ORAMFS:
Achieving Storage-Agnostic Privacy

Nils Amiet, Tommaso Gagliardoni

July 7, 2021

KUDELSKI SECURITY

Who am I?

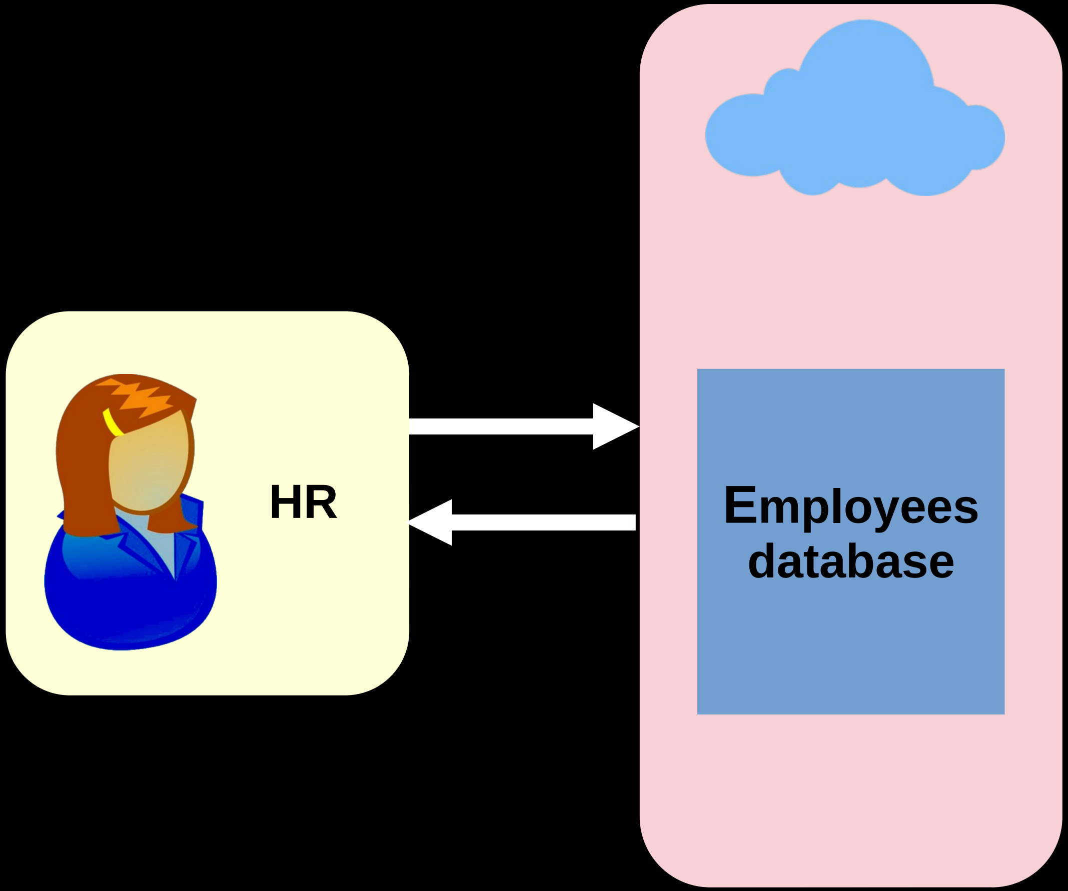
- Nils Amiet
- Research team @ KUDELSKI SECURITY
- Main tech interests:
  - Open source software
  - Big data analytics
  - Modern programming languages

Who am I?

- Tommaso Gagliardoni
- Research team @ KUDELSKI SECURITY
- Main tech interests:
  - Cryptography
  - Quantum computing & quantum security
  - Anonymity and Privacy

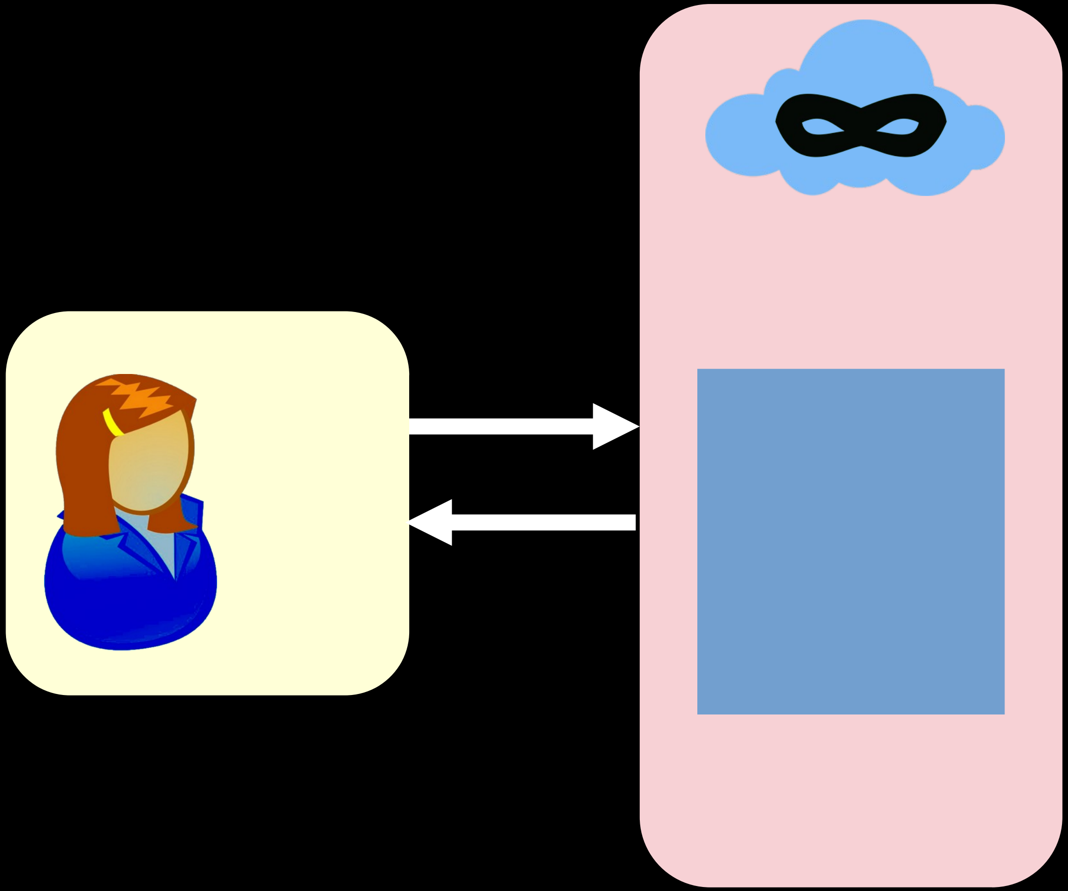
Once Upon a Time… Encrypted Data at Rest

- HR dept. keeps database of employees’ salary records on public cloud (AWS/Azure)



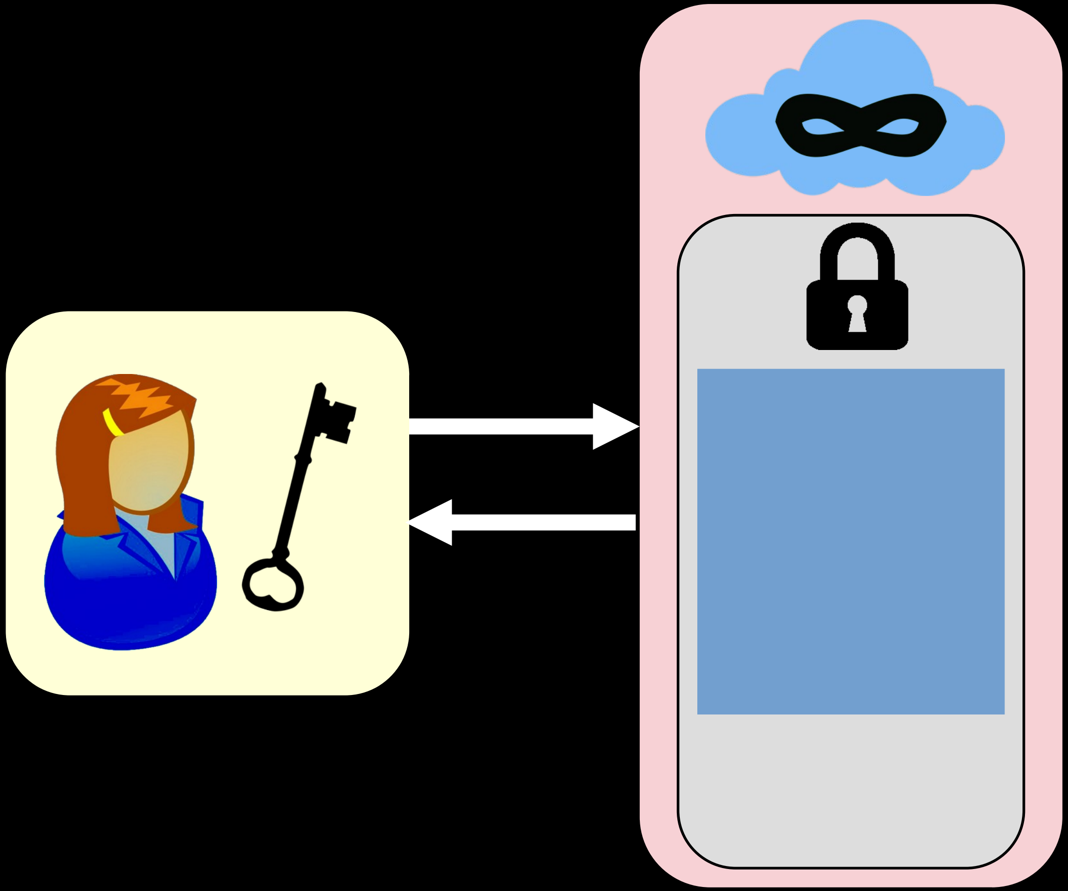
Once Upon a Time… Encrypted Data at Rest

