The GNOME™ Conference

GUADEC

Desktop secrets management for the future

Daiki Ueno

Agenda

- Objective
- Solutions: past, present, and future
- Discussion

Objective

Allow applications to **store user credentials** in a uniform way

User credentials = some attributes + secret

- Sign-in forms on websites
- Access tokens for web services
- Wi-Fi passwords
- Protection passwords for SSH keys

Solutions

Past: `~/.netrc`

Invented for FTP clients, adopted by other applications

- User credentials are in plain text with matching attributes

```
machine foo login ueno password baz
machine bar port 587 login ueno password baz
default login anonymous password user@site
```

Potential attackers

- Other users on the system

Protection

- File permissions

Present: gnome-keyring and libsecret

The central daemon manages all the user credentials

- Applications are supposed to use a client library to access

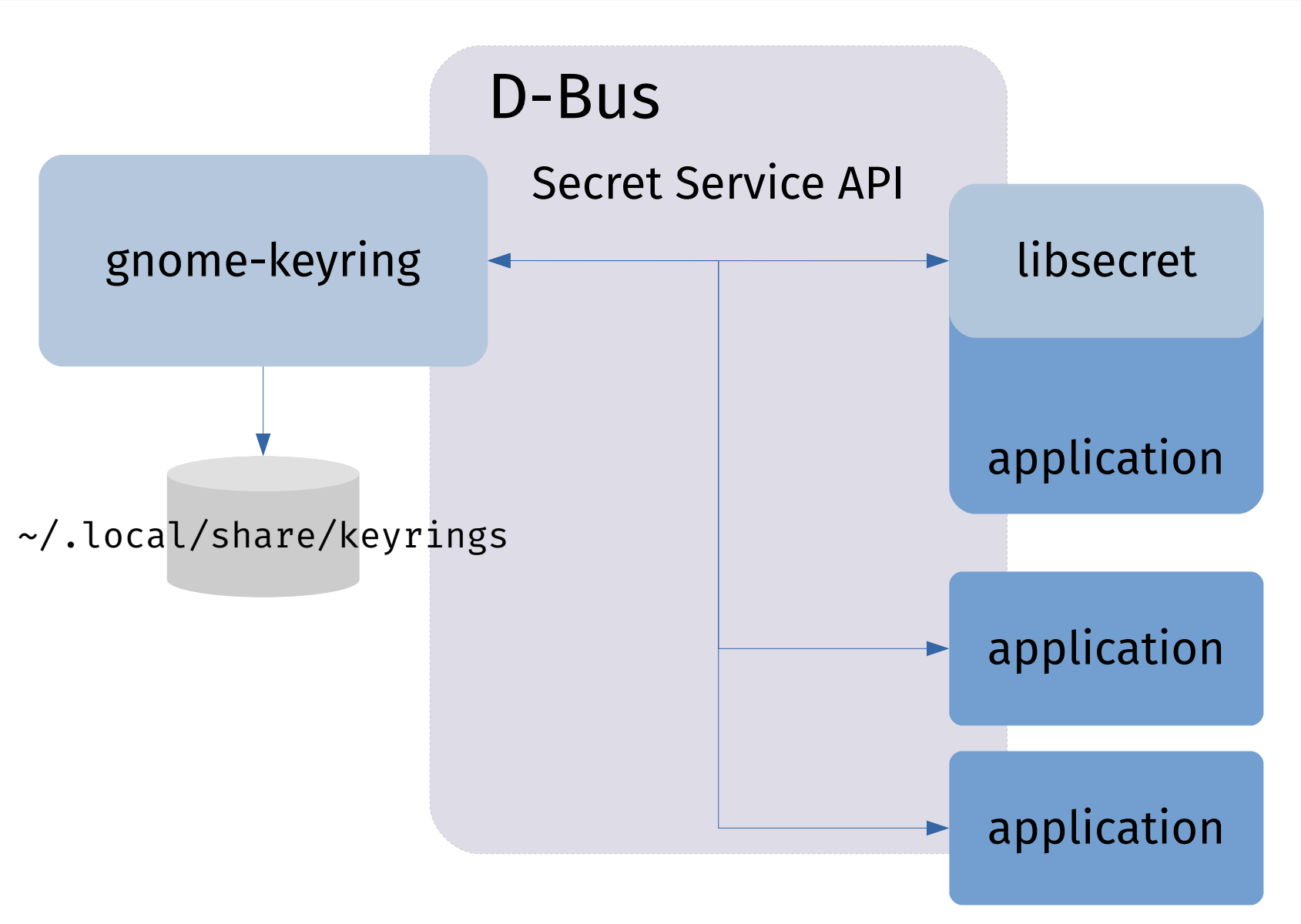
Potential attackers

- Other users on the system
- Active attackers who gain access to the disk

Protection

- File permissions
- Encryption

Present: gnome-keyring and libsecret



Future: is the current technology still relevant?

