Security enforcement by privilege aware launcher

Interesting research trial in **Tizen** security

José Bollo - Eurogiciel

The problem

- Security policy of Tizen:
  - At the API level, managed by privileges
  - Privileges are not enforced to be linked to a system resource, Privileges can be “logical”
  - Strict separation of data per application (for privacy)
- How can programs written in C respect that ?

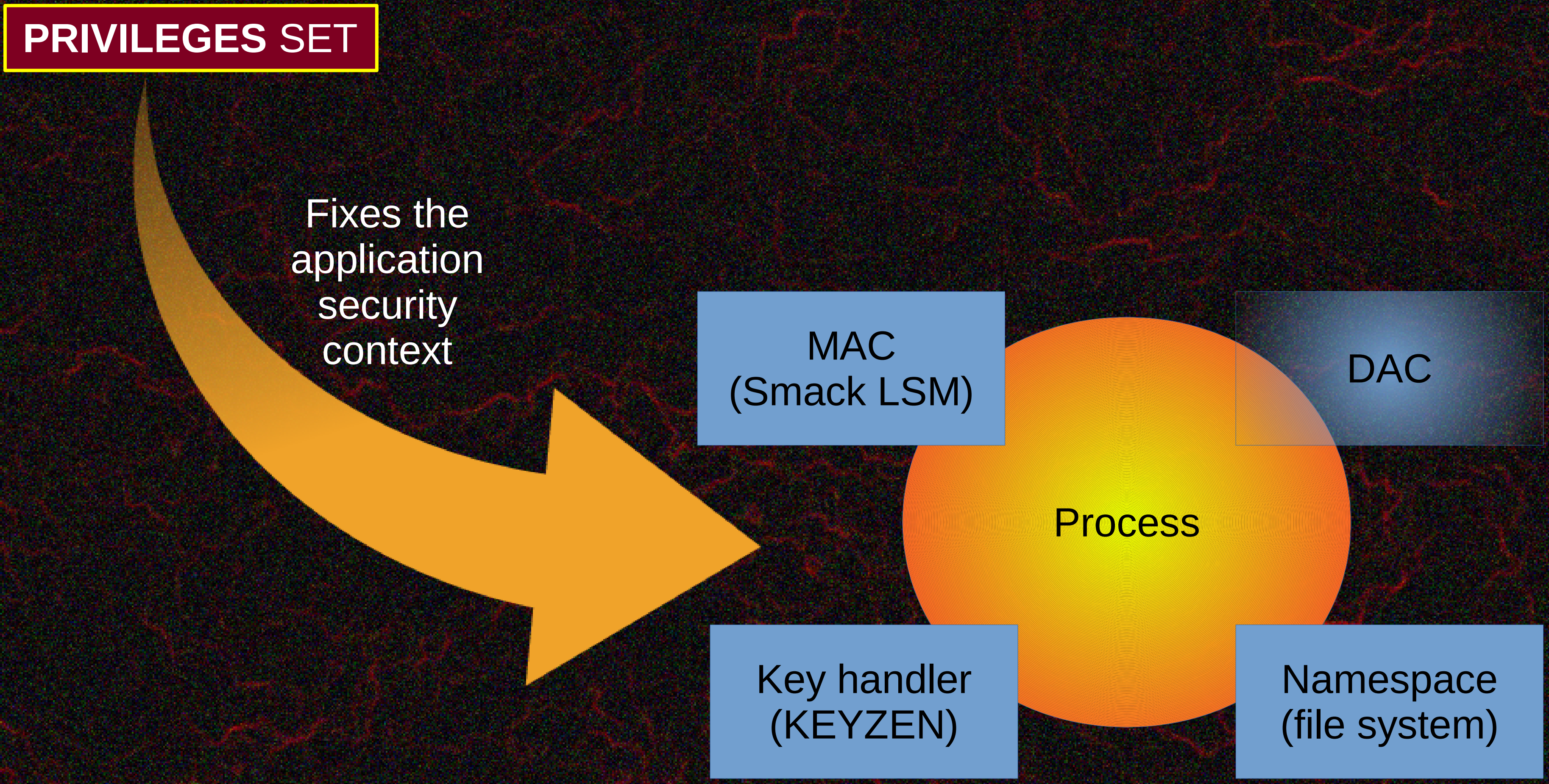
The idea

- Based on the idea that any application:
  - is installed by an installer
  - the installer can configure the starting mechanism
  - services will check granted privileges
- The idea is to enforce the use of a launcher for launching applications