- HR dept. keeps database of employees’ salary records on public cloud (AWS/Azure)
- Cloud provider is untrusted (honest-but-curious)



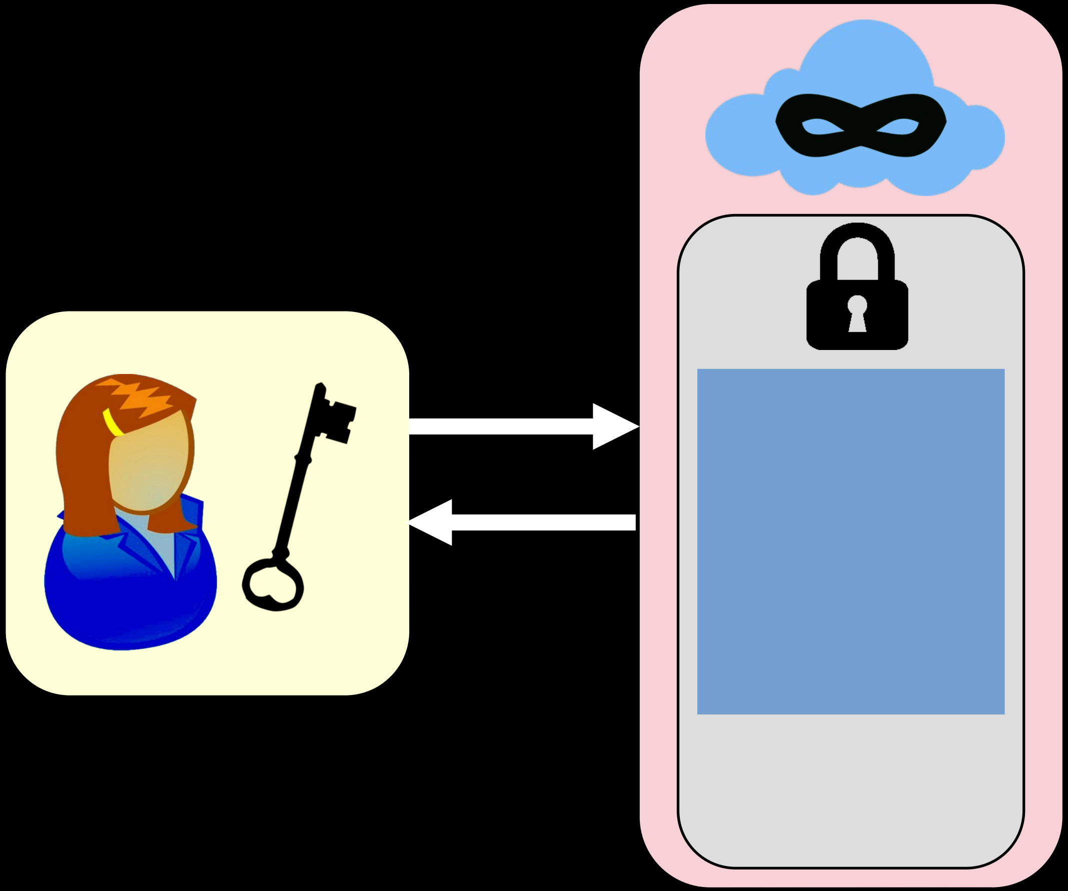
Once Upon a Time… Encrypted Data at Rest

- HR dept. keeps database of employees' salary records on public cloud (AWS/Azure)
- Cloud provider is untrusted (honest-but-curious)
- All information is encrypted (financial information, private)



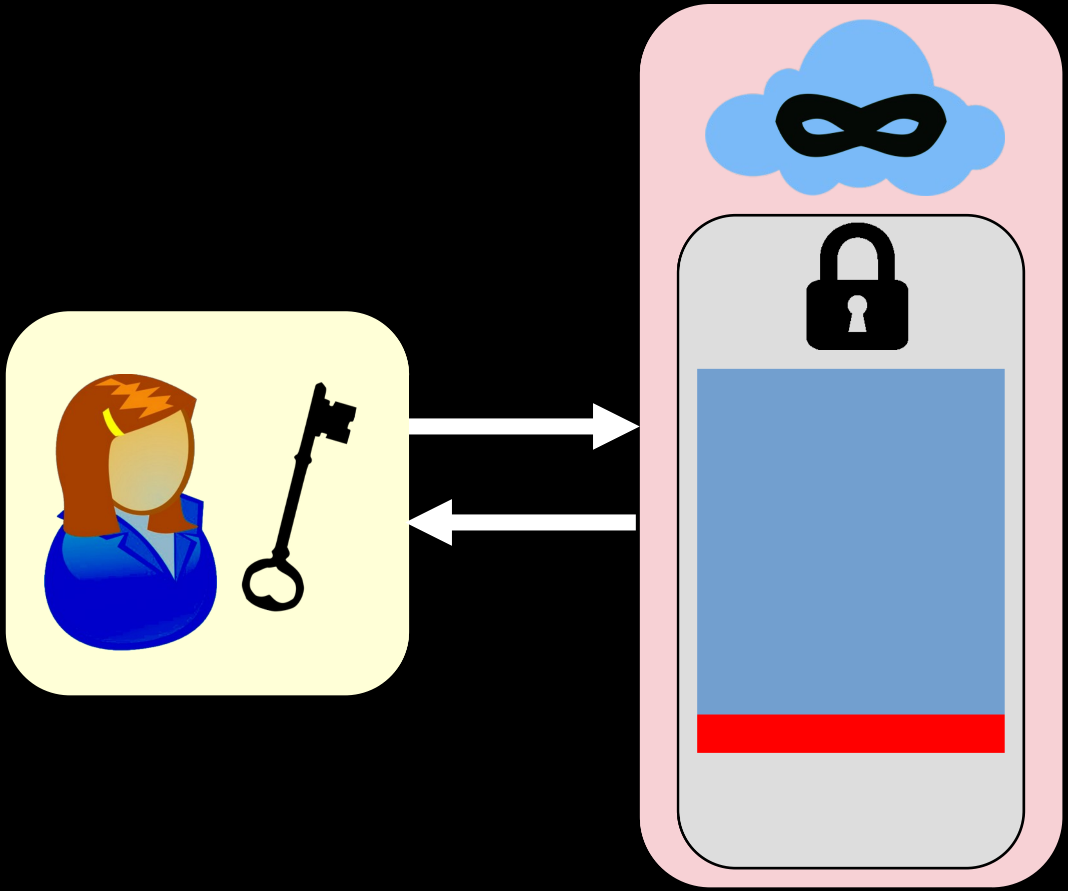
Once Upon a Time… Encrypted Data at Rest

- HR dept. keeps database of employees’ salary records on public cloud (AWS/Azure)
- Cloud provider is untrusted (honest-but-curious)
- All information is encrypted (financial information, private)
- So all good?



