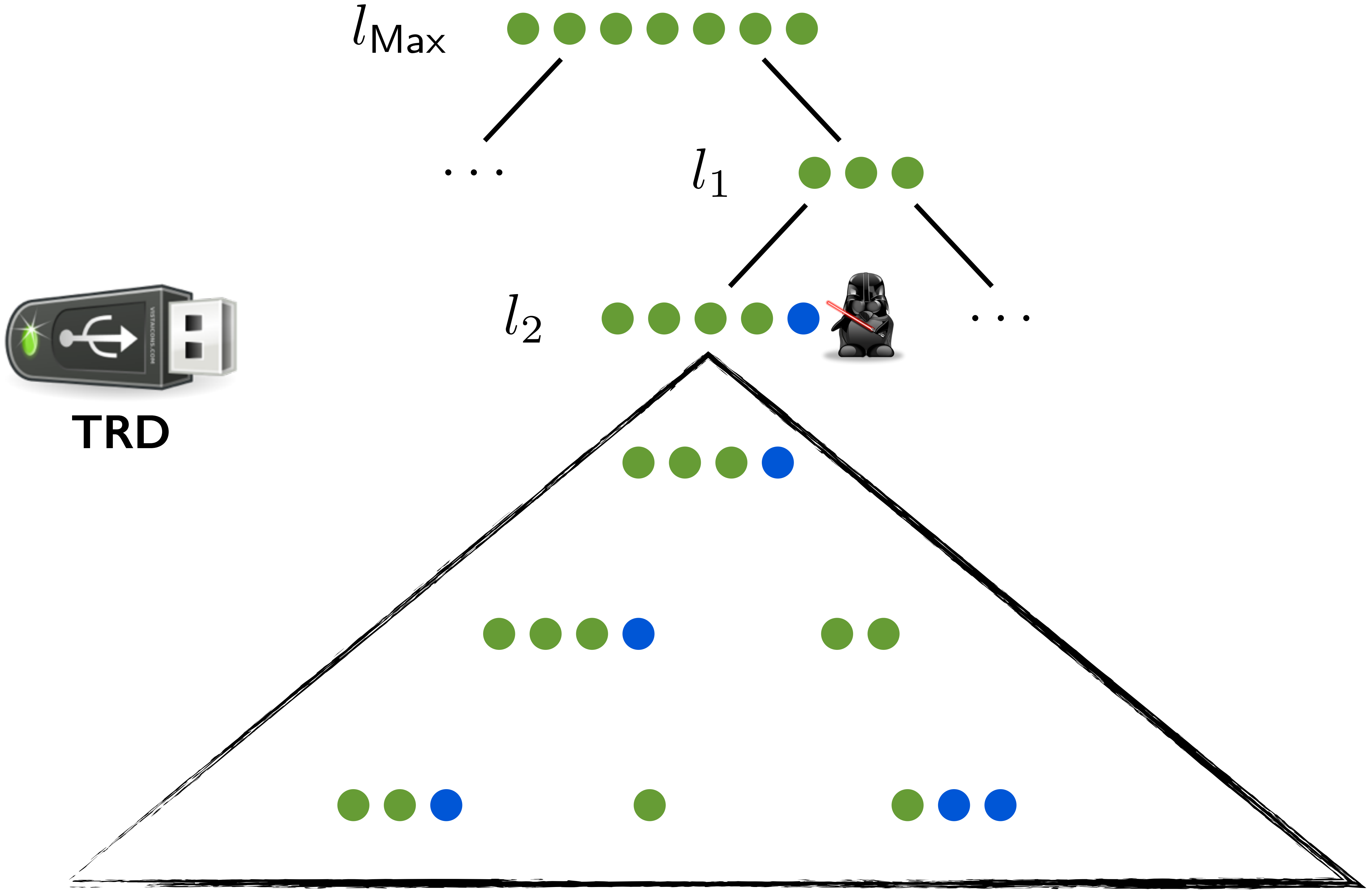
$\dots$ l_1

l_2 $\dots$

TRD

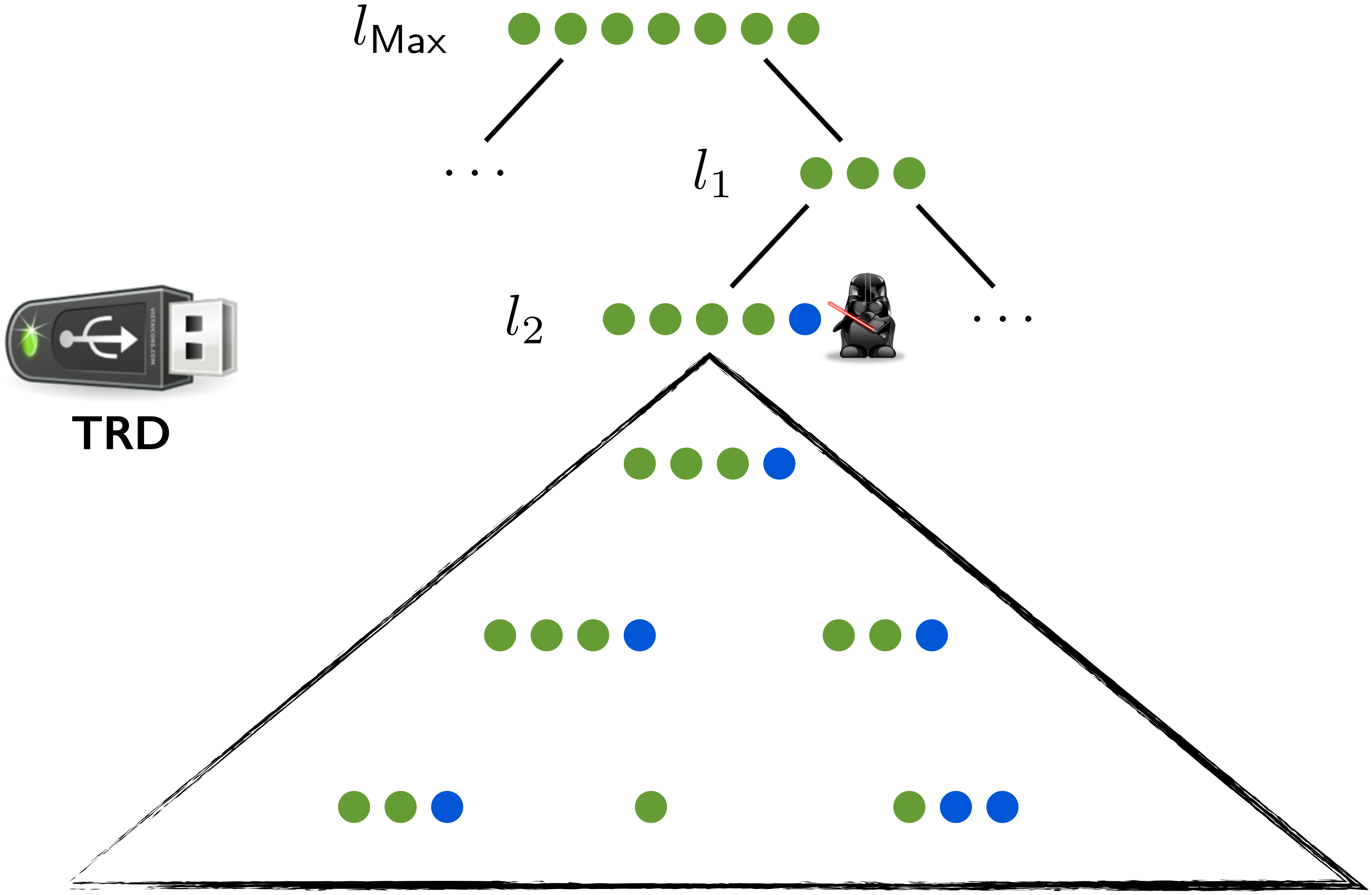
Self Repair Property

«It's just a flesh wound !»



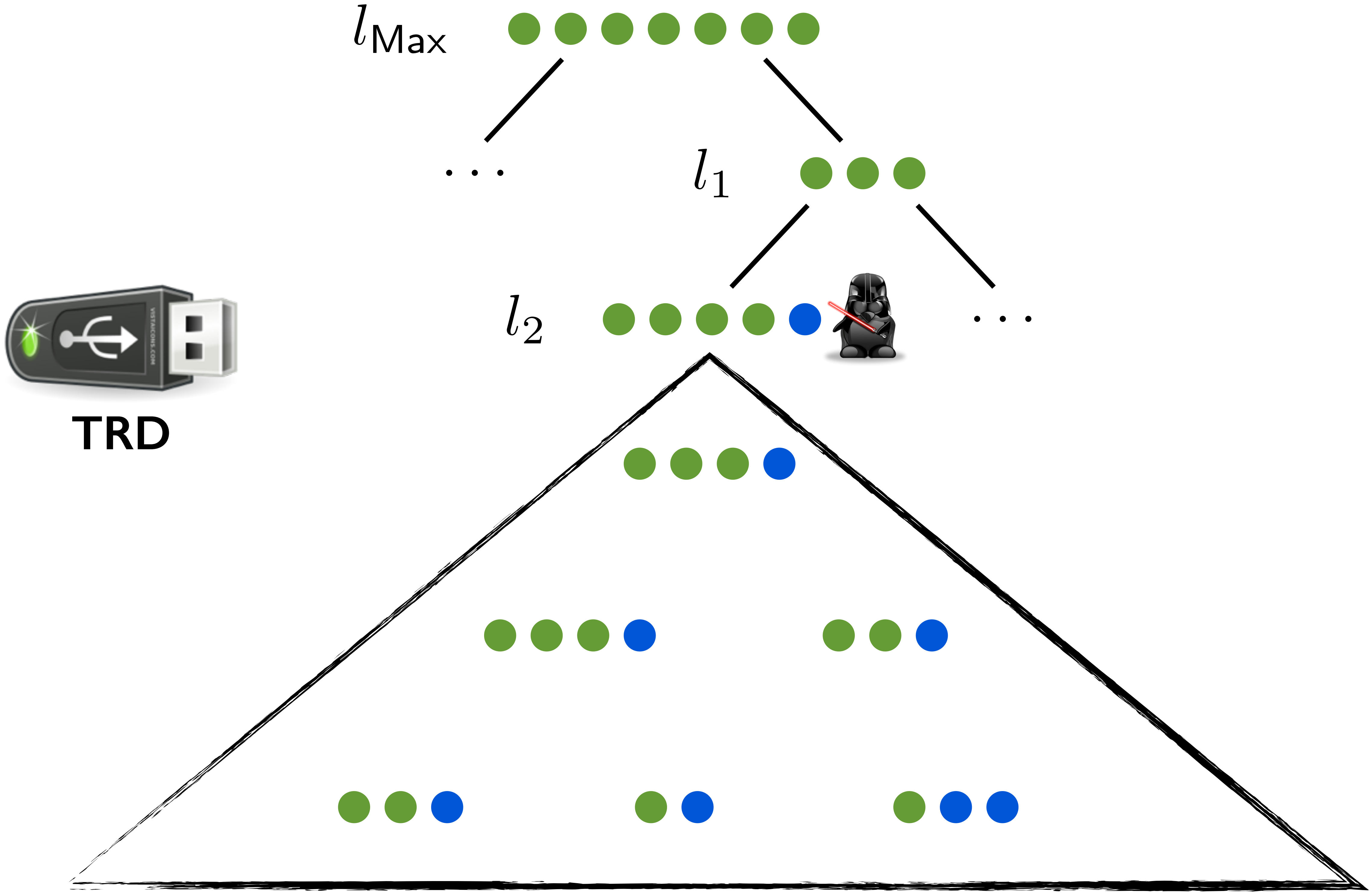
Self Repair Property

«It's just a flesh wound !»



