Secure Storage

Generated by Doxygen 1.9.3

1 Namespace Index 1

1.1 Packages . . . 1

2 Class Index 3

2.1 Class List . . . 3

3 Namespace Documentation 5

3.1 SecureStorage Namespace Reference . . . 5

3.1.1 Detailed Description . . . 5

4 Class Documentation 9

4.1 SecureStorage.DataStorage Class Reference . . . 9

4.1.1 Detailed Description . . . 9

4.1.2 Constructor & Destructor Documentation . . . 9

4.1.2.1 DataStorage() . . . 9

4.1.3 Member Function Documentation . . . 10

4.1.3.1 BinaryDeserialize() . . . 10

4.1.3.2 BinarySerialize() . . . 10

4.1.3.3 LoadData() . . . 10

4.1.3.4 SaveData() . . . 11

4.2 SecureStorage.Initializer Class Reference . . . 11

4.2.1 Constructor & Destructor Documentation . . . 12

4.2.1.1 Initializer() . . . 12

4.2.2 Member Function Documentation . . . 12

4.2.2.1 SetKeyValue_Default() . . . 12

4.3 SecureStorage.ObjectStorage Class Reference . . . 13

4.3.1 Detailed Description . . . 13

4.3.2 Constructor & Destructor Documentation . . . 13

4.3.2.1 ObjectStorage() . . . 13

4.3.3 Member Function Documentation . . . 14

4.3.3.1 DeleteAllObject() . . . 14

4.3.3.2 DeleteObject() . . . 14

4.3.3.3 GetAllKey() . . . 14

4.3.3.4 GetAllObjects() . . . 15

4.3.3.5 LoadObject() . . . 15

4.3.3.6 SaveObject() . . . 15

4.4 SecureStorage.Values Class Reference . . . 16

4.4.1 Detailed Description . . . 17

4.4.2 Member Function Documentation . . . 17

4.4.2.1 Get() [1/5] . . . 17

4.4.2.2 Get() [2/5] . . . 17

4.4.2.3 Get() [3/5] . . . 17

4.4.2.4 Get() [4/5] . . . 18

4.4.2.5 Get() [5/5] 18
4.4.2.6 Set() 18

Index **19**

Chapter 1

Namespace Index

1.1 Packages

Here are the packages with brief descriptions (if available):

SecureStorage
5

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

SecureStorage.DataStorage
: This class has been conceived to be able to save data in a safe way, that is: The saved data of any application cannot be considered safe if it will be accessible in clear text to other applications or resident software. For encryption to be active, the library must be initialized using the Initializer class, enabling encryption (encryption is enabled by default). 9

SecureStorage.Initializer 11

SecureStorage.ObjectStorage
: This class has been conceived to be able to save objects in a safe way, that is: The saved data of any application cannot be considered safe if it will be accessible in clear text to other applications or resident software. For encryption to be active, the library must be initialized using the Initializer class, enabling encryption (encryption is enabled by default). 13

SecureStorage.Values
: This class is used to save in an encrypted and secure way the values, which can be used by any application that needs to store parameters, such as configuration values, flags, names, and variables that you do not want to be lost after restarting of the application 16

Chapter 3

Namespace Documentation

3.1 SecureStorage Namespace Reference

Classes

- class **Cryptography**

  Class to encrypt and decrypt the data.
- class DataStorage

  This class has been conceived to be able to save data in a safe way, that is: The saved data of any application cannot be considered safe if it will be accessible in clear text to other applications or resident software. For encryption to be active, the library must be initialized using the Initializer class, enabling encryption (encryption is enabled by default).
- class Initializer
- class ObjectStorage

  This class has been conceived to be able to save objects in a safe way, that is: The saved data of any application cannot be considered safe if it will be accessible in clear text to other applications or resident software. For encryption to be active, the library must be initialized using the Initializer class, enabling encryption (encryption is enabled by default).
- class Values

  This class is used to save in an encrypted and secure way the values, which can be used by any application that needs to store parameters, such as configuration values, flags, names, and variables that you do not want to be lost after restarting of the application

3.1.1 Detailed Description

The necessity and desire to secure personal information is one thing that everyone shares around the world in the recent times, ranging from businesses to governments to military structures. Data security is critical whether it is being stored, sent, or delivered. Data breaches, hacking, and lost or stolen devices can have catastrophic financial and reputational costs. The need for a Library to protect data generated and handled by applications arose from a desire to protect not only public structures, but also individual citizens, who are even more at risk if their freedom of expression, gender, religion, and any data relating to their person and loved ones is not protected.