- Per-process isolation
- Keyring format

Per-process isolation

gnome-keyring doesn't provide isolation between processes

- One app can retrieve the other app's credentials
- flatpak makes installing third party apps easier than ever
- flatpak apps need a **hole** to access the Secret Service API

```
 /* Secret Service API */
"--talk-name=org.freedesktop.secrets"
```

Potential attackers

- Other users on the system
- Active attackers who gain access to the disk
- Malicious third party apps

Can we close the hole?

Can we close the hole?

Solution: Let's store credentials in application side

- Analogous to GSettings: "Settings, in a sandbox world"
- Master secrets are still needed for encryption

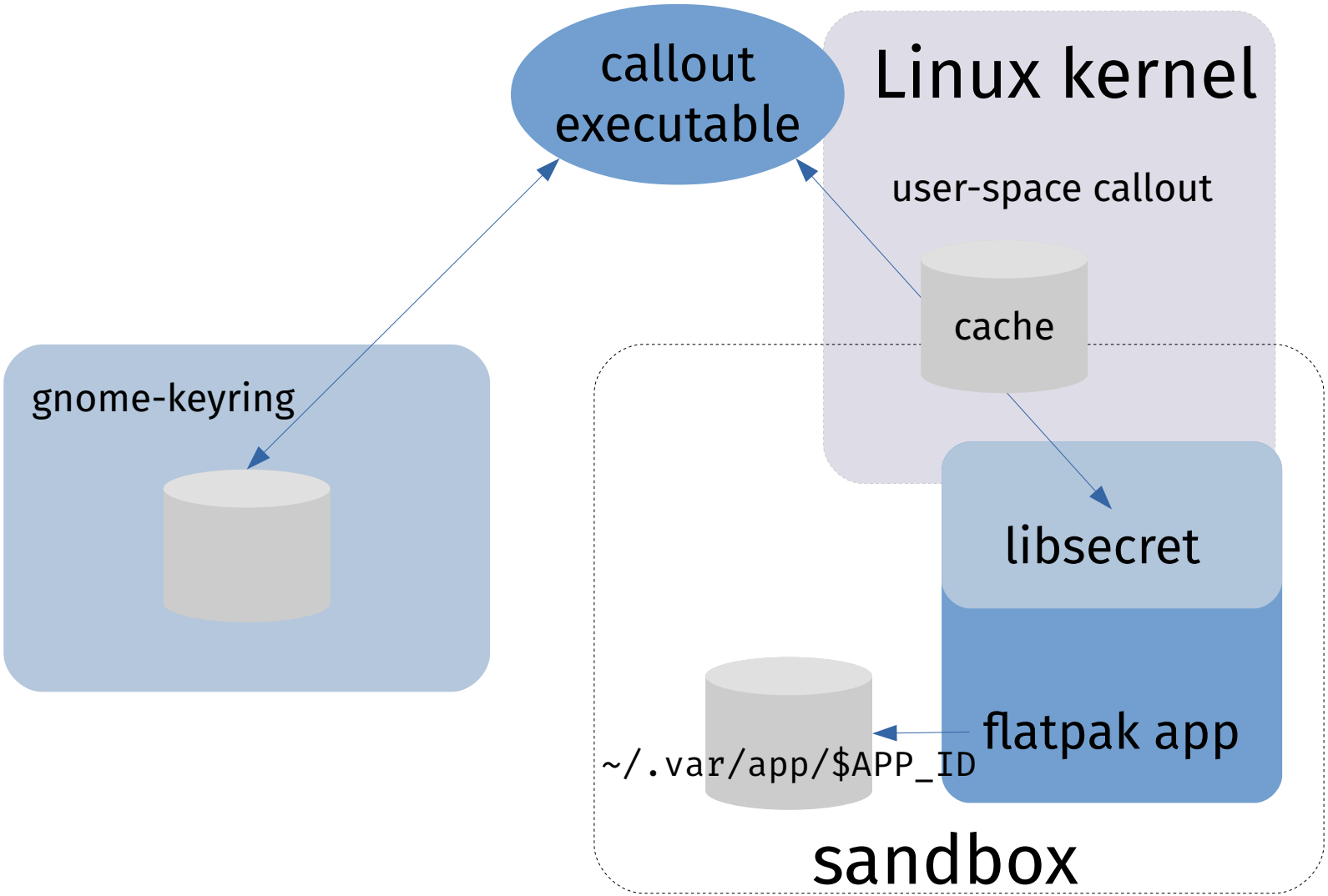
Approaches

- Kernel keyring upcall
- Portal and FD passing

Retrieving master secret: kernel keyring upcall

1. Look up kernel keyring for master secret
2. If not found, access gnome-keyring through upcall
3. Cache the retrieved secret in kernel keyring

Retrieving master secret: kernel keyring upcall



Retrieving master secret: kernel keyring upcall

Pros

- Master passwords are cached securely in the kernel
- No need for wire encryption

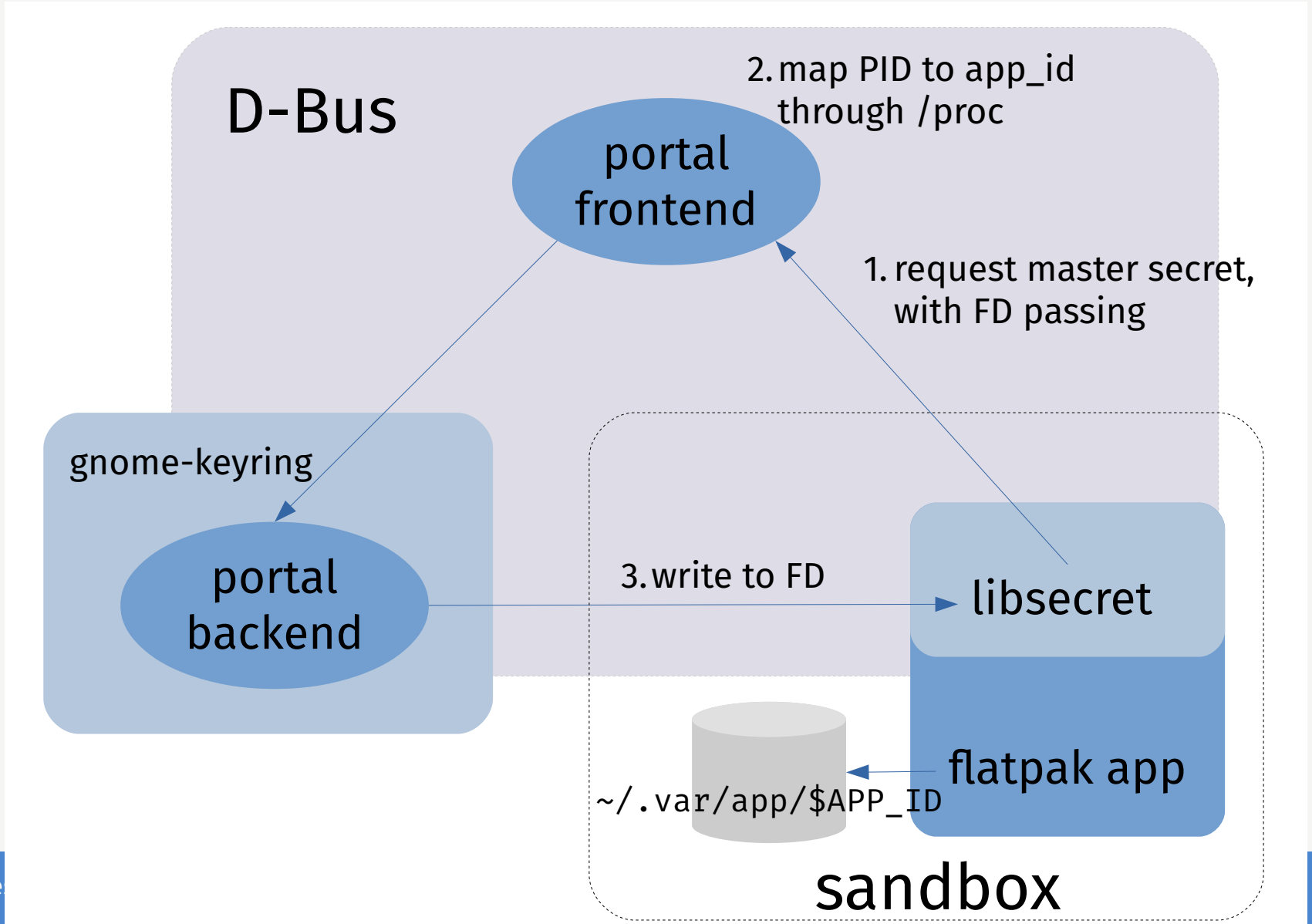
Cons

- The callout program lacks app information
- Impossible to cancel prompting if keyring is locked
- Linux only