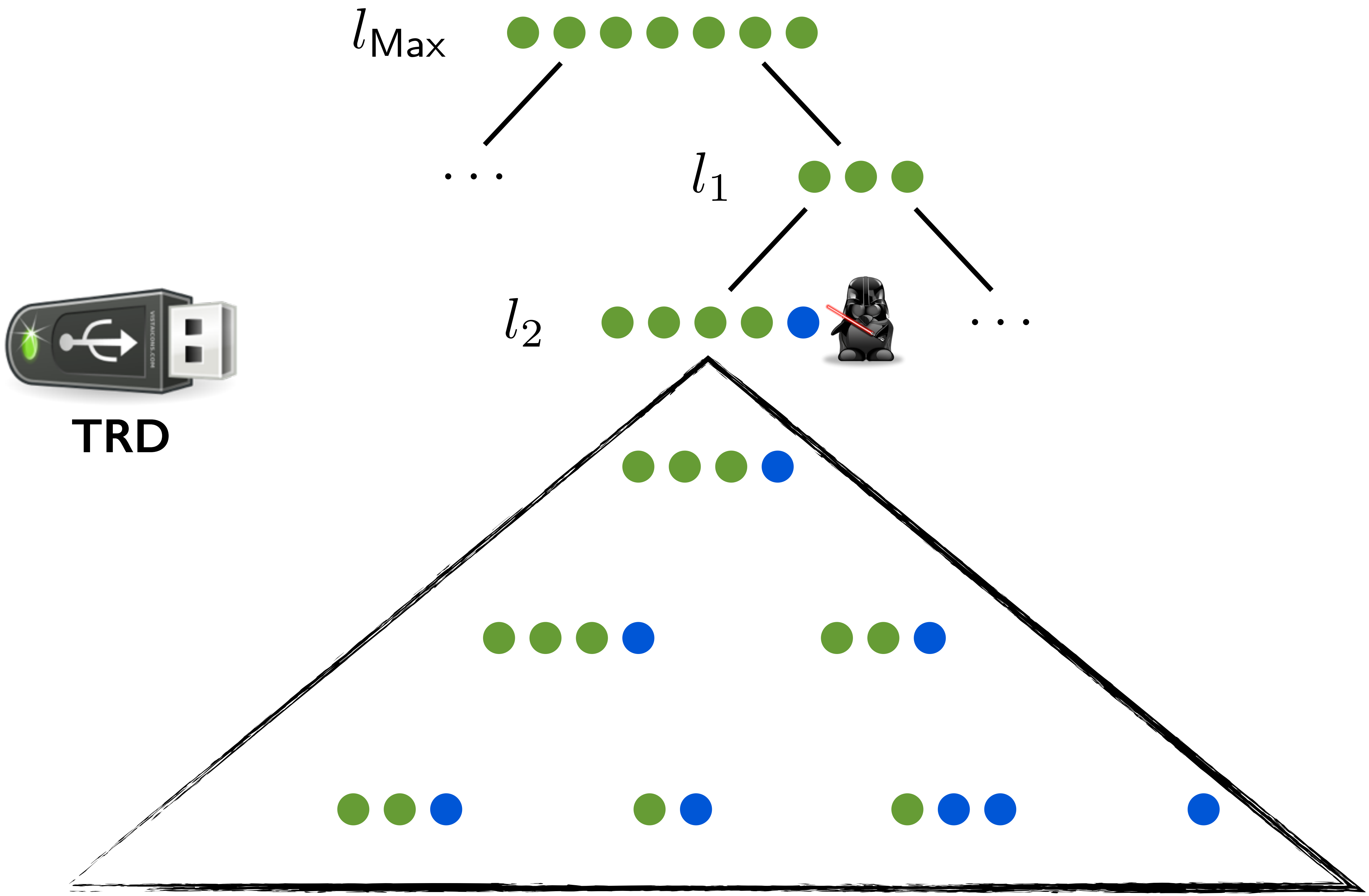
Self Repair Property

«It's just a flesh wound !»



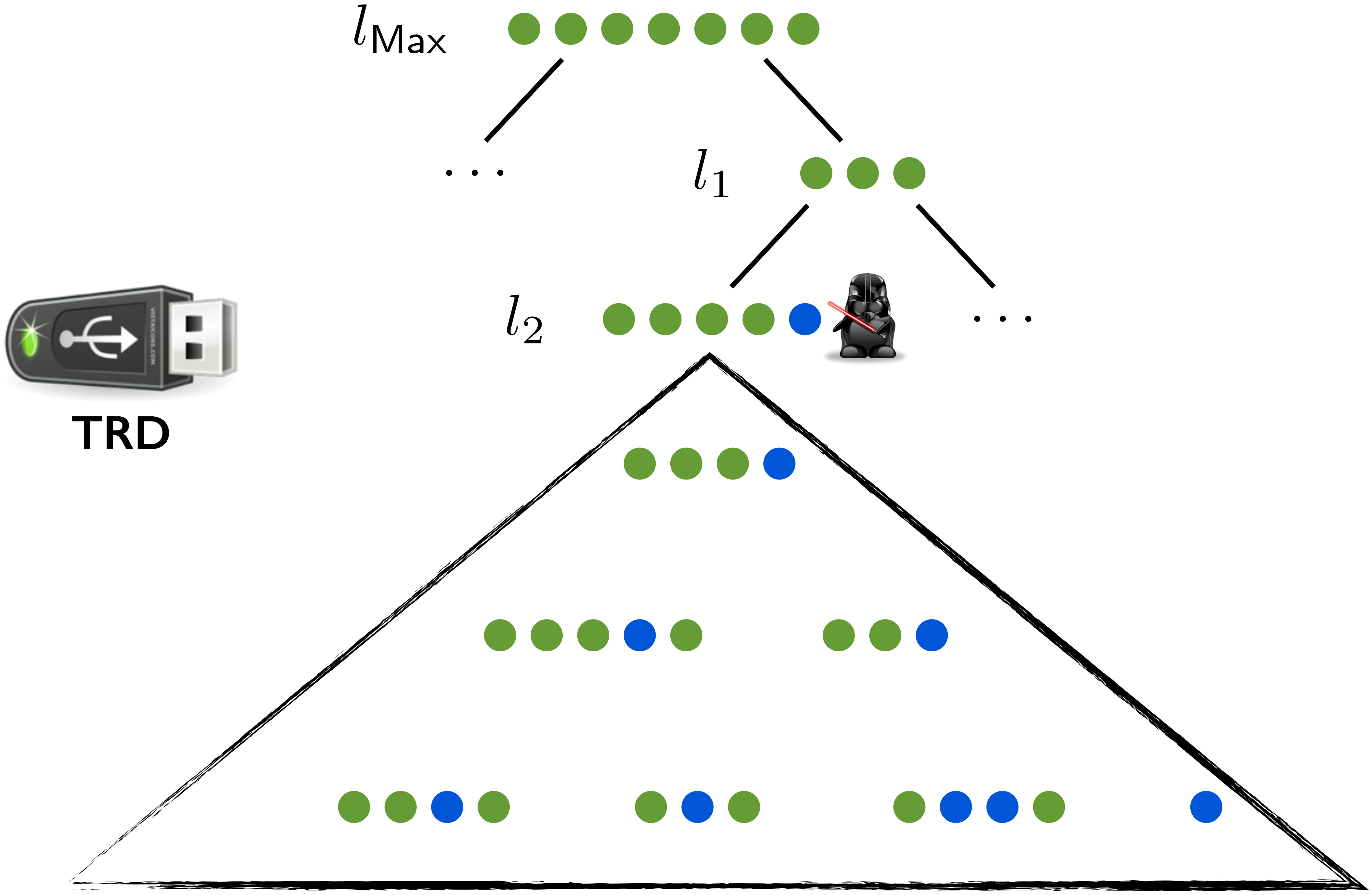
Self Repair Property

«It's just a flesh wound !»