Any application that does not secure the data it generates and manages carries the risk of revealing sensitive information that can be used to profile users, scammers to invent scams, and hackers to carry out their plans to pirated programs. The information created by the applications can easily be gathered and marketed on the dark web.

SecureStorage is a library that provides effective encryption to the apps that use it, making the data generated by it inaccessible and inviolable.

Any application creates a large quantity of data; some of it serves just as a warning, while others are essential to the application's operation and users, and some of it, if interfered with, can allow the application and its content to be hacked.

To protect yourself from malicious hackers and organizational data breaches, encrypt all data generated by the application and prevent it from being saved in a way that may be read externally. In the case that unwanted access is permitted to a computer network or storage device, other apps on the same device, or system applications designed with fraudulent purpose by the device's maker, encryption provides an extra level of protection. The hacker will be unable to access the application data encrypted through SecureStorage.

What is encryption?

Simply said, encryption transforms data entered into a digital device into gibberish-like pieces. The encrypted data becomes more unreadable and indecipherable as the encryption technique becomes more complex. Decryption, on the other hand, restores the encrypted data to its original state, making it readable again. Unencrypted data is referred to as normal data, and encrypted data is referred to as encrypted data.

Software vs Hardware encryption

Software encryption encrypts data on a logical disk using a number of software packages. A unique key is created and saved in the computer's memory when a drive is encrypted for the first time. A user passcode is used to encrypt the key. When a user enters the passcode, the key is unlocked, allowing access to the drive's unencrypted data. The drive also stores a copy of the key. When data is written to the drive, it is encrypted using the key before it is physically committed to the disk; software encryption works as an intermediate between application read / write data on the device. Before being given to the software, data read from the drive is decrypted using the same key. Hardware - level encryption is possible on some devices: Hardware - based encryption is used in Self - Encrypting Drives(SEDs), which takes a more comprehensive approach to encrypting user data. SEDs include an AES encryption chip that encrypts data before it is written to NAND media and decrypts it before it is read. Between the operating system loaded on the drive and the system BIOS is where hardware encryption takes place. An encryption key is generated and stored on NAND flash memory when the drive is encrypted for the first time. A custom BIOS is loaded when the system is first booted, prompting for a user password. The contents of the drive are decrypted and access to the operating system and user data is provided once the pass is entered.

Self-encrypting drives also encrypt and decrypt data on the fly, with the built-in cryptographic chip encrypting and decrypting data before it is written to NAND flash memory. Because the encryption procedure does not use the host CPU, the performance penalty associated with software encryption is reduced. The encryption key is typically placed in the SSD's built-in memory at system startup, which complicates recovery and makes it less vulnerable to low-level attacks. This hardware-based encryption solution provides strong data security in the event that the device is lost, cannot be disabled, and has no performance impact. However, it is a type of low-level encryption that is completely transparent to the device that uses these storage units, as well as to all software programs that run on the device. As a result, this type of encryption does not protect the data of individual applications and users from other resident programs that can see all of the data stored in clear text.

SecureStorage provides an additional layer of security for individuals who utilize primary hardware encrypted devices, rendering the data unreadable outside of the single program that created and is using it. </ para >

The Advanced Encryption Standard (AES) is a cryptographic technique that is based on the Rijndael family of algorithms. It is now one of the most widely used encryption and decryption techniques. Vincent Rijmjen and Joan Daemen created the Rijndael algorithm, which is a block cipher. It's a symmetric-key algorithm, which means it encrypts and decrypts data with the same key. As a consequence of the NIST Advanced Encryption Standard competition, the Rijndael algorithm was chosen as an Advanced Encryption Standard and the successor to the Data Encryption Standard (DES). The competition was held in order to produce a new cryptographic standard as a replacement for the obsolete DES. Because to the modernization of computer technologies, the Data Encryption Standard's key length (56 bits) was insecure at the time. The Rijndael family of functions is represented by three algorithms in the AES standard. They have varying key lengths of 128, 192, and 256 bits, but they all use the same 128-bit block length. More variations of encryption algorithms, cyphers, and other cryptographic functions are included in the Rijndael family of hashing functions than in AES. The Advanced Encryption Standard was designed to work equally well in software and hardware implementations. With the deployment of the substitution–permutation network design, it was possible. This network design is similar to the Feistel network, which was utilized in DES,

but it is faster to compute on both hardware and software, which was critical given DES's software implementation inefficiency.