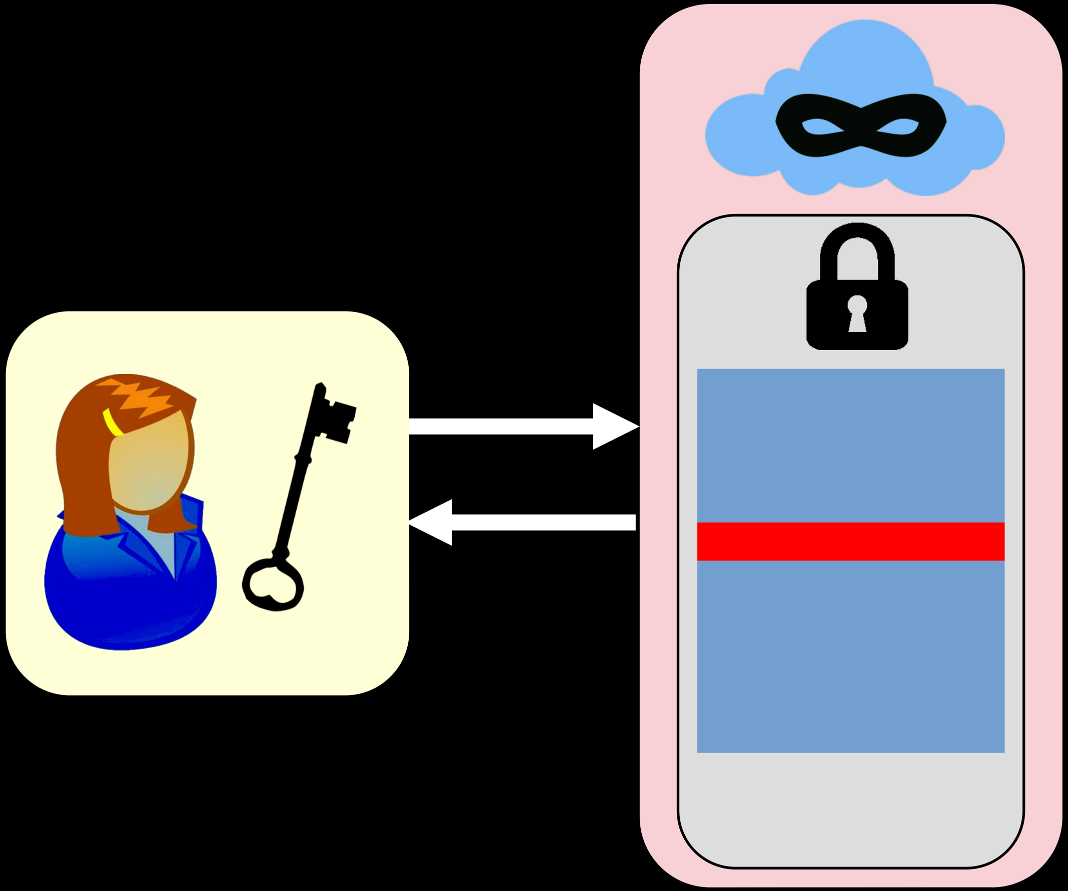
Once Upon a Time… Encrypted Data at Rest

- HR dept. keeps database of employees’ salary records on public cloud (AWS/Azure)
- Cloud provider is untrusted (honest-but-curious)
- All information is encrypted (financial information, private)
- So all good?
- New employee joins



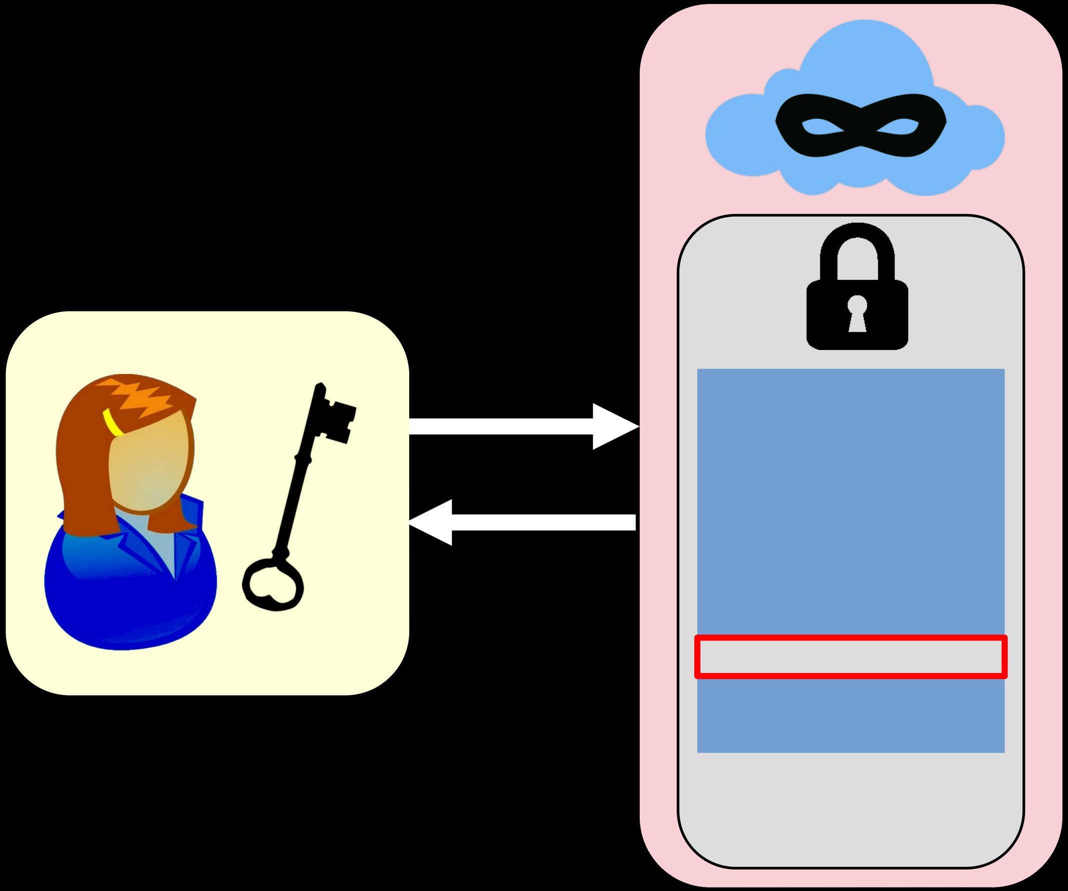
Once Upon a Time… Encrypted Data at Rest

- HR dept. keeps database of employees’ salary records on public cloud (AWS/Azure)
- Cloud provider is untrusted (honest-but-curious)
- All information is encrypted (financial information, private)
- So all good?
- New employee joins
- Employee gets a raise



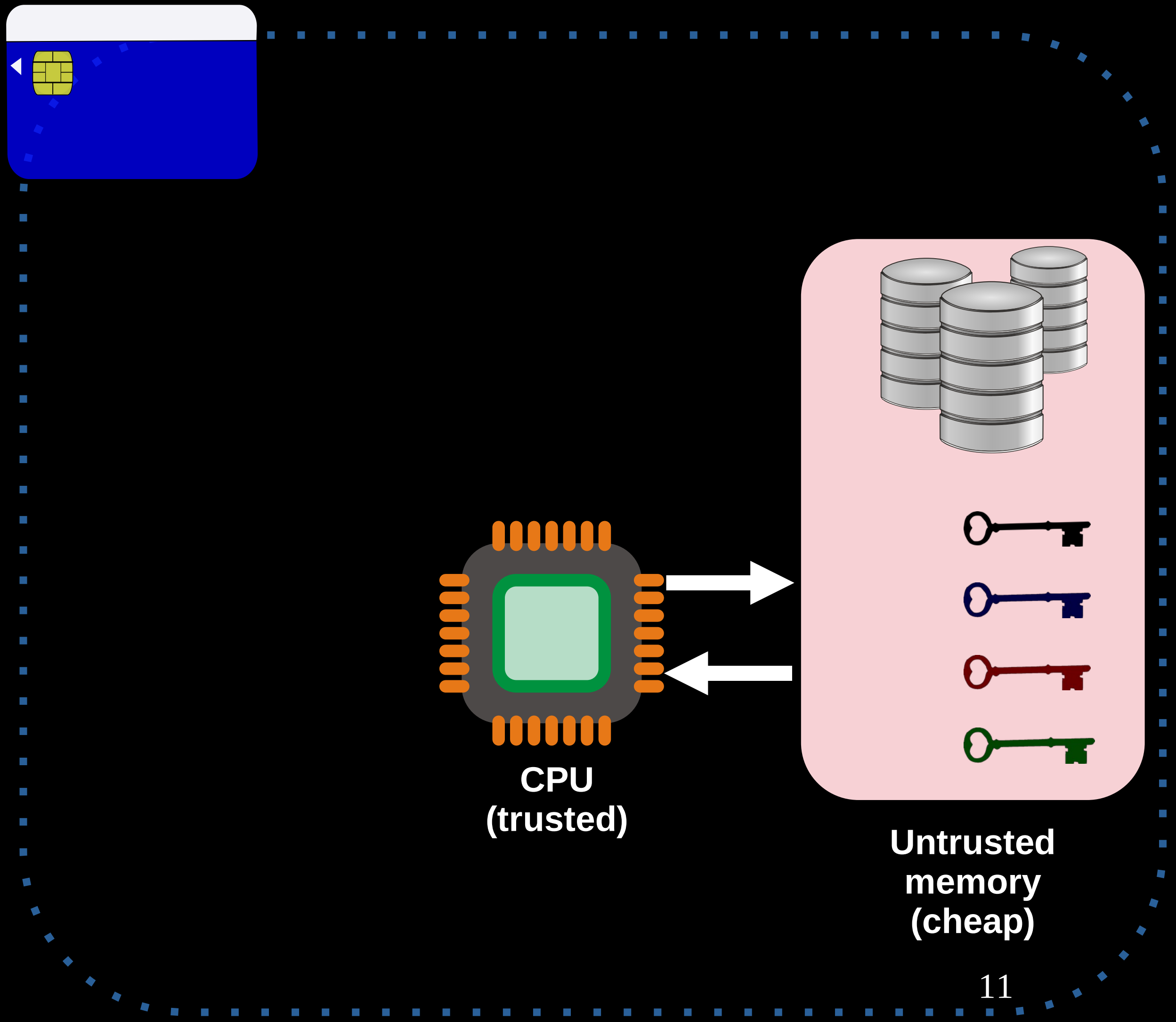
Once Upon a Time… Encrypted Data at Rest

