SafeNet Authentication Service Authentication API for Microsoft .Net

Developer Guide

gemalto
security to be free

Document Part Number: 007-012398-002, Rev. G

Release Date: 14 July 2016

Contents

Preface

Audience

This document is intended for personnel responsible for maintaining your organization's security infrastructure.

All products manufactured and distributed by SafeNet, Inc. are designed to be installed, operated, and maintained by personnel who have the knowledge, training, and qualifications required to safely perform the tasks assigned to them. The information, processes, and procedures contained in this document are intended for use by trained and qualified personnel only.

Related Documents

The following document contains related information:

- SafeNet Authentication Service Authentication API for Microsoft .Net 1.11 Customer Release Notes (PN: 007-013255-001)

Support Contacts

If you encounter a problem while installing, registering or operating this product, please make sure that you have read the documentation. If you cannot resolve the issue, contact your supplier or Gemalto Customer Support. Gemalto Customer Support operates 24 hours a day, 7 days a week. Your level of access to this service is governed by the support plan arrangements made between Gemalto and your organization. Please consult this support plan for further information about your entitlements, including the hours when telephone support is available to you.

Contact Method	Contact Information	
Address	Gemalto, Inc. 4690 Millennium Drive Belcamp, Maryland 21017, USA	
Phone	US	1-800-545-6608
	International	1-410-931-7520
Technical Support Customer Portal	https://serviceportal.safenet-inc.com Existing customers with a Technical Support Customer Portal account can log in to manage incidents, get the latest software upgrades, and access the Gemalto Knowledge Base.	

Introduction

This document describes the SafeNet Authentication Service (SAS) Authentication API for Microsoft .Net that allows agents to support all the functionality that is required to interact with SAS. An SAS agent is essentially a third-party application having embedded plug-in code that passes the collected user names and one-time passwords (OTPs) to SAS for authentication.

Applicability

The information in this document applies to the following:

- **SafeNet Authentication Service, Cloud Edition** - a cloud authentication service.
- **SafeNet Authentication Service, Service Provider Edition (SAS-SPE)** - the software used to build a SafeNet Authentication Service.
- **SafeNet Authentication Service, Private Cloud Edition (SAS-PCE)** - the term used to describe the implementation of SAS-SPE on-premises.

Security Notes

To enhance security, we strongly recommend the following actions.

Subject	Action	For More Details
Installation Folder	During installation, change the default destination folder to a system-protected folder accessible only by an account with management (read) privileges and by the account that initiates the API call.	See “Installation”, step 6 on page 8.
SSL Server Certificate Validation	Activate SSL server certificate validation.	See “SSL Server Certificate Validation when using API” on page 14.
Agent Key File	Place the Key File in a system-protected folder accessible only by an account with management (read) privileges and by the account that initiates the API call.	See “Load Key File” on page 19.

Prerequisites

Microsoft .Net Framework 2.0 or later.

Upgrade

Upgrade to SAS Authentication API for Microsoft .Net 1.11 is not supported.