Retrieving master secret: portal and FD passing

1. Ask flatpak portal for master secret
2. Portal redirects the request to gnome-keyring
3. Secrets are transported through FD passing

Retrieving master secret: portal and FD passing



Retrieving master secret: portal and FD passing

Pros

- No need for wire encryption
- Ability to cancel prompting by application
- Portable

Cons

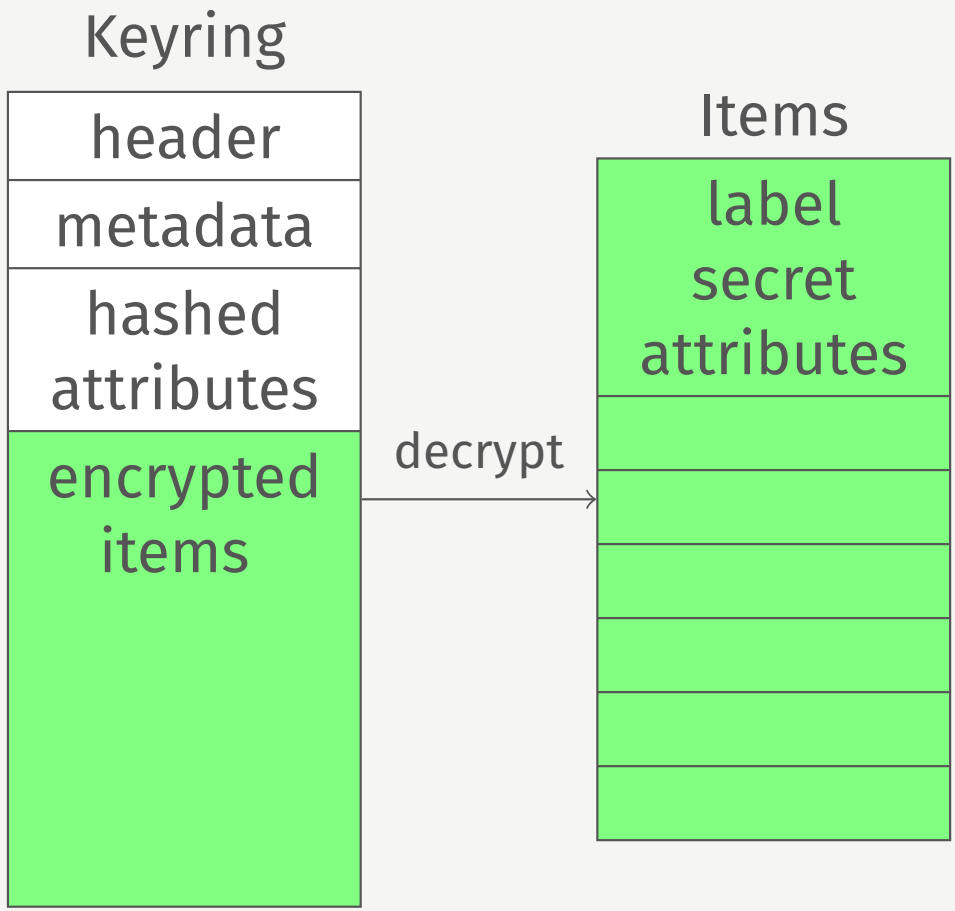
- Applications have more responsibility

Future: is the current technology still relevant?