- HR dept. keeps database of employees' salary records on public cloud (AWS/Azure)
- Cloud provider is untrusted (honest-but-curious)
- All information is encrypted (financial information, private)
- So all good?
- New employee joins
- Employee gets a raise
- Employee quits



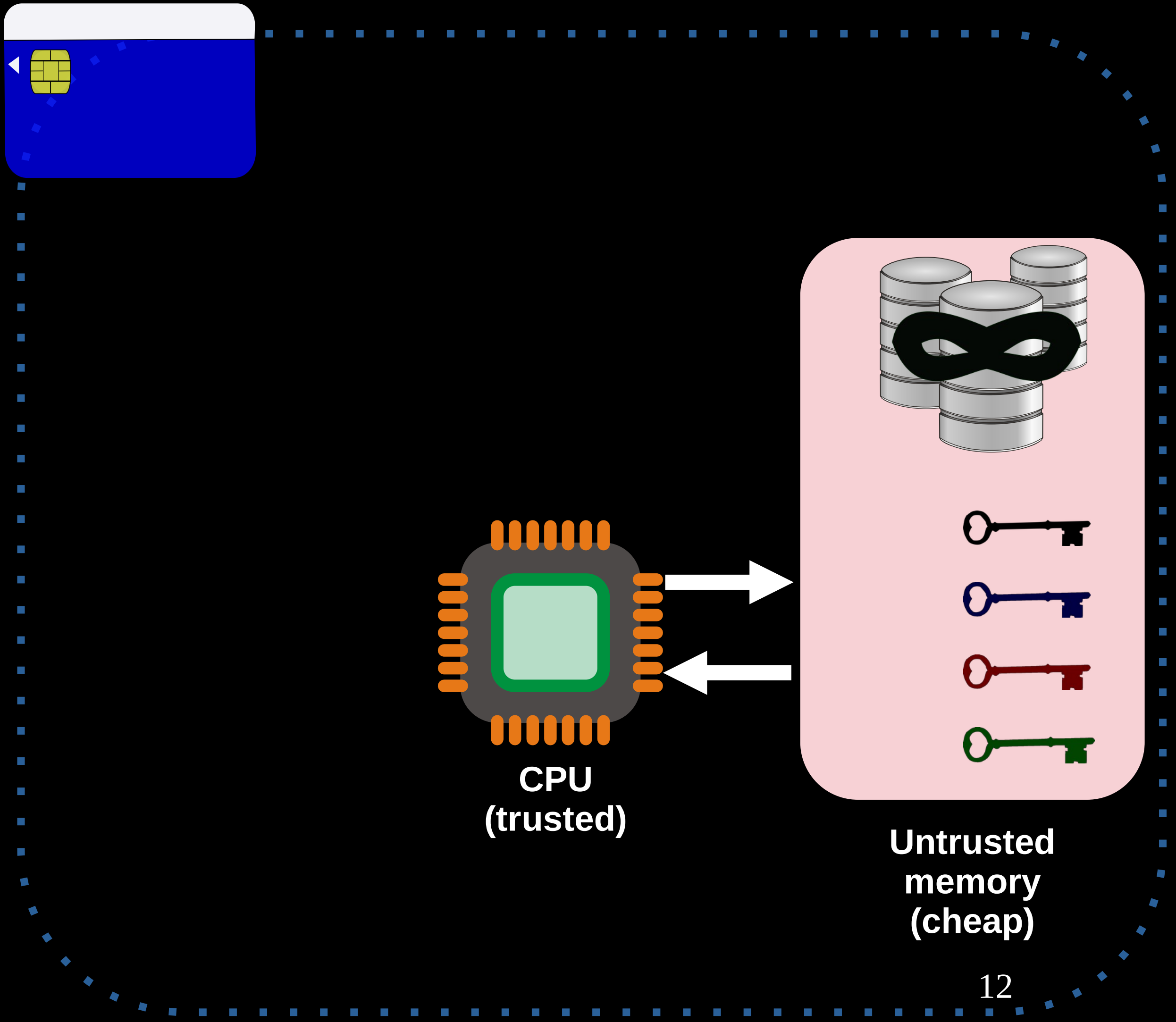
Trusted Storage

- Secure smartcard needs to manage independent keys



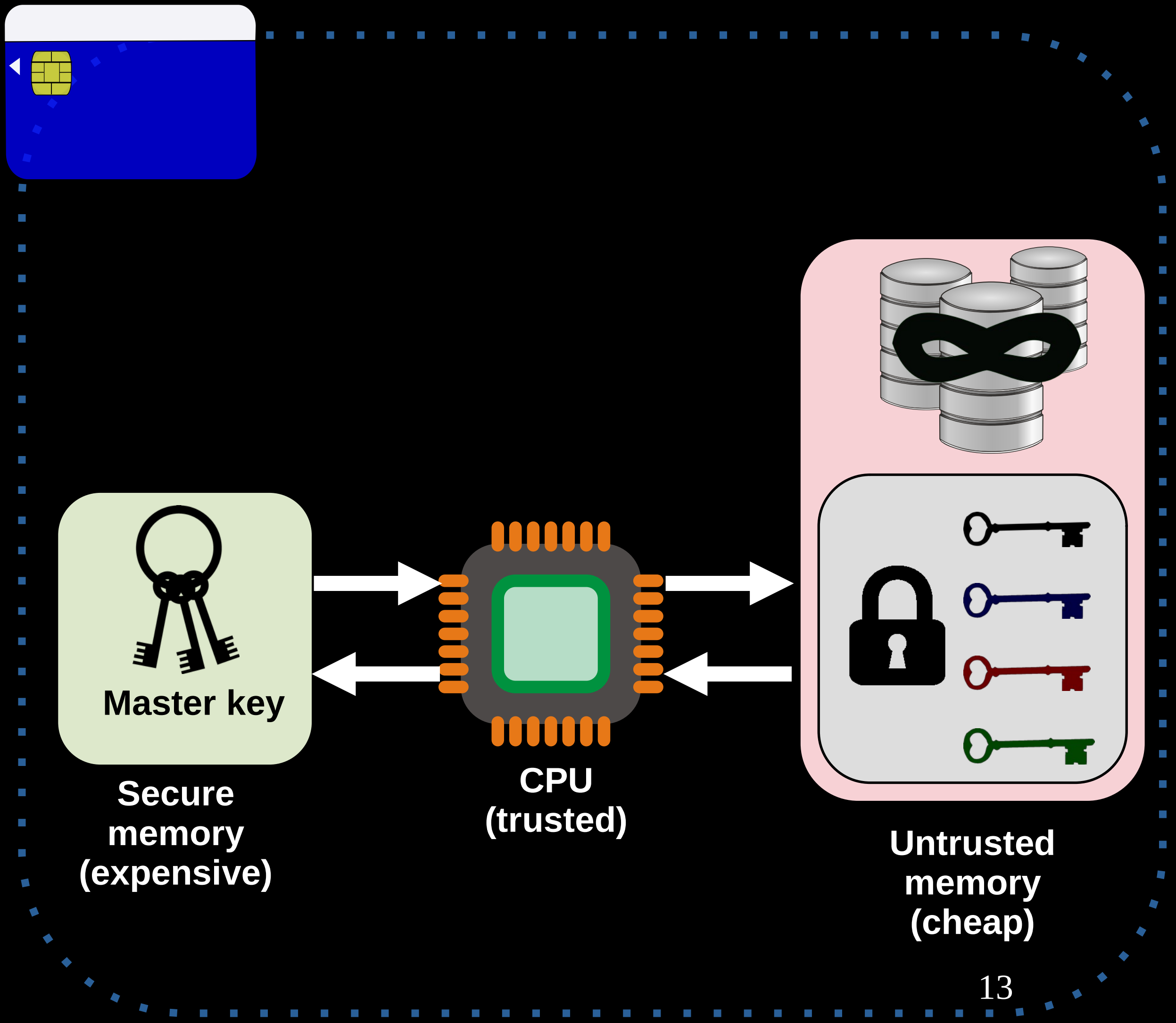
Trusted Storage

- Secure smartcard needs to manage independent keys
- Main storage untrusted: can be analyzed, modified