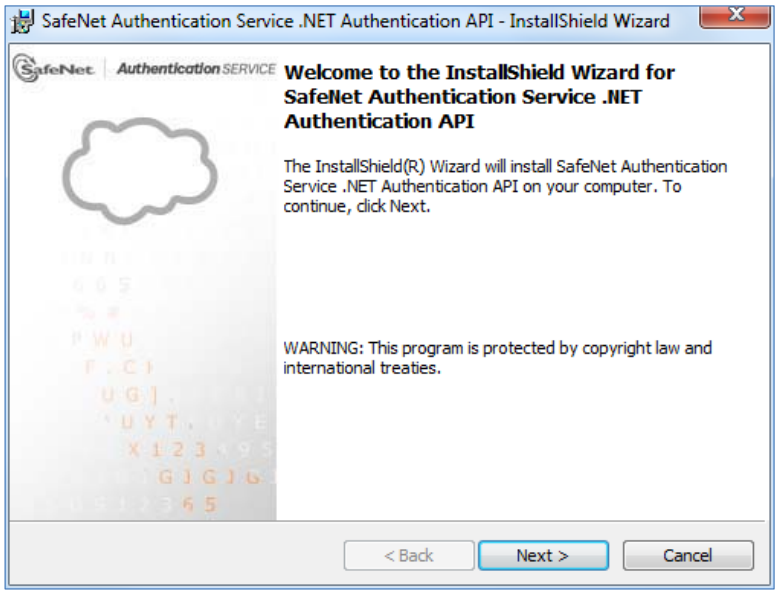
Installation

1. Run the required installation file:

   32-bit: BlackShield ID .NET Authentication API.exe

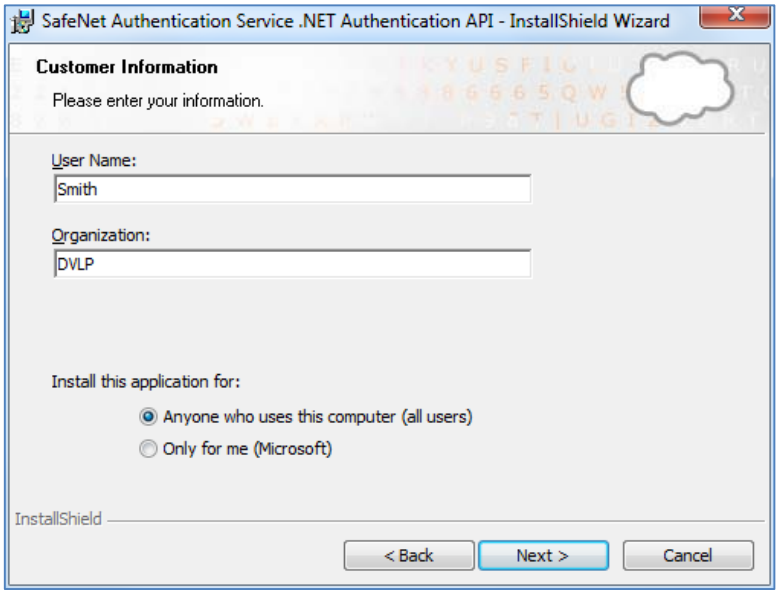
   64-bit: BlackShield ID .NET Authentication API x64.exe

2. Click **Next.**



3. Accept the terms in the license agreement and click **Next**.

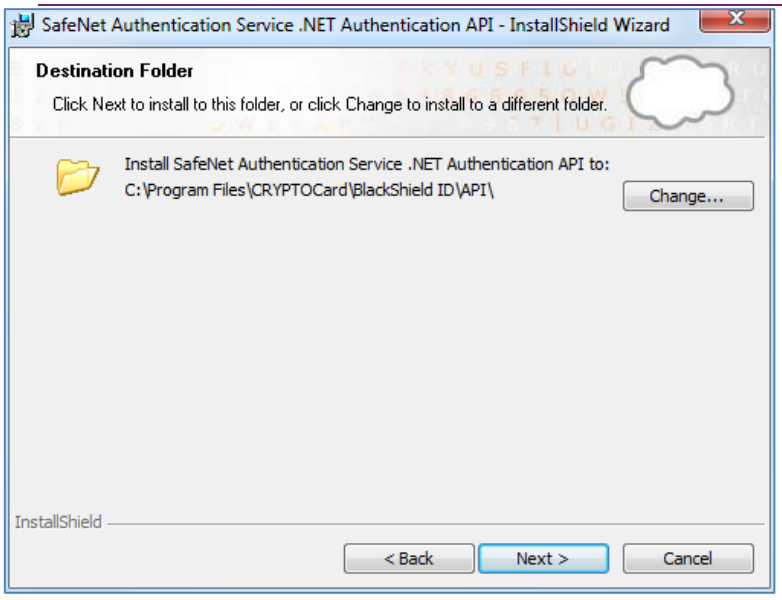
4. Enter **User Name** and **Organization**.
5. To determine who will have access to the application, select one of the following:
   - **Anyone who uses this computer (all users)**
   - **Only for me**



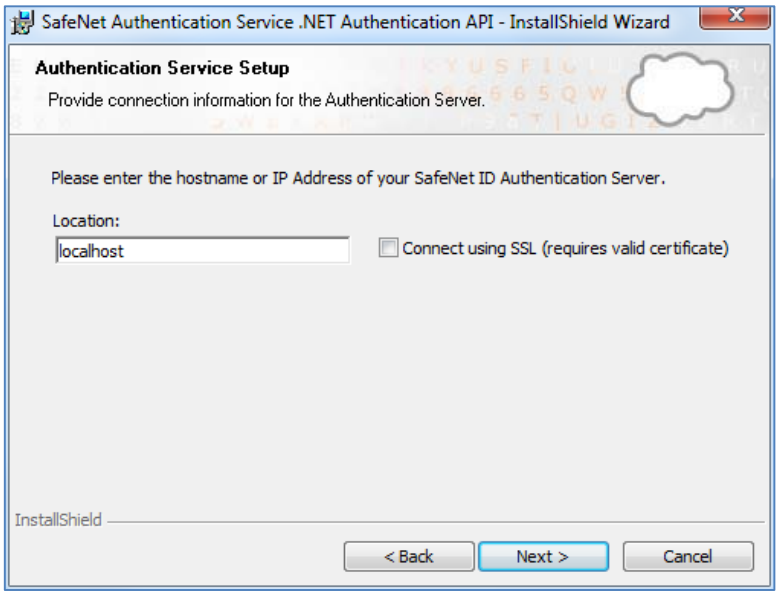
6. To change the default destination folder, click **Change** and navigate to the required location.

NOTE: We strongly recommend installing the application in a system protected folder accessible only by an account with management (read) privileges and by the account that initiates the API call.

