New E2EE Content Types Epic - Business Case - PUB...

New E2EE Content Types Epics

Business Case & High Level Approach

Abstract

"Vous trouverez dans ces potains-là une foultitude de raisons pour que je me libertise." — (Victor Hugo, Les Misérables)

This document describes the business case for a collection of epics (a saga?) that will introduce, in passbolt, new end-to-end encrypted (E2EE) resource types, and bring an evolution to existing ones. It touches on how these changes will all fit together as part of the passbolt product evolutions, while maintaining backward compatibility.

It is aimed at stakeholders and developers to help drive the discussion about a major overhaul of what can be stored in passbolt, via multiple incremental building blocks that could be shipped with confidence and minimum end-user disruption.

For brevity's sake, we assume the reader has knowledge of client and server side core data structures (e.g. resources, resource types, secrets, folders, tags, comments tables/fields and models/entities) as well as the current security architecture. Check out the security white paper if you do not know about these.

Status: APPROVED
Classification: PUBLIC CC BY-SA 4.0

Related documentation:

- E2EE - FE Resource metadata usage - Analysis - INTERNAL
- E2EE - FE WP1 Resource metadata as associated entity and support v4 resou…
- E2EE - FE WP2 Preparation for Supporting v5 Resource Types - Front-End Te…
- E2EE - FE WP3 Support of v5 Resource Types - Front-End technical specificat…
- v5 - WP4 UI Redesign - Functional specifications
- E2EE - iOS new content types support - Requirements & technical approach
- E2EE Content Types Android
- E2EE - WP5 encrypted metadata administration & settings - Requirements & s…
- E2EE - WP6 v5 resource types creation/edition - Requirements & specifications

- v5 - WP7 Multiple URIs
- v5 - WP8 Icons
- v5 - WP9 Admin workspace redesign - Requirements & specifications

Change history

Date	Author	Changes
May 9, 2024	Remy Bertot	Initial draft
Q2/3 2024	Cedric, Remy	Many updates
Nov 18, 2024	Remy Bertot	Added migration
Apr 27, 2025	Remy Bertot	Post implementation review
Date	Person	

Table of content

Introduction

Problem definition

This section presents the high level functional requirements and the high level supporting technical approaches. While the actual work is/was broken down into smaller parts, the goal here is to try to understand the problem as a whole so that the future multiple incremental evolutions can be understood together.

Q1. What are the problems that we are trying to solve?

We consider the following problems, in scope of the discussion: the type of content that is stored passbolt is limited and not always end to end encrypted. Moreover true end-to-end encryption creates on-boarding and usability issues (a public key must be shared before anything else can happen, keys can be lost, etc.). It also creates observability issues for both administrators and SOC analysts (e.g. if the server has zero knowledge then how can it be expected to report accurately on usage).

Issue 1. Limited number of resource types

In a nutshell, as of version 4, while it is possible to store multiple resource types, their number and variation is limited. It includes:

- Password or passphrase or PIN, with or without a free text description
- TOTP, as standalone or combined with password

This combination has allowed passbolt users quite a bit of mileage, however it is not perceived as “feature complete”. Here are some examples of resource types that have been requested throughout the years:

DevSecOps items:

- Key/Value items, (typically an environment variable name and its value)
- JSON snippets
- YML snippets

PKI items (e.g. private key materials):

- SSL certificate (X.509)
- OpenPGP Keys
- SSH Keys
- Passkeys

Financial and personal information items:

- Credit cards
- Bank account details (IBAN, BIC, etc.)
- Contact information (Phone, Addresses)
- ID/membership information (number, name, dob, expiration date, issue date, etc.)

The “where do I put these” items:

- Markdown / readme notes
- PIN (phone, physical locks, luggage)
- Software license codes
- Recovery codes

Files and file attachment items:

- TXT
- Images: jpg, png, etc.
- PDF
- Others: Zip, Word, CSV, etc.

Issue 2. The resource types lacks some properties

The current content types lacks some functionalities from the above content types or properties such as:

- Icons and color
- Multiple URLs
- Password history
- Autofill guides
- Custom fields (Key/Value)

Issue 3. Some E2EE content is not searchable

We already know that searchability should be thought of as part of the solution.

For example in passbolt if the description is encrypted, it cannot be made searchable via the UI, without downloading the secret and thus triggering an entry in the audit logs. This additionally may push users to prefer unencrypted content as it remains more easily searchable.

User would face similar issues should passbolt store “key/value” items: users may expect to search for items using the “key” (e.g. a variable name, for example from an environment

variable on their CI for example) to get a “value” (e.g. a secret token, aka the concealed value in our CI example).

On the other hand some items do not need to be searchable. For example a password or a SSH/OpenPGP key cryptographic material is generally not expected to be searchable by part of its content or itself.

For some it's a mix of both, for example for a credit card, it could be expected to use the last few digits as a search, but not the entire number.

Issue 4. “Lack of zero knowledge” e.g. some content is readable server side

Some parts of the resources are not end-to-end encrypted, and thus available to an attacker that is able to compromise the server component. For example, the following parts can be perceived as part of the secrets by some/most users:

- Usernames
- URLs
- Resource names

Some parts are not directly attached to the resource and are also not encrypted:

- Comments
- Tag names
- Folder names
- Resource types slug and schema

As seen with the previous section about search issues, moving these parts of the resources to the secret part would not solve the issue, as they should be searchable client side.

Note: some parts are not encrypted and are actually needed by the server side logic, so they will remain out of scope for the foreseeable future:

- Creation/Edition/Deletion dates
- Expiry dates
- Permissions or group related items (group names, groups memberships)
- User info (email, name, roles)
- Object types (whether an entry is a Folder or Resource)

Issue 5. E2EE creates on-boarding issues

There are several issues with multiple use cases.

Administrators want to be able to on-board users and start sharing credentials with users before they complete the setup, e.g. before they register their public key in the system.

So it goes without saying that the E2EE metadata feature should not create additional on-boarding issues.

Issue 6. E2EE creates observability issues

Administrators want to be able to have information such as password entropy and/or whether a credential is/was part of a breach.

Additionally people working as security analysts, for example as part of an outsourced SOC, may not be users of the passbolt instance themselves, but must be able to review some of the information, such as whether a credential was accessed by which user, from where and when. Therefore this data sent to SIEMs is not end-to-end encrypted, as the SIEM integration happens server side.

Q2 - Who is impacted?

Pretty much all users of passbolt are affected by one or more of these problems. With some notable differences in terms of importance and/or scope. For now let's just say we've heard "everyone" complain about it, you included.

Q3 - Why is it important and/or urgent?

"M. de Lassay, a very indulgent man, but with a great knowledge of society, said that we should swallow a toad every morning, in order to fortify ourselves against the disgust of the rest of the day, when we have to spend it in society." — (Chamfort, Causeries Du Lundi)

As mentioned previously, the product is not perceived as feature complete without solving most of these issues. Additionally the fact that some data is not end to end encrypted makes the security-first positioning of passbolt more complicated to explain. Moreover some new content types such as passkeys require metadata to be encrypted twice in transit for FIDO future potential credential manager certification.

Finally this problem can be considered as passbolt greatest "toad": it is a complex and delicate problem that is unpleasant to get started with. However not solving it once and for all, is preventing us from feeling confident and ready to make strides in other areas. All in all,

It may well be the one task that can bring the greatest positive impact and results, and therefore by transitivity solve all the world's problems.

Q4 - What has been discussed as a solution?

Over the years some proposals have been discussed at the coffee machine, while some remained shower thoughts until now. Here is a dump so that you can get up to speed:

Assumptions

Server-side “zero-knowledge” of metadata is desirable for personal data for privacy reasons, but is not necessarily expected for enterprise deployment, where observability can be more important than privacy considerations.

Most password managers do not encrypt all metadata. Most have some form of escrow mechanism that allows the decryption of all the content.

Encryption of the metadata at rest and in transit is still a must-have requirement, and multiple layers can be implemented to provide defense in depth.

High level proposal

With v5.x we aim to encrypt all content that the database currently does not need for application logic (such as resource names, urls, etc.). Personal metadata can be made E2EE using the user key. Shared metadata can be made E2EE or encrypted at least one more time at rest and in transit. Shared metadata is available to administrators to allow for meaningful data transfer when deleting users/groups.

The content of what is encrypted varies for both secret and metadata parts. Their structure is flexible enough to accommodate the desired content types variations. The searchable content resides in the metadata and is decrypted upfront, when the user logs in. Secret behavior remains the same: the content that triggers secret access entry in the audit trail is not searchable and downloaded/decrypted only when used.

Remaining questions to answer

This proposal should raise the following eyebrows:

How can we deal with the performance hit of having to decrypt data prior to showing something on the screen?

How do we deal with the previously unencrypted data?

How to send E2EE encrypted data to SIEM? Or, how can data be both encrypted in one system and not another and stay somewhat secure?

High Level Approach

Main problem resolution

Solving the limited content type issue

New default content type

We propose to introduce a new content type which will be as follow:

```
{
  "id": <uuid>,
  "resource_type_id": <uuid>,
  "folder_parent_id": <uuid>,
  "metadata": "-----BEGIN PGP MESSAGE-----",
  "metadata_key_id": <uuid>,
  "metadata_key_type": <enum|null>,
  "secret": ["-----BEGIN PGP MESSAGE-----"],
  "personal": <bool>
  "created": <datetime>
  "modified": <datetime>
  "created_by": <uuid>,
  "modified_by": <uuid>,
  "deleted": <bool>,
  "expired": <datetime|null>,
}
```

```
{
  "object_type": "PASSBOLT_RESOURCE_METADATA", // sanity check
  "resource_type_id": <uuid>,            // idem
  "name": <string>,
  "username": <string>,
  "uris": [<string>]|[],                 // allow multiple uris
  "description": <string|null>,          // maybe in secret if not 🔎
  "custom_fields": [{                    // if unset <>resource_type_id
    "id": <uuid>,
```

```
    "key": <string>,
    "value": <string>,
    "type": <enum:"text", "boolean", etc.>
    "protec": <bool>
  }],
  "icon": {
    "type": "<enum:"icon-set", "base64-image", "emoji">",
    "value": "<string>",
    "background_color": <?string:7-9>, // null or #000000[00]
  },
}
```

Metadata, once decrypted, will contain the following for the new default contain type:

The secret part contains:

```
{
  "object_type": "PASSBOLT_SECRET_DATA",   // used for sanity check
  "resource_type_id": <uuid>,              // idem
  "password": <string|null>,
  "description": <string|null>,            // could be in metadata if 🔎
  "totp": <object>,                        // if unset <>resource_type_id
  "password_history": [<string>]|null,     // keep X prev passwords
  "custom_fields": [{
      "id": <uuid>,
      "type": <enum: key, value>,
      "value_type": <enum:"string">
      "value": <string>
  }],
 }
```

Solving performance issues on decryption of metadata

Using cached session keys

Why use session keys?

As we have seen, one of the challenges is the time it will take to decrypt the resources metadata if they are encrypted. There is one possible way to speed this up.

OpenPGP is a hybrid cryptosystem: it generally uses both asymmetric and symmetric encryption to encrypt a plaintext. In all cases the plaintext is encrypted symmetrically using a random session key (used directly or derived as a message key using a salt in OpenPGP v6). The session key is then in turn encrypted (once or separately multiple times), using asymmetric OpenPGP key material(s) (e.g. one or more user public keys), and/or a

symmetric key (e.g. a key derived from a passphrase using a key derivation function, either Iterated and Salted S2K or Argon2 in v6).

In short, in OpenPGP one can see the following packets:

Term	Description
SEIPD Packet	Symmetrically Encrypted, Integrity Protected Data packet; contains the encrypted plaintext
SKESK Packet	Symmetric-Key Encrypted Session Key packet; contains or provides a passphrase-encrypted session key
PKESK Packet	Public-Key Encrypted Session Key packet; contains a session key encrypted using an asymmetric public key
Session Key	Symmetric encryption key, which is either used directly or to derive the message key
Message Key (v6)	Symmetric encryption key used to encrypt the contents of the SEIPD packet in OpenPGP v6.

What is currently the most costly is not the decryption of the ciphertext using the session key, but the process of decrypting the session key.

Therefore we propose to cache session keys when encrypting/decrypting metadata so that once the session key is known, it can be reused in the future, without the initial cost. It also doesn't affect the security model or the usability of the OpenPGP messages.