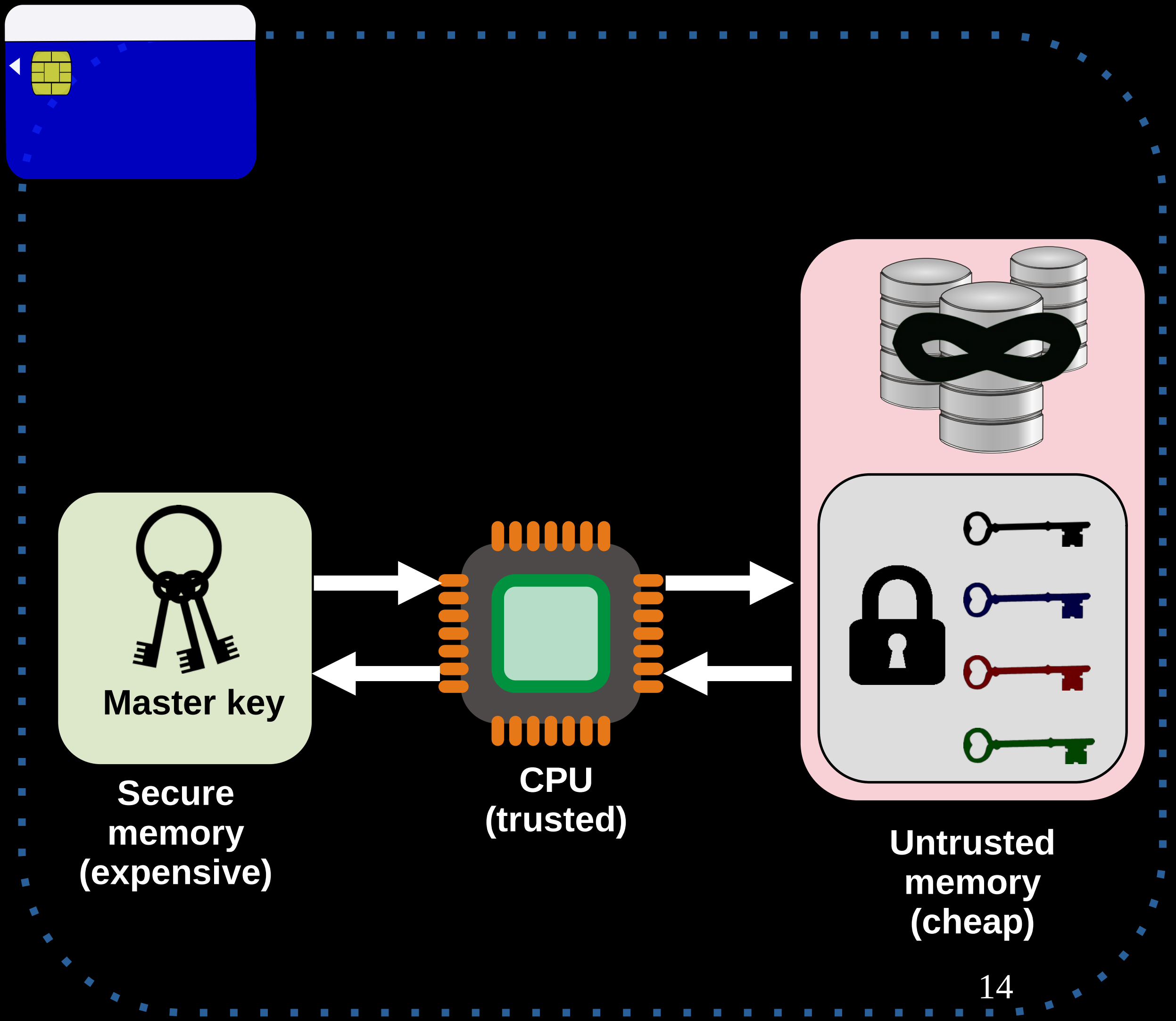
Trusted Storage

- Secure smartcard needs to manage independent keys
- Main storage untrusted: can be analyzed, modified
- Encrypt and use a master key stored on secure memory



Trusted Storage

- Secure smartcard needs to manage independent keys
- Main storage untrusted: can be analyzed, modified
- Encrypt and use a master key stored on secure memory
- “event X triggers key Y”
- “this key opens door Z”
- “door Z is CEO’s office”
- “key has been updated/added/removed”



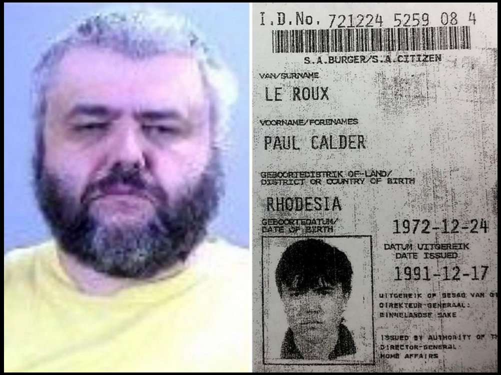
The Good’Ol Times: Truecrypt

TrueCrypt: one of the earliest, efficient full-disk encryption software (released 2004)

The Good’Ol Times: Truecrypt

TrueCrypt: one of the earliest, efficient full-disk encryption software (released 2004)

Troubled history, discontinued in 2014, replaced by VeraCrypt



I.D.No. 721224 5259 08 4

S.A.BURGER/S.A.CITIZEN

VAN/SURNAME

LE ROUX

VOORNAME/FORENAMES

PAUL CALDER

GEBOORTEDISTRIK OF-LAND/
DISTRICT OR COUNTRY OF BIRTH

RHODESIA

GEBOORTEDATUM/
DATE OF BIRTH

1972-12-24

DATUM UITGEREIK
DATE ISSUED

1991-12-17

UITGEREIK OP GESAG VAN D
DIREKTEUR-GENERAAL:
BINNELANDSE SAKE

ISSUED BY AUTHORITY OF TH
DIRECTOR-GENERAL
HOME AFFAIRS

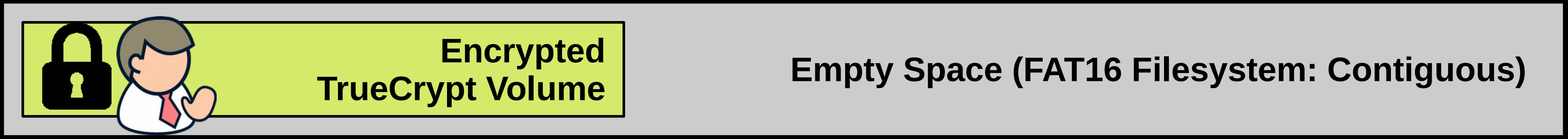
Also: check out this guy, LOL

The Good’Ol Times: Truecrypt

TrueCrypt: one of the earliest, efficient full-disk encryption software (released 2004)

Troubled history, discontinued in 2014, replaced by VeraCrypt

Physical volume (hard disk/partition)



Encrypted TrueCrypt Volume

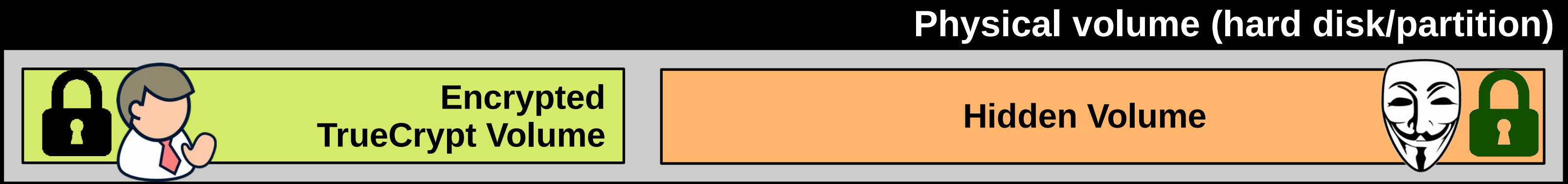
Empty Space (FAT16 Filesystem: Contiguous)

The Good’Ol Times: Truecrypt

TrueCrypt: one of the earliest, efficient full-disk encryption software (released 2004)

Troubled history, discontinued in 2014, replaced by VeraCrypt

Plausible deniability: *hidden volumes*



Physical volume (hard disk/partition)

Encrypted TrueCrypt Volume

Hidden Volume

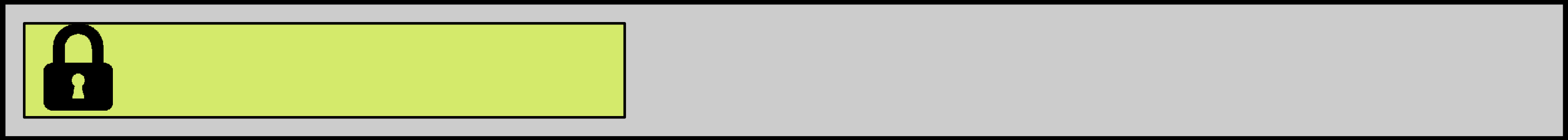
The Good’Ol Times: Truecrypt

TrueCrypt: one of the earliest, efficient full-disk encryption software (released 2004)

Troubled history, discontinued in 2014, replaced by VeraCrypt

Plausible deniability: *hidden volumes*

Physical volume (hard disk/partition)



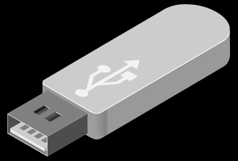
The Good’Ol Times: Truecrypt

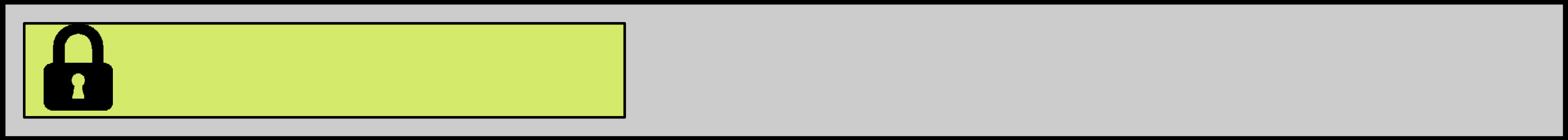
TrueCrypt: one of the earliest, efficient full-disk encryption software (released 2004)

