

# Protected PlayerPrefs Toolkit

Protected PlayerPrefs Toolkit is a lightweight Unity package for project-local data protection, key management, and editor-side inspection of stored values. It wraps Unity `PlayerPrefs`, hashes keys, encrypts values, and adds a guided setup flow so the package can be configured before it ships.

## Screenshot Placeholder

Package Overview - Add a screenshot of the package files and folders inside the Unity Project window. Replace this box with the final package screenshot before submission.

## What Is Included

- `ProtectedPlayerPrefs`: runtime API for string, int, float, and bool values.
- `ProtectedPlayerPrefsSettings`: project settings asset that stores the SHA-256 hash derived from your password.
- Getting Started window: first-run setup that appears automatically when the hash password has not been configured.
- `PlayerPrefsKeyRegistry`: registry asset for known keys and descriptions.
- `PlayerPrefs Editor`: editor window for registered keys, raw lookup, delete actions, and key-code generation.
- `PlayerPrefsKeys`: generated constants class that removes string literals from gameplay and service code.

## Quick Start

- Import the package into your project.
- If no hash password is configured, Unity creates the settings asset and opens `Tools/Darkmatter/Protected PlayerPrefs/Getting Started`.
- Enter a strong password and save it. The package stores only the SHA-256 hash, not the raw password.
- Keep the original password in a secure team password manager.
- Use `ProtectedPlayerPrefs` from runtime code and `Tools/Darkmatter/PlayerPrefs Editor` from the editor.

### Screenshot Placeholder

Getting Started Window - Add a screenshot of the first-run setup window with the password fields and action buttons. Replace this box with the final package screenshot before submission.

## Settings Asset and Hash Password

The package creates or reuses this asset path by default:

```
Assets/Darkmatter/Data/Settings/Persistence/Resources/ProtectedPlayerPrefsSettings.asset
```

Important behavior:

- The runtime loads the settings asset through `Resources`, so the configured hash is available in builds.
- If no configured hash exists, the runtime falls back to the built-in default passphrase for backward compatibility.
- Changing the password later changes the derived encryption key and IV. Existing protected values will no longer decrypt.
- The package stores only the SHA-256 hash of the password. The original password cannot be recovered from the asset.

### Screenshot Placeholder

Settings Asset Inspector - Add a screenshot of the generated `ProtectedPlayerPrefsSettings` asset in the Inspector. Replace this box with the final package screenshot before submission.

## Runtime API

Use `ProtectedPlayerPrefs` exactly like a narrow, protected subset of Unity `PlayerPrefs`.

```
using Darkmatter.Libs.PlayerPrefs;

ProtectedPlayerPrefs.SetString(PlayerPrefsKeys.Accounts.SavedAuthRequest, json);
ProtectedPlayerPrefs.SetInt("Profile.Level", 12);
ProtectedPlayerPrefs.SetFloat("Audio.MasterVolume", 0.8f);
ProtectedPlayerPrefs.SetBool("Onboarding.Completed", true);
ProtectedPlayerPrefs.Save();
```

Reading values:

```
var savedAuth = ProtectedPlayerPrefs.GetString(PlayerPrefsKeys.Accounts.SavedAuthRequest, string.Empty);
var level = ProtectedPlayerPrefs.GetInt("Profile.Level", 1);
var volume = ProtectedPlayerPrefs.GetFloat("Audio.MasterVolume", 1f);
var onboardingDone = ProtectedPlayerPrefs.GetBool("Onboarding.Completed", false);
```

Available API surface:

- `Init(string passphrase)`: optional manual initialization from a raw password.
- `InitWithHash(string hashedPassphrase)`: optional manual initialization from a SHA-256 hash.
- `ComputePassphraseHash(string passphrase)`: helper used by the setup flow.
- `SetString, GetString`
- `SetInt, GetInt`
- `SetFloat, GetFloat`
- `SetBool, GetBool`
- `HasKey, DeleteKey, DeleteAll, Save`

## Key Registry and Code Generation

Use `PlayerPrefsKeyRegistry` to maintain a curated list of known keys, types, and descriptions. The editor window can generate a strongly named `PlayerPrefsKeys` class from the registry.