Example using GnuPG:

```
# View a session key for a given message
$ gpg --show-session-key file.gpg
gpg: session key: '9:AA95E9BD1..'

# Decrypt using a session key
$ gpg --override-session-key 9:AA95E9BD1.. -d < file.gpg
```

Example using OpenPGP.js:

```
// Generate session key
const sessionKey = await openpgp.generateSessionKey(
```

```
  openpgp.enums.symmetric.aes256
);

// Or reuse one
const hexString = 'AA95E9BD10D4...';
const byteArray = new Uint8Array(hexString.match(/.{1,2}/g)
                        .map(byte => parseInt(byte, 16)));
const sessionKey = { data: byteArray, algorithm: "aes256" };

// Encrypt
const armoredMessage = await openpgp.encrypt({
    message: await openpgp.createMessage({ text: '<json>' }),
    encryptionKeys: publicKey,
    sessionKey: sessionKey
});

// Decrypt
const {data: decryptedMessage, signatures} = await openpgp.decrypt({
    message: armoredMessage,
    sessionKeys: [sessionKey]
});
```

To get an idea of benchmarks we get the following for a 10K of operations:

Case	Results (in ms)
OpenPGP.js v5 / Node / M1 Max	
Encrypt & sign with curve25519 keys	43000
Encrypt with curve25519 public key	16000
Encrypt with curve25519 public key & session keys	**15000**
Encrypt with pre-selected session keys only	450
Decrypt & verify with curve25519 keys	62000
Decrypt with curve25519 key	9000
Decrypt with session key	**1200**
OpenPGP.js v5 / Chrome 127 (debugger off) / M1 Max	
Encrypt with curve25519 public key & session keys	**11000**
Decrypt with session key	**676**
GopenPGP v?/ iOS ? / Iphone ? (source)	

Encrypt with curve25519 public key & session keys	**1000**
Decrypt with session key	**2200**
GopenPGP v?/ Android / Pixel (source)	
Encrypt with curve25519 public key & session keys	**800**
Decrypt with session key	**1900**

Note: Using binary OpenPGP messages format (instead of armored), can also improve performance by around 30%. To stay consistent with secret entries, we'll keep using the armored format.

How will the session key cache work?

The idea is to make the session key cache optional. If a client uses the cache they will be able to decrypt resources metadata faster using session keys. If not they will be able to decrypt normally using the private key.

The cache will look like this:

```
[{
  "foreign_key_id":"<UUID>",
  "foreign_model":"<enum:Resource|Folder|Tag>",
  "session_key_data": <string>,
  "session_key_algorithm":"aes256"
}]
```

The high level logic for the client would be as follow:

```
1. When the user login
2. If there are session key cache file(s)
       a. Download cache file(s)
       b. Decrypt
       c. (Merge them)
3. Download the resource metadata
4. For each resources
       a. If there is a session key for the resource id in cache
            i.   Decrypt using session key
       b. Else
            i.   Decrypt using user or metadata private key
           ii.   Add session key to the cache
5. If cache has changed
       a. Encrypt the new cache
```

```
        b. Sync' it server side
```

Note: It could be possible that passbolt also makes use of this system to allow users doing a large share with other users by allowing the person who is doing the share to create a session key cache file related to that share to help the other users. This will not be implemented in the first iteration.

Solving the "Zero-knowledge" conflicting requirements

We propose to introduce a public/private key pair shared by all users of the passbolt instance. This key will be used to encrypt metadata for the resources that are shared (or created with the intent of being shareable, more on that later).

We also propose to allow users to use their personal OpenPGP key to encrypt data that is not shared with other users. There will be a setting for the administrators to disable this behavior, e.g. force all the metadata to be shared.

Shared Metadata Keys

The shared public/private key pairs will be used by users for shared resources to access resource metadata. These keys are shared with new users when they complete the setup, and stored encrypted once per user using the user public key.

To accomplish this we propose to build two strategies:

- **"User-friendly mode"**: the shared metadata keys are available to the server and can be shared by the server when a user completes the setup. The shared session keys values are available to the passbolt application. The application can in practice see shared metadata. There is no actual zero-knowledge, but this is still a nice improvement, since for example data is encrypted additionally at rest (in the database and backups) and in transit.
- **"Zero-knowledge mode"**: the shared metadata keys are not available to the server and must be shared with users by the admins. Users are not allowed to create shared content unless they have access to a valid metadata key. This will be a behavior that is supported in both modes, to allow administrators to recover from situations for example when sharing a metadata key with a user failed during setup.

This will allow these metadata shared keys management to handle different security requirements.

Future iterations could introduce other modes such as relying on a service hosted on a different server, or even be done in collaboration with an app client side push, or work with dedicated hardware devices, etc. to offer different security properties.

What will key rotation look like?

We propose to support key rotation upfront. It may be required to rotate the keys, for example when a key is leaked, when a user leaves the organization or when some legal requirements are met (some regulated industries require this). It could also be used to switch from “User friendly mode” to “Zero-knowledge mode” once the user onboarding phase is completed for example.

Note that in this document we talk about metadata keys in plural, we indeed aim to support having multiple keys in parallel to allow for key rotation.

The process will be as follow:

1. Admin create a new metadata key
2. Admin share new metadata key with users
3. Admin migrate items metadata to use the new keys
4. Admin delete the old key if there is no item using that key

We allow these steps to be performed independently, so between step 1 and 4 there might be several keys supported at the same time, with the goal of converging ultimately to one.

What does it mean in terms of security compared to what we have now?

Risk scenario	Improvement?
1. Unencrypted backup access: attacker manages to decrypt / access cleartext backups and view metadata.	All modes
2. SQL Injection: an attacker manages to bypass application control to view metadata that is stored in the database.	All modes
3. MiiM: an attacker manages to break SSL to view metadata that is transmitted on the wire.	All modes
4. Remote code injection: attacker manages to execute some PHP code as part of a specific request.	Zero-knowledge mode

5. Complete server access control: an attacker manages to execute any code on the passbolt server.	Zero-knowledge mode
6. Complete infrastructure control: an attacker can access any server and run any code on the infrastructure side. (Aka “Zero-knowledge”)	Zero-knowledge mode
7. Complete client control: an attacker can access memory or have full file access on the client side.	Out of scope
8. OpenPGP public key injection: an attacker is able to impersonate as a legitimate user / trick a user to share a secret with them.	Out of scope

What is the impact on API backward compatibility with v4?

The benefits of shared metadata keys are interesting in terms of API backward compatibility:

- The orchestration of the create / edit / share resource operations remains the same. When sharing a resource there is no need to re-encrypt the metadata to share the resource with a user if a shared metadata key was used.
- The orchestration of the update / delete users and groups operations remains the same. Administrators need access to all resources to see basic information in order to transfer the resource when deleting users and groups. Promoting a user to admin will not require re-encryption of all resources metadata.

Personal Resource Metadata

When data is not shared it must be possible to use the end user key to encrypt the metadata, providing full zero-knowledge in this case.

Note that administrators with account recovery in place will be able to recover user data if they have access to the organization recovery key and access to the user mailbox (or server access). However the server will not have access to this information.

What does it mean in terms of security compared to the “shared metadata key” scenario?

Risk scenario	Improvement?
5. Complete server access control: an attacker manages to execute any code on the server.	Yes

6. Complete infrastructure control: an attacker can access any server on the infrastructure side.	Yes

What happens when private/shared session keys switching is needed?

Scenario Personal → Shared, a private resource is shared

In this scenario when calling the share API and sending the permission and secret for the new user(s), the API must detect that an end-user key was used instead of the shared metadata key and throw a new error. The client must update the resource entry and encrypt the metadata with the session key, then share.

Note that it should be possible to create a resource with the shared metadata key, in order to speed up the process when sharing (see next session), e.g. when the client knows that the next action is to share the secret (like when creating a resource in a folder with more than one permission).

Moreover in the future we could also envision a setting where the administrator chooses at an organization level a policy that enforces the use of the shared metadata key, e.g. refuse the use of personal keys, for example for compliance reasons.

Scenario Shared → Personal, a shared resource is now owned by one person

In this scenario the share happens normally. The resource is marked as expired by the API. The next time the resource is updated, the client could reencrypt the metadata and make use of the user private session key.

Solving SIEM Integration issues

Obviously if our aim is for the server to not know about the metadata in “Zero-knowledge mode”, it can not spill the beans in the logs.

In the middle-term the SIEM integration will need to be moved to a separate service, tied to a user/service account with access to the shared metadata key. This or not sending any metadata information to the SIEM in “Zero-knowledge mode” beyond phase 0.

Solving rollout and backward compatibility issues

The approach towards backward compatibility is to progressively roll out the new format and new resource types, while leaving the administrators in control of the timeline. This means that by default the v5 will accept v4 entities in and out and will not migrate or transform them, until the administrator has selected it. Once upgraded however it is not possible to go back.

Here are the migration steps:

Passbolt version	Stage
Version 4.9.X - Starting point.	Phase 0 (Q3 2024)
Version 4.10.X - Introduces most v5 functionalities behind a feature flag. This is mostly to allow developers to work on migrating clients, it is not meant for production use.	Phase 1 (Q4 2024)
Version 5.0.0 - Introduces officially the new default resource types. It introduces migration tools to migrate resources from v4 to v5 as well as admin settings to define which resource types are allowed to be used.	Phase 2 (Q1 2025)
Version 5.X.0 - Continues with the remaining metadata encryption, such as folder name, comments and tags. Add an unencrypted resource metadata type if there is a push back. After six months warnings start to appear that v4 format will be dropped.	Phase 3 (Q2 2025)
Version 6.0.0. Deprecates officially v4 format. Support for unencrypted resource metadata v4 format is dropped by default and cannot be re-enabled in settings.	Phase 4 (2026)

Security considerations

Risk analysis

Risk	Mitigation
A user leaving the organization while they don't have logical access to resources, could still decrypt using previously known shared metadata keys.	Allow for the creation of multiple shared keys. Allow for the rotation of shared keys. Administrators can choose to rotate shared keys as one off every X days at their convenience. (Note: This will reset session key cache / impact performance)
An attacker with server access / infrastructure wide access is able to decrypt shared metadata if the key is shared with the server to ease user onboarding.	Provide settings for the administrator to decide to accept the risk or not.
An attacker with admin access is able to change the configuration to downgrade to unencrypted types.	Allow to lock configuration using environment variables to prevent edition from the UI. Allow settings to prevent downgrade. Changes of default resource type from encrypted to unencrypted should trigger a user confirmation dialog in the clients.
An administrator may upgrade a resource to v5 even though it's used by a client that is not compatible with v5 creating compatibility issues.	Provide administrator with a setting to prevent upgrade of resource types. V5 rollout must make sure all official clients are ready. Allow the v5 to function in v4 mode.

Technical Specifications

Encrypted metadata

Resource types needed for backward compatibility

For backward compatibility keep the existing resource types:

- password-string
- password-and-description
- password-description-totp
- totp

The v4 types can be migrated to v5 using an endpoint (see below). Additionally it will be possible for an administrator to mark v4 as deleted.

New E2EE Resource Types

We also introduce the new following types:

- v5-password-string
- v5-default
- v5-totp-standalone
- v5-default-with-totp

```
GET /resources/<uuid>.json

{
  "id": <uuid>,
  "resource_type_id": <uuid>,
  "folder_parent_id": <uuid>,
  "metadata_key_id": <uuid>,
  "metadata_key_type": <enum:user_key, shared_key>,
  "metadata": "-----BEGIN PGP MESSAGE-----",
  "secrets": ["-----BEGIN PGP MESSAGE-----"],
  //etc.
}
```

Here are the supported transitions for resources:

From	To
Resources type	

password-string	v5-password-string
password-and-description	v5-default
password-description-totp	v5-default-with-totp
totp	v5-totp-standalone

Note that v5 will not accept the creation of v5-password-string, they can only be used when migrating from v4.

Support for E2EE Folders & Tags

We follow a similar approach for encrypted folders names and tags slugs. We add an additional field metadata, and some settings to allow migrating from versions to versions. The logic to de-duplicate tags is moved client side.

Support for E2EE Comments

Comments can now also be encrypted. They are encrypted directly using the users public keys, e.g. all the users that have access to the secret at the time the comment is written can decrypt it, much like an encrypted email message. Users that are added later cannot see these comments. Comments are not displayed by the clients if they cannot be decrypted.

Settings management

It's important for the application to function properly that the configuration is kept in a working state. For example, it should not be possible to delete a key that is still in use. Or you cannot require a user to encrypt a metadata key if one refuses the use of personal keys and provides no shared key.