The launcher

- The launcher will:
  - retrieve the privileges granted to the application
  - prepare a secured runtime environment according to that privileges
  - launch the application

Target security context



PRIVILEGES SET

Fixes the application security context

MAC (Smack LSM)

DAC

Process

Key handler (KEYZEN)

Namespace (file system)

About privileges

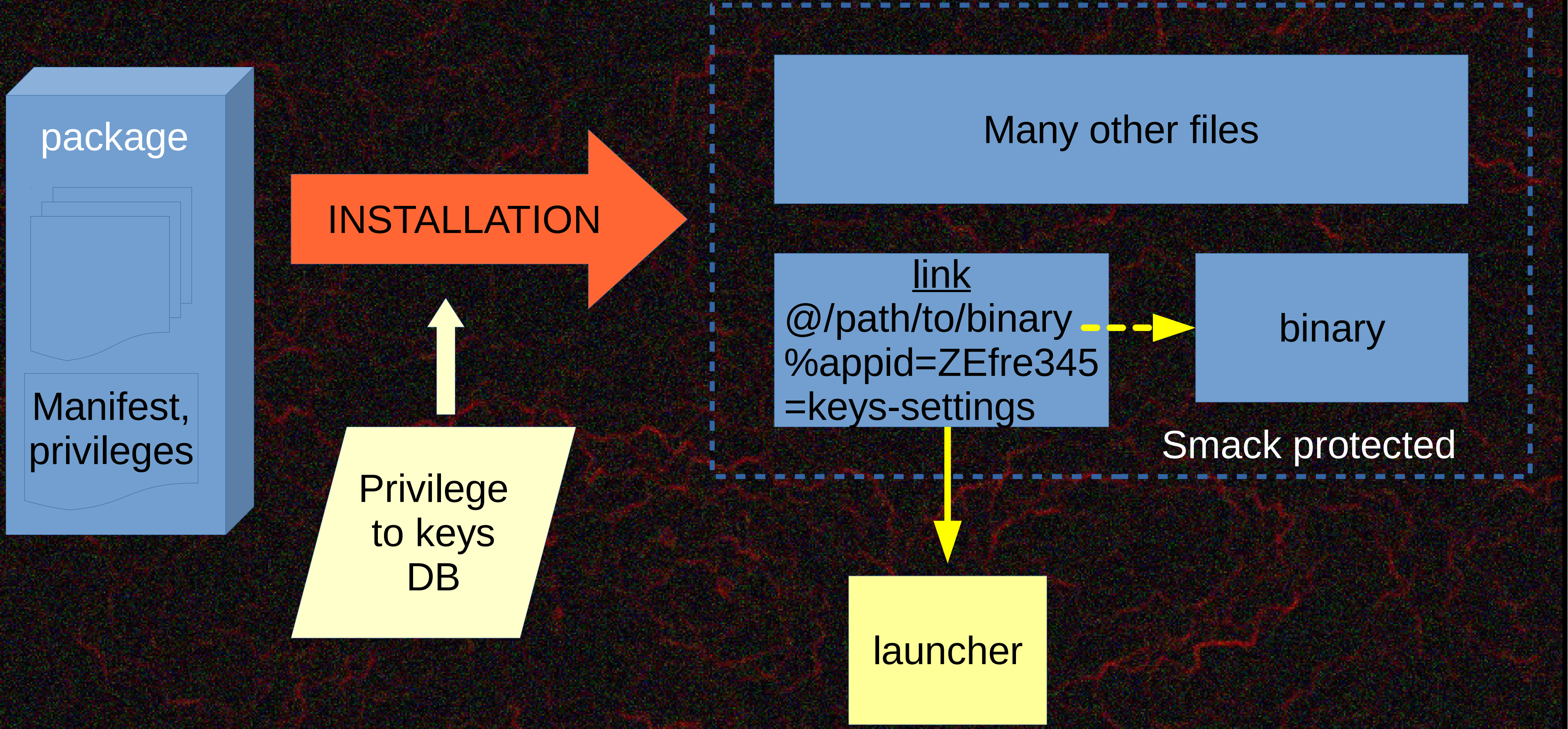
- The privileges of Tizen are dynamics, they can be of the types below
- Privileges can be used as key “as is” or mapped to some key values