Troubled history, discontinued in 2014, replaced by VeraCrypt

Plausible deniability: *hidden volumes*

“modern” solid-state drives: caching / layering / TRIM





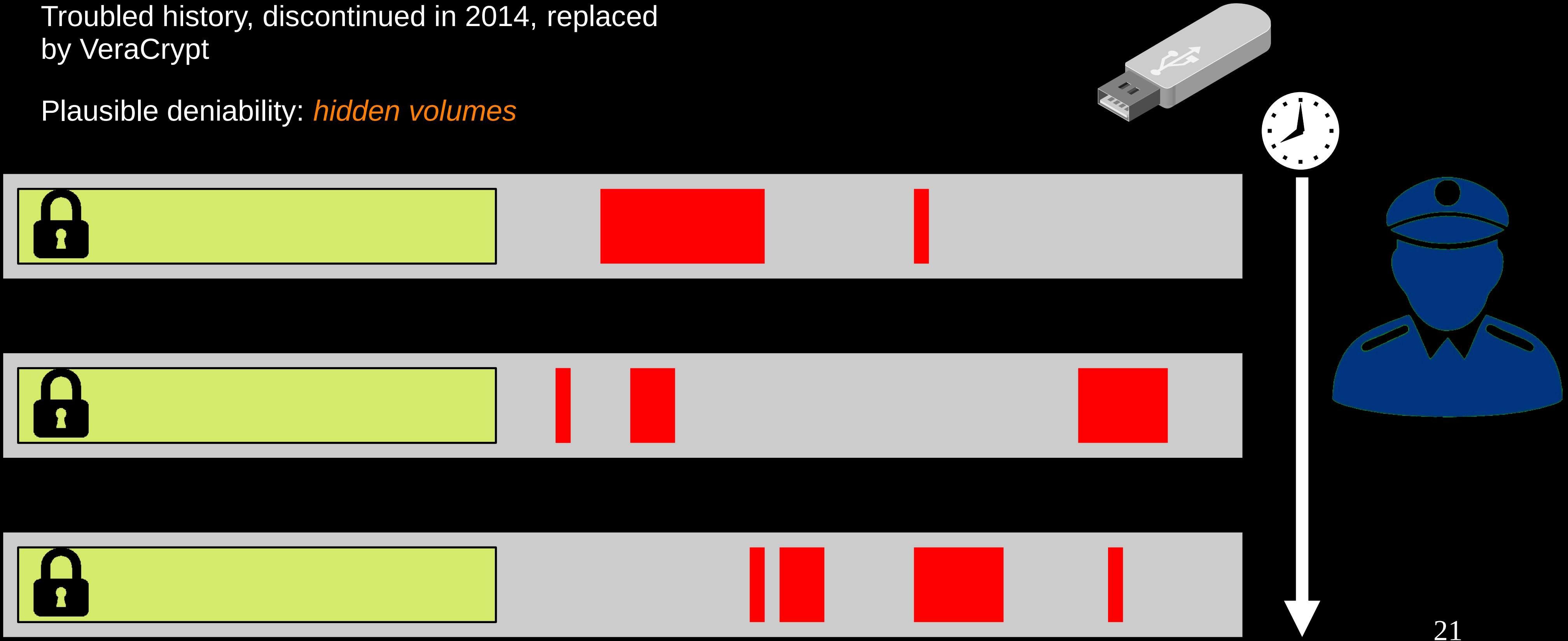
The Good’Ol Times: Truecrypt

TrueCrypt: one of the earliest, efficient full-disk encryption software (released 2004)

Troubled history, discontinued in 2014, replaced by VeraCrypt

Plausible deniability: *hidden volumes*

“modern” solid-state drives: caching / layering / TRIM



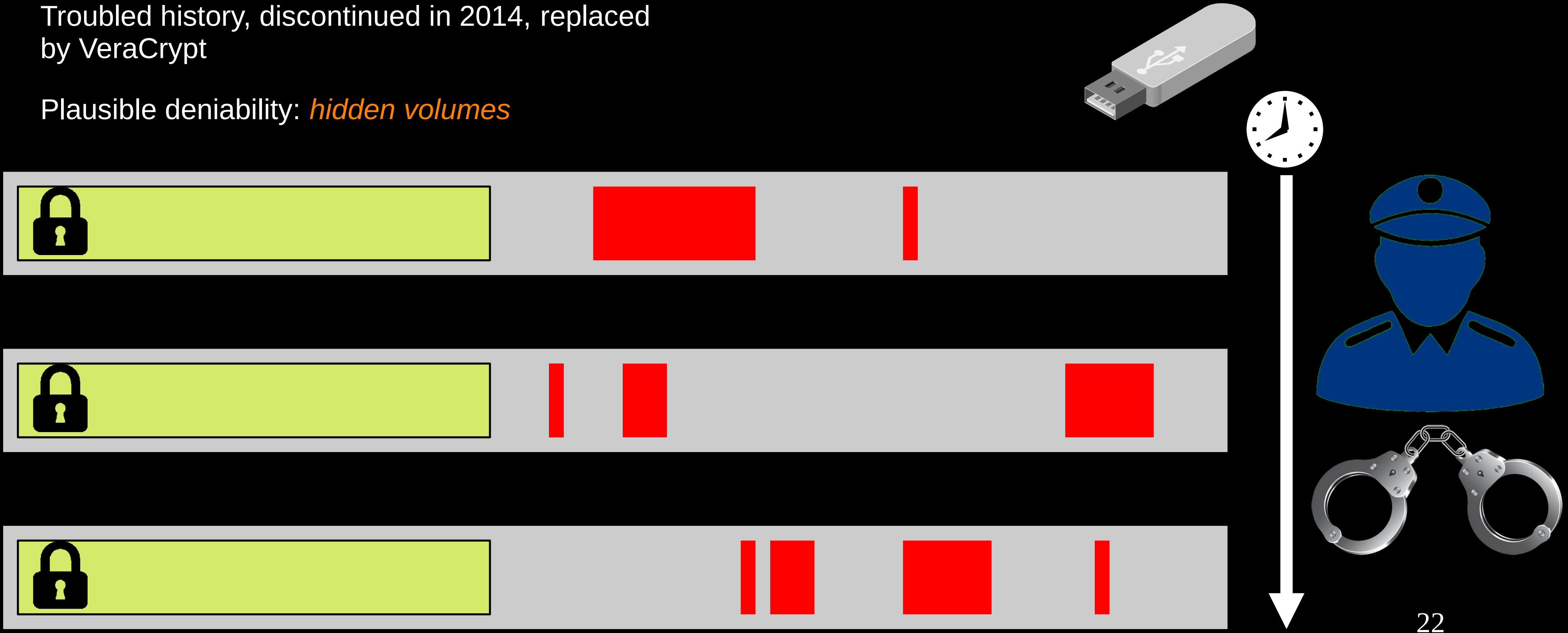
The Good’Ol Times: Truecrypt

TrueCrypt: one of the earliest, efficient full-disk encryption software (released 2004)