Metadata key management

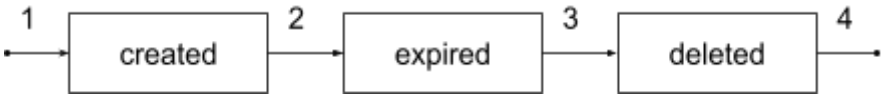

Fig. state machine for metadata key management

The possible actions to change the states for metadata key management are quite simple:

1. **Create** (from nothing to created, aka “active”): it’s possible for an administrator to create a new metadata key, e.g. a new “active” key. There can only be max two in non-expired / non-deleted ones at a given time. We don’t want to have multiple active keys for now.
2. **Expire** (from created to expired): In this state there might still be some content using the key, the content can be decrypted, but it’s not possible to create new content or edit existing content with the expired key. New users will be given access to expired keys in order to decrypt legacy content. In order to move an active key to the expired state, the key must already exist, e.g. you cannot create an expired key. If it is the last “active” key, then the user must not be required to create v5 content by the content type settings and not be allowed to use personal keys.
3. **Delete** (from expired to deleted): it is possible to delete an expired key, but only if there is no more content associated with it. With this the key is officially retired and needs not be shared between users. When a key is deleted the private key materials are also deleted.
4. **Erase** (from deleted to nothing): currently not on the menu for the API, as we want to keep an history of the keys. It’s still possible to be performed manually in the database by adventurous admins.

Type settings management

The type settings must be coherent with the metadata key settings:

- There must be at least one resource type of the default selected preferred version (e.g. one resource type v5 if v5 is default).
- If v5 is default, one must be allowed to use personal keys, or there must be at least one shared key.

Database tables

Planned migrations

v4.1X

- Add new resource_types entries: v5 resource types (PB-34446)
- Update resource_types table to add deleted field (PB-34448)
- Update resources table for v5 resource type support (PB-34450)
- Update resource table to make name nullable

- Update folders table to add encrypted metadata fields (PB-34452)
- Update folders table to make name nullable (covered with PB-35361)
- Update tags table to add encrypted metadata fields (PB-34451)
- Update tags table to make slug nullable (covered with PB-35416)
- Update comments table (PB-34455)
- Add metadata_session_keys table (PB-34453)

v5.0

Migrations wise we're squishing v2 and v3 migrations. Additionally there is migration to detect that if there are no v4 resources in the database, then these types are disabled (they can be re-enabled in the admin, but we're not encouraging the use).

Other migrations:

- Migrate resources.deleted to timestamp|null (instead of boolean)

v6.0

- Delete v4 resource types if not done already
- Update resource tables
    - Make metadata field is not nullable
    - drop name, url, username fields
- Update tags table
    - Drop slug field
- Update folders table
    - Drop name field
- Update comments table
    - Drop content field

resource_types (new entries + updated table)

See. previous section for new entries. We add two fields to the `resource_types` table, one to allow to soft delete entries,

Field	Type	Comment
deleted	datetime or null	

resources (updated table)

We add three fields to the `resources` table:

Field	Type	Comment
metadata	mediumtext, null	16MB max
metadata_key_id	uuid, null	Ref. metadata_keys.id or gpgkeys.id
metadata_key_type	enum, null	user_key, metadata_key, or null

folders (updated table)

We add three fields to the `folders` table:

Field	Type	Comment
metadata	mediumtext, null	
metadata_key_id	uuid, null	Ref. metadata_keys.id or gpgkeys.id
metadata_key_type	enum, null	user_key, metadata_key, or null

tags (updated table)

We add three fields to the `tags` table:

Field	Type	Comment
metadata	mediumtext, null	
metadata_key_id	uuid, null	Ref. metadata_keys.id or gpgkeys.id
metadata_key_type	enum, null	user_key, metadata_key, or null

comments (updated table)

We create one field to the `comment` table, this will allow us to store encrypted comments.

Field	Type	Comment

data	mediumtext, null	16MB

metadata_keys (new table)

This table stores the information related to a given metadata public key.

Field	Type	Comment
id	uuid, not null	PK
fingerprint	varchar(51)	
armored_key	text, not null	
created	datetime, not null	
modified	datetime, not null	
deleted	datetime, null	
created_by	uuid, not null	Ref. users.id
modified_by	uuid, not null	Ref. users.id

metadata_private_keys (new table)

This table stores the information related to a given metadata private key. The data contains an asymmetrically encrypted OpenPGP message, encrypted with a given user (or server) public key. This decrypted message contains the shared private key pair. The key that does not have a user_id is the one used by the server.

Field	Type	Comment
id	uuid, not null	PK
metadata_key_id	uuid, not null	Ref. metadata_keys.id
user_id	uuid, null	Ref. users.id
data	mediumtext, not null	
created	datetime, not null	
modified	datetime, not null	

created_by	uuid	Ref Users.id
modified_by	uuid	Ref Users.id

Note: no deleted field on private keys. They will be hard deleted only as they contain sensitive information. Created_by and modified_by can be null as server may manipulate them.

metadata_session_keys (new table)

This table stores the information related to the OpenPGP session key cache for a given user.

Field	Type	Comment
id	uuid, not null	PK
user_id	uuid, not null	Ref. users.id
data	mediumtext, not null	
created	datetime, not null	
modified	datetime, not null	

Decrypted metadata entities

Resource

Field	Type	Comment/Validation
object_type	string, not null	Constant: PASSBOLT_RESOURCE_METADATA required
resource_type_id	uuid, not null	required
name	string, not null	required, length: 255
username	string	length: 255
uris	array<string>	uri length: 1024
description	string	length: 10000

icon	object	TBD

Example:

```
{
  "object_type": "PASSBOLT_RESOURCE_METADATA",
  "resource_type_id": "<uuid>",
  "name": "my password",
  "username": "john",
  "uris": ["https://passbolt.com"],
  "description": "description",
  "autofill_mappings": null,
  "custom_fields": null,
  "color": "#000000",
  "icon": null
}
```

Folders

Field	Type	Comment
object_type	string, not null	PASSBOLT_FOLDER_METADATA
name	string, not null	
color	string, nullable	Hex color format
description	string, nullable	length: 10.000
icon	string, nullable	See resource

The information will look like:

```
{
  "object_type": "PASSBOLT_FOLDER_METADATA",
  "name": "folder",
  "color": "#000000",
  "description": null,
  "icon": null,
}
```

Tags

Field	Type	Comment

object_type	string, not null	PASSBOLT_TAG_METADATA
slug	string, not null	
color	string, nullable	Hex color format
description	string, nullable	
icon	string, nullable	See resource

The information will look like:

```
{
  "object_type": "PASSBOLT_TAG_METADATA",
  "slug": "tag",
  "color": "#000000",
  "description": null,
  "icon": null,
}
```

Comments

TODO

Decrypted secret entities

Default v5 secret

Field	Type	Comment
object_type	string	PASSBOLT_SECRET_DATA
password	string	maxLength: 4096 allow empty nullable: true (**vary from v4**) required: true
description	string	maxLength: 50000 (**vary from v4**), nullable: true (vary from v4) required: true (vary from v4)

The information will look like:

```
{
  "object_type": "PASSBOLT_SECRET_DATA",
```

```
  "password": "password",
  "description": "description",
}
```

V5 standalone totp secret

Field	Type	Comment
object_type	string	PASSBOLT_SECRET_DATA
totp	TotpEntity	Object See TOTP secret part schema

The information will look like:

```
{
  "object_type": "PASSBOLT_SECRET_DATA",
  "totp": {
    "secret_key": "DAV3DS4ERAAF5QGH",
    "period": 30,
    "digits": 6,
    "algorithm": "SHA1"
  }
}
```

TOTP secret part

Field	Type	Validation
secret_key	string	maxLength: 1024
period	integer	min: 1
digits	integer	minength: 6 maxLength: 8
algorithm	string	enum: "SHA1", "SHA256", "SHA512"

The information will look like:

```
{
  "secret_key": "DAV3DS4ERAAF5QGH",
  "period": 30,
  "digits": 6,
  "algorithm": "SHA1"
```

```
}
```

Default v5 totp secret

Field	Type	Validation
object_type	string	PASSBOLT_SECRET_DATA
password	string	See Default v5 secret schema
description	string	See Default v5 secret schema
totp	TotpEntity	Object See TOTP secret part schema

The information will look like:

```
{
  "object_type": "PASSBOLT_SECRET_DATA",
  "password": "password",
  "description": "description",
  "totp": {
    "secret_key": "DAV3DS4ERAAF5QGH",
    "period": 30,
    "digits": 6,
    "algorithm": "SHA1"
  }
}
```

v5 password string secret

A password string has no schema, this schema is used by the browser extension to ease the manipulation of password string type.

Field	Type	Comment
object_type	string	PASSBOLT_SECRET_DATA
password	string	maxLength: 4096 allow empty nullable: true

The information will look like:

```
{
  "object_type": "PASSBOLT_SECRET_DATA",
  "password": "password",
}
```

Default v4 (previously password and encrypted description) ●

Field	Type	Comment
password	string	maxLength: 4096 required allow empty
description	string	maxLength: 10000, nullable: true

The information will look like:

```
{
  "password": "password",
  "description": "description",
}
```

Default v4 totp secret (previously password and encrypted description with totp) ●

Field	Type	Comment
password	string	maxLength: 4096 required
description	string	maxLength: 10000, nullable: true
totp	TotpEntity	association required

The information will look like:

```
{
  "password": "password",
  "description": "description",
  "totp": {
```

```
    "secret_key": "DAV3DS4ERAAF5QGH",
    "period": 30,
    "digits": 6,
    "algorithm": "SHA1"
  }
}
```

v4 standalone totp secret (previously standalone totp)

Field	Type	Comment
totp	TotpEntity	association required

The information will look like:

```
{
  "totp": {
    "secret_key": "DAV3DS4ERAAF5QGH",
    "period": 30,
    "digits": 6,
    "algorithm": "SHA1"
  }
}
```

v4 password string secret (previously password string)

A password string has no schema, this schema is used by the browser extension to ease the manipulation of password string type.

Field	Type	Comment
password	string	maxLength: 4096 allow empty

The information will look like:

```
{
  "password": "password",
}
```

API Specifications - Metadata keys

POST /metadata/keys.json (new) ● PB-34467 ●

Allow an administrator to create a new metadata key. It is done by sending the public key and multiple OpenPGP messages encrypted for the server key and all the existing users using a private key armored block and the associated passphrase if any.

```
POST /metadata/keys.json
Authentication: Session or JWT

{
  "armored_key": "-----BEGIN PGP PUBLIC KEY BLOCK-----",
  "fingerprint": "<string:40>",
  "metadata_private_keys": [{
    "user_id": null,                    // for server (optional)
    "data": "-----BEGIN PGP MESSAGE-----"
  }, {
    "user_id": <uuid>,                  // for users
    "data": "-----BEGIN PGP MESSAGE-----"
  }]
}
```

The “data” once decrypted contains:

```
{
  "object_type": "PASSBOLT_METADATA_PRIVATE_KEY",  // sanity check
  "domain": "https://<domain>/[<workspace>]",      // sanity check
  "fingerprint": "<string:40>,                     // sanity check
  "armored_key": "-----BEGIN PRIVATE KEY -----",
  "passphrase": "", // must be empty if server, could be set for users
}
```

Response

See. GET /metadata/keys.json

Error response

New errors:

- 401 if the user is not logged in
- 403 if the user is not an administrator
- 400 if armored key is missing, not valid text, not valid public key, expires
- 400 if fingerprint key is missing, not valid fingerprint, not matching public key

- 400 if the metadata_private_keys data is not an array of user_id and data
- 400 if in metadata_private_keys there is more than one item with user_id set at null
- 400 if there is one or more metadata_private_keys data that is not a valid OpenPGP message
- 400 if metadata_private_keys data is not encrypted with the server key if user_id i set to null
- 400 if in metadata_private_keys there is one or more of the user_id is not a uuid
- 400 if metadata_private_keys user_id does not belong to an existing active user (deleted or not active, disabled user is allowed)
- 400 if there are already two active metadata keys you cannot create a third one.

Note for frontend: What about a large payload? Like 10k users? The idea is to start sending them using POST /metadata/keys.json and then use POST /metadata/keys/privates.json to complete. This way you can control how many private keys you send at once.

