SV Anti-Cheat Secure Variables

Table of contents

[SV Anti-Cheat Secure Variables](#)
- [Description](#)
- [Features](#)
- [How to use](#)
- [Supported types](#)
- [Some example screenshot](#)

Description

Protect your game from cheaters who are trying to change your variables in the memory.
It's a lightweight easy to use out of box solution to encrypt your variables in RAM. It protects your variables from memory scanners and searchers. A lot of cheaters use these scanners and searchers to increase their money and items count. Or, for example, to freeze health to stay immortal.

Features

* Out of box solution
* Easy to use
* Lightweight
* Protect your game from cheaters who trying change your variables in the memory

How to use

1. Import asset package
2. Use secure types instead of regular types for the data that you want to protect from memory cheating.

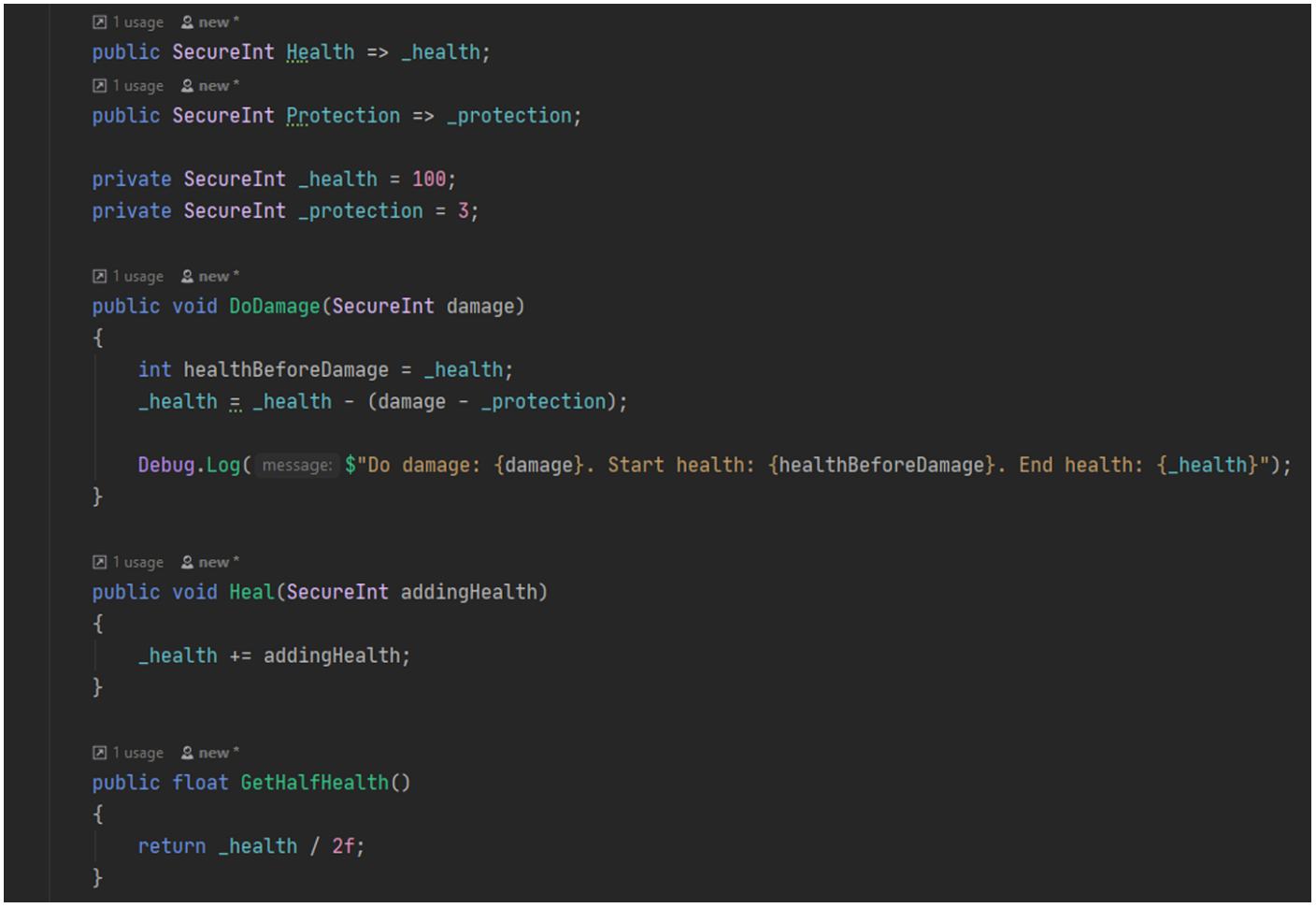
Use secure types exactly as you used the corresponding types.
Also the asset package contains example scene with using secure types.

Supported types

- SecureBool
- SecureDouble
- SecureFloat
- SecureInt
- SecureLong
- SecureString
- SecureByte
- SecureChar

- SecureShort
- SecureUInt
- SecureUShort
- SecureSByte
- SecureULong
- SecureVector2
- SecureVector2Int
- SecureVector3
- SecureVector3Int
- SecureQuaternion

Some example screenshot



```
1 usage  new *
public SecureInt Health => _health;
1 usage  new *
public SecureInt Protection => _protection;

private SecureInt _health = 100;
private SecureInt _protection = 3;

1 usage  new *
public void DoDamage(SecureInt damage)
{
    int healthBeforeDamage = _health;
    _health = _health - (damage - _protection);

    Debug.Log( message: $"Do damage: {damage}. Start health: {healthBeforeDamage}. End health: {_health}");
}

1 usage  new *
public void Heal(SecureInt addingHealth)
{
    _health += addingHealth;
}

1 usage  new *
public float GetHalfHealth()
{
    return _health / 2f;
}
```