Troubled history, discontinued in 2014, replaced by VeraCrypt

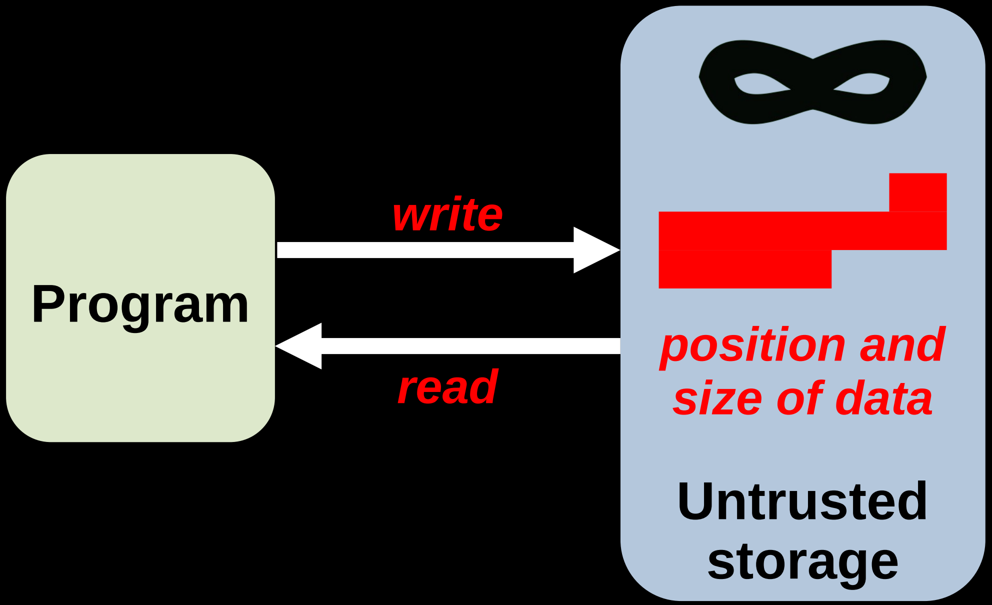
Plausible deniability: *hidden volumes*

“modern” solid-state drives: caching / layering / TRIM

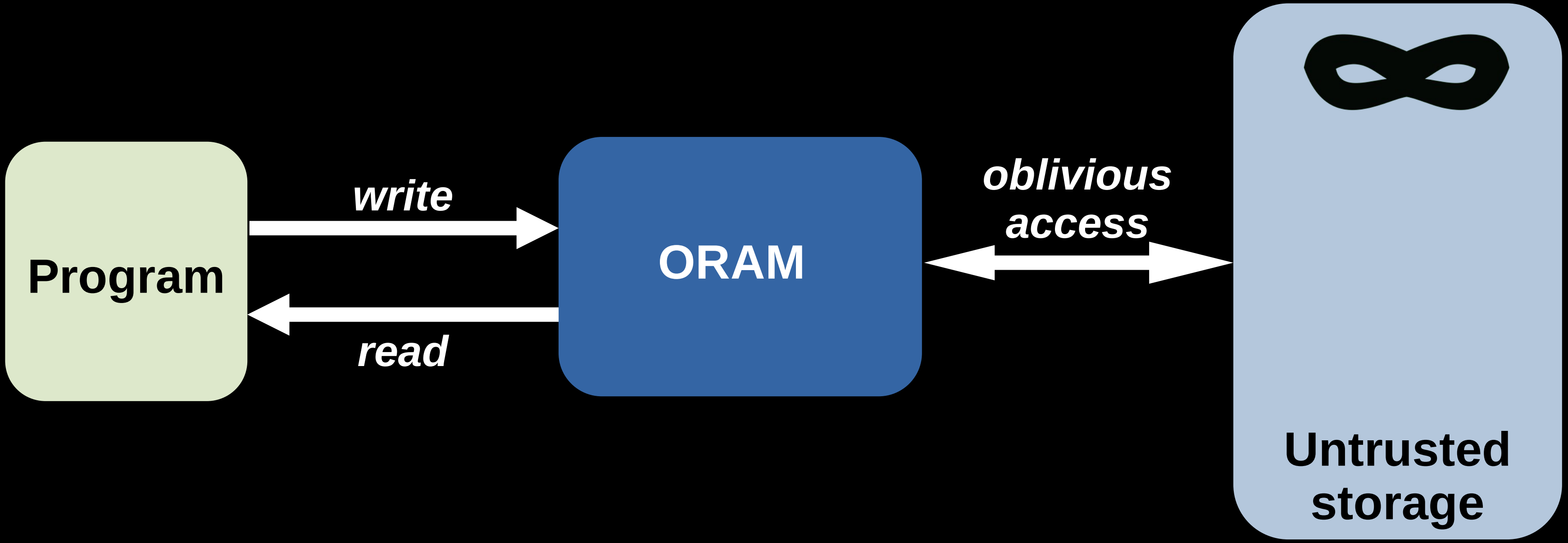


Access Patterns Matter

- Encryption alone does not hide access patterns
- These can leak sensitive information



ORAM (Oblivious Random Access Machines)



Program

write

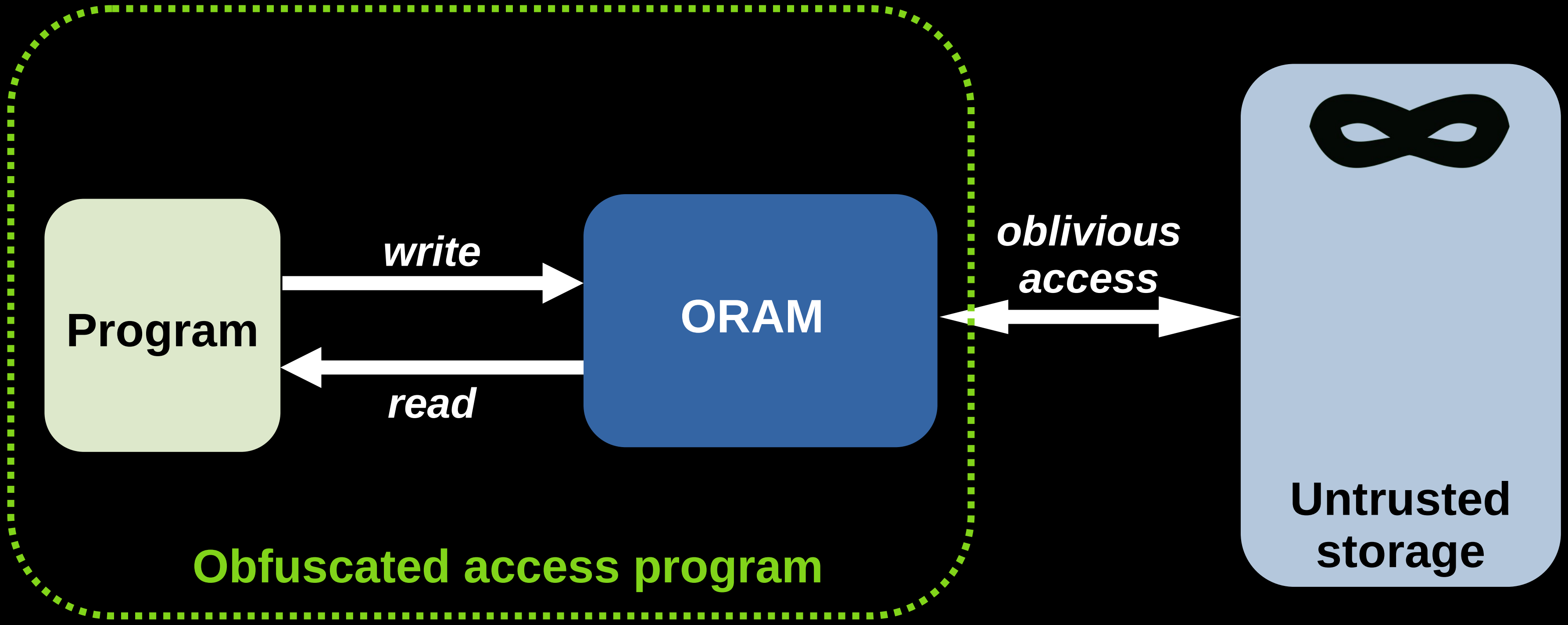
read

ORAM

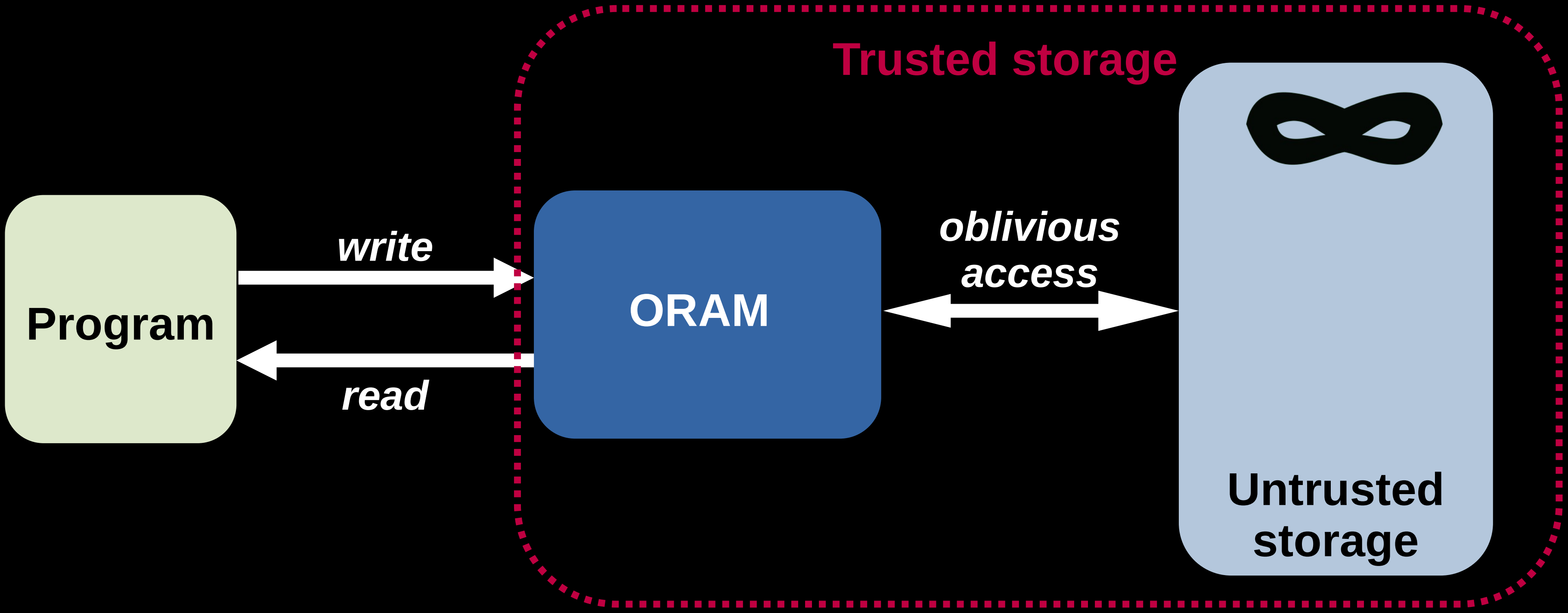
oblivious access

Untrusted storage

ORAM (Oblivious Random Access Machines)



ORAM (Oblivious Random Access Machines)