GET /metadata/keys.json (new) ● ▾ PB-34471 ● ▾

Logged in users can use this endpoint to get the list of metadata keys available to them (encrypted with the user public key). In most cases there will be only one, unless administrators elect to rotate them, or we offer multiple ones (e.g. future evolution). A filter allows users to get the expired, deleted ones also (as may be needed if there is still non-migrated content associated with them).

```
GET /metadata/keys.json
  ?filter[deleted]=1
  &filter[expired]=1
  &contain[metadata_private_keys]=1
  &contain[creator]=1
  &contain[creator.profile]=1
Authentication: Session or JWT
```

Response

```
{
  "header": {
      "id": "<uuid>",
      "status": "success",
      "servertime": 1723453376,
      "action": "<uuid>",
      "message": "The operation was successful.",
      "url": "/metadata/keys.json",
      "code": 200
```

```
  },
  "body": [{
    "id": "<uuid>",
    "fingerprint": "<string:40>",
    "armored_key": "-----BEGIN PGP PUBLIC KEY BLOCK-----",
    "created": "<datetime>",
    "modified": "<datetime>",
    "expired": null,
    "deleted": null,
    "created_by": <uuid|null>,
    "modified_by": <uuid|null>
    "metadata_private_keys": [{
      "id": "<uuid>",
      "metadata_key_id": "<uuid>",
      "user_id": <uuid>,
      "data": "-----BEGIN PGP MESSAGE-----",
      "created": "<datetime>",
      "modified": "<datetime>",
      "created_by": <uuid|null>,
      "modified_by": <uuid|null>
    }],
    "creator": {
      "id": "<uuid>",
      "role_id": "<uuid>",
      "username": "kristofer.kohler@passbolt.local",
      "active": true,
      "deleted": false,
      "disabled": null,
      "created": "2016-08-21T11:43:51+00:00",
      "modified": "2025-04-24T11:43:51+00:00",
      "profile": {
        "id": "<uuid>",
        "user_id": "<uuid>",
        "first_name": "Isabel",
        "last_name": "Kemmer",
        "created": "2009-10-12T11:43:51+00:00",
        "modified": "2024-08-05T11:43:51+00:00",
        "avatar": {
          "url": {
            "medium":
"https://passbolt.local/img/avatar/user_medium.png",
            "small": "https://passbolt.local/img/avatar/user.png"
          }
        }
      },
      "last_logged_in": null
    }
  }]
}
```

Errors:

- 401 if the user is not logged in

POST /metadata/keys/privates.json ● PB-35986 ●

This endpoint allows an administrator to share/create the missing private key(s) for one or more users. For example after a user completed a setup in zero-knowledge mode or if after a server error in the user friendly mode.

Note: There's also a POST /metadata/keys/private.json endpoint that aliases this endpoint. Any of both endpoints can be used interchangeably.

```
POST /metadata/keys/privates.json
Authentication: Session or JWT

[{
    "metadata_key_id": "<uuid>",
    "user_id": <uuid>,
    "data": "-----BEGIN PGP MESSAGE-----"
 }, {
    "metadata_key_id": "<uuid>",
    "user_id": <uuid>,
    "data": "-----BEGIN PGP MESSAGE-----"
}]
```

Response (success):

```
{
  "header": {
    "id": "<uuid>",
    "status": "success",
    "servertime": 1723453376,
    "action": "<uuid>",
    "message": "The operation was successful.",
    "url": "/metadata/keys/privates.json",
    "code": 200
  },
  "body": {}
}
```

```
{
  "header": {
    "id": "<uuid>",
    "status": "error",
```

```
    "servertime": 1723453376,
    "action": "<uuid>",
    "message": "The metadata private key data is not valid.",
    "url": "/metadata/keys/privates.json",
    "code": 400
  },
  "body": {
    "2": {
      "user_id": {
        "uuid": "The user identifier should be a valid UUID."
      }
      "data": {
        "_empty": "The data should not be empty."
      }
    }
  }
}
```

Response (validation errors): *Note that if multiple dataset contains the errors then the body part will only show the errors of the first data set. This behaviour is consistent with other parts (i.e. rbacs update endpoint).*

Response (not admin error):

```
{
  "header": {
    "id": "<uuid>",
    "status": "error",
    "servertime": 1723453376,
    "action": "<uuid>",
    "message": "Access restricted to administrators.",
    "url": "/metadata/keys/privates.json",
    "code": 403
  },
  "body": ""
}
```

Errors:

- 401 if the user is not logged in
- 403 if the user is not an administrator
- 400 if the key already exists for the `metadata_key_id / user_id` combination
- 400 if the key is not encrypted for the corresponding user key
- 400 if the key has been deleted.

Note: we accept private keys, even for the expired metadata_key, if there is still some content for that key that has not been migrated.

POST /metadata/keys/private.json ● PB-42706 ●

Alias of POST /metadata/keys/privates.json endpoint. See
New E2EE Content Types Epic - Business Case - PUBLIC

PUT /metadata/keys/private/<uuid>.json ● PB-35275 ●

This endpoint allows for a user to re-upload the private key encrypted and signed with his own key. It can be used by the client to increase the trust of the key.

Note for clients: Prior to doing that the client should also validate the signature (if created_by is null, use server key, if uuid is set, then fetch the corresponding user key). Display a warning and require user approval if the key signature fails.

```
PUT /metadata/keys/private/<uuid>.json
Authentication: Session or JWT

{
  "data": "-----BEGIN PGP MESSAGE-----"
}
```

Response

Returned the updated private key with the current user id set to correct modified_by field.

```
{
  "header": {
      "id": "<uuid>",
      "status": "success",
      "servertime": 1723453376,
      "action": "<uuid>",
      "message": "The operation was successful.",
      "url": "/metadata/keys/private/<uuid>.json",
      "code": 200
  },
  "body": {
      "user_id": <uuid>,
      "data": "-----BEGIN PGP MESSAGE-----",
      "created_by": <uuid|null>,
      "modified_by": <user.id|null>
  }
}
```

Error response

New errors:

- 401 if the user is not logged in
- 404 the private key does not exist, or does not belong to the current user
- 400 the user already uploaded a valid modified version of the key
- 400 the private key is not a valid openpgp message encrypted for the user

PUT /metadata/keys/<uuid>.json (new) PB-35990

This endpoint is used to mark a metadata key as expired. The revoked public key is sent with the request. This will allow an administrator to trigger the refusal of the creation/edition of items that are using an expired key.

```
PUT /metadata/keys/<uuid>.json
Authentication: Session or JWT

{
  "fingerprint": "<fingerprint>",
  "armored_key": "-----BEGIN PGP PUBLIC KEY BLOCK-----"
  "expired": <datetime>
}
```

Response

```
{
  "header": {
      "id": "<uuid>",
      "status": "success",
      "servertime": 1723453376,
      "action": "<uuid>",
      "message": "The operation was successful.",
      "url": "/metadata/keys/<uuid>.json",
      "code": 200
  },
  "body": null
}
```

New errors:

- 401 if the user is not logged in
- 403 if the user is not an administrator

- 404 if the key does not exist or is already expired
- 400 if the key is the last active metadata key and v5 entities are set as default
- 400 if the sent armored key not a valid public OpenPGP key
- 400 if the sent fingerprint does not match the original OpenPGP key
- 400 if the sent fingerprint does not match the sent OpenPGP key
- 400 if the sent key is not set as expired (sig validation to be done in future)
- 400 if the expired date is not sent or is not a valid date
- 400 if the expired date does not match the one in the key
- 400 if the expired date is in the future

DELETE /metadata/keys/<uuid>.json (new) PB-35990

This endpoint is used to soft delete a metadata key. The revoked public key is sent with the request.

```
DELETE /metadata/keys/<uuid>.json
Authentication: Session or JWT
```

Response

```
{
  "header": {
      "id": "<uuid>",
      "status": "success",
      "servertime": 1723453376,
      "action": "<uuid>",
      "message": "The operation was successful.",
      "url": "/metadata/keys/<uuid>.json",
      "code": 200
  },
  "body": null
}
```

Error response

New errors:

- 401 if the user is not logged in
- 403 if the user is not an administrator
- 404 if the key does not exist or is already deleted
- 400 if the key is the last metadata key and v5 entities are set as default
- 400 the fingerprint does not match

Note for backend: private keys are not deleted as some content may still use the deleted key and need to be migrated.

API Specifications - Settings

GET /settings (updated) PB-34461

Logged in users can see if the metadata plugin is enabled or not. This prevents creating unnecessary 404 to see if certain new features can be used or not. Not visible if PASSBOLT_V5_ENABLED false.

```
GET /settings.json
Authentication: Session or JWT
```

Response

```
{
  "header": {
      "id": <uuid>,
      "status": "success",
      "servertime": 1723453376,
      "action": <uuid>,
      "message": "The operation was successful.",
      "url": "/settings.json",
      "code": 200
  },
  "body": {
    "app": {
       // etc.
    },
    "passbolt": {
      "plugins": {
         "metadata": {
            "version": "1.0.0",
            "enabled": true
          },
         // etc.
       }
    },
    //etc.
}
```

POST /metadata/types/settings.json (new) ● PB-34472 ●

Administrators can define which resource type is the default and if v5 folders/tags/comments are enabled.

```
POST /metadata/types/settings.json
Authentication: Session or JWT

{
    "default_resource_types": "<enum:v4|v5">, // plural, multiple types
    "default_folder_type": "<enum:v4|v5">,    // singular, one type
    "default_tag_type": "<enum:v4|v5">,
    "default_comment_type": "<enum:v4|v5">,
    "allow_creation_of_v5_resources": <bool>,
    "allow_creation_of_v5_folders": <bool>,
    "allow_creation_of_v5_tags": <bool>,
    "allow_creation_of_v5_comments": <bool>,
    "allow_creation_of_v4_resources": <bool>,
    "allow_creation_of_v4_folders": <bool>,
    "allow_creation_of_v4_tags": <bool>,
    "allow_creation_of_v4_comments": <bool>,
    "allow_v4_v5_upgrade": <bool>,
    "allow_v5_v4_downgrade": <bool>,
}
```

Response

```
{
  "header": {
      "id": <uuid>,
      "status": "success",
      "servertime": 1723453376,
      "action": <uuid>,
      "message": "The operation was successful.",
      "url": "/metadata/types/settings.json",
      "code": 200
  },
  "body": {
    "default_resource_types": "<enum:v4|v5">,
    "default_folder_type": "<enum:v4|v5">,
    "default_tag_type": "<enum:v4|v5">,
    "default_comment_type": "<enum:v4|v5">,
    "allow_creation_of_v5_resources": <bool>,
    "allow_creation_of_v5_folders": <bool>,
    "allow_creation_of_v5_tags": <bool>,
    "allow_creation_of_v5_comments": <bool>,
    "allow_creation_of_v4_resources": <bool>,
```

```
    "allow_creation_of_v4_folders": <bool>,
    "allow_creation_of_v4_tags": <bool>,
    "allow_creation_of_v4_comments": <bool>,
    "allow_v4_v5_upgrade": <bool>,
    "allow_v5_v4_downgrade": <bool>,
  }
}
```

Error response

New errors:

- 401 if the user is not logged in
- 403 the user is not an administrator
- 400 invalid values are sent in the body
- 400 admin select a default version but all resource types are deleted for this version
- 400 v5 selected but metadata key does not exist or is not available to all

Back-end Implementation notes

To prevent downgrade attack scenarios, it should be possible to disable the v4 from accepted value using server configuration (e.g. environment variable / config file), and/or lock the configuration like for other endpoints.

GET /metadata/types/settings.json (new) ● PB-35927 ●

Logged in users can now get information from the resource types settings, as selected by their administrators.

```
GET /metadata/types/settings.json
Authentication: Session or JWT
```

Response

```
{
  "header": {
      "id": <uuid>,
      "status": "success",
      "servertime": 1723453376,
      "action": <uuid>,
      "message": "The operation was successful.",
      "url": "/metadata/types/settings.json",
      "code": 200
```

```
  },
  "body": {
    "default_resource_types": "<enum:v4|v5">,
    "default_folder_type": "<enum:v4|v5">,
    "default_tag_type": "<enum:v4|v5">,
    "default_comment_type": "<enum:v4|v5">,
    "allow_creation_of_v5_resources": <bool>,
    "allow_creation_of_v5_folders": <bool>,
    "allow_creation_of_v5_tags": <bool>,
    "allow_creation_of_v5_comments": <bool>,
    "allow_creation_of_v4_resources": <bool>,
    "allow_creation_of_v4_folders": <bool>,
    "allow_creation_of_v4_tags": <bool>,
    "allow_creation_of_v4_comments": <bool>,
    "allow_v4_v5_upgrade": <bool>,
    "allow_v5_v4_downgrade": <bool>,
  }
}
```

Error response

New errors:

- 401 if the user is not logged in
- 404 if the software is running version 4

GET /metadata/keys/settings.json (new) ● PB-35367 ●

Logged in users can now get information from the key sharing settings, as selected by their administrators.

```
GET /metadata/keys/settings.json
Authentication: Session or JWT
```

Response

```
{
  "header": {
      "id": <uuid>,
      "status": "success",
      "servertime": 1723453376,
      "action": <uuid>,
      "message": "The operation was successful.",
      "url": "/metadata/keys/settings.json",
      "code": 200
```

```
  },
  "body": {
    "allow_usage_of_personal_keys": <bool>, //default: true
    "zero_knowledge_key_share": <bool>,     //default: false
  }
}
```

Error response

New errors:

- 401 if the user is not logged in
- 404 if the software is running version 4

Back-end Implementation notes

Like resource type settings, this endpoint always returns something, e.g. the default values if the setting is not found.