Our cryptography is the same as that used in Bitcoin, which has been put to the test by hackers all around the world without ever being broken: Breaking this form of cryptography would give you access to coins stored in wallets, which no one has ever done before.

The Advanced Encryption Algorithm (AES256) is an AES algorithm with a key length of 256 bits.The computational difficulty of the decryption is affected by the length of the AES version. The key recovery for AES 256-encrypted data requires more computational power than the 128 and 192-bit variants. The biclique attack, for example, can decrypt AES128 with a computational complexity of 2126. The computational complexity of biclique attacks on AES 192 and AES 256 are 2189.9 and 2254.3, respectively.However, for every key length, real execution of the attacks on the AES-protected data is currently impractical. All of the AES attacks are hypothetical. Every known AES attack would take millions of years to complete, regardless of the algorithm's key length.

Chapter 4

Class Documentation

4.1 SecureStorage.DataStorage Class Reference

This class has been conceived to be able to save data in a safe way, that is: The saved data of any application cannot be considered safe if it will be accessible in clear text to other applications or resident software. For encryption to be active, the library must be initialized using the Initializer class, enabling encryption (encryption is enabled by default).

Public Member Functions

- DataStorage (Initializer secureStorage)

  Initialized object storageusing the Initializer class, enabling encryption (encryption is enabled by default).
- void SaveData (byte[ ] data, string key)

  This method is used to encrypt and securely save data with their public properties.
- byte[ ] LoadData (string key)

  This method is used to load a previously saved data.
- void BinarySerialize (object obj, string key)

  Serialize the object using the key.
- object BinaryDeserialize (string key)

  Deserialize the binarydata to object using the key.

4.1.1 Detailed Description

This class has been conceived to be able to save data in a safe way, that is: The saved data of any application cannot be considered safe if it will be accessible in clear text to other applications or resident software. For encryption to be active, the library must be initialized using the Initializer class, enabling encryption (encryption is enabled by default).

4.1.2 Constructor & Destructor Documentation

4.1.2.1 DataStorage()

```
SecureStorage.DataStorage.DataStorage (
            Initializer secureStorage )
```

Initialized object storageusing the Initializer class, enabling encryption (encryption is enabled by default).

Parameters

secureStorage	Storage name

4.1.3 Member Function Documentation

4.1.3.1 BinaryDeserialize()

```
object SecureStorage.DataStorage.BinaryDeserialize (
            string key )
```

Deserialize the binarydata to object using the key.

Parameters

key	Key used to deserialize

Returns

object

4.1.3.2 BinarySerialize()

```
void SecureStorage.DataStorage.BinarySerialize (
            object obj,
            string key )
```

Serialize the object using the key.

Parameters

obj	Object to be serialized
key	Key used to serialize the object

4.1.3.3 LoadData()

```
byte[] SecureStorage.DataStorage.LoadData (
            string key )
```

This method is used to load a previously saved data.

Parameters

key	Key used to save data

Returns

Saved data

4.1.3.4 SaveData()

```
void SecureStorage.DataStorage.SaveData (
            byte[] data,
            string key )
```

This method is used to encrypt and securely save data with their public properties.

Parameters

data	Data to save
key	Key used to save the Data.

The documentation for this class was generated from the following file:

- C:/documentation/CryptoMessenger/SecureStorageGit/SecureStorage/DataStorage.cs

4.2 SecureStorage.Initializer Class Reference

Public Member Functions

- Initializer (string domain, Func< string, string > getSecureKeyValue=null, SetKeyKalueSecure setSecure↩ KeyValue=null, bool encrypted=true)

  Prepares the library for using cryptography. Before using the library, initialization is mandatory!
- void **Destroy** ()

  Delete all the directory with all the content in it.
- delegate void **SetKeyKalueSecure** (string key, string value)
- void SetKeyValue_Default (string key, string value)

  Secure function provided by the hardware to be able to save keys

Public Attributes

- bool **SecureKeyValueCapability**
- readonly DataStorage **DataStorage**
- readonly ObjectStorage **ObjectStorage**
- readonly Values **Values**

4.2.1 Constructor & Destructor Documentation

4.2.1.1 Initializer()

```
SecureStorage.Initializer.Initializer (
            string domain,
            Func< string, string > getSecureKeyValue = null,
            SetKeyKalueSecure setSecureKeyValue = null,
            bool encrypted = true )
```

Prepares the library for using cryptography. Before using the library, initialization is mandatory!