Type	Description	Prefix
blanket prompt	The user has to validate at least one time	!
session prompt	The user has to validate at least one time per session	+
one-shot prompt	The user has to validate each time	*
permit	The privilege is granted	=
deny	The privilege is denied	-

Installing an application

- The executable file is renamed and hidden
- The executable file is protected by Smack:
  - ✓ only the launcher can execute it
  - ✓ its “execute” Smack label is set to good value
- A symbolic link takes place of the original executable file (before renaming), it point to the launcher
- Security attributes of the symbolic link (extended attribute **security.smaunch-launcher**) tells:
  - ✓ What is the executable file
  - ✓ What are the keys to be set for that executable

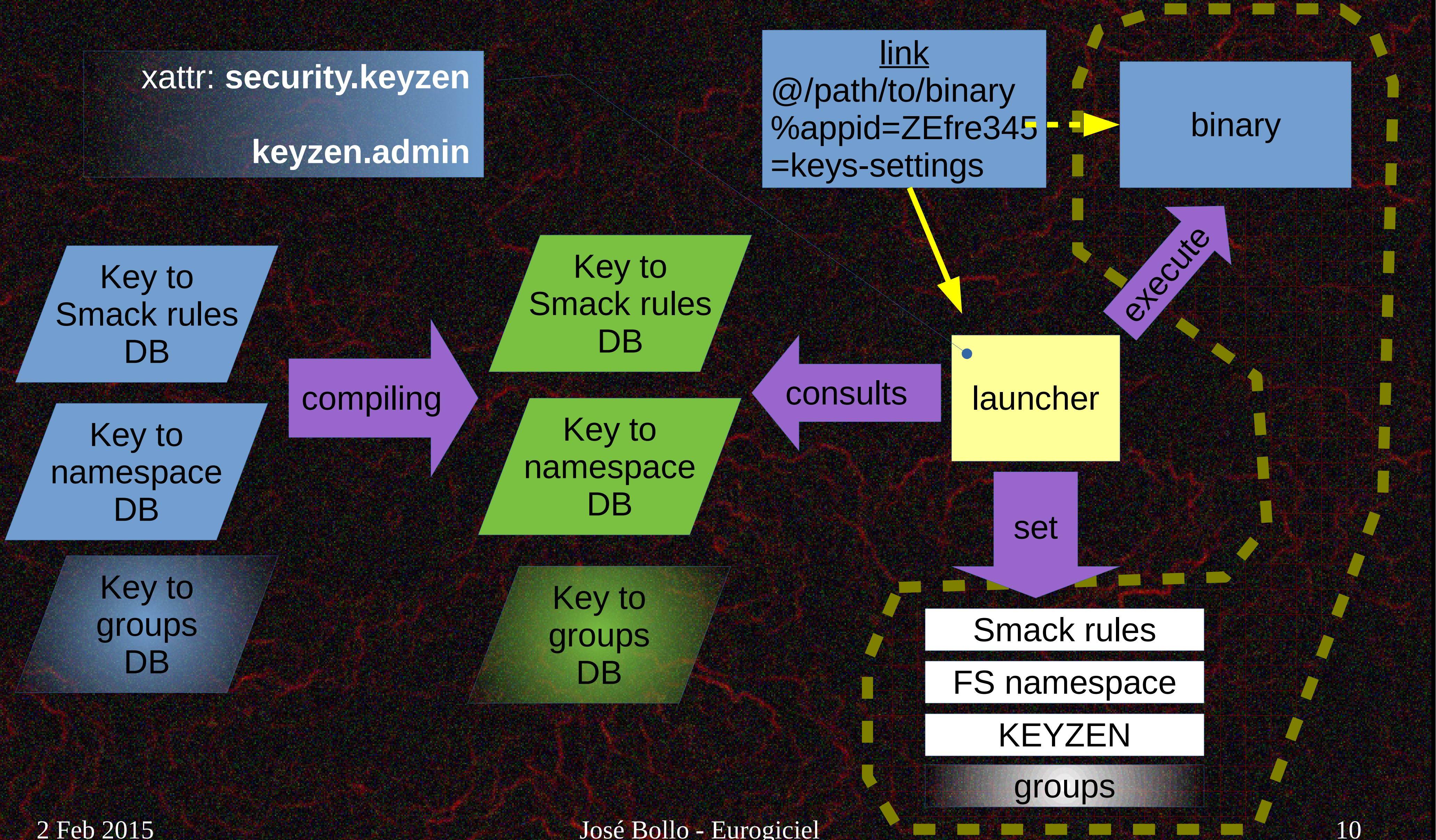
The installer



Launching an application

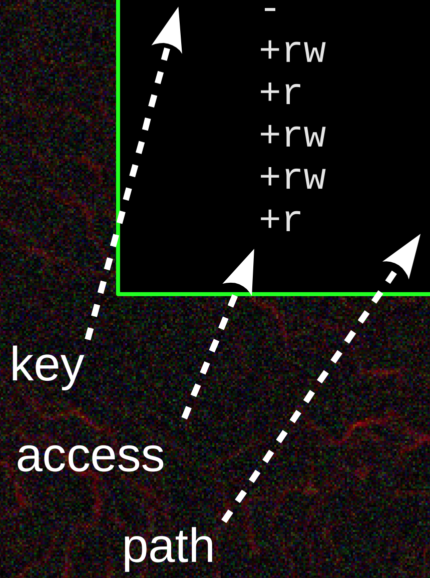
1) The launcher is executed through the installed link
2) The launcher reads security extended attribute **security.smaunch-launcher** and retrieves executable path and security keys for it
3) The launcher uses several databases to set the security context:
   a) The Smack context
   b) The filesystem context
   c) The KEYZEN context
   d) The DAC added groups
4) The launcher finally launches the executable in the set context