Benefits:

- Central place to document keys.
- Safer refactoring than raw string literals.
- Cleaner service and feature code.
- Easier QA and debugging in the editor window.

Example generated structure:

```
public static class PlayerPrefsKeys
{
    public static class Accounts
    {
        public const string SavedAuthRequest = "Accounts.SavedAuthRequest";
    }

    public static class SaveGame
    {
        public const string Session = "SaveGame.Session";
        public const string Vehicle = "SaveGame.Vehicle";
    }
}
```

## PlayerPrefs Editor

Open the tool from:

```
Tools/Darkmatter/PlayerPrefs Editor
```

The editor window includes three areas:

- Registered Keys: edit keys defined in the registry and save protected values.

- Raw Lookup: inspect or delete unprotected third-party or legacy keys by name.
- Settings: manage registry generation output and open the Protected PlayerPrefs setup flow.

### Screenshot Placeholder

Registered Keys Tab - Add a screenshot of the Registered Keys tab with a few example entries. Replace this box with the final package screenshot before submission.

### Screenshot Placeholder

Raw Lookup Tab - Add a screenshot of the Raw Lookup tab showing a direct key lookup. Replace this box with the final package screenshot before submission.

## Integration Notes

- Treat this package as protection against casual local tampering, not as a replacement for a trusted backend.
- Use generated keys for feature and service code whenever the key is known ahead of time.
- Save immediately after write operations that must survive app termination.
- For sensitive gameplay state, pair local protection with server validation when the product design requires trust.

## Recommended Workflow for Shipping

- Configure the hash password on first import.
- Register the keys that belong to your project.
- Generate `PlayerPrefsKeys.cs`.
- Replace raw string literals with generated constants.
- Run a smoke test with a fresh install and with existing saved data.
- Capture the screenshots marked in this documentation before publishing the package.

## Troubleshooting

Problem: protected values return defaults after you change the password.

- Cause: the encryption key changed because the stored hash changed.
- Fix: restore the original password or clear and rebuild the protected local cache.

Problem: the setup window keeps appearing.

- Cause: the settings asset exists, but its hash password is still blank.
- Fix: open the Getting Started window and save a password.

Problem: a key is visible in PlayerPrefs but not in the Registered Keys tab.

- Cause: the value exists, but the key is not in `PlayerPrefsKeyRegistry`.
- Fix: add the key to the registry or inspect it from Raw Lookup.

## Package Files

Core files:

- `Assets/Darkmatter/Code/Libs/PlayerPrefs/Runtime/ProtectedPlayerPrefs.cs`
- `Assets/Darkmatter/Code/Libs/PlayerPrefs/Runtime/ProtectedPlayerPrefsSettings.cs`
- `Assets/Darkmatter/Code/Libs/PlayerPrefs/Runtime/PlayerPrefsKeyRegistry.cs`
- `Assets/Darkmatter/Code/Libs/PlayerPrefs/Runtime/PlayerPrefsKeys.cs`
- `Assets/Darkmatter/Code/Libs/PlayerPrefs/Editor/PlayerPrefsEditorWindow.cs`
- `Assets/Darkmatter/Code/Libs/PlayerPrefs/Editor/ProtectedPlayerPrefsGettingStartedWindow.cs`
- `Assets/Darkmatter/Code/Libs/PlayerPrefs/Editor/ProtectedPlayerPrefsSetupBootstrap.cs`

## Final Checklist Before Asset Store Submission

- Verify the configured hash password is set.
- Remove project-specific sample keys that should not ship.
- Replace every screenshot placeholder in this PDF.
- Confirm the generated documentation opens correctly on a clean machine.
- Test the package in a fresh Unity project before export.