Parameters

domain	The domain allows you to use multiple instances of the library. Then use different domains to have multiple instances
getSecureKeyValue	Function to get keys safely save in hardware.
setSecureKeyValue	Secure function provided by the hardware to be able to save keys
encrypted	Enable encryption (by default it is active and it is recommended not to delete it to keep your data safe)

4.2.2 Member Function Documentation

4.2.2.1 SetKeyValue_Default()

```
void SecureStorage.Initializer.SetKeyValue_Default (
            string key,
            string value )
```

Secure function provided by the hardware to be able to save keys

Parameters

key	Key to identify the users and will be used to save and delete data on the device
value	Encrypted Key Value

The documentation for this class was generated from the following file:

- C:/documentation/CryptoMessenger/SecureStorageGit/SecureStorage/initializer.cs

4.3 SecureStorage.ObjectStorage Class Reference

This class has been conceived to be able to save objects in a safe way, that is: The saved data of any application cannot be considered safe if it will be accessible in clear text to other applications or resident software. For encryption to be active, the library must be initialized using the Initializer class, enabling encryption (encryption is enabled by default).

Public Member Functions

- ObjectStorage (Initializer secureStorage)

  Initialized object storageusing the Initializer class, enabling encryption (encryption is enabled by default).
- string SaveObject (object obj, string key)

  This method is used to encrypt and securely save objects with their public properties. Only public properties will be saved via serializations, so it is important that the class has a parameterless constructor for deserialization.In case the class has only parameterized constructors, it will be necessary to add an empty parameterless constructor, otherwise the deserialization fails.
- object LoadObject (Type type, string key)

  This method is used to load a previously saved object.
- string[ ] GetAllKey (Type type)

  Get all the keys used to save a certain type of objects.
- object[ ] GetAllObjects (Type type)

  Get all the save objects by the type it was saved.
- void DeleteObject (Type type, string key)

  Delete the saved object.
- void DeleteAllObject (Type type)

  Delete all the object of certain type

4.3.1 Detailed Description

This class has been conceived to be able to save objects in a safe way, that is: The saved data of any application cannot be considered safe if it will be accessible in clear text to other applications or resident software. For encryption to be active, the library must be initialized using the Initializer class, enabling encryption (encryption is enabled by default).

4.3.2 Constructor & Destructor Documentation

4.3.2.1 ObjectStorage()

```
SecureStorage.ObjectStorage.ObjectStorage (
            Initializer secureStorage )
```

Initialized object storageusing the Initializer class, enabling encryption (encryption is enabled by default).

Parameters

secureStorage	storage name

4.3.3 Member Function Documentation

4.3.3.1 DeleteAllObject()

```
void SecureStorage.ObjectStorage.DeleteAllObject (
            Type type )
```

Delete all the object of certain type

Parameters

type	The type of the object

4.3.3.2 DeleteObject()

```
void SecureStorage.ObjectStorage.DeleteObject (
            Type type,
            string key )
```

Delete the saved object.

Parameters

type	The type of object
key	The key that was used to save the object

4.3.3.3 GetAllKey()

```
string[] SecureStorage.ObjectStorage.GetAllKey (
            Type type )
```

Get all the keys used to save a certain type of objects.

Parameters

type	The type of object whose keys you want to get

Returns

The keys used to save the object

4.3.3.4 GetAllObjects()

```
object[] SecureStorage.ObjectStorage.GetAllObjects (
            Type type )
```

Get all the save objects by the type it was saved.

Parameters

type	The type of object you want to get

Returns

The objects by type

4.3.3.5 LoadObject()

```
object SecureStorage.ObjectStorage.LoadObject (
            Type type,
            string key )
```

This method is used to load a previously saved object.

Parameters

type	The type of the object you want to load. Represents type declarations: class types, interface types, array types, value types, enumeration types, type parameters, generic type definitions, and open or closed constructed generic types
key	The key that was used to save the object

Returns

saved object

Exceptions

ArgumentException	

4.3.3.6 SaveObject()

```
string SecureStorage.ObjectStorage.SaveObject (
            object obj,
            string key )
```

This method is used to encrypt and securely save objects with their public properties. Only public properties will be saved via serializations, so it is important that the class has a parameterless constructor for deserialization.In case the class has only parameterized constructors, it will be necessary to add an empty parameterless constructor, otherwise the deserialization fails.