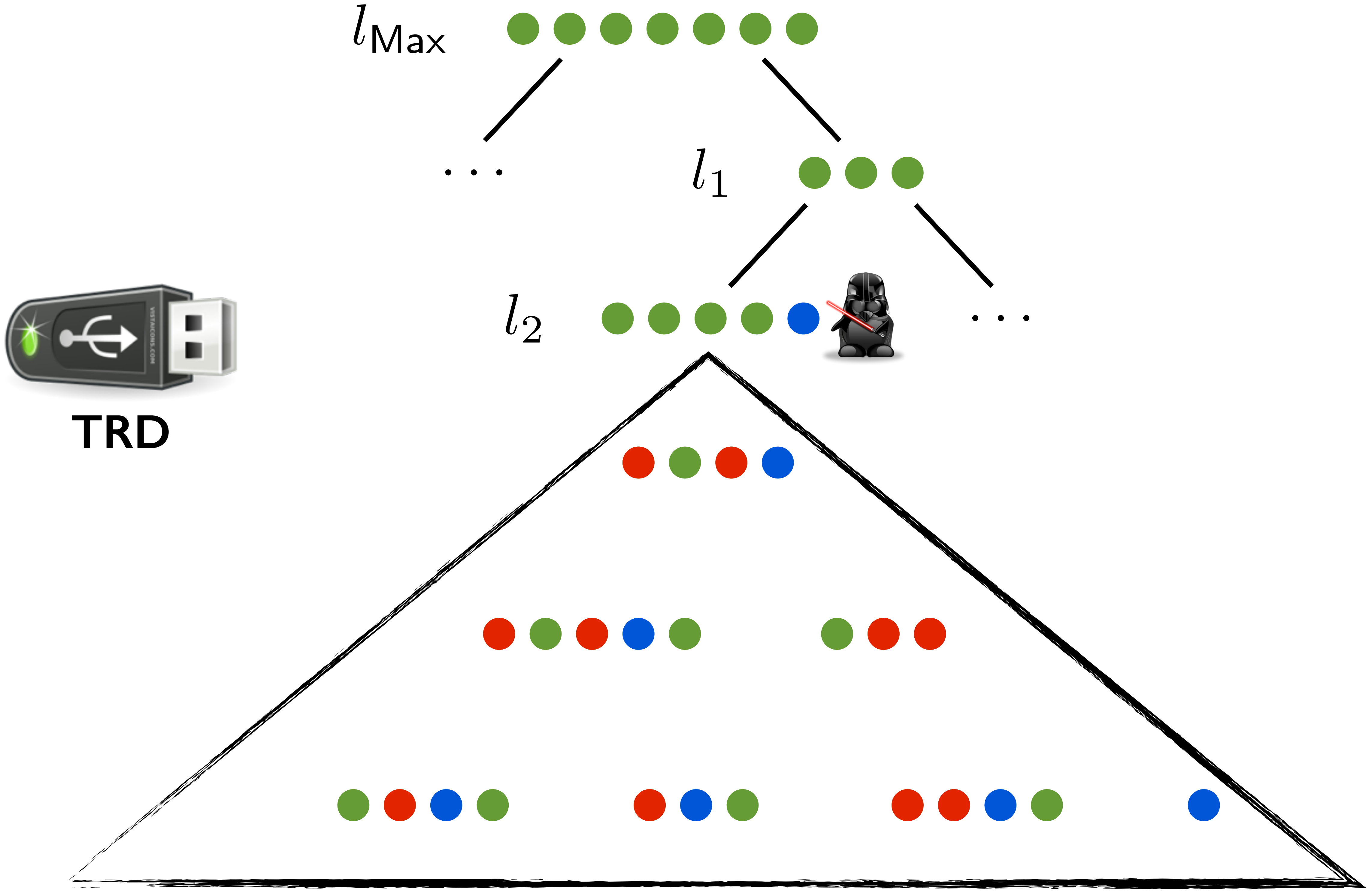
Self Repair Property

«It's just a flesh wound !»



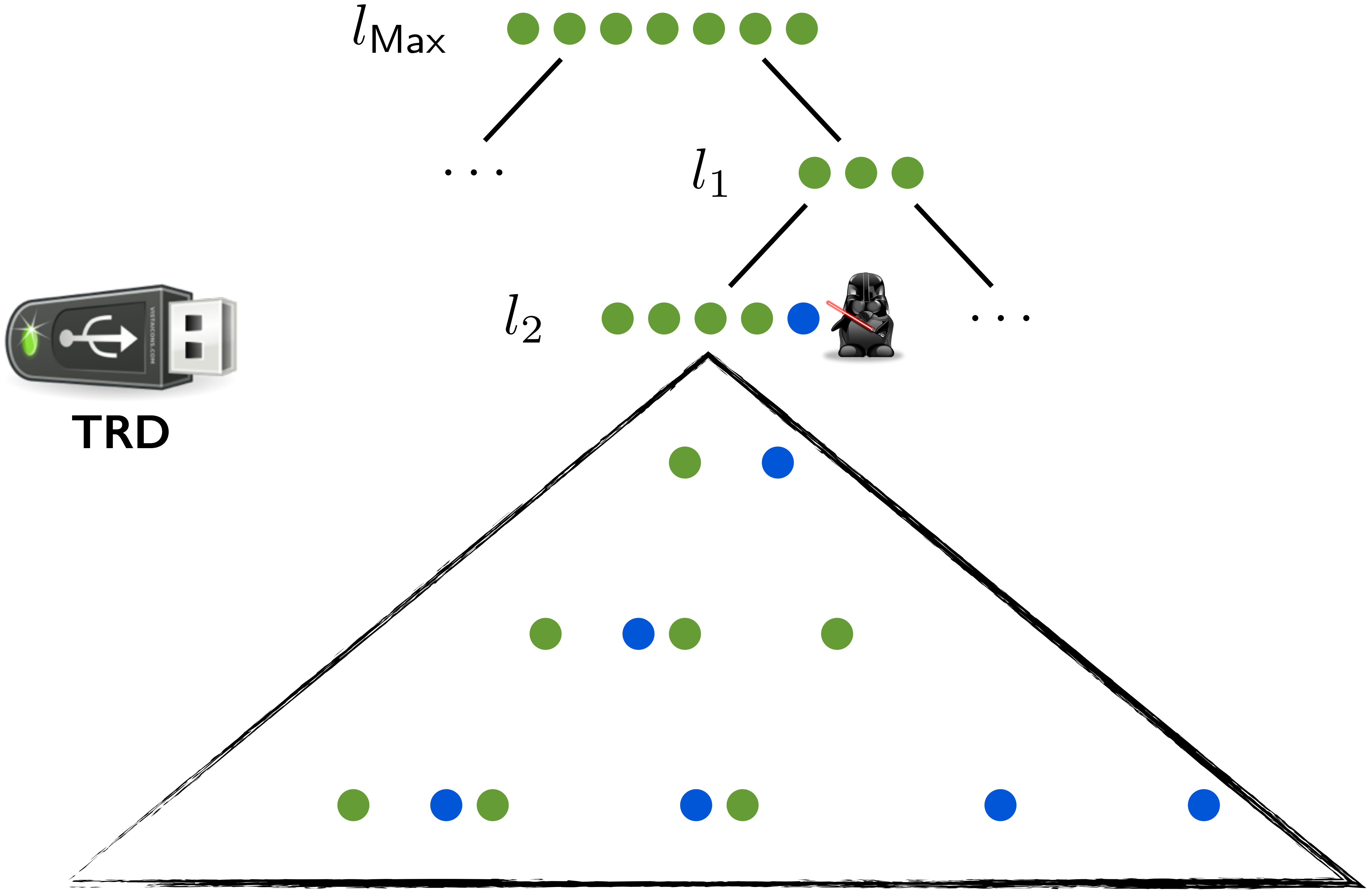
Self Repair Property

«It's just a flesh wound !»



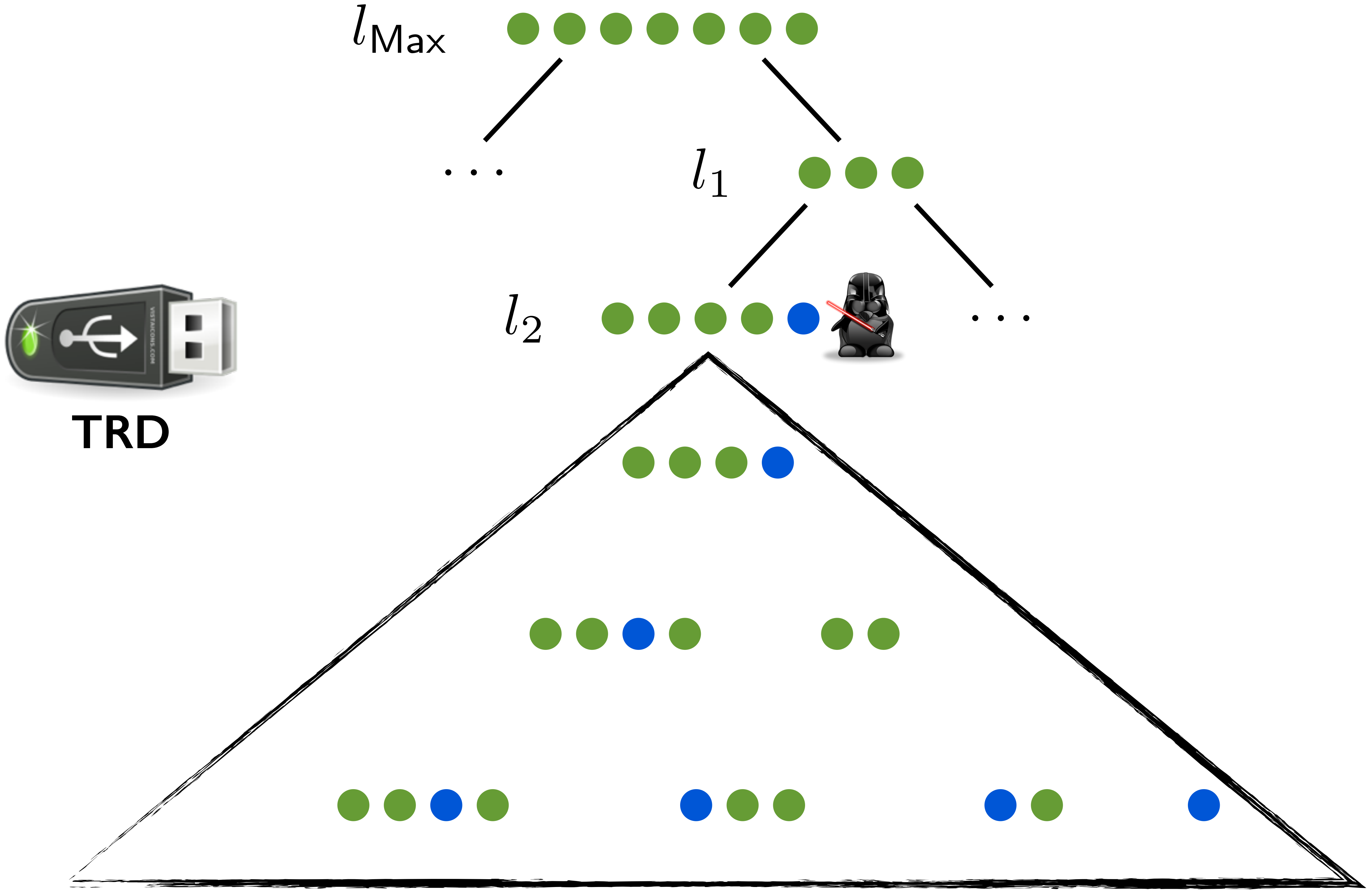
Self Repair Property

«It's just a flesh wound !»