A Brief History of ORAM Schemes

- Idea started in 1987 (by cryptographer Oded Goldreich)
- Trivial scheme: encrypt database, and then at every read or write, download whole database, decrypt, and then re-encrypt with a randomized cipher
- Subsequent works: hierarchical buffers, Bloom filters, cuckoo hashing (security and efficiency issues)

A Brief History of ORAM Schemes

- Idea started in 1987 (by cryptographer Oded Goldreich)
- Trivial scheme: encrypt database, and then at every read or write, download whole database, decrypt, and then re-encrypt with a randomized cipher
- Subsequent works: hierarchical buffers, Bloom filters, cuckoo hashing (security and efficiency issues)
- Basic principles for all schemes:
  1) Store data in encrypted blocks and keep track of their index (position)
  2) If you need a certain block, never download only that block; download some more instead
  3) Every time decrypt and re-encrypt the downloaded blocks with a randomized cipher
  4) But also shuffle somehow blocks' positions at every access
- Need to reach a tradeoff between security and performance

A Brief History of ORAM Schemes

- Idea started in 1987 (by cryptographer Oded Goldreich)
- Trivial scheme: encrypt database, and then at every read or write, download whole database, decrypt, and then re-encrypt with a randomized cipher
- Subsequent works: hierarchical buffers, Bloom filters, cuckoo hashing (security and efficiency issues)
- Basic principles for all schemes:
  1) Store data in encrypted blocks and keep track of their index (position)
  2) If you need a certain block, never download only that block; download some more instead
  3) Every time decrypt and re-encrypt the downloaded blocks with a randomized cipher
  4) But also shuffle somehow blocks’ positions at every access
- Need to reach a tradeoff between security and performance
- 2011: tree-based ORAM (Shi et al.)
- 2012: Path-ORAM (Stefanov et al.)

Path ORAM

- Regular block access
  - Just access the physical block by its logical block ID
- Path ORAM
  - We don’t want to leak that information
  - Cannot access physical blocks directly by logical block ID

Regular block access

- read(block: int)
  -
  - return os.read(block)
- write(block: int, data: [byte])
  -
  - return os.write(block, data)

Path ORAM block access

- read(block: int)
    - **b = f(block)**
    - return os.read(**b**)
- write(block: int, data: [byte])
    - **b = f(block)**
    - return os.write(**b**, data)

```
function f(block: int) {
        … ?
}
```

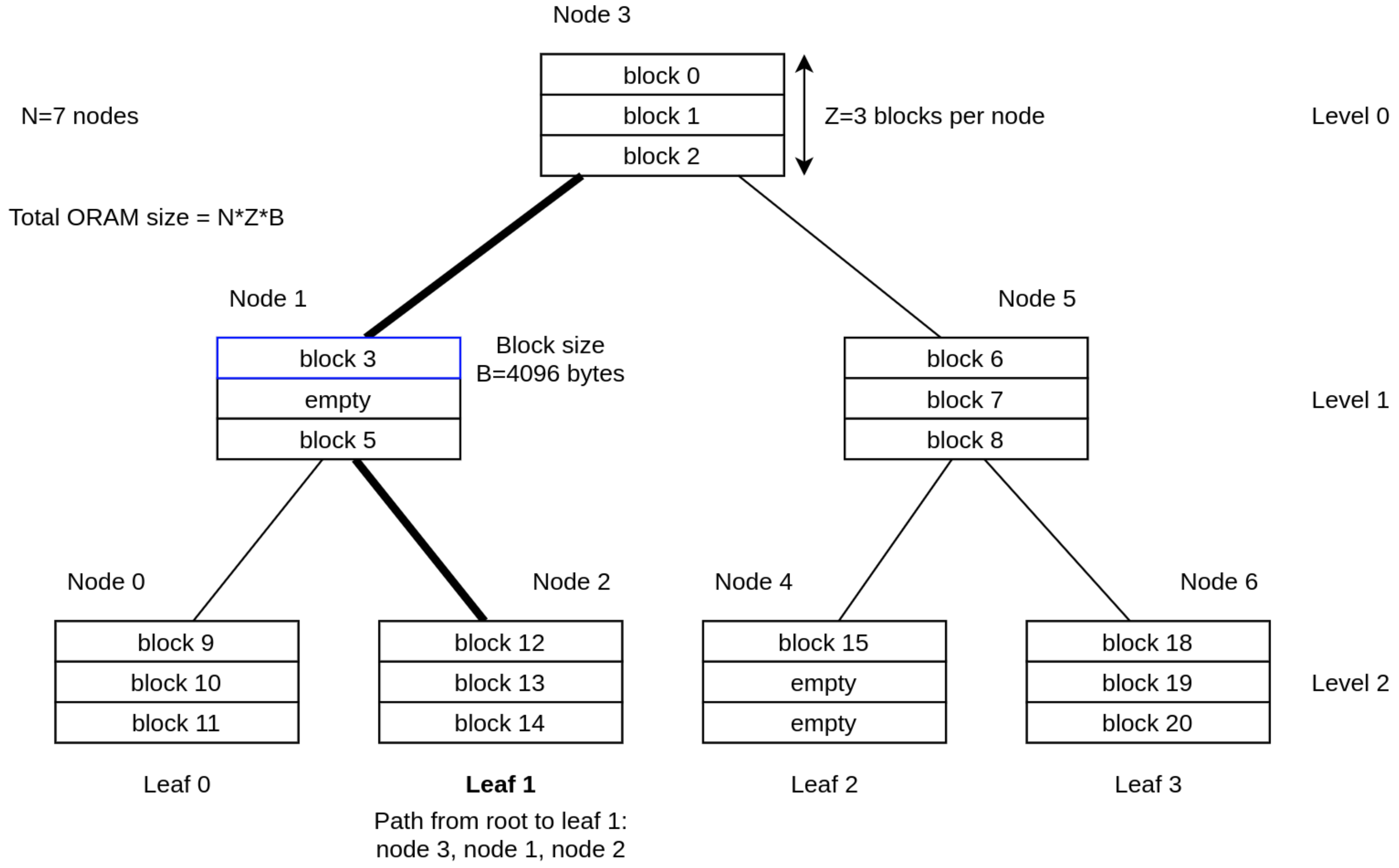
Path ORAM idea

- What if we access **more blocks than required**?
- Which blocks should we access? How can we be sure that the “true” block is in there?
- Solution: group blocks in nodes, represent nodes in a tree
  - And **map blocks to tree leaves**
  - **Path from root to leaf is unique** and defines list of blocks to access
  - Guaranteed that “true” block is contained in that list
  - Requires storing small amount of client data

Stash
block 4
block 17
block 16

Position map

Block ID	Leaf ID
Block 0	Leaf 0
Block 1	Leaf 1
⋮	
Block 20	Leaf 3



Introducing Oramfs

- https://github.com/kudelskisecurity/oramfs
- Storage-agnostic
- GPL 3.0
- ORAM filesystem written in **Rust**
- Resizing supported
- Built to support multiple ORAM schemes (Path ORAM, ...)
- Multiple encryption ciphers (AES-GCM, etc.)

Inputs

- **Public** directory (the “server”)
  - This can be stored on untrusted storage
  - Anything that appears as a local directory (e.g. mount remote storage as local directory using Rclone)
- **Private** directory (the “client”)
  - This is what the user accesses
  - Just a regular directory where files can be read or written

Architecture

```
+---------------------------------------------------+
|                                                   |
|                   ext4 filesystem                 | <---+ or any other FS or your choice
|                                                   |
+---------------------------------------------------+
|                                                   |
|               Loop device (/dev/loop0)            | <---+ created with losetup
|                                                   |
+---------------------------------------------------+
|                                                   |
|                    ORAMFS (FUSE)                  | <---+ Input  : *public* local directory
|                                                   |       Output : *private* “oram” single file,
+-------------------+-----------------+-------------+                for use with loop device
|                   |                 |             |
|  Local directory  |  Cloud storage  |    SSHFS    | <---+ Input directory can be anything
|                   |                 |             |       that appears as a local directory,
+-------------------+-----------------+-------------+       including mounted remote directories.
                                                            Examples: SSH, FTP, anything supported
                                                            by rclone or similar tools,
                                                            any mounted FUSE filesystem, etc.
```

Performance with default settings

- UtahFS
  - Encrypted storage system, FUSE-based, backed by cloud storage
  - Optionally supports ORAM (Path ORAM)
  - https://github.com/cloudflare/utahfs
- Write 10MB random data to ORAM
  - UtahFS (local disk, oram=true): 30sec
  - Oramfs (local disk, AES-GCM): 15 sec
    - => **2x speedup (write)**
- Read 10MB random data from ORAM
  - UtahFS: 9.37 sec
  - Oramfs: 1.05 sec
    - => **9x speedup (read)**

Demo

Conclusions

- Increased privacy for untrusted storage users
- Ease of use
- Still a prototype

Future work

- Performance improvements
- Support more platforms
- Implement more ORAM schemes

More resources

- Oramfs on Github
  - https://github.com/kudelskisecurity/oramfs
- https://research.kudelskisecurity.com
  - Path ORAM blog post
- Path ORAM paper
  - https://eprint.iacr.org/2013/280.pdf

Thank you

- Questions?