@push.rocks/smartsecret

Documentation for @push.rocks/smartsecret

- readme.md for @push.rocks/smartsecret
- changelog.md for @push.rocks/smartsecret

readme.md for @push.rocks/smartsecret

OS keychain-based secret storage with encrypted-file fallback for Node.js.

Install

To install `@push.rocks/smartsecret`, use pnpm:

```
pnpm install @push.rocks/smartsecret
```

Usage

`@push.rocks/smartsecret` provides a unified API for storing and retrieving secrets. It automatically selects the best available backend for the current platform: macOS Keychain on macOS, `secret-tool` (libsecret / GNOME Keyring) on Linux, or an AES-256-GCM encrypted file as a universal fallback.

Basic Setup

```
import { SmartSecret } from '@push.rocks/smartsecret';

// Create an instance with default settings
const secretStore = new SmartSecret();

// Or specify a custom service name and vault path
const secretStore = new SmartSecret({
  service: 'my-application',
  vaultPath: '/path/to/custom/vault.json',
});
```

The `service` option acts as a namespace, isolating secrets so that different applications do not collide. It defaults to `'smartsecret'` when omitted.

The `vaultPath` option only applies to the encrypted-file backend and controls where the vault JSON file is stored. It defaults to `~/.config/smartsecret/vault.json`.

Storing a Secret

```
await secretStore.setSecret('api-key', 'sk-abc123xyz');
```

If a secret with the same account name already exists under the configured service, it is overwritten.

Retrieving a Secret

```
const apiKey = await secretStore.getSecret('api-key');

if (apiKey !== null) {
  console.log('Retrieved secret:', apiKey);
} else {
  console.log('Secret not found');
}
```

Returns `null` when no secret exists for the given account.

Deleting a Secret

```
const wasDeleted = await secretStore.deleteSecret('api-key');
console.log(wasDeleted); // true if the secret existed and was removed
```

Returns `false` if the secret did not exist.

Listing Accounts

```
const accounts = await secretStore.listAccounts();
console.log(accounts); // e.g. ['api-key', 'db-password', 'oauth-token']
```

Returns an array of account names that have stored secrets under the configured service.

Checking the Active Backend

```
const backendType = await secretStore.getBackendType();
console.log(backendType);
// 'macos-keychain' | 'linux-secret-service' | 'file-encrypted'
```

This is useful for logging or diagnostics to understand which storage mechanism is in use at runtime.

Service-Based Isolation

Different `SmartSecret` instances with different `service` names maintain completely separate secret namespaces, even when sharing the same underlying storage:

```
const appSecrets = new SmartSecret({ service: 'my-app' });
const ciSecrets = new SmartSecret({ service: 'ci-pipeline' });

await appSecrets.setSecret('token', 'app-token-value');
await ciSecrets.setSecret('token', 'ci-token-value');

const appToken = await appSecrets.getSecret('token');  // 'app-token-value'
const ciToken = await ciSecrets.getSecret('token');     // 'ci-token-value'
```

API Reference

`SmartSecret`

The main class. Instantiate it to store and retrieve secrets.

Constructor

```
new SmartSecret(options?: ISmartSecretOptions)
```

Option	Type	Default	Description
`service`	`string`	`'smartsecret'`	Namespace for secret isolation

Option	Type	Default	Description
`vaultPath`	`string`	`~/.config/smartsecret/vault.json`	Path to the encrypted vault file (file backend only)

Methods

Method	Signature	Description
`setSecret`	`(account: string, secret: string) => Promise<void>`	Store or overwrite a secret
`getSecret`	`(account: string) => Promise<string | null>`	Retrieve a secret, or `null` if not found
`deleteSecret`	`(account: string) => Promise<boolean>`	Delete a secret; returns `true` if it existed
`listAccounts`	`() => Promise<string[]>`	List all account names for the configured service
`getBackendType`	`() => Promise<TBackendType>`	Returns the active backend identifier

Types

```
type TBackendType = 'macos-keychain' | 'linux-secret-service' | 'file-encrypted';

interface ISmartSecretOptions {
  service?: string;
  vaultPath?: string;
}

interface ISecretBackend {
  readonly backendType: TBackendType;
  isAvailable(): Promise<boolean>;
  setSecret(account: string, secret: string): Promise<void>;
  getSecret(account: string): Promise<string | null>;
  deleteSecret(account: string): Promise<boolean>;
  listAccounts(): Promise<string[]>;
}
```

Backend Classes

Each backend implements `ISecretBackend` and can be used directly if needed:

- `MacosKeychainBackend` -- macOS Keychain via the `security` CLI
- `LinuxSecretServiceBackend` -- Linux Secret Service via `secret-tool`
- `FileEncryptedBackend` -- AES-256-GCM encrypted JSON vault file

Backends

`SmartSecret` tries each backend in order and uses the first one that reports itself as available.

macOS Keychain (`macos-keychain`)

Used automatically on macOS when the `security` command-line tool is present (ships with macOS by default). Secrets are stored as generic password items in the user's default keychain. The `service` option maps to the keychain service name, and the `account` parameter maps to the keychain account name.

Linux Secret Service (`linux-secret-service`)

Used automatically on Linux when `secret-tool` is installed. This integrates with GNOME Keyring, KDE Wallet, or any other provider that implements the freedesktop.org Secret Service D-Bus API. Install the tool on Debian/Ubuntu with:

```
sudo apt install libsecret-tools
```

Secrets are stored with `service` and `account` as lookup attributes.

Encrypted File (`file-encrypted`)

The universal fallback that works on all platforms. Secrets are encrypted with AES-256-GCM and stored in a JSON vault file. A random 32-byte key is generated on first use and stored alongside the vault at `~/.config/smartsecret/.keyfile` (with `0600` permissions). The encryption key is derived from the keyfile using PBKDF2 with 100,000 iterations of SHA-512, salted with the service name.

Vault writes are atomic (write to a temporary file, then rename) to prevent corruption. Both the keyfile and the vault file are created with restrictive file permissions.

File locations (defaults):

File	Path
Vault	`~/.config/smartsecret/vault.json`
Keyfile	`~/.config/smartsecret/.keyfile`

Both paths can be influenced by providing a custom `vaultPath` in the constructor options. The keyfile is always stored in the same directory as the vault.

License and Legal Information

This project is licensed under the MIT license. For more details, see the `license` file in the repository.

By using this package, you agree to the licensing terms.

changelog.md for @push.rocks/smartsecret

2026-02-24 - 1.0.2 - fix()

no changes detected

- No files were modified in this diff; no release required.

2026-02-24 - 1.0.1 - fix(smartsecret)

no changes

- No files changed in this diff; no code or dependency updates to release.

2025-02-24 - 1.0.0 - Initial Release

- macOS Keychain backend via `security` CLI
- Linux secret-tool backend via libsecret
- AES-256-GCM encrypted file vault fallback
- Automatic backend detection with graceful fallback