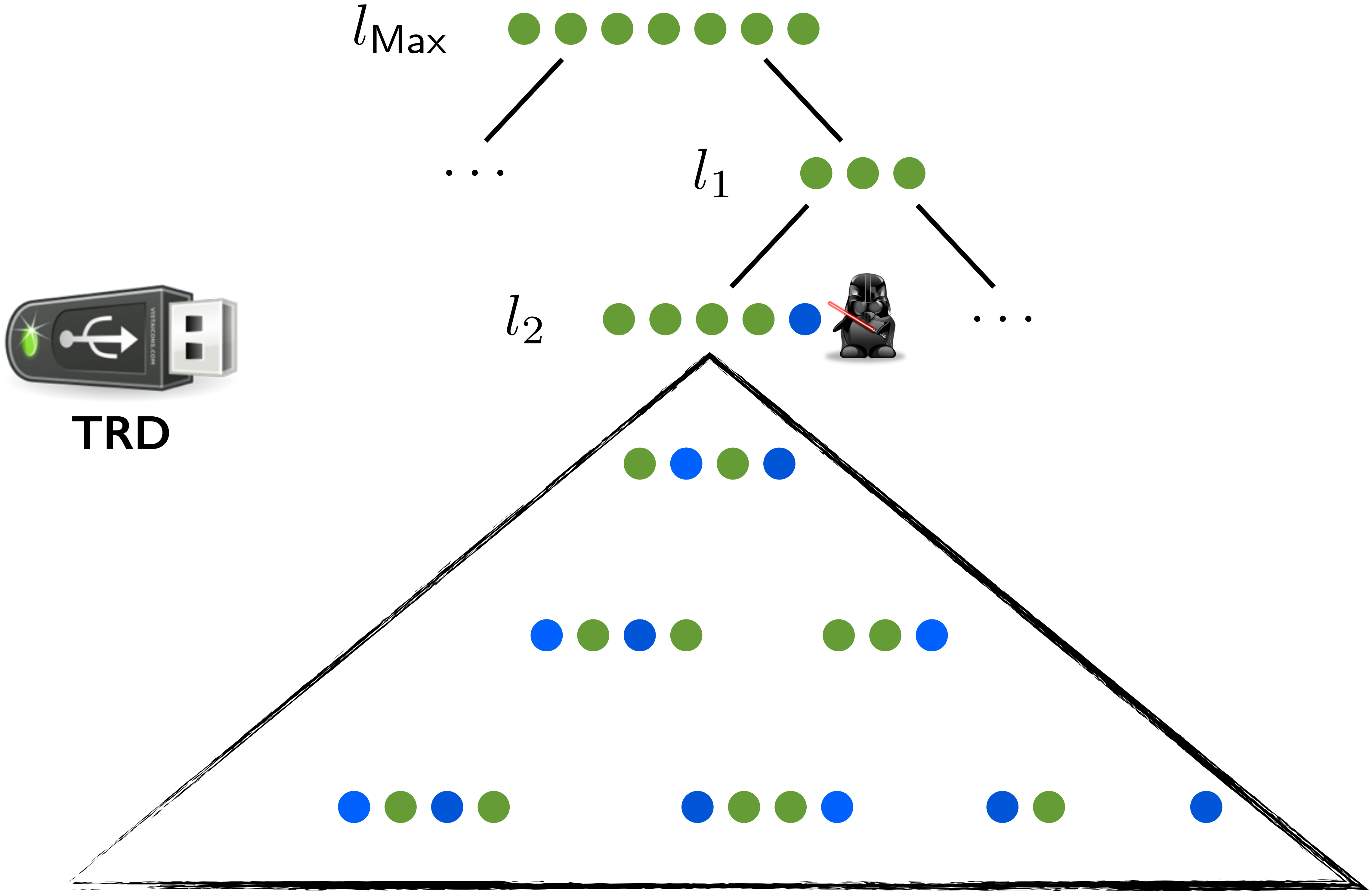
Self Repair Property

«It’s just a flesh wound !»



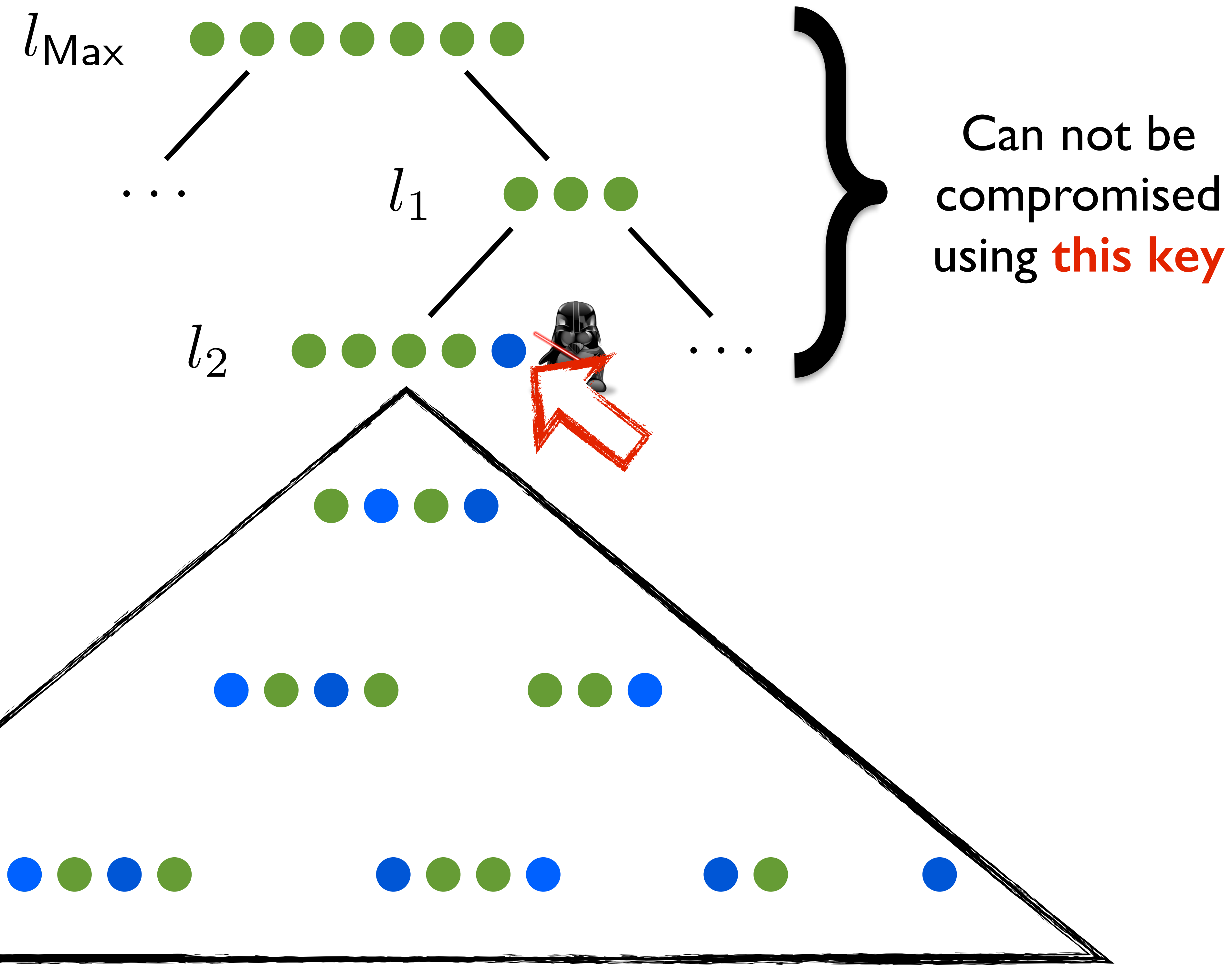
Self Repair Property

«It’s just a flesh wound !»



Self Repair Property

«It's just a flesh wound !»