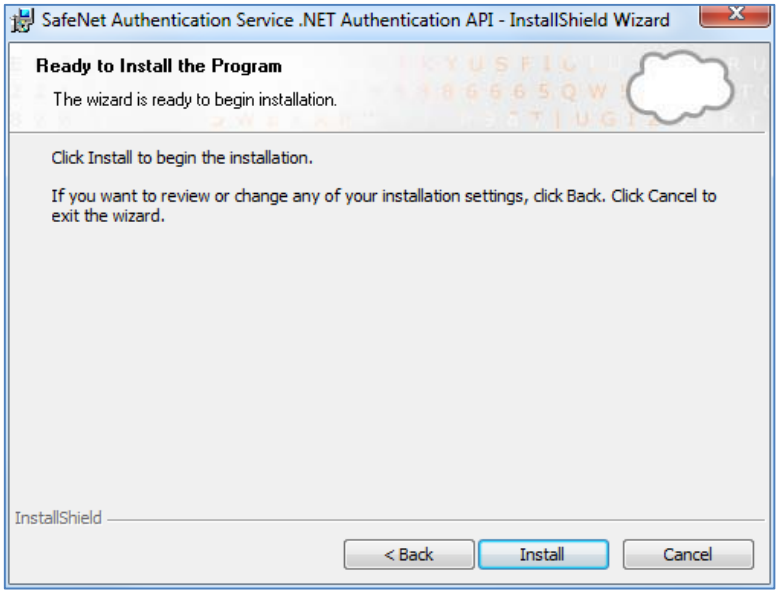
The default location is in the **Program Files** folder, which is not read protected. This means that the location must be changed to a protected location that cannot be read by non-administrative accounts.



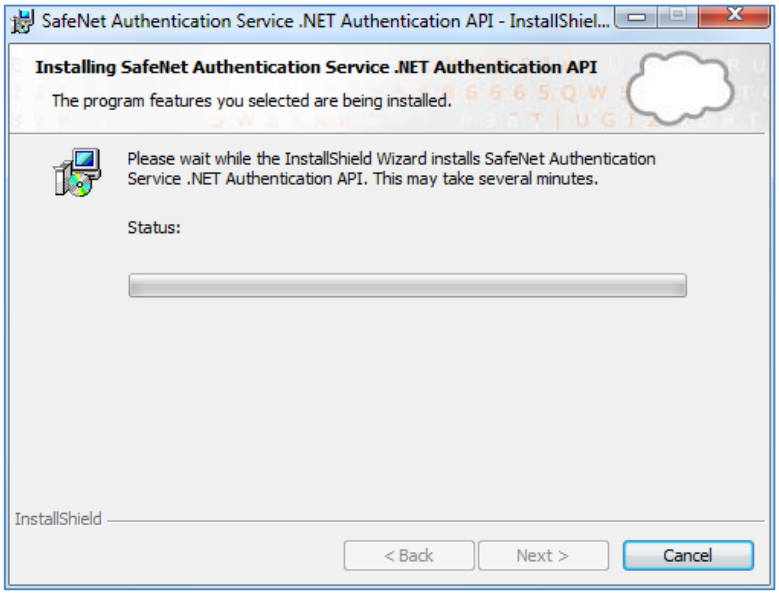
7. In the **Location** field, enter the hostname or IP address of the SafeNet Authentication Service server and click **Next**.



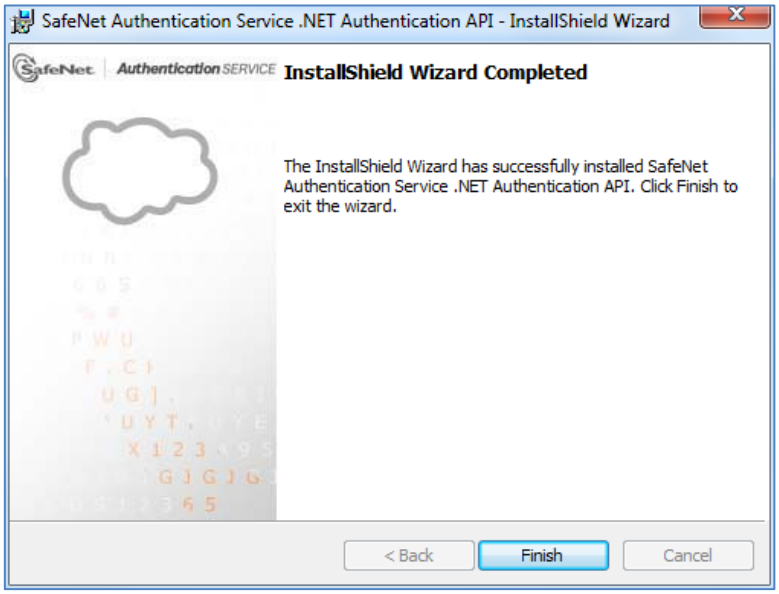
8. Click **Install** to begin the installation.



The installation proceeds.



9. Click **Finish** to exit the installation wizard.



SAS Authentication API C# Class

The SafeNet Authentication Service authentication API is represented by a single C# class **BSIDAPI**.

This class includes the following:

- A default constructor that loads its configuration information from a default Registry location
- An alternate constructor that allows the