The launcher



Preparinging the filesystem

Example of configuration file



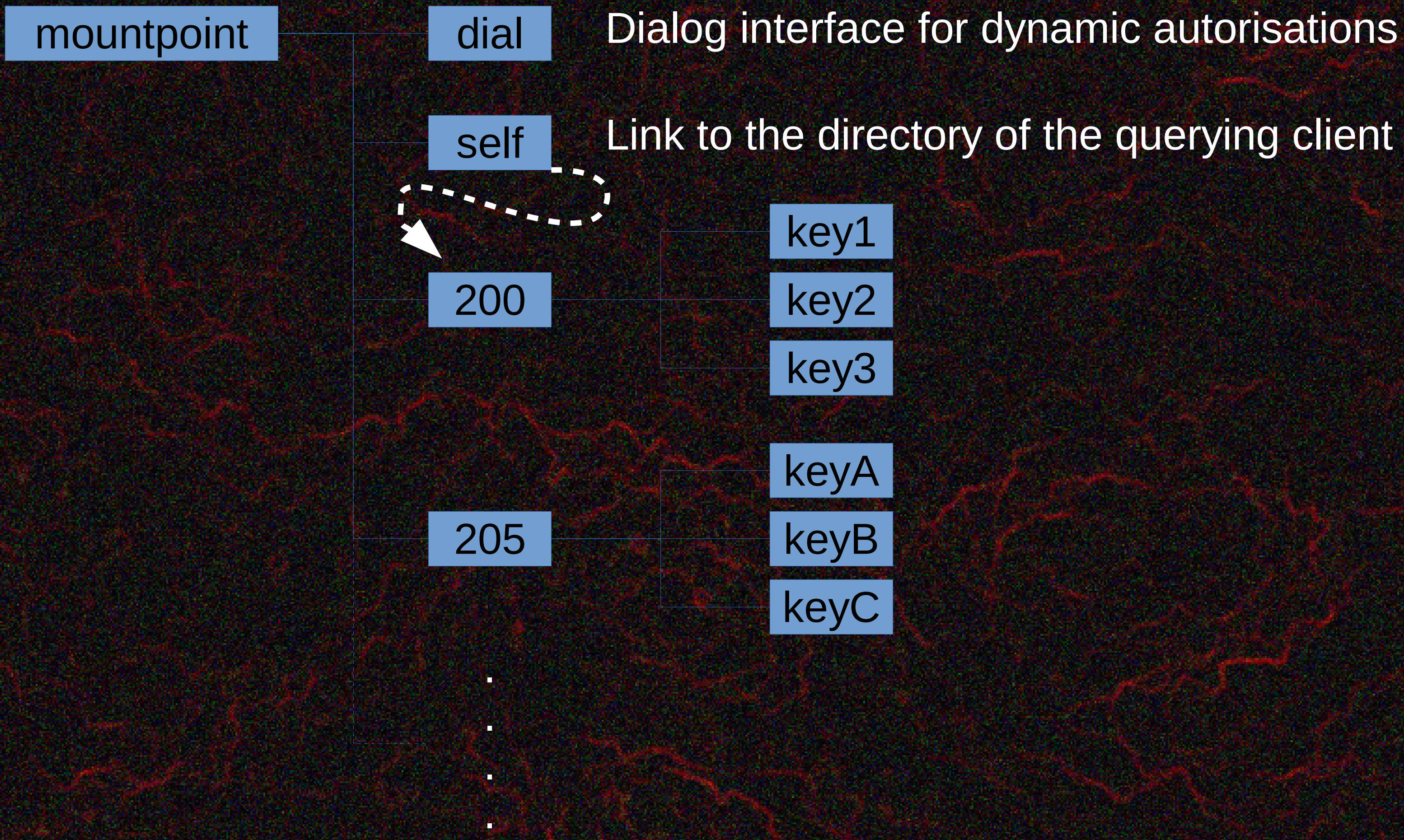
```
-- user access
user
    -       /home                   -- dont see other users
    +rw     /home/%user             -- see itself
    +r      /sys/fs/smackfs         -- disable write access to change-rule

-- basic restricted access
restricted
    -       /home                                   -- dont see any other user
    +rw     /home/%user/.config/%appid              -- access to config
    +r      /home/%user/share                       -- shared data
    +rw     /home/%user/share/%appid                -- own shared data
    +rw     /home/%user/share/.cert/%cert           -- same certificate
    +r      /sys/fs/smackfs                         -- disable change-rule
```

%user is a predefined substitution replaced with $USER

The key handler KEYZEN

- It handles the keys per process
- The keys are handled through the filesystem
- Any keys is of one of the dynamic type: blanket prompt, session prompt, one-shot prompt, permit, deny
- Keys are static or dynamic
- The current implementation is using FUSE

example



mountpoint

dial

Dialog interface for dynamic autorisations

self

Link to the directory of the querying client

200

key1

key2

key3

205

keyA

keyB

keyC

. . . .