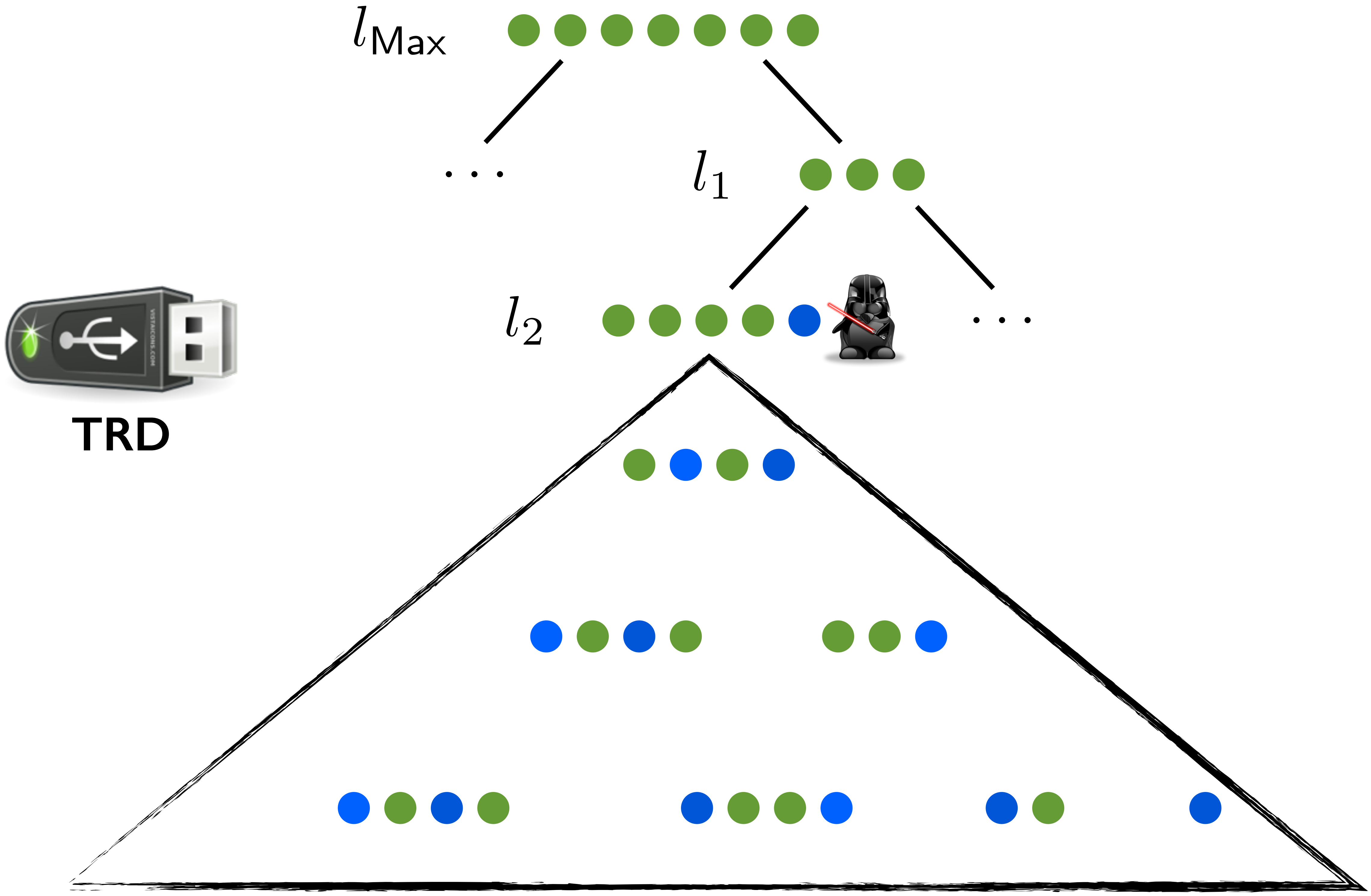


Can not be compromised using **this key**

TRD

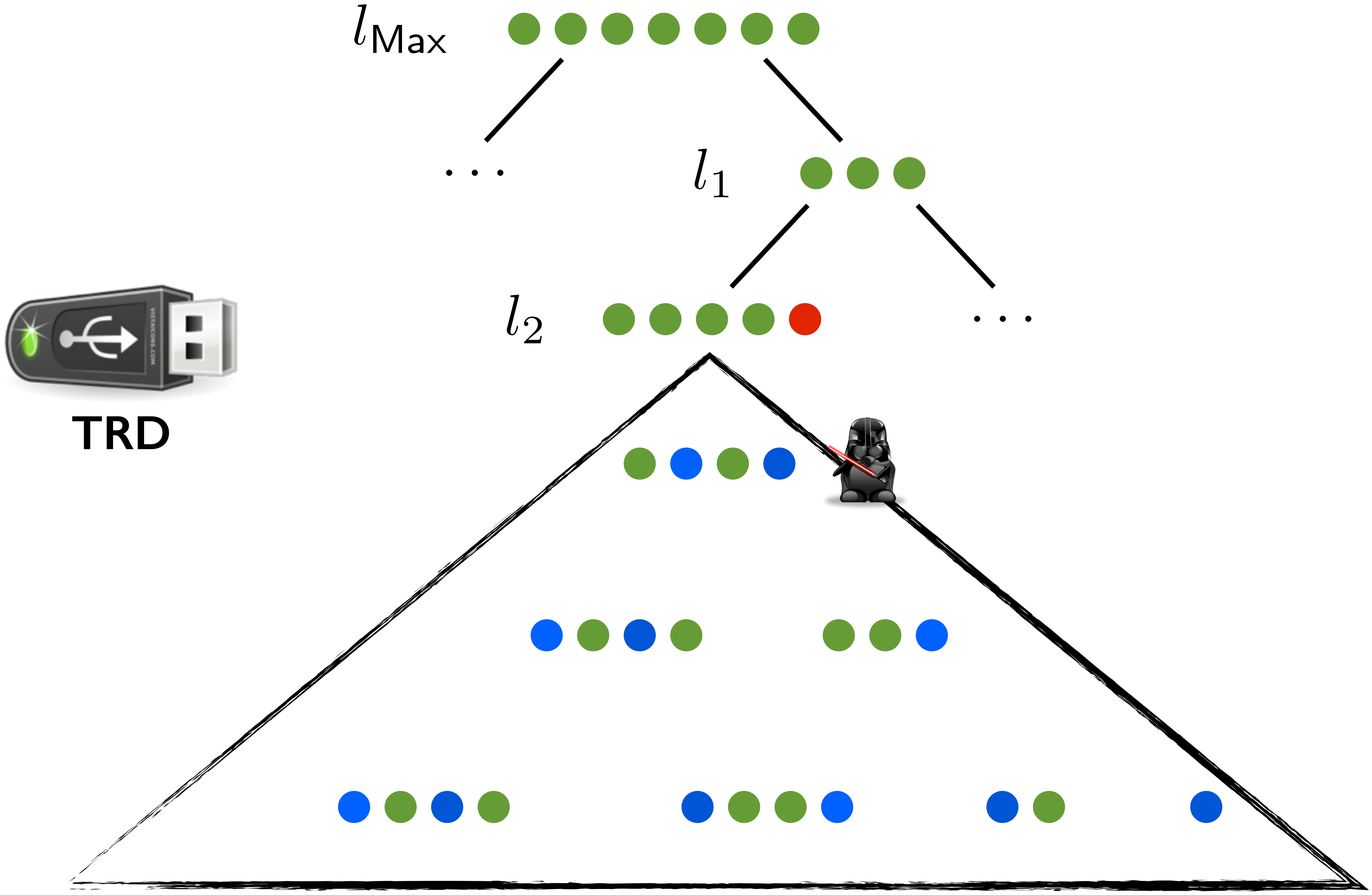
Self Repair Property

«It's just a flesh wound !»



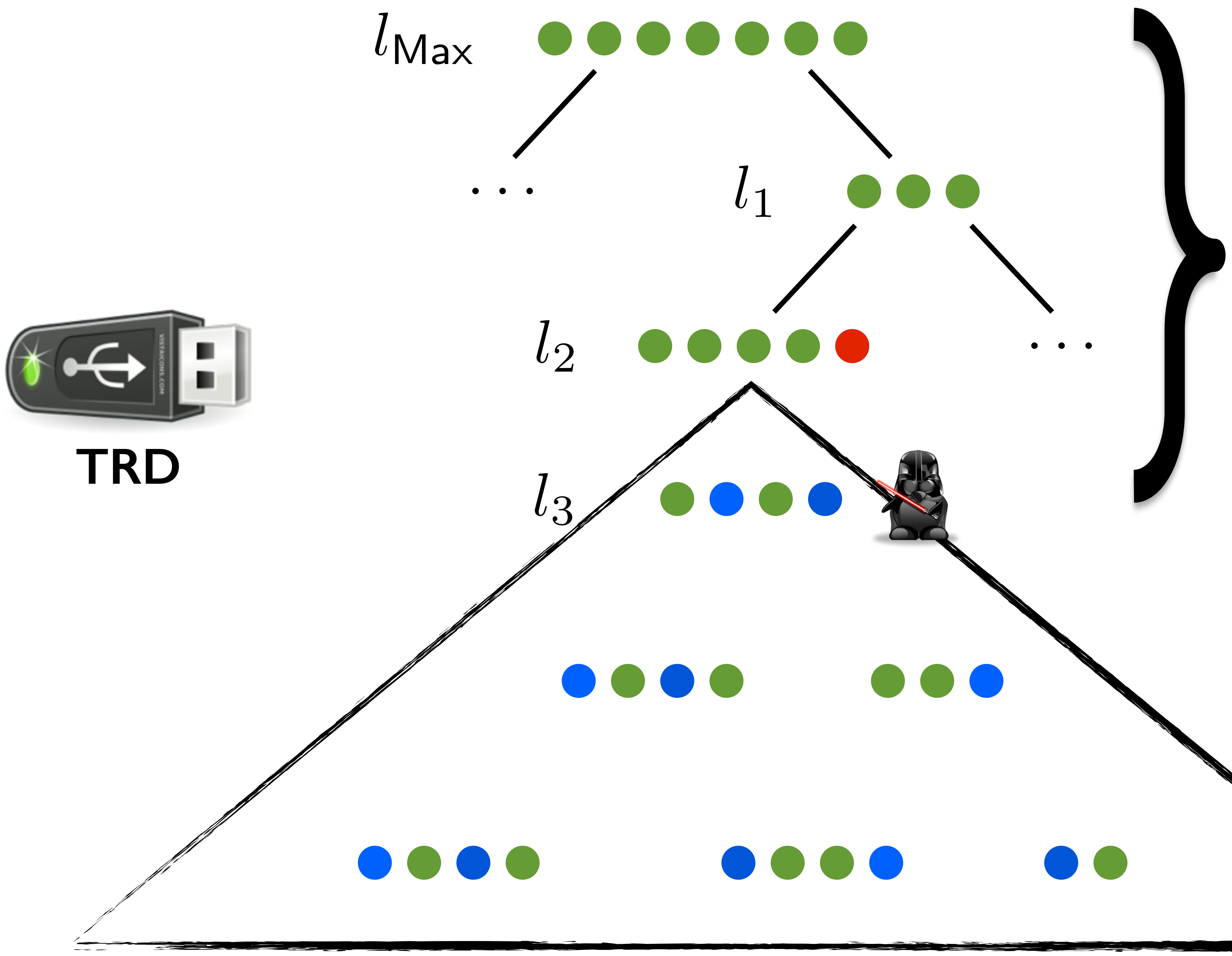
Self Repair Property

«It's just a flesh wound !»



Self Repair Property

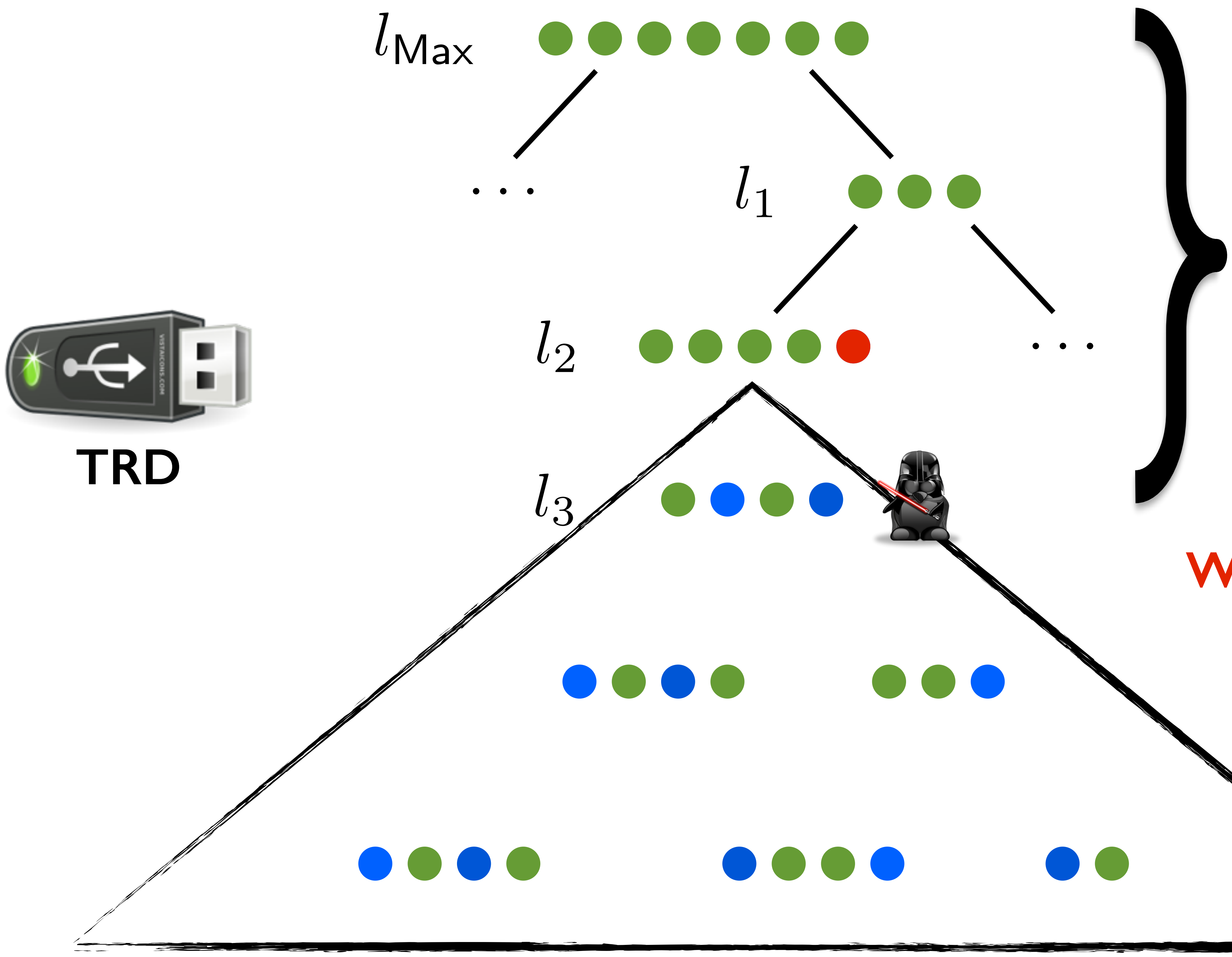
«It’s just a flesh wound !»



