**Flutter Secure Storage vs Shared Preferences with Table**

Flutter offers two common ways to store small amounts of data on the device: **Secure Storage** and **Shared Preferences**. Here's a detailed comparison:

**1. Flutter Secure Storage**

* **Package**: flutter\_secure\_storage
* **Purpose**: Used for securely storing sensitive information like API tokens, user credentials, or other confidential data.

**Key Features**

* **Security**: Uses platform-specific secure storage:
  + Android: **Encrypted Shared Preferences** or **KeyStore**
  + iOS: **Keychain**
* **Encryption**: Data is encrypted at rest.
* **Accessibility**: Supports secure access, and the data can only be retrieved by the app that stored it.

**Use Cases**

* Storing **tokens** (e.g., OAuth tokens).
* Storing **user credentials**.
* Storing **sensitive app configurations**.

**Advantages**

* Highly secure.
* Data is encrypted both at rest and in transit.
* Easy integration with platform-specific security mechanisms.

**Disadvantages**

* Slower than SharedPreferences because of encryption/decryption processes.
* Less suitable for large datasets or frequent reads/writes.

**Example**

dart

Copy code

import 'package:flutter\_secure\_storage/flutter\_secure\_storage.dart';

final storage = FlutterSecureStorage();

// Write data

await storage.write(key: 'token', value: 'secureToken123');

// Read data

String? token = await storage.read(key: 'token');

// Delete data

await storage.delete(key: 'token');

**2. SharedPreferences with Table**

* **Package**: shared\_preferences
* **Purpose**: Used for storing non-sensitive data in key-value pairs.

**Key Features**

* **No Encryption**: Data is stored as plain text, making it less secure.
* **Simple Storage**: Data is stored in shared preferences files.
* **Performance**: Faster for frequent reads/writes.

**Use Cases**

* Storing **user preferences** (e.g., theme, language).
* Storing **non-sensitive settings**.
* Simple **data persistence** for lightweight apps.

**Advantages**

* Lightweight and fast.
* Easy to use for non-sensitive data.
* Good for small datasets and frequent access.

**Disadvantages**

* No built-in encryption; not suitable for sensitive information.
* SharedPreferences is accessible to other apps on rooted/jailbroken devices.

**Example**

dart

Copy code

import 'package:shared\_preferences/shared\_preferences.dart';

// Write data

SharedPreferences prefs = await SharedPreferences.getInstance();

await prefs.setString('username', 'Arbaaz');

// Read data

String? username = prefs.getString('username');

// Delete data

await prefs.remove('username');

**Key Differences**

| **Feature** | **Flutter Secure Storage** | **SharedPreferences with Table** |
| --- | --- | --- |
| **Encryption** | Yes | No |
| **Purpose** | Secure, sensitive data | Non-sensitive data |
| **Performance** | Slower due to encryption | Faster |
| **Data Type Support** | Strings only | Strings, Int, Double, Bool |
| **Use Case** | Passwords, Tokens, Keys | App settings, Preferences |

**Which to Use?**

* **Choose Flutter Secure Storage**:
  + When storing **sensitive data** like passwords, tokens, or encryption keys.
  + If security and data protection are critical.
* **Choose SharedPreferences**:
  + For **non-sensitive data** like user preferences, app configurations, or temporary settings.
  + When performance is a priority, and encryption isn't required.

Let me know if you need code examples tailored to a specific scenario! 😊

4o

**You said:**

give me as documentation

**ChatGPT said:**

ChatGPT

**Flutter Secure Storage vs SharedPreferences Documentation**

**Overview**

Flutter provides two primary methods for local data storage:

1. **Flutter Secure Storage**: Designed for securely storing sensitive data.
2. **SharedPreferences**: Used for lightweight, non-sensitive data storage in key-value pairs.

**Flutter Secure Storage**

**Introduction**

The flutter\_secure\_storage package is a secure storage solution for sensitive data such as passwords, tokens, or private keys. It uses platform-specific secure storage mechanisms:

* **Android**: EncryptedSharedPreferences or Keystore.
* **iOS**: Keychain.

**Installation**

Add the package to your pubspec.yaml:

yaml

Copy code

dependencies:

flutter\_secure\_storage: ^8.0.0

Run flutter pub get to install the package.

**Key Features**

* Data is encrypted at rest.
* Platform-specific secure storage.
* Supports read, write, delete, and deleteAll operations.

**Usage**

1. **Initialization**

dart

Copy code

import 'package:flutter\_secure\_storage/flutter\_secure\_storage.dart';

final storage = FlutterSecureStorage();

1. **Write Data**

dart

Copy code

await storage.write(key: 'token', value: 'secureToken123');

1. **Read Data**

dart

Copy code

String? token = await storage.read(key: 'token');

1. **Delete Data**

dart

Copy code

await storage.delete(key: 'token');

1. **Delete All Data**

dart

Copy code

await storage.deleteAll();

**Advantages**

* Provides high security for sensitive data.
* Data is encrypted on-device.

**Disadvantages**

* Slower compared to SharedPreferences.
* Limited to storing strings only.

**SharedPreferences**

**Introduction**

The shared\_preferences package is used for lightweight key-value storage, suitable for non-sensitive data like user preferences and app settings.

**Installation**

Add the package to your pubspec.yaml:

yaml

Copy code

dependencies:

shared\_preferences: ^2.1.0

Run flutter pub get to install the package.

**Key Features**

* Stores data as plain text (no encryption).
* Supports multiple data types: String, int, double, bool, and List<String>.
* Easy to use and highly performant.

**Usage**

1. **Initialization**

dart

Copy code

import 'package:shared\_preferences/shared\_preferences.dart';

SharedPreferences prefs = await SharedPreferences.getInstance();

1. **Write Data**

dart

Copy code

await prefs.setString('username', 'Arbaaz');

await prefs.setInt('age', 25);

1. **Read Data**

dart

Copy code

String? username = prefs.getString('username');

int? age = prefs.getInt('age');

1. **Delete Data**

dart

Copy code

await prefs.remove('username');

1. **Clear All Data**

dart

Copy code

await prefs.clear();

**Advantages**

* Lightweight and fast.
* Supports various data types.

**Disadvantages**

* No built-in encryption; not suitable for sensitive data.

**Comparison Table**

| **Feature** | **Flutter Secure Storage** | **SharedPreferences** |
| --- | --- | --- |
| **Security** | Encrypted data storage | Plain text storage |
| **Purpose** | Sensitive data | Non-sensitive data |
| **Data Types** | Strings only | Strings, int, bool, etc. |
| **Performance** | Slower (encryption overhead) | Faster |
| **Best Use Case** | Tokens, passwords | Preferences, settings |

**When to Use Which?**

* **Flutter Secure Storage**:
  + Store sensitive data like passwords, API tokens, and encryption keys.
* **SharedPreferences**:
  + Store non-sensitive data such as app configurations, user preferences, and temporary settings.

**References**

* Flutter Secure Storage
* SharedPreferences