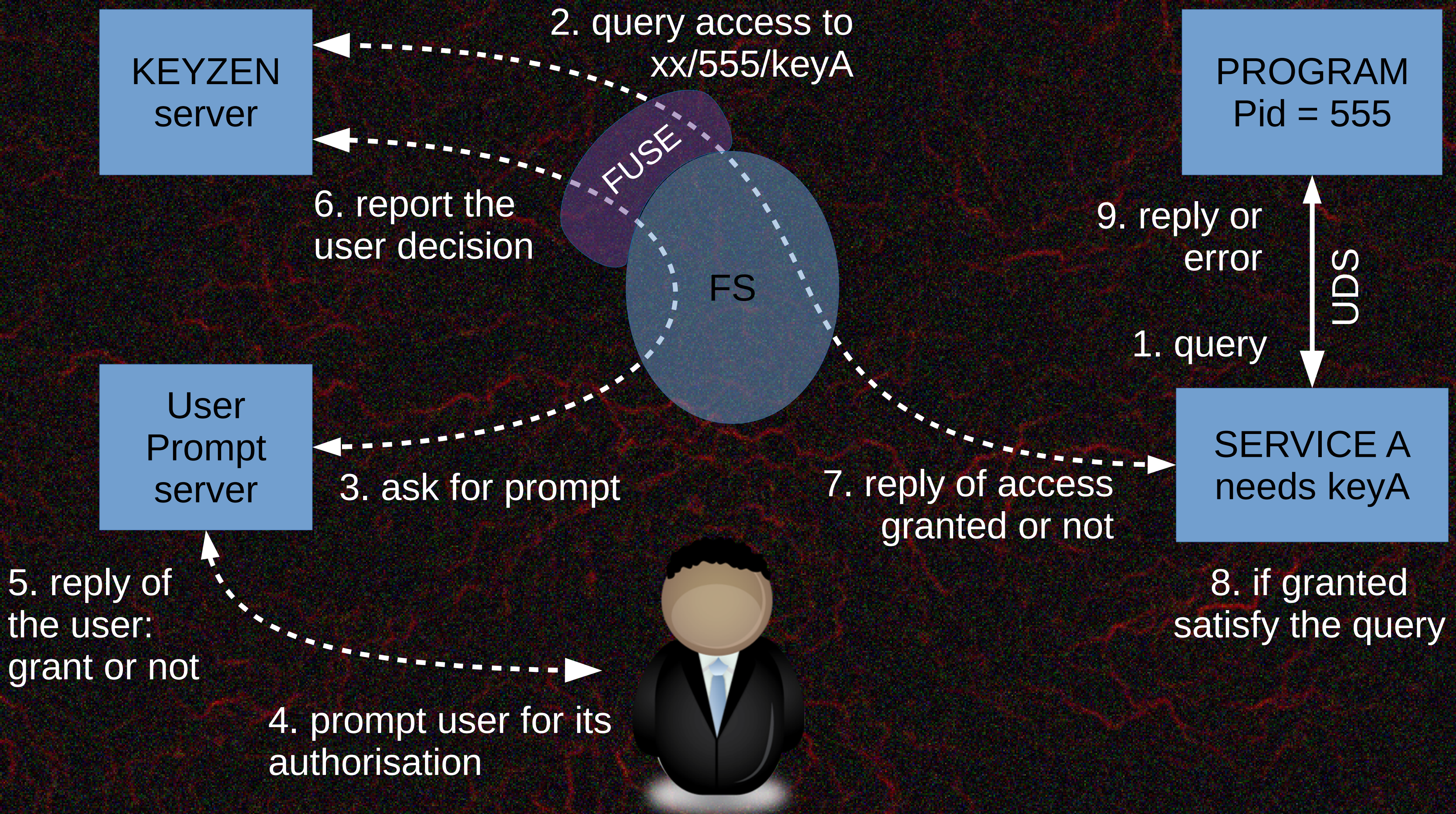
Operations of KEYZEN FS

- Adding a key ==> **mknod**
  - Example: mknod(“+keyname”, S_IFREG|0644, 0)
  - Creates a session key
- Removing a key ==> **unlink**
- Asking a key ==> **access**
  - Fast syscall
- Listing potential keys ==> opendir/readdir/closedir

KEYZEN security

- Dropping keys
  - Any process can drop any of its key at any time
  - No process can drop a key of other processes
- Adding keys
  - To add itself a permission key, a process must have the permission key **keyzen.admin**
  - No process can add a key to an other process
- At startup, a process gains the keys set in the extended security attribute of name **security.keyzen**