Can not be compromised using remaining corrupted keys.

Self Repair Property

«It's just a flesh wound !»

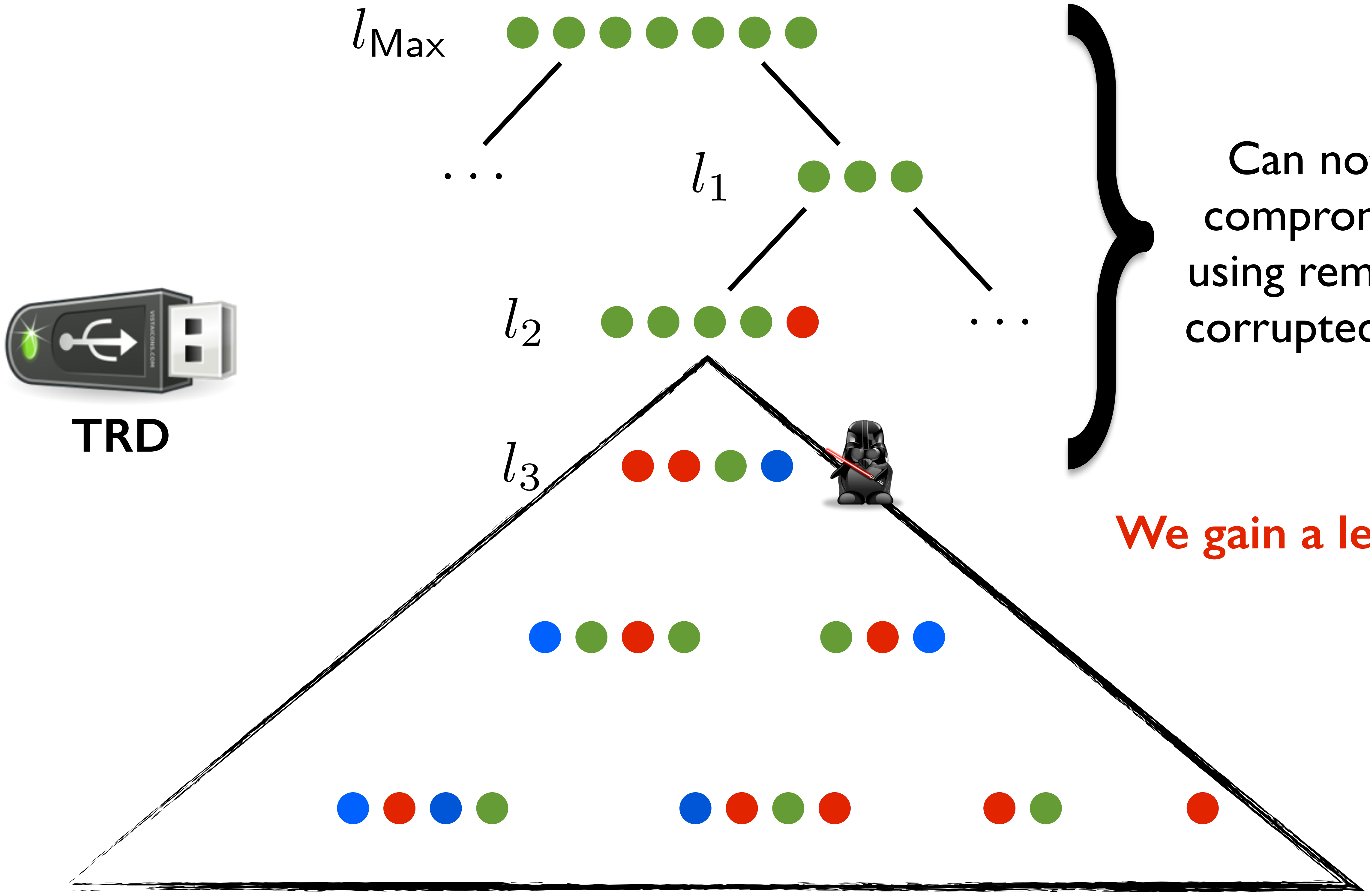


Can not be compromised using remaining corrupted keys.

We gain a level !

Self Repair Property

«It's just a flesh wound !»

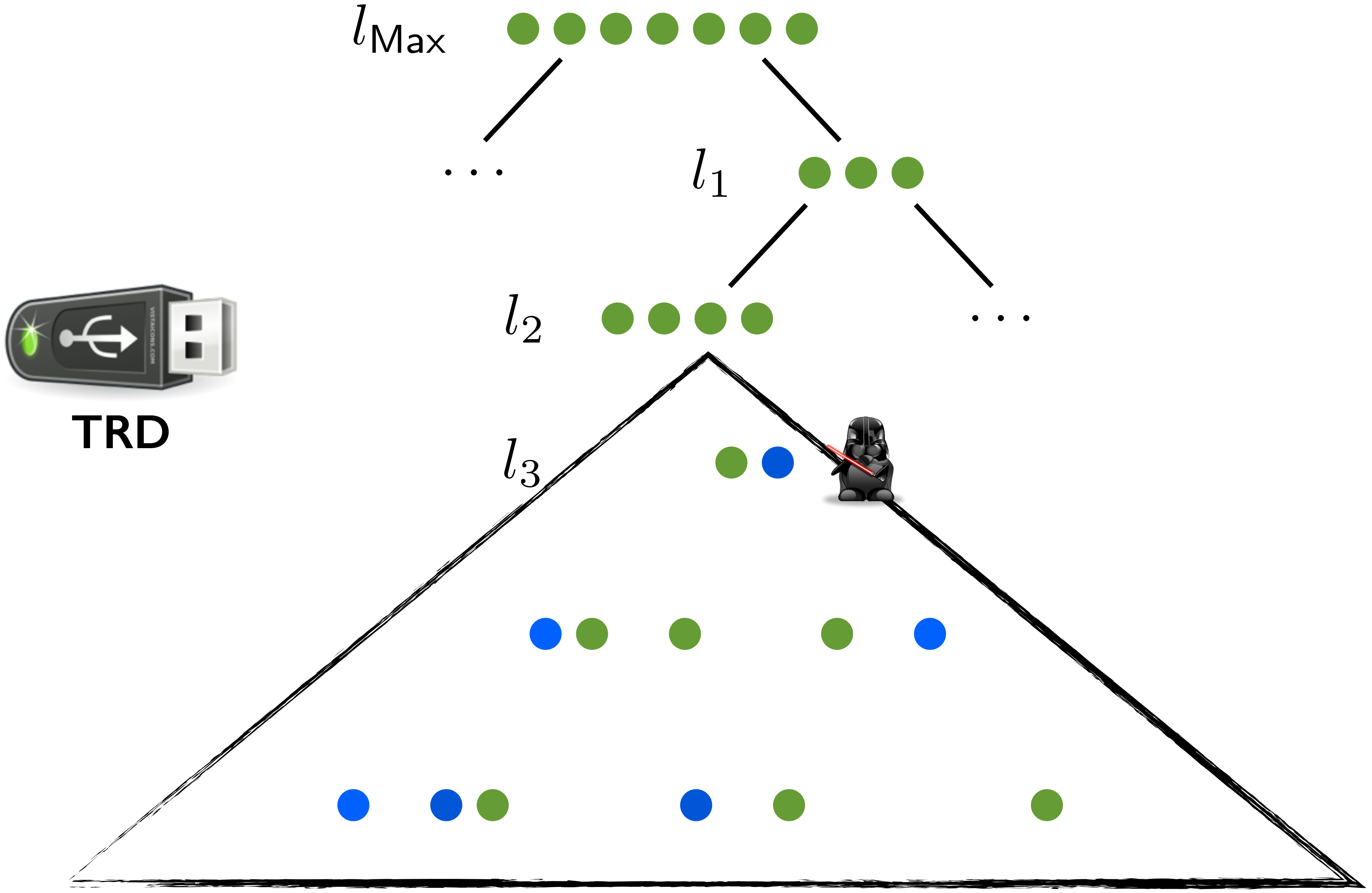


Can not be compromised using remaining corrupted keys.

We gain a level !

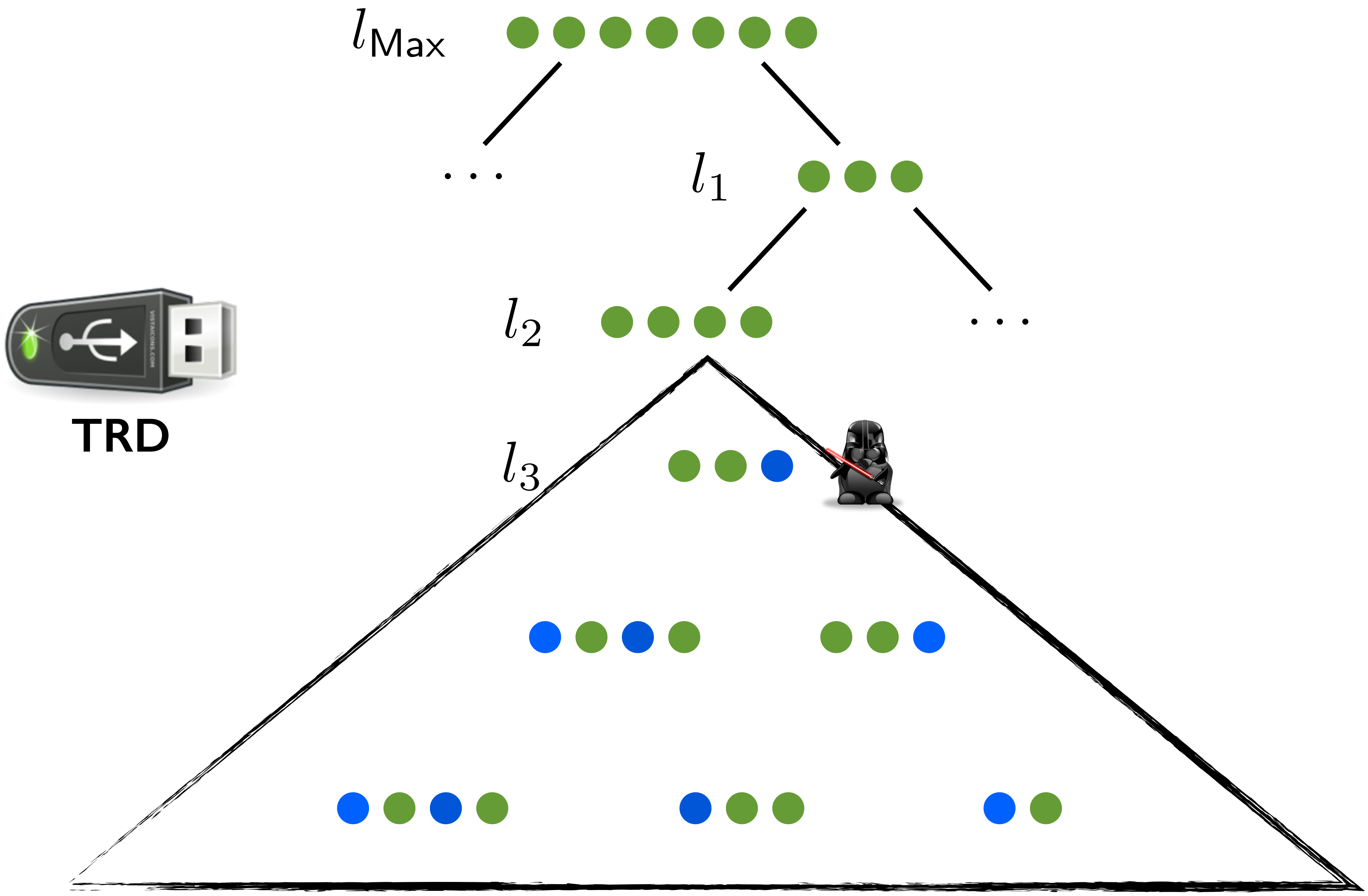
Self Repair Property

«It's just a flesh wound !»