Parameters

obj	Object to save
key	Key used to save the object. This key will be used to upload the object in the future

Returns

The key used to save the object

Exceptions

ArgumentException	Object to save

The documentation for this class was generated from the following file:

- C:/documentation/CryptoMessenger/SecureStorageGit/SecureStorage/ObjectStorage.cs

4.4 SecureStorage.Values Class Reference

This class is used to save in an encrypted and secure way the values, which can be used by any application that needs to store parameters, such as configuration values, flags, names, and variables that you do not want to be lost after restarting of the application

Public Member Functions

- void Set (string name, bool value)

  Permanently save the value of a variable, for possible use after reloading the application
- bool Get (string name, bool defaultValue=default)

  Load a previously saved value of a variable, if it has not been previously saved then the returned value will be by default indicated in the parameter
- void **Set** (string name, string value)
- string Get (string name, string defaultValue=default)
- void **Set** (string name, int value)

  inheritdoc cref="Set(string, bool)"/>
- int **Get** (string name, int defaultValue=default)

  inheritdoc cref="Set(string, bool)"/>
- void **Set** (string name, uint value)
- uint Get (string name, uint defaultValue=default)
- void **Set** (string name, long value)
- long Get (string name, long defaultValue=default)
- void **Set** (string name, ulong value)
- ulong Get (string name, ulong defaultValue=default)
- void **Set** (string name, DateTime value)
- DateTime **Get** (string name, DateTime defaultValue=default)

4.4.1 Detailed Description

This class is used to save in an encrypted and secure way the values, which can be used by any application that needs to store parameters, such as configuration values, flags, names, and variables that you do not want to be lost after restarting of the application

4.4.2 Member Function Documentation

4.4.2.1 Get() [1/5]

```
bool SecureStorage.Values.Get (
            string name,
            bool defaultValue = default )
```

Load a previously saved value of a variable, if it has not been previously saved then the returned value will be by default indicated in the parameter

Parameters

name	The name assigned to the variable
defaultValue	This value will be returned if the variable has never been previously saved

Returns

The value of the previously saved variable, or the default value

4.4.2.2 Get() [2/5]

```
long SecureStorage.Values.Get (
            string name,
            long defaultValue = default )
```

inheritdoc cref="Set(string, bool)"/>

4.4.2.3 Get() [3/5]

```
string SecureStorage.Values.Get (
            string name,
            string defaultValue = default )
```

inheritdoc cref="Set(string, bool)"/>

4.4.2.4 Get() [4/5]

```
uint SecureStorage.Values.Get (
            string name,
            uint defaultValue = default )
```

inheritdoc cref="Set(string, bool)"/>

4.4.2.5 Get() [5/5]

```
ulong SecureStorage.Values.Get (
            string name,
            ulong defaultValue = default )
```

inheritdoc cref="Set(string, bool)"/>

4.4.2.6 Set()

```
void SecureStorage.Values.Set (
            string name,
            bool value )
```

Permanently save the value of a variable, for possible use after reloading the application

Parameters

name	Name to assign to the variable
value	The value of the variable to save

The documentation for this class was generated from the following file:

- C:/documentation/CryptoMessenger/SecureStorageGit/SecureStorage/Values.cs

Index

BinaryDeserialize
    SecureStorage.DataStorage, 10
BinarySerialize
    SecureStorage.DataStorage, 10

DataStorage
    SecureStorage.DataStorage, 9
DeleteAllObject
    SecureStorage.ObjectStorage, 14
DeleteObject
    SecureStorage.ObjectStorage, 14

Get
    SecureStorage.Values, 17, 18
GetAllKey
    SecureStorage.ObjectStorage, 14
GetAllObjects
    SecureStorage.ObjectStorage, 14

Initializer
    SecureStorage.Initializer, 12

LoadData
    SecureStorage.DataStorage, 10
LoadObject
    SecureStorage.ObjectStorage, 15

ObjectStorage
    SecureStorage.ObjectStorage, 13

SaveData
    SecureStorage.DataStorage, 11
SaveObject
    SecureStorage.ObjectStorage, 15
SecureStorage, 5
SecureStorage.DataStorage, 9
    BinaryDeserialize, 10
    BinarySerialize, 10
    DataStorage, 9
    LoadData, 10
    SaveData, 11
SecureStorage.Initializer, 11
    Initializer, 12
    SetKeyValue_Default, 12
SecureStorage.ObjectStorage, 13
    DeleteAllObject, 14
    DeleteObject, 14
    GetAllKey, 14
    GetAllObjects, 14
    LoadObject, 15
    ObjectStorage, 13
    SaveObject, 15
SecureStorage.Values, 16
    Get, 17, 18
    Set, 18
Set
    SecureStorage.Values, 18
SetKeyValue_Default
    SecureStorage.Initializer, 12