prompting



KEYZEN server

2. query access to xx/555/keyA

PROGRAM Pid = 555

FUSE

FS

6. report the user decision

9. reply or error

UDS

1. query

User Prompt server

3. ask for prompt

7. reply of access granted or not

SERVICE A needs keyA

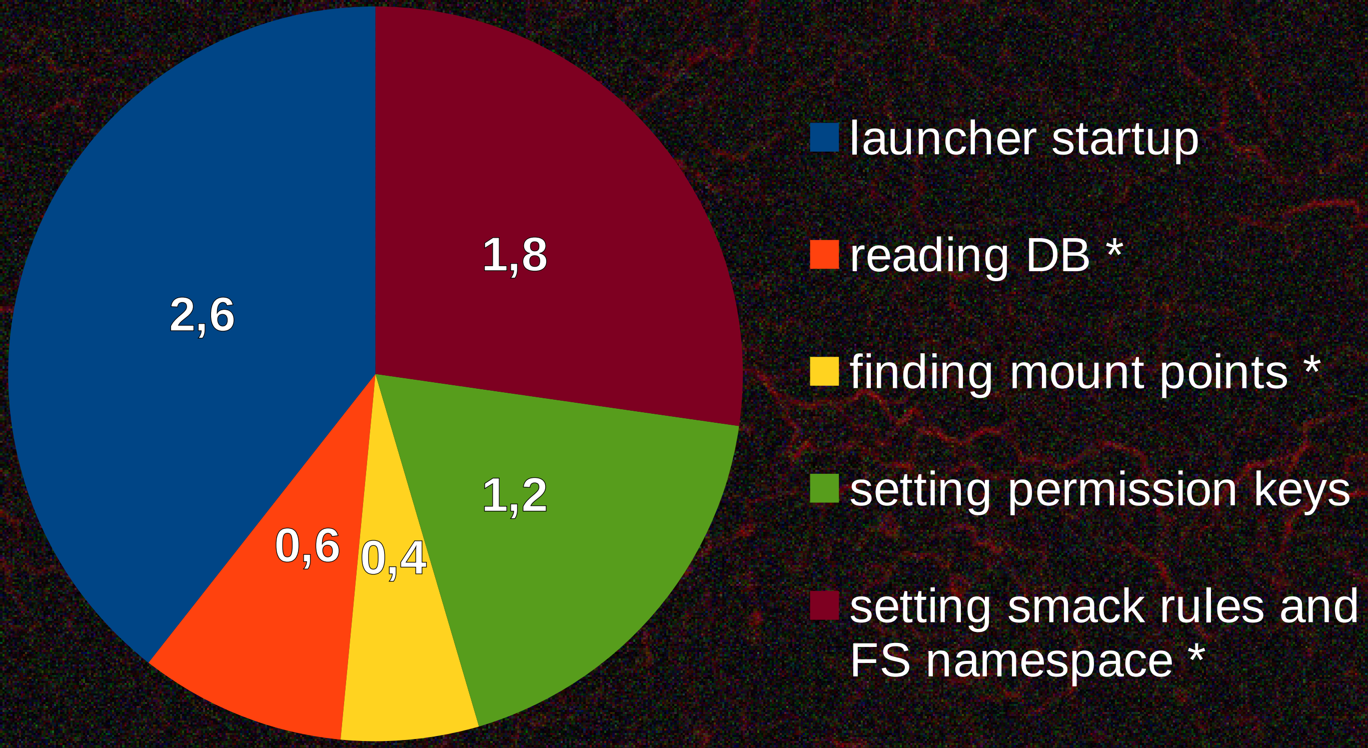
5. reply of the user: grant or not

8. if granted satisfy the query

4. prompt user for its authorisation

Timings

decomposition of the launching time of 6,6 ms



*: divisible at least by 2
when not using smack
(5,2 ms)

(measured on Intel's i7-2600 @ 3.4GHz)

Issues

- /proc is rotten
  - No notification on it (inotify)
  - Not extendible by LSM
  - Not extendible by FUSE (overlays not tried)
- LSM is half-rotten
  - Only one LSM at a time is possible
- Posix is rotten
  - exec does not plan any kind of context switch

Links

- KEYZEN
  - https://github.com/jobol/keyzen
- SMAUNCH
  - https://github.com/jobol/smaunch
- STAUNCH
  - https://github.com/jobol/staunch