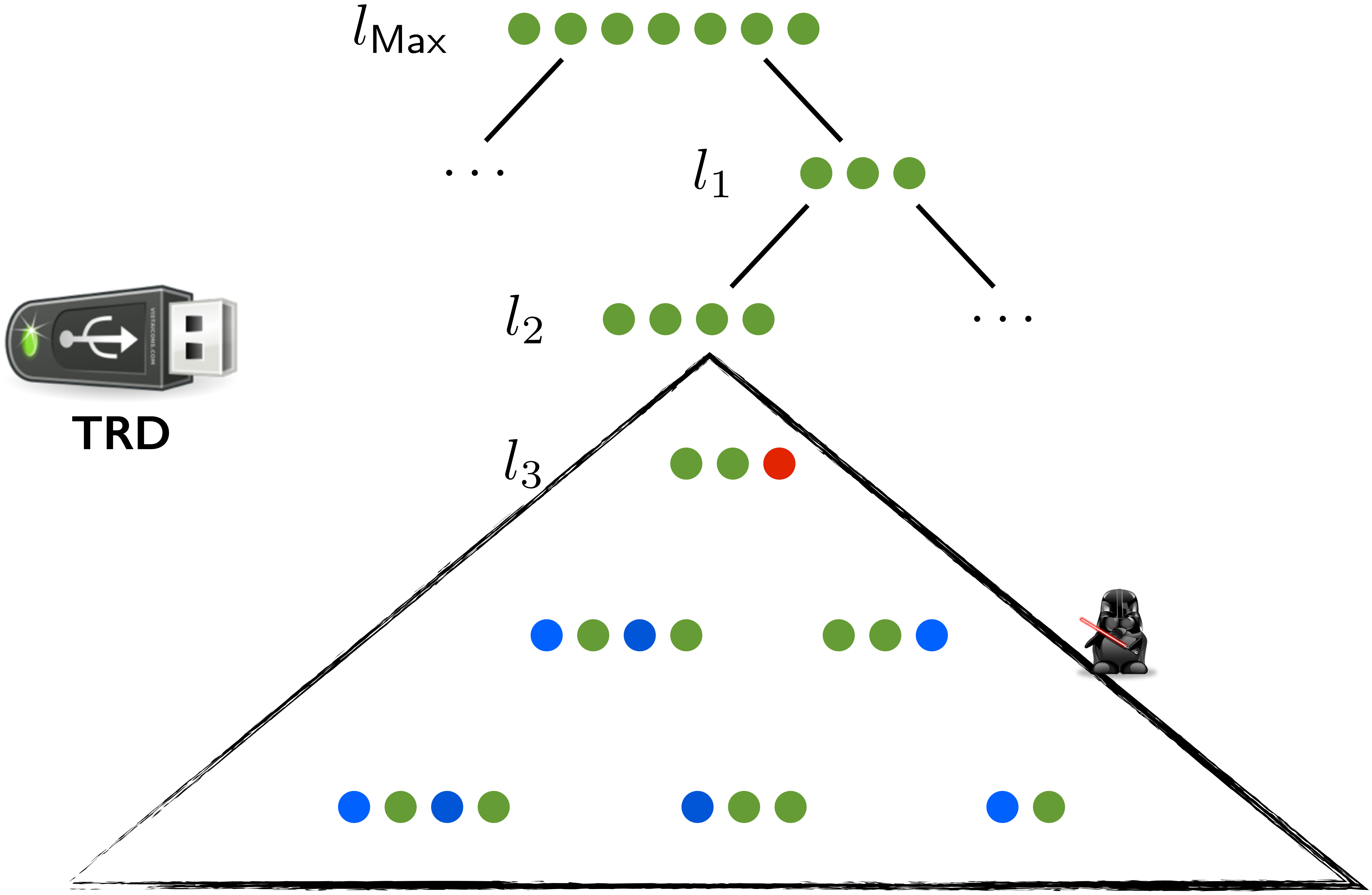
Self Repair Property

«It's just a flesh wound !»



Self Repair Property

«It's just a flesh wound !»



Self Repair Property

«It’s just a flesh wound !»

Then, the story went, until the TRD was fully repaired and it lived happily ever after...

Self Repair Property

«It’s just a flesh wound !»

Then, the story went, until the TRD was fully repaired and it lived happily ever after...

Theorem 2 (Stated for one level)

Assume that all keys are secret at time t except those under a level l.

Then at time $t + \Delta(l)$, all keys are secret except those under levels $l_1, \ldots, l_n$ such that $l_i < l$.

Self Repair Property

«It's just a flesh wound !»

Then, the story went, until the TRD was fully repaired and it lived happily ever after...

Theorem 2 (Stated for one level)

Assume that all keys are secret at time t except those under a level l.

Then at time $t + \Delta(l)$, all keys are secret except those under levels $l_1, \ldots, l_n$ such that $l_i < l$.

It assumes that, during time $\Delta(l)$, you **do not lose** a level higher than the one you «try» to repair.

Self Repair Property

«It’s just a flesh wound !»

Then, the story went, until the TRD was fully repaired and it lived happily ever after...

Theorem 2 (Stated for one level)

Assume that all keys are secret at time t except those under a level l.

Then at time $t + \Delta(l)$, all keys are secret except those under levels $l_1, \ldots, l_n$ such that $l_i < l$.

It assumes that, during time $\Delta(l)$, you **do not lose** a level higher than the one you «try» to repair.

ALL APIS REPAIRED AT ONCE

Blacklist Option

«For those who are in a hurry...»

$\mathsf{blacklist}(C, h_1, \ldots, h_n)$

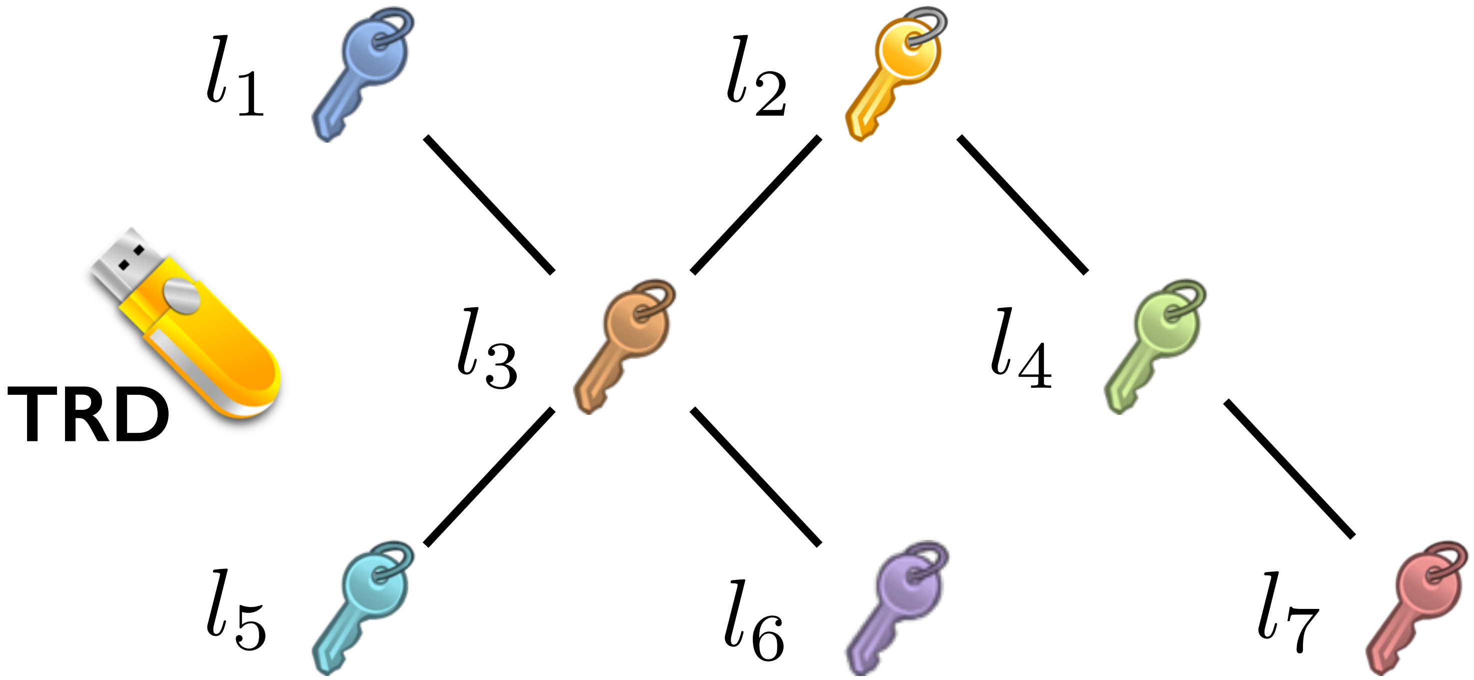
Ex : $C = \left\{ \langle \mathsf{blacklist}, \langle l_3, t \rangle \rangle \right\}$ 🔑 $\ldots$ 🔑

Blacklist Option

«For those who are in a hurry...»

blacklist$(C, h_1, \ldots, h_n)$

Ex : $C = \left\{ \langle \text{blacklist}, \langle l_3, t \rangle \rangle \right\}$