POST /metadata/keys/settings.json (new) ● PB-35367 ●

Logged in users can now get information from the key sharing settings, as selected by their administrators.

```
POST /metadata/keys/settings.json
Authentication: Session or JWT

{
    "allow_usage_of_personal_keys": <bool>,
    "zero_knowledge_key_share": <bool>,
    "metadata_private_keys": [{           // Optional, only if
!zero-knowledge and updating the settings
      "metadata_key_id": <uuid>,
      "user_id": null,
      "data": "-----BEGIN PGP MESSAGE-----"
    }]
}
```

Response

See `GET /metadata/keys/settings.json`.

Error response

New errors:

- 401 if the user is not logged in
- 400 if allow_usage_of_personal_keys or zero_knowledge_key_share is not sent or not booleans
- 400 if zero_knowledge_key_share is true and metadata_private_keys is sent
- 400 if zero_knowledge_key_share is false and metadata_private_keys is not sent and server didn't have yet access to it.
- 400 if metadata_private_keys are sent and user_id is not null
- 400 if metadata_private_keys are sent and data is not valid OpenPGP message encrypted for the server key and signed by the current user key.
- 400 if the incorrect amount of metadata_private_keys is sent, e.g. for example there are multiple metadata keys available and only one is sent.
- 404 if the software is running version 4

Back-end Implementation notes

Like resource type settings, this endpoint always returns something, e.g. the default values if the setting is not found.
If zero_knowledge_key_share is set to true and was previously set to false, the metadata private key entry for the server is deleted.

API Specification - Resource types changes

GET /resource-types.json (updated) ● PB-34609 ●

Get the list of supported resource types. The main changes are the addition of new entries, as well as a "is-deleted" filter to show entries that have been deleted (default false). For administrators It is also possible to get optionally the number of count of resources associated to this type of resource.

```
GET
/resources-types.json?filter[is-deleted]=1&contain[resources_count]=1
Authentication: Session or JWT
```

Response

```
{
  "header": {
      "id": <uuid>,
      "status": "success",
      "servertime": 1723453376,
      "action": <uuid>,
```

```
      "message": "The operation was successful.",
      "url": "/resource-types.json?filter[is-deleted]=1",
      "code": 200
  },
  "body": [
      {
        "id": <uuid>,
        "slug": "totp",
        "name": "Standalone TOTP",
        "description": "A resource with standalone TOTP fields.",
        "definition": <object>,
        "created": "2023-05-17T07:49:15+00:00",
        "modified": "2023-07-05T09:04:40+00:00",
        "deleted": "2023-07-05T09:04:40+00:00"   // datetime or null
        "default": false,
        "resources_count": 0
      },
    //etc.
  ]
}
```

DELETE /resource-types/<uuid>.json (new) ● PB-34610 ●

Soft delete a given resource type. This allows administrators to control which resource types are available, e.g. which one can be created or not.

```
DELETE /resources-types.json/1870775b-fb57-4bb1-ad49-b2ba36c7dcba.json
Authentication: Session or JWT
```

Error response

New errors:

- 401 if the user is not logged in
- 403 if the user is not an administrator
- 404 the resource type id does not exist
- 400 the resource type is already soft deleted
- 400 it is the last resource type available for this version and the resource type version is marked as the default one in the /metadata/settings
- 400 there are still resources with this type
- 400 it is the last resource type available

PUT /resource-types/<uuid>.json (new) ● PB-34609 ●

Can be used to un-delete a given resource type. It can be used to set the new default resource type.

```
PUT /resources-types/1870775b-fb57-4bb1-ad49-b2ba36c7dcba.json
Authentication: Session or JWT

{
  "deleted": null,
}
```

Response

```
{
  "header": {
      "id": <uuid>,
      "status": "success",
      "servertime": 1723453376,
      "action": <uuid>,
      "message": "The operation was successful.",
      "url":
"/resources-types/1870775b-fb57-4bb1-ad49-b2ba36c7dcba.json",
      "code": 200
  },
  "body": ""
}
```

Error response

Errors:

- 401 if the user is not logged in
- 403 if the user is not an administrator
- 404 resource type id does not exist
- 400 other fields than deleted is provided
- 400 deleted field is missing or set to something else than null

API Specification - Changes to user setup

POST /setup/complete/<uuid>.json - Complete user setup (Updated) ● ▾

PB-35119 ● ▾

Request / Response

No changes, if zero knowledge is set to false, the server tries to share the metadata keys with the end user. Server decrypts the shared key and re-encrypt for the given user and inserts it in the private key table.

If zero knowledge is on, the email notification received by the administrator prompt them to go the user workspace and share the key(s) manually.

API Specification - Changes to resources and related

POST /resources.json - Create new resource (Updated) ● ▾

Request

```
POST /resources.json
Authentication: Session or JWT

{
  "metadata": "-----BEGIN PGP MESSAGE-----">|null,
  "metadata_key_id": <uuid|null>,
  "metadata_key_type": "enum:user_key|shared_key|null",
  "expired: null,
  "folder_parent_id": null,
  "resource_type_id": <uuid>,
  "secrets": ["-----BEGIN PGP MESSAGE-----"],
}
```

Note:

- Accept both personal or shared key id on creation (client may know already there is a share coming), or it may be needed to have a policy to share everything (org setting).

Response

```
200
```

```
{
 "header": {
   "id": <uuid>,
   "status": "success",
   "servertime": <unix_timestamp>,
   "action": <uuid>,
   "message": "The operation was successful.",
   "url": "/resources.json",
   "code": 200
 },
 "body": {
   "id": <uuid>,
   "personal": true,
   "metadata": "-----BEGIN PGP MESSAGE-----",
   "metadata_key_id": <uuid>,
   "metadata_key_type": "user_key",
   "expired: null,
   "folder_parent_id": null
   "resource_type_id":<uuid>,
   "created_by": <uuid>,
   "modified_by": <uuid>,
   "created": <datetime>,
   "modified": <datetime>,
 }
}
```

Error response

New errors:

- 400 if trying to create a resource with a resource type that is deleted
- 400 if both v4 and v5 fields are sent
- 400 if one of (url, name, description, username) is sent with any of (metadata*)
- 400 if metadata is not an OpenPGP message when resource_type_id is v5_*
- 400 if metadata_key_id is not a uuid if resource_type_id is v5_*
- 400 if metadata_key_type is not user_key or shared_key if resource_type_id is v5_*
- 400 if resource_type_id belong to a resource type that is deleted
- 400 if trying to build a resource of v5-password-string type
- 400 if the metadata_key_id is for a deleted or expired key

PUT /resources/<uuid>.json - Update resource (Updated) ● ▾

Request

```
POST /resources/<uuid>.json
Authentication: Session or JWT

{
  "metadata": "-----BEGIN PGP MESSAGE-----",
  "metadata_key_id": <uuid>,
  "metadata_key_type": <enum>,
  "expired: null,
  "folder_parent_id": null,
  "resource_type_id": <uuid>,
  "secrets": ["-----BEGIN PGP MESSAGE-----"],
}
```

Error response

Two new errors:

- 400 if `metadata_key_id` is personal one and the resource has more than one permission and/or is shared with a group
    - Note: We don't care that the group has only one member, it counts as shared with multiple people, there is no such thing as a private group.
- 400 if resource is shared and the metadata cannot be decrypted with the shared session key
- 400 if the metadata_key_id is for a deleted or expired key

GET /resources/<uuid>.json - View resource (Updated) ● ▾

Viewing a resource can return either resource in v4 or v5 format.

Request

```
GET /resources/<uuid>.json
Authentication: Session or JWT
```

Response

```
200

{
 "header": {
```

```
    "id": <uuid>,
    "status": "success",
    "servertime": <unix_timestamp>,
    "action": <uuid>,
    "message": "The operation was successful.",
    "url": "/resources/<uuid>.json",
    "code": 200
  },
  "body": {
    "id": <uuid>,
    "metadata": "-----BEGIN PGP MESSAGE-----",
    "metadata_key_id": <string>,
    "metadata_key_type": <enum>,
    "personal": true,
    "expired: null,
    "folder_parent_id": null
    "resource_type_id":<uuid>,
    "created_by": <uuid>,
    "modified_by": <uuid>,
    "created": <datetime>,
    "modified": <datetime>,
  }
}
```

GET /resources.json - View all resources (Updated)

If a resource is a v5 one, render the v5 metadata properties. Otherwise render the v4 properties. Additionally it is possible to filter on metadata_key_type.

Request

```
GET /resources.json?filter[metadata_key_type]=<enum:user_key,...>
Authentication: Session or JWT
```

Response

```
200

{
  "header": {
    "id": <uuid>,
    "status": "success",
    "servertime": <unix_timestamp>,
```

```
   "action": <uuid>,
   "message": "The operation was successful.",
   "url": "/resources.json",
   "code": 200
 },
 "body": {[
{
   // if v5
   "id": <uuid>,
   "metadata": "-----BEGIN PGP MESSAGE-----",
   "metadata_key_id": <string>,
   "metadata_key_type": <enum>,
   "personal": true,
   "expired: null,
   "folder_parent_id": null
   "resource_type_id":<uuid>,
   "created_by": <uuid>,
   "modified_by": <uuid>,
   "created": <datetime>,
   "modified": <datetime>,
 },
   //if v4
   "id": <uuid>,
   "name": <string>,
   "username": <string|null>,
   "uri": <string>,
   "description": <string>,
   "personal": true,
   "expired: null,
   "folder_parent_id": null
   "resource_type_id":<uuid>,
   "created_by": <uuid>,
   "modified_by": <uuid>,
   "created": <datetime>,
   "modified": <datetime>,
 }
], ...}
```

Validation:

- 400 if metadata_key_type filter is not valid

Note: If the v5 flag is disabled we ignore v5 resources and the filter.
The filtering of v5 resources if the flag is down is done based on the resource_type_id.

POST /share/simulate/resources/<uuid>.json (Updated) & POST /share/resources/<uuid>.json (Updated)

Only new errors are added, the logic stays the same.

Error responses

New errors:

- 400 trying to share a resource that has user_key as metadata_key_type.

GET /users.json - View Users (Updated) PB-35925

Here we added a contain `missing_metadata_key_ids` option (default false) available to administrators only. This allows them to see the list of metadata_keys ids where a metadata_private_key is missing for a given user (including the expired), allowing them to then share the private key with them later on. If none is missing an empty array is returned.

Request

```
GET /users.json?contain[missing_metadata_key_ids]=<bool>
Authentication: Session or JWT
```

Response

```
200

{
 "header": {
   "id": <uuid>,
   "status": "success",
   "servertime": <unix_timestamp>,
   "action": <uuid>,
   "message": "The operation was successful.",
   "url": "/use.json",
   "code": 200
 },
 "body": [{
     "id": <uuid>,
     "username": <uuid>,
     "missing_metadata_key_ids": [<uuid>, <uuid>], // new
```

```
      ...
  },...]
}
```

New errors:

- 403 missing_metadata_key_ids set to true and user is not admin
- 400 missing_metadata_key_ids is not boolean

GET /users/<me|uuid>.json - View Users (Updated) ● PB-37068 ●

Same as the previous point, we add a new contain option called `missing_metadata_key_ids` (default false). This allows users to know if they are missing private keys.

Request

```
GET /users/me.json?contain[missing_metadata_key_ids]=<bool>
Authentication: Session or JWT
```

Response

```
200

{
 "header": {
   "id": <uuid>,
   "status": "success",
   "servertime": <unix_timestamp>,
   "action": <uuid>,
   "message": "The operation was successful.",
   "url": "/resources.json",
   "code": 200
 },
 "body": {
    "id": <uuid>,
    "username": <uuid>,
    "missing_metadata_key_ids": [<uuid>, <uuid>], // new
     ...
  }
}
```

DELETE /users/<uuid>.json - Delete user (Updated) ● ▾

Changes:

- Resources that are returned as part of an error when trying to delete a user that is sole owner of resources may be returned in v4 or v5 format.
    - Similar to GET /resources.json endpoint, there should be a reformatting
- Deleting the user should also delete:
    - Metadata private key entries for that user
    - Metadata ses sion keys entries for that user

DELETE /users/<uuid>/dry-run.json - Dry-run delete user (Updated) ● ▾

Same changes as DELETE /users/<uuid>.json endpoint (but without delete)

DELETE /groups/<uuid>.json - Delete group (Updated) ● ▾

Changes:

- Resources that are returned as part of an error when trying to delete a user that is sole owner of resources may be returned in v4 or v5 format.
    - Similar to GET /resources.json endpoint, there should be a reformatting

DELETE /groups/<uuid>/dry-run.json - Dry-run delete group (Updated) ● ▾

Same changes as DELETE /groups/<uuid>.json endpoint

API Specifications - Session key cache

GET /metadata/session-keys.json (New) ● ▾ PB-35150 ● ▾

This endpoint is used to get the list of the available encrypted session keys cache for a given user.