- Per-process isolation
- Keyring format

Keyring format

- Custom binary format
- **Items are encrypted in a single chunk**
- SHA-256 for key derivation
- AES-128-CBC for encryption
- **MD5 for hashing attributes**



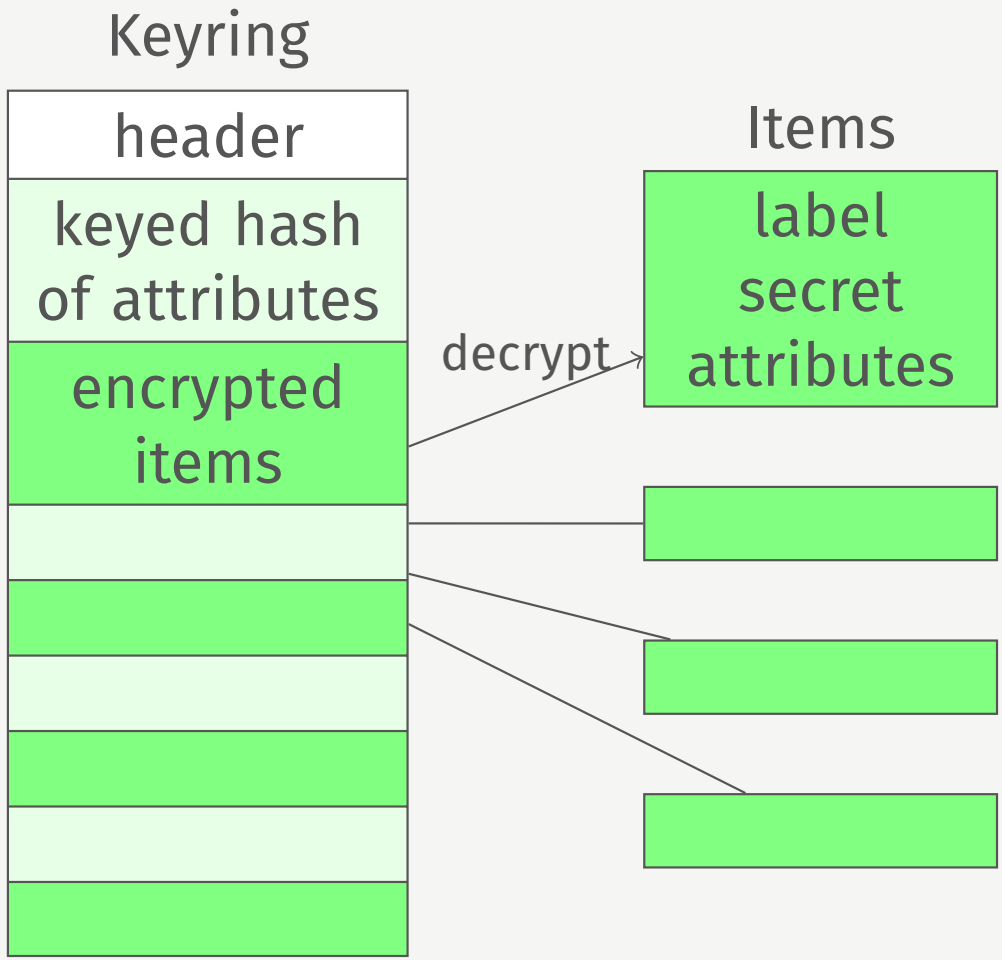
Keyring format

```
$ strings ~/.local/share/keyrings/login.keyring
GnomeKeyring
Login
host
 5f4a67b0504e75a1b659441bfdddd948
keyring
 42a31547b300ad0c1123774ec91f1a9b
unique
 8ae8c34a73360d83a5be7bbe68e930cd

$ echo -n ssh-store:/home/dueno/.ssh/sakura_rsa | md5sum
8ae8c34a73360d83a5be7bbe68e930cd  -
```

Proposed keyring format

- GVariant serialization format
- Items are encrypted individually
- PBKDF2
- AES-256-CBC
- MAC instead of simple hashing



Current status

Demo

- user credentials isolation under flatpak

4 open merge requests

- libsecret: Add local-storage backend
- libsecret: New interface to represent storage backend
- xdg-desktop-portal: Add a secret portal
- gnome-keyring: Implement secret portal backend

Discussion

Discussion

What happens if app ID is forged after reinstall?

- Remove app's keyring file on uninstall, or
- Ensure all apps are digitally signed

Can we close the hole entirely?

- Anonymize the host access using public key crypto
- TLS exporters, Clevis / Tang
- Need a physical boundary between the sandbox and host

Discussion

How about backup and mobility of secrets?

- Not feasible to store arbitrary credentials on HSM
- Only store a key pair used to encrypt master secrets
- Share the secrets on remote machine, as an encrypted file

Thanks!

Questions, comments or suggestions?

Image credit: view from a hole by spDuchamp, CC-BY-SA 2.0