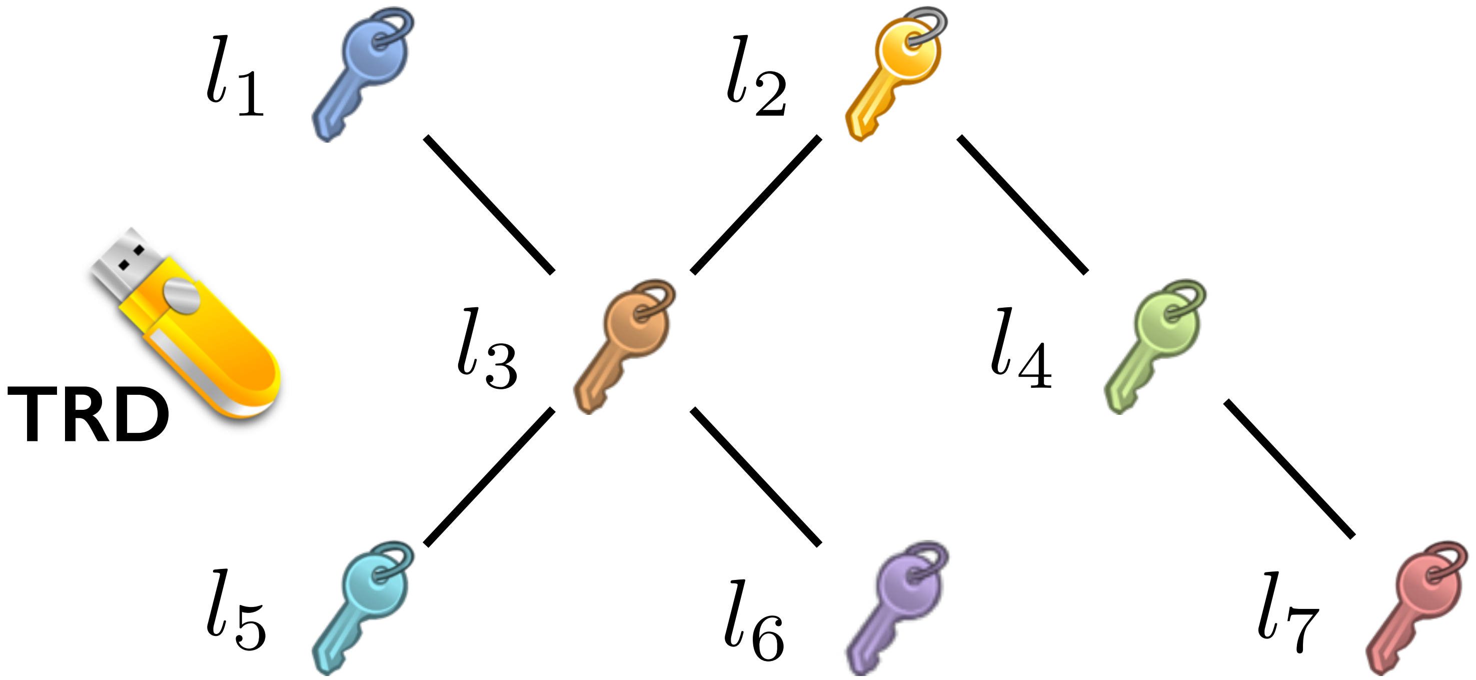


Blacklist Option

«For those who are in a hurry...»

$\text{blacklist}(C, h_1, \ldots, h_n)$

Ex : $C = \Big\{ \langle \text{blacklist}, \langle l_3, t \rangle \rangle \Big\}$



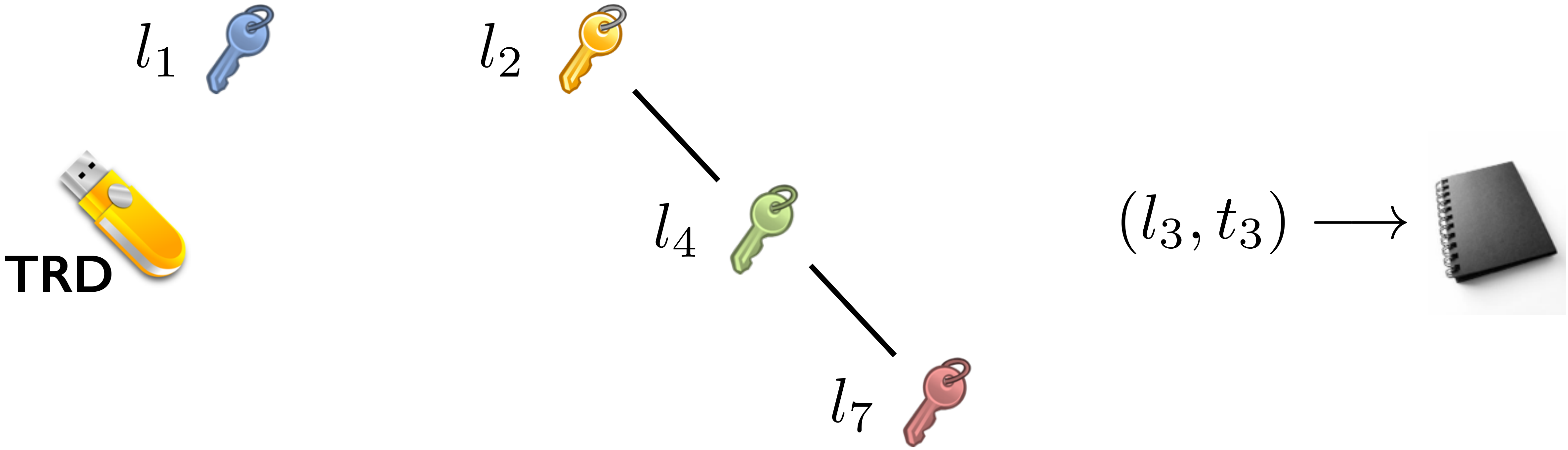
$(l_3, t_3) \longrightarrow$

Blacklist Option

«For those who are in a hurry...»

$\text{blacklist}(C, h_1, \ldots, h_n)$

Ex : $C = \left\{ \langle \text{blacklist}, \langle l_3, t \rangle \rangle \right\} \ldots$



Blacklist Option

«For those who are in a hurry...»

Theorem 3 (Stated for one level)

Assume that all keys are secret at time t except those under a level l.

If we blacklist level l on a TRD , then, **immediately**, all keys are secret.

Blacklist Option

«For those who are in a hurry...»

Theorem 3 (Stated for one level)

Assume that all keys are secret at time t except those under a level l.

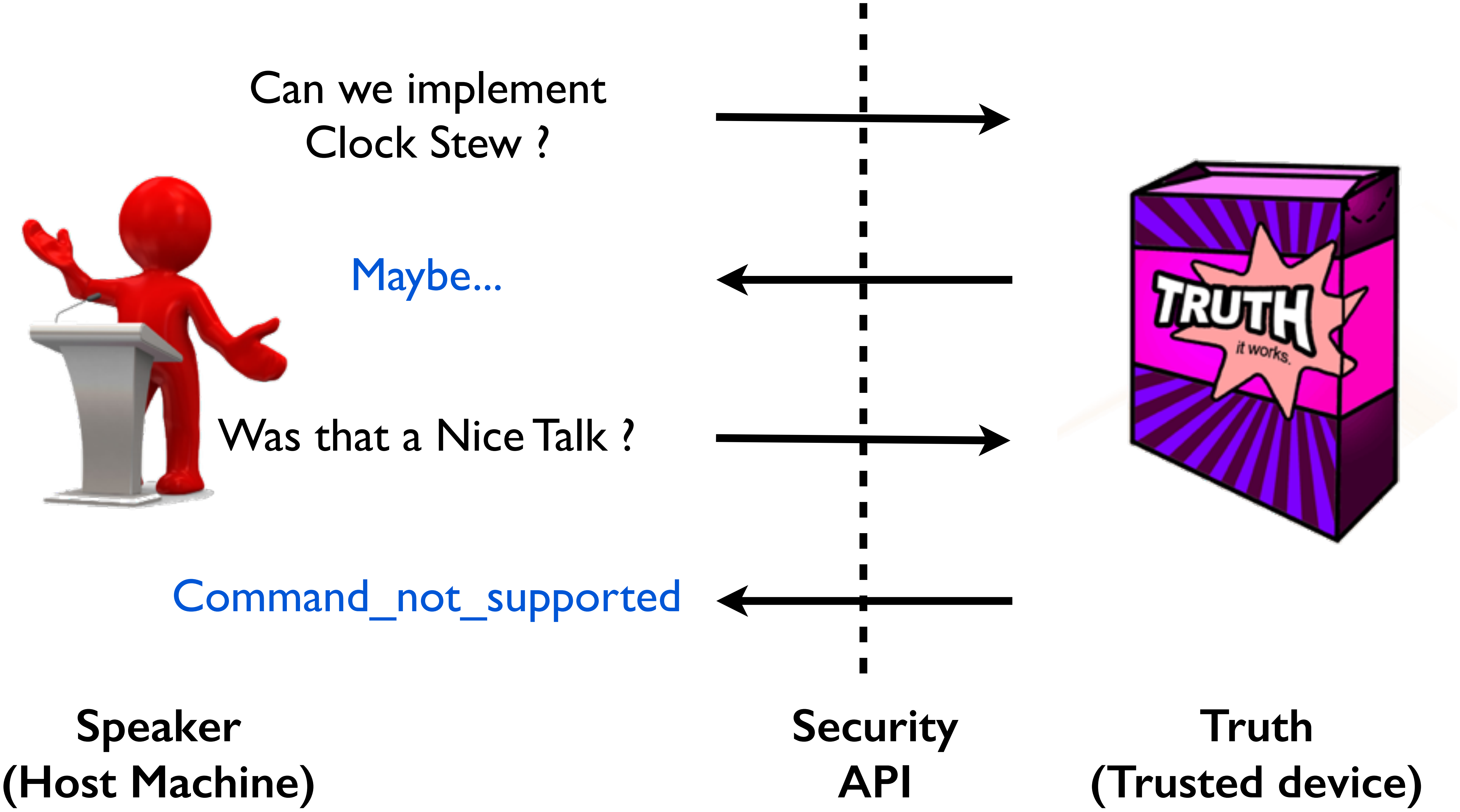
If we blacklist level l on a TRD , then, **immediately**, all keys are secret.

- It **only works** for the blacklisted TRD.
- The time of the blacklist should be long enough.
- It **prevents the attacker** to operate on the TRD.

Future Work

- **Weaken assumptions**, especially on hidden level Max messages (maybe requiring more cryptographic primitives),
- **Extend** revocation to **asymmetric encryption**,
- **Adapt** the result taking account of possible **clock skew**, or replacing the clock by some sort of nonce based freshness test,
- **Implement** the API in order to carry out some performance tests. [Ongoing work in JavaCard]

Thank you for your attention !



Can we implement Clock Stew ?

Maybe...

Was that a Nice Talk ?

Command_not_supported

Speaker
(Host Machine)

Security
API

Truth
(Trusted device)