Request

```
GET /metadata/session-keys.json
Authentication: Session or JWT
```

Response

```
200

{
 "header": {
   "id": <uuid>,
   "status": "success",
   "servertime": <unix_timestamp>,
   "action": <uuid>,
   "message": "The operation was successful.",
   "url": "/resources/<uuid>.json",
   "code": 200
 },
 "body": [{
    "id": "<uuid>",
    "user_id": "<uuid>",
    "data": "-----BEGIN PGP MESSAGE-----",
    "created": "<datetime>",
    "modified": "<datetime>"
  }]
}
```

Error codes

- 401 if the user is not logged in

POST /metadata/session-keys.json (New) ● PB-35151 ●

This endpoint is used to save encrypted session keys cache for a given user.

Request

```
POST /metadata/session-keys.json
Authentication: Session or JWT

{
   "data": "-----BEGIN PGP MESSAGE-----"
}
```

Decrypted the OpenPGP message will look like this:

```
{
  "object_type": "PASSBOLT_SESSION_KEYS", // sanity check
  "session_keys": [{
     "foreign_model": "<enum: Resource, Secret, Folder, etc.>",
     "foreign_id": "<uuid>",
     "session_key": "<string:66>", //ex. 9:1AEA6EE66..051BD2D3357F38
     "modified": <datetime>
  }]
}
```

Error codes

- 401 if the user is not logged in
- 400 data is not an asymmetrically encrypted OpenPGP message
- 400 data is not encrypted for the current user key
- 500 entry could not be saved because of internal error

PUT /metadata/session-keys/<uuid>.json (New) ● PB-35921 ●

Update a given session-key entry.

Request

```
POST /metadata/session-keys/<uuid>.json
Authentication: Session or JWT

{
      "data": "-----BEGIN PGP MESSAGE-----",
      "modified": "<datetime>"
}
```

Response

```
200

{
 "header": {
   "id": <uuid>,
   "status": "success",
   "servertime": <unix_timestamp>,
   "action": <uuid>,
   "message": "The operation was successful.",
```

```
    "url": "/metadata/session-keys/<uuid>.json",
    "code": 200
  },
  "body": {
     "id": "<uuid>",
     "user_id": "<uuid>",
     "data": "-----BEGIN PGP MESSAGE-----",
     "created": "<datetime>",
     "modified": "<datetime>"
   }
}
```

Error codes

- 409 if the modified date is not equal to the persisted session key one
- 401 if the user is not logged in
- 404 session key entry does not exist
- 404 session key entry exist but is not for the current user user_id
- 400 data is not a valid OpenPGP message or not for current user
- 500 entry could not be deleted because of some internal error

DELETE /metadata/session-keys/<uuid>.json (New) ● PB-35152 ●

Delete a given session-key entry.

Request

```
DELETE /metadata/session-keys/<uuid>.json
Authentication: Session or JWT

{}
```

Error codes

- 401 if the user is not logged in
- 404 session key entry does not exist
- 404 session key entry exist but is not for the current user user_id
- 500 entry could not be deleted because of some internal error

API Specifications - Folders

POST /folders.json (Updated) ● PB-35361 ●

It is now possible to create folders with encrypted metadata as follow:

Request

```
POST /folders.json
Authentication: Session or JWT

{
   "metadata": "-----BEGIN PGP MESSAGE-----",
   "metadata_key_id": <string>,
   "metadata_key_type": <enum>,
}
```

Response

```
200

{
 "header": {
   "id": <uuid>,
   "status": "success",
   "servertime": <unix_timestamp>,
   "action": <uuid>,
   "message": "The operation was successful.",
   "url": "/folders.json",
   "code": 200
 },
 "body": {
    "id": "<uuid>",
    "metadata": "-----BEGIN PGP MESSAGE-----",
    "metadata_key_id": <string>,
    "metadata_key_type": <enum>,
    "created": "<datetime>",
    "modified": "<datetime>",
    "created_by": "<uuid>",
    "modified_by": "<uuid>",
    "folder_parent_id": null,
    "personal": true
  }
```

```
}
```

Error responses

New errors:

- 400 name and metadata fields are both sent in data request
- 400 metadata fields are set even though encrypted metadata for folders is disabled.
- 400 same logical validation rules than resource metadata (if type is user it must be encrypted with user, key)
- 400 if the metadata_key_id is for a deleted or expired key

POST /share/folders/<uuid>.json (Updated) • PB-35365 •

Only new errors are added, the logic stays the same.

Error responses

New errors:

- 400 trying to share an item that has user_key as metadata_key_type.

PUT /folders/<uuid>.json (Updated) • PB-35362 •

It is now possible to update folders with encrypted metadata as follow:

Request

```
PUT /folders/<uuid>.json
Authentication: Session or JWT

{
   "metadata": "-----BEGIN PGP MESSAGE-----",
   "metadata_key_id": <string>,
   "metadata_key_type": <enum>,
}
```

Response

```
200

{
 "header": {
```

```
   "id": <uuid>,
   "status": "success",
   "servertime": <unix_timestamp>,
   "action": <uuid>,
   "message": "The operation was successful.",
   "url": "/folders/<uuid>.json",
   "code": 200
 },
 "body": {
    "id": "<uuid>",
    "metadata": "-----BEGIN PGP MESSAGE-----",
    "metadata_key_id": <string>,
    "metadata_key_type": <enum>,
    "created": "<datetime>",
    "modified": "<datetime>",
    "created_by": "<uuid>",
    "modified_by": "<uuid>",
    "folder_parent_id": null,
    "personal": true
  }
}
```

Error responses

New errors:

- 400 is both v5 and v4 data (e.g. folder name) is sent as part of the request
- 400 metadata fields are set even though encrypted metadata for folders is disabled.
- 400 same logical validation rules than resource metadata (if type is user it must be encrypted with user, key)
- 400 if the metadata_key_id is for a deleted or expired key

GET /folders.json (Updated) PB-35363

Folders can be received as v4 or v5 format. Like resources only the corresponding fields are sent.

Request

```
GET /folders.json
Authentication: Session or JWT
```

Response


```
{
    "header": {
        "id": "b26ba81b-937f-4dfa-a14b-954fcd80b5d3",
        "status": "success",
        "servertime": 1619762222,
        "action": "1cd53591-cb6b-5b03-b0be-05a54644263d",
        "message": "The operation was successful.",
        "url": "/folders.json",
        "code": 200
    },
    "body": [
        {
          "id": "<uuid>",
          "name": "Test",
          "created": "2021-04-30T05:56:06+00:00",
          "modified": "2021-04-30T05:56:06+00:00",
          "created_by": "f848277c-5398-58f8-a82a-72397af2d450",
          "modified_by": "f848277c-5398-58f8-a82a-72397af2d450",
          "permissions": [...],
          "folder_parent_id": null,
          "personal": true
        },
        {
          "id": "<uuid>",
          "metadata": "-----BEGIN PGP MESSAGE-----",
          "metadata_key_id": <string>,
          "metadata_key_type": <enum>,
          "created": "2021-04-30T05:56:06+00:00",
          "modified": "2021-04-30T05:56:06+00:00",
          "created_by": "f848277c-5398-58f8-a82a-72397af2d450",
          "modified_by": "f848277c-5398-58f8-a82a-72397af2d450",
          "permissions": [...],
          "folder_parent_id": null,
          "personal": true
         }
     ]
}
```


GET /folders/<uuid>.json (Updated) ● PB-35364 ●

Same as folders.json but with one folder.

API Specifications - Tags

GET /tags.json (Updated) • PB-35415 •

Tags can be received as v4 or v5 format. Like resources & folders only the corresponding version's fields are sent.

Request

```
GET /tags.json
Authentication: Session or JWT
```

Response

```
{
    "header": {
        "id": "b26ba81b-937f-4dfa-a14b-954fcd80b5d3",
        "status": "success",
        "servertime": 1619762222,
        "action": "1cd53591-cb6b-5b03-b0be-05a54644263d",
        "message": "The operation was successful.",
        "url": "/tags.json",
        "code": 200
    },
    "body": [
        {
          "id": "<uuid>",
          "slug": "#test",
          "is_shared": true
        },
        {
          "id": "<uuid>",
          "metadata": "-----BEGIN PGP MESSAGE-----",
          "metadata_key_id": <string>,
          "metadata_key_type": <enum>,
          "is_shared": false
        }
    ]
}
```

POST /tags/<resource-uuid>.json (Updated) ● PB-36518 ●

This endpoint already existed in v4. Here the entire list of assigned tags to a given resource are sent to request every time. If any existing tag is missing from the request then it gets deleted from resources_tags entry.

With v5 it is now possible to create tags with encrypted metadata.

If the `id` field is passed the backend will reuse an existing tag (if it exists it is shared, if it is accessible for the user if it's a personal one), otherwise an error will be thrown for that tag.

It is always possible to reuse shared tags. However it is only possible to to reuse a personal tag if the tag was created by the current user, and the metadata key type is personal.

Request

```
POST /tags/<resource-uuid>.json
Authentication: Session or JWT

{
   "tags": [
      {
         "metadata": "-----BEGIN PGP MESSAGE-----",
         "metadata_key_id": <string|null>,
         "metadata_key_type": <enum>,
         "is_shared": <bool>
      },
      {
         "metadata": "-----BEGIN PGP MESSAGE-----",
         "metadata_key_id": null,
         "metadata_key_type": <user_key>,
         "is_shared": <bool>
      },
      "#tag-2",
      {
         "id": "<uuid>", // reuse existing shared/personal tag
      }
   ]
}
```

Response

```
200

{
 "header": {
   "id": <uuid>,
   "status": "success",
   "servertime": <unix_timestamp>,
   "action": <uuid>,
   "message": "The operation was successful.",
   "url": "/tags/b26ba81b-937f-4dfa-a14b-954fcd80b5d3.json",
   "code": 200
 },
 "body": {
    [
        {
           "id": "<uuid>",
           "metadata": "-----BEGIN PGP MESSAGE-----",
           "metadata_key_id": <string>,
           "metadata_key_type": <enum>,
           "is_shared": false
        },
        {
           "id": "<uuid>",
           "slug": "#tag-2",
           "is_shared": true
        }
    ]
  }
}
```

Error responses

New errors:

- 400 slug and metadata fields are both sent in data request
- 400 metadata fields are set even though encrypted metadata for tags is disabled.
- 400 if Id is specified and the tag does not exist
- 400 if Id is specified and the tag is personal and the tag was not created by the current user, e.g. the user doesn't have right to view that tag
- 400 if the tag is shared and it is not encrypted using shared key
- 400 if tag is personal and it is not encrypted using user key
- 400 if the metadata_key_id is for a deleted or expired key

PUT /tags/<uuid>.json (Updated) ● PB-36518 ●

It is now possible to update a tag. It can be used in several scenarios:

- Upgrade from v4 to v5
- Update the content of a shared or personal tag (e.g. color or slug for example)

However it cannot be used for:

- Downgrading from v5 to v4 (unless a special security flag is set)
- Moving from shared to personal (either in v4 or v5).

Request

```
PUT /tags/<uuid>.json
Authentication: Session or JWT

{
   "metadata": "-----BEGIN PGP MESSAGE-----",
   "metadata_key_id": <string|null>,
   "metadata_key_type": <enum>,
   "is_shared": <bool>
}
```

Response

```
200

{
 "header": {
   "id": <uuid>,
   "status": "success",
   "servertime": <unix_timestamp>,
   "action": <uuid>,
   "message": "The operation was successful.",
   "url": "/tags/b26ba81b-937f-4dfa-a14b-954fcd80b5d3.json",
   "code": 200
 },
 "body": {
    "id": "<uuid>",
    "metadata": "-----BEGIN PGP MESSAGE-----",
    "metadata_key_id": <string>,
    "metadata_key_type": <enum>,
    "is_shared": false
  }
```

```
}
```

Error responses

New errors:

- 400 slug and metadata fields are both sent in data request
- 400 metadata fields are set even though encrypted metadata for tags is disabled.
- 400 if tag is v5 and user is trying to downgrade to v4
- 400 if metadata_key_type is shared and key id is a user id
- 400 if metadata_key_type is user_id and key id is metadata key type
- 400 if the tag type is updated, e.g. it's changing from shared to non shared or vice versa. (a new tag should be created then)
- 400 if `is_shared` is sent in request and metadata is already of type v5
- 400 if `metadata_key_type` is sent in request and metadata is already of type v5, e.g. it's not possible to change this once set (see previous rule)
- 400 if the metadata_key_id is for a deleted or expired key

API Specification - Metadata upgrade (migration v4 to v5)

GET /metadata/upgrade/resources.json (New) ● PB-39394 ●

With this endpoint administrator can see the resources that can / needs to be upgraded.

Request

For convenience the data is paginated. For security reasons the administrator cannot control which page to see, they only see the first page. If they define the page, it is ignored and replaced to page 1. By default the page size is hardcoded to 20 (and cannot be controlled).

The endpoint contains a filter "is-shared" that returns shared items if true and non-shared items if false. If no filter is provided both records are provided. Results are ordered by creating date ascending, migrating older items first.

The endpoint also supports a contain permissions option, to allow the administrator to know, for personal secrets, which user the metadata should be encrypted for.

```
GET /metadata/upgrade/resources.json
  ?filter[is-shared]=1&contain[permissions]=1
Authentication: Session or JWT
```

Response

Returns the resources that are in v4 format and needs to be migrated to v5.

```
200

{
 "header": {
   "id": <uuid>,
   "status": "success",
   "servertime": <unix_timestamp>,
   "action": <uuid>,
   "message": "The operation was successful.",
   "url": "/metadata/upgrade/resources.json",
   "code": 200,
   "pagination": {...} // admin cannot control, but can see values
 },
 "body": {...} //same than resources.json
}
```

Errors

- 401 if user is not logged in
- 400 is-shared filter is invalid
- 403 if the user is not an administrator

POST /metadata/upgrade/resources.json (New) PB-39394

Request

With this endpoint an administrator can upgrade the metadata of a given set of resources, made available to them via the endpoint in the previous section. The administrator must send the modified and modified_by value received by the previous endpoint, this will be used to ensure that they are editing the last known version and not reverting back to some other version, in case some things are happening in parallel. It doesn't solve all race conditions, but safe as a good enough failsafe.

```
POST /metadata/upgrade/resources.json
  ?filter[is-shared]=1&contain[permissions]=1
Authentication: Session or JWT
```

```
[{
  id: <uuid>,
  metadata_key_id: <uuid>,
  metadata_key_type: <enum>, //personal items allowed
  metadata: <string>,
  modified: <datetime>, // previously modified date, not now
  modified_by: <uuid>,  // previous value, not updated to current user
},
{...}
]
```

Response

In case of success this endpoint returns the next batch of resources that needs to be migrated to v5. See GET /metadata/upgrade/resources.json.

```
200

{
 "header": {
   "id": <uuid>,
   "status": "success",
   "servertime": <unix_timestamp>,
   "action": <uuid>,
   "message": "The operation was successful.",
   "url": "/metadata/upgrade/resources.json",
   "code": 200,
   "pagination": {...} // admin cannot control, but can see
 },
 "body": {...} //GET /metadata/migrate/resources.json with filters
}
```

Errors

- 401 user is not logged in
- 403 user is not administrator
- 404 entity is not found or deleted
- 409 entity was updated (modified or modified_by changed)
- 400 is-shared filter is invalid
- 400 metadata OpenPGP is not valid, or not encrypted for the right key (user/shared)
- 400 metadata_key_id is expired or deleted
- 400 modified_by is not UUID
- 400 modified is not a valid date

- 400 id is not a valid uuid
- 400 more than 20 items are send
- 400 more or less entity data than the expected fields is sent
- 400 upgrade from v4 to v5 is disabled in settings
- 400 item is already v5

If more data is sent, like expiry, or even associated data like secrets, the data is ignored.

GET /metadata/upgrade/folders.json (New) PB-39395

Similar to the same endpoint for resources.

Request

```
GET /metadata/upgrade/folders.json
  ?filter[is-shared]=1&contain[permissions]=1
Authentication: Session or JWT
```

Response

```
200

{
 "header": {
   "id": <uuid>,
   "status": "success",
   "servertime": <unix_timestamp>,
   "action": <uuid>,
   "message": "The operation was successful.",
   "url": "/metadata/upgrade/folders.json",
   "code": 200,
   "pagination": {...} // admin cannot control, but can see values
 },
 "body": {...} //same than folders.json
}
```

Errors

- 401 if user is not logged in
- 400 is-shared filter is invalid
- 403 if the user is not an administrator

POST /metadata/upgrade/folders.json (New) ● PB-39395 ●

Request

```
POST /metadata/upgrade/folders.json?contain[permissions]=1
Authentication: Session or JWT

[{
  id: <uuid>,
  metadata_key_id: <uuid>,
  metadata_key_type: "shared_key",
  metadata: <string>,
  modified: <datetime>, // previously modified date, not now
  modified_by: <uuid>,  // previous value, not updated to current user
},
{...}
]
```

Response

In case of success this endpoint returns the next batch of resources that needs to be migrated to v5. See GET /metadata/upgrade/folders.json.

```
200

{
 "header": {
   "id": <uuid>,
   "status": "success",
   "servertime": <unix_timestamp>,
   "action": <uuid>,
   "message": "The operation was successful.",
   "url": "/metadata/upgrade/folders.json",
   "code": 200,
   "pagination": {...} // admin cannot control, but can see
 },
 "body": {...} //GET /metadata/migrate/folders.json with filters
}
```

Errors

- 401 user is not logged in
- 403 user is not administrator

- 404 entity is not found or deleted
- 409 entity was updated (modified or modified_by changed)
- 400 is-shared filter is invalid
- 400 metadata OpenPGP is not valid, or not encrypted for the right key (user/shared)
- 400 metadata_key_id is expired or deleted
- 400 modified_by is not UUID
- 400 modified is not a valid date
- 400 id is not a valid uuid
- 400 more than 20 items are send
- 400 more or less entity data than the expected fields is sent
- 400 upgrade from v4 to v5 is disabled in settings
- 400 item is already v5

GET /metadata/upgrade/tags.json (New) ● PB-37700 ●

Request

```
GET /metadata/migrate/tags.json?filter[is-shared]=1
Authentication: Session or JWT
```

Response

```
200
```

```
{
 "header": {
   "id": <uuid>,
   "status": "success",
   "servertime": <unix_timestamp>,
   "action": <uuid>,
   "message": "The operation was successful.",
   "url": "/metadata/upgrade/tags.json",
   "code": 200,
   "pagination": {...} // admin cannot control, but can see values
 },
 "body":  [{
    "id": "<uuid>",
    "user_id": <uuid|null>, // null if shared = true
    "slug": "#test",
    "is_shared": true
  }]
}
```

Errors

- 401 if user is not logged in
- 400 is-shared filter is invalid
- 403 if the user is not an administrator

POST /metadata/upgrade/tags.json (New) PB-37702

Similar to resources and folders, the only difference here is that there is no need to send “modified” and “modified_by” as slug edition is not allowed.

Request

```
POST /metadata/upgrade/tags.json?filter[is-shared]=1
Authentication: Session or JWT

[{
  id: <uuid>,
  metadata_key_id: <uuid>,
  metadata_key_type: "shared_key",
  metadata: <string>,
},
{...}
]
```

Response

In case of success this endpoint returns the next batch of resources that needs to be migrated to v5. See GET /metadata/upgrade/tags.json.

```
200

{
 "header": {
   "id": <uuid>,
   "status": "success",
   "servertime": <unix_timestamp>,
   "action": <uuid>,
   "message": "The operation was successful.",
   "url": "/metadata/upgrade/tags.json",
   "code": 200,
   "pagination": {...} // admin cannot control, but can see
 },
 "body": {...} //GET /metadata/upgrade/tags.json with filters
}
```

Errors

- 401 user is not logged in
- 403 user is not administrator
- 404 entity is not found or deleted
- 400 is-shared filter value is invalid
- 400 metadata OpenPGP is not valid, or not encrypted for the right key
- 400 metadata_key_id is expired or deleted
- 400 modified_by is not UUID
- 400 modified is not a valid date
- 400 id is not a valid uuid
- 400 more than 20 items are send
- 400 more or less entity data than the expected fields is sent
- 400 upgrade from v4 to v5 is disabled in settings
- 400 item is already v5

API Specification - Metadata Key Rotation

GET /metadata/rotate-key/resources.json (New) ● PB-37360 ●

With this endpoint administrator can see the resources that are using an expired key that needs to/can be rotated.

Please note that unlike upgrades there is no filter on whether the resource is shared or not, we only focus on metadata_key_type of type shared_key, i.e. rotation of the key for content encrypted with a user key will be done on another endpoint.

Request

For convenience the data is paginated. For security reasons the administrator cannot control which page to see, they only see the first page. If they define the page, it is ignored and replaced to page 1. By default the page size is hardcoded to 20 (and cannot be controlled via the UI).

```
GET /metadata/rotate-key/resources.json
Authentication: Session or JWT
```

Response

Returns the resources that are using v5 keys that are expired and need to be migrated to a new key (if it exists).

```
200

{
 "header": {
   "id": <uuid>,
   "status": "success",
   "servertime": <unix_timestamp>,
   "action": <uuid>,
   "message": "The operation was successful.",
   "url": "/metadata/rotate-key/resources.json",
   "code": 200,
   "pagination": {...} // admin cannot control, but can see values
 },
 "body": {...} //same than resources.json
}
```

Errors

- 401 if user is not logged in
- 400 there is no active key to use (all keys are expired or deleted)
- 403 if the user is not an administrator

POST /metadata/rotate-key/resources.json (New) ● PB-37361 ●

Request

With this endpoint an administrator can rotate the metadata of a given set of resources, made available to them via the endpoint in the previous section. The administrator must send the modified and modified_by value received by the previous endpoint, this will be used to ensure that they are editing the last known version and not reverting back to some other version, in case some things are happening in parallel.

```
POST /metadata/rotate-key/resources.json
Authentication: Session or JWT

[{
  id: <uuid>,
  metadata_key_id: <uuid>,
  metadata_key_type: "shared_key", // user key not allowed
  metadata: <string>,
  modified: <datetime>, // previously modified date, not now
  modified_by: <uuid>,  // previous value, not updated to current user
},
{...}
]
```

Response

In case of success this endpoint returns the next batch of resources that needs to be migrated to the new key. See GET /metadata/rotate-key/resources.json.

```
200

{
 "header": {
   "id": <uuid>,
   "status": "success",
```

```
   "servertime": <unix_timestamp>,
   "action": <uuid>,
   "message": "The operation was successful.",
   "url": "/metadata/rotate-key/resources.json",
   "code": 200,
   "pagination": {...} // admin cannot control, but can see
 },
 "body": {...} //GET /metadata/rotate-key/resources.json with filters
}
```

Errors

- 401 user is not logged in
- 403 user is not administrator
- 400 more than maximum supported items are send
- 400 no item was sent
- 400 metadata OpenPGP is not valid, or not encrypted for the right key
- 400 if there is more than one active key (should be in the error of that entity)
- 400 metadata_key_id is expired or deleted
- 400 metadata_key_type is not shared_key
- 400 modified_by is not UUID
- 400 modified is not a valid date
- 400 resource id is not a valid uuid
- 400 less entity data than the expected fields is sent
- 404 one or more entity is not found or deleted
- 409 one or more entity was updated (modified or modified_by changed)

If more data is sent, like expiry, or even associated data like secrets, the data is ignored.

Note:
Server will break on the first entity that fails validation. It will return in the body to the first entity that breaks. It is an all or nothing update. The client is supposed to restart the process if it fails (get the data, reprepare them, resend).

Example error response:

```
{
  "header": {
    "id": "<uuid>",
    "status": "error",
```

```
    "servertime": 1723453376,
    "action": "<uuid>",
    "message": "The resource is not valid",
    "url": "/metadata/rotate-key/resources.json",
    "code": 400
  },
  "body": {
    "2": {
      "id": {
        "uuid": "The identifier should be a valid UUID."
      }
      "data": {
        "_empty": "The data should not be empty."
      }
    }
  }
}
```

Response (validation errors):*Note that if multiple dataset contains the errors then the body part will only show the errors of the first data set. This behaviour is consistent with other parts (i.e. rbacs update endpoint).*

GET /metadata/rotate-key/folders.json (New) • PB-37362 •

Similar to the same endpoint for resources.

Request

```
GET /metadata/rotate-key/folders.json
Authentication: Session or JWT
```

Response

```
200

{
 "header": {
   "id": <uuid>,
   "status": "success",
   "servertime": <unix_timestamp>,
   "action": <uuid>,
   "message": "The operation was successful.",
   "url": "/metadata/rotate-key/folders.json",
```

```
   "code": 200,
   "pagination": {...} // admin cannot control, but can see values
 },
 "body": {...} //same than folders.json
}
```

Errors

- 401 if user is not logged in
- 400 there is no active key to use (all keys are expired or deleted)
- 403 if the user is not an administrator

POST /metadata/rotate-key/folders.json (New) • PB-37363 •

Request

```
POST /metadata/rotate-key/folders.json
Authentication: Session or JWT

[{
  id: <uuid>,;
  metadata_key_id: <uuid>,
  metadata_key_type: "shared_key",
  metadata: <string>,
  modified: <datetime>, // previously modified date, not now
  modified_by: <uuid>,  // previous value, not updated to current user
},
{...}
]
```

Response

In case of success this endpoint returns the next batch of folders that needs to be migrated. See GET /metadata/rotate-key/folders.json.

```
200

{
 "header": {
   "id": <uuid>,
   "status": "success",
   "servertime": <unix_timestamp>,
```

```
   "action": <uuid>,
   "message": "The operation was successful.",
   "url": "/metadata/rotate-key/folders.json",
   "code": 200,
   "pagination": {...} // admin cannot control, but can see
 },
 "body": {...} //GET /metadata/rotate-key/folders.json with filters
}
```

Errors

- 401 user is not logged in
- 403 user is not administrator
- 404 entity is not found or deleted
- 409 entity was updated (modified or modified_by changed)
- 400 there is no active key to use (all keys are expired or deleted)
- 400 metadata OpenPGP is not valid, or not encrypted for the right key
- 400 if there is more than one active key (should be in the error of that entity)
- 400 metadata_key_id is expired or deleted
- 400 metadata_key_type is not shared_key
- 400 modified_by is not UUID
- 400 modified is not a valid date
- 400 id is not a valid uuid
- 400 more than 20 items are send
- 400 more or less entity data than the expected fields is sent

GET /metadata/rotate-key/tags.json (New) PB-37364

Request

```
GET /metadata/rotate-key/tags.json
Authentication: Session or JWT
```

Response

```
200
```

```
{
 "header": {
   "id": <uuid>,
   "status": "success",
   "servertime": <unix_timestamp>,
   "action": <uuid>,
   "message": "The operation was successful.",
   "url": "/metadata/upgrade/tags.json",
   "code": 200,
   "pagination": {...} // admin cannot control, but can see values
 },
 "body":  [{
    "id": "<uuid>",
    "slug": "#test",
    "is_shared": true
  }]
}
```

Errors

- 401 if user is not logged in
- 400 there is no active key to use (all keys are expired or deleted)
- 403 if the user is not an administrator

POST /metadata/rotate-key/tags.json (New) ● PB-37365 ●

Similar to resources and folders, the only difference here is that there is no need to send “modified” and “modified_by” as slug edition is not allowed.

Request

```
POST /metadata/rotate-key/tags.json
Authentication: Session or JWT

[{
  id: <uuid>,
  metadata_key_id: <uuid>,
  metadata_key_type: "shared_key",
  metadata: <string>,
},
{...}
```

```
]
```

Response

In case of success this endpoint returns the next batch of tags that needs to be migrated. See GET /metadata/rotate-key/tags.json.

```
200

{
 "header": {
   "id": <uuid>,
   "status": "success",
   "servertime": <unix_timestamp>,
   "action": <uuid>,
   "message": "The operation was successful.",
   "url": "/metadata/rotate-key/tags.json",
   "code": 200,
   "pagination": {...} // admin cannot control, but can see
 },
 "body": {...} //GET /metadata/rotate-key/tags.json with filters
}
```

Errors

- 401 user is not logged in
- 403 user is not administrator
- 404 entity is not found or deleted
- 400 there is no active key to use
- 400 there is no active key to use (all keys are expired or deleted)
- 400 if there is more than one active key (should be in the error of that entity)
- 400 metadata OpenPGP is not valid, or not encrypted for the right key
- 400 metadata_key_id is expired or deleted
- 400 metadata_key_type is not shared_key
- 400 id is not a valid uuid
- 400 more than 20 items are send
- 400 more or less entity data than the expected fields is sent

Shell Command Specifications

$./bin/cake passbolt create_metadata_key

This command allows you to create the shared session_key.

What it does:

- Assert that there not already an entry in session_keys table with user set to null and personal set to false

$./bin/cake passbolt update_metadata_types_settings

This command is used to update metadata settings via command-line. Administrators can set default types (v4/v5), allow creation, disallow creation, etc.

Usage:

```
bin/cake passbolt update_metadata_types_settings \
  --default_resource_types=v5 \
  --allow_creation_of_v5_resources=1 \
  --allow_creation_of_v4_resources=0 \
  ...
```

Available options:

- default_resource_types
- default_folder_type
- default_tag_type
- default_comment_type
- allow_creation_of_v5_resources
- allow_creation_of_v5_folders
- allow_creation_of_v5_tags
- allow_creation_of_v5_comments
- allow_creation_of_v4_resources
- allow_creation_of_v4_folders
- allow_creation_of_v4_tags
- allow_creation_of_v4_comments
- allow_v4_v5_upgrade
- allow_v5_v4_downgrade

$./bin/cake passbolt share_metadata_key PB-37069

As a system administrator I can run a command to share metadata keys with users that are missing them, if the server have access to the private key(s).

Success:

- Output: Key <uuid> was shared with user <username>
    - Or: Nothing to share. All users have access to the required keys.
    - Or: Nothing to share. No key set. Create a key first.

DoD:

- Does not share with deleted users
- Does not share with non active users (no user key)
- Does share with suspended users
- Does share expired keys
- Does not share deleted keys
- Try everything possible, do not exit on the first error.

Errors:

- Code 0: nothing to share
- Code 0: everything that could be shared was shared
- Code 1: server doesn't have access to one or more keys
- Code 1: server could not share a key for one or more users

Appendix

Backend Inventory (PB-34112)

List of files impacted

As per investigation, resource metadata (uri, username, name) is used in below places:

App\Notification\Email\Redactor\Resource\ResourceCreateEmailRedactor

- Inside `createResourceCreateEmail` method
  - Change `$resource->name` to <todo>

templates/email/html/LU/resource_create.php

- Change `$resources` array occurrences
  - `$resource['name']` -> <todo>
  - `$resource['username']` -> <todo>
  - `$resource['uri']` -> <todo>

App\Notification\Email\Redactor\Resource\ResourceUpdateEmailRedactor

- Inside `createUpdateEmail` method
  - Change `$resource->name` to <todo>

templates/email/html/LU/resource_update.php

- Change `$resources` array occurrences
  - `$resource['name']` -> <todo>
  - `$resource['username']` -> <todo>
  - `$resource['uri']` -> <todo>

App\Notification\Email\Redactor\Resource\ResourceDeleteEmailRedactor

- Inside `createDeleteEmail` method
  - Change `$resource->name` to <todo>

templates/email/html/LU/resource_delete.php

- Change `$resource` array occurrences
  - `$resource['name']` -> <todo>
  - `$resource['username']` -> <todo>
  - `$resource['uri']` -> <todo>

App\Notification\Email\Redactor\Share\ShareEmailRedactor

- Inside `createShareEmail` method
  - Change `$resource->name` to <todo>

templates/email/html/LU/resource_share.php

- Change `$resource` array occurrences
  - `$resource['name']` -> <todo>
  - `$resource['username']` -> <todo>
  - `$resource['uri']` -> <todo>

App\Notification\Email\Redactor\Comment\CommentAddEmailRedactor

- Inside `createCommentAddEmail` method
  - Change `$resource->name` to <todo>

templates/email/html/LU/comment_add.php

- Change `$resource` array occurrences
  - `$resource['name']` -> <todo>

App\Controller\Resources\ResourcesAddController

- Request data and response needs to be adapted according to new structure

App\Service\Resources\ResourcesAddService

- Inside `buildEntity` method, entity build creation needs to be adapted (i.e. store in separate resource metadata column, table)

App\Controller\Resources\ResourcesUpdateController

- Request data and response needs to be adapted according to new structure

App\Service\Resources\ResourcesAddService

- `extractDataResourceMeta` method will be changed
- Inside `patchEntity` method, patch entity code needs to be adapted (i.e. store in separate resource metadata column, table)

App\Model\Table\ResourcesTable

- `softDelete` method > patch entity data (to clean up sensitive data)

Passbolt\AuditLog\Utility\ResourceActionLogsFinder (PassboltEe/AuditLog)

- `_findActionLogIdsForResources` method

    - `innerJoinWith('EntitiesHistory.Resources')` - inner join with resources to show resource name, etc.

Migration - V350IncreaseResourcesNameUsernameLengthInResourceTypes

- `updateResourceTypeDefinition` method
    - Updates `name` and `username` resource properties array

Future Ideas

This is our kitchen sink. Don't trust anything there.

API Specifications - Comments

GET /comments/resource/<uuid>.json (Updated)

TODO

POST /comments/resource/<uuid>.json (Updated)

TODO

PUT /comments/<uuid>.json (Updated)

TODO

Other ideas

Solving unplanned migration issues

metadata_history (new table)

This table could be used to prevent data loss when performing breaking changes, for example when migrating a resource type from unencrypted to encrypted metadata, or when rotating a session_key. This will be our life raft when (if not if) something goes wrong. It should be flushed once you're confident things are working fine.

Field	Type	Index
id	uuid, not null	PK
action_id	uuid	
foreign_key	uuid, not null	For ex. Ref Resources.id
foreign_model	uuid, not null	For example: Resources
previous_metadata_version	mediumtext	could be json or OpenPGP
created	datetime	
created_by	uuid	Ref. users.id

Solving performance issues on decryption of metadata (MOAR)

Option 2. Using encrypted local cache

Another approach that can be used in addition with option 1. would be to store in local storage encrypted using another local symmetric encryption key (encrypted itself with the private key and stored locally). The goal here would be to keep the local storage (encrypted) and not flushing it anymore (unless extension is uninstalled) and then rely on the `diff` mechanism to update it.

Keeping an encrypted local cache would reduce the time needed to decrypt content at login in most use cases during daily use on a configured device.

The high level logic for the client would be as follow:

```
1. When the user login (or refreshes page)
2. If there is no encrypted local cache
      a. Generate new symmetric key
      b. Encrypt and save empty local cache from memory on disk
3. Try to decrypt and load the resource cache in memory
      a. If it fails delete local cache and goto 2
4. Fetch resource list server side
5. Calculate diff between local cache and server answer
6. If resources that are in cache are not found server side
      a. Delete them locally in memory
7. If resources are present locally and remote modified date more recent
      a. Decrypt updated resources metadata from server request
      b. Update them locally in memory
8. If resources are not present locally
      a. Decrypt new resources metadata from server request
      b. Add them locally in memory
9. If some changes have been made
      a. encrypt and save local cache from memory to disk
```

Solving fetching only changes?

Get permissions.json
Goal: Get list of all my permissions with modified date
Returns: {user_id/group_id, resource_id/folder_id, permission_type, aro_type, aco_type, modified}

Get resources from local storage

Goal: Do a diff of result of permissions.json with local storage based on resource ids
Returns
=> list of the resources that the user lost access (deleted or lost permission)
=> list of the resources that the user gain access (created or added permission)

Get resources.json?modified-after=<last_sync_date>
Goal: Get list of resources created/modified after that date
Returns
=> list of resources where secret or metadata have been updated

Get list of all folders relations
{user_id, foreign_id, folder_parent_id, modified}

Resources.json
What is considered a change:

- Metadata
- Secret

What is not considered a change:

- Tag
- **Permission**
- **Folder relation**

Other commands

$./bin/cake passbolt rotate_metadata_key

This command must be confirmed with user input as it can potentially break things.
When this command is run, the server should not be allowed to serve requests as it can mess things up royally.

What it does:

- Assert nginx or apache is not running
- Assert that there is already an entry in session_keys table with user set to null and personal set to false
- Assert that it can be decrypted with the current server private key
- Assert that it is a metadata key object
- Generate a new metadata key object
- In one transaction:
    - Assert server public key can be used

- Encrypt the metadata key object with server public key
- Create entry in session_key table with personal
- For each users
  - Get the user OpenPGP key from gpgkeys table
  - Assert the public key can be used
  - Encrypt the metadata key object for the given public key
  -
- For each resources with the matching metdata_key_id
  - Decrypt the metadata with the old key
  - Assert the metadata
  - Encrypt the metadata with the new key
  - Update the new resources with the new metadata and metdata_key_id
  - Create metadata_history entry with old value (to avoid data loss)
- Set deleted date to old session_key entry

$./bin/cake passbolt delete_metadata_key

TODO. once rotated, allow to delete a key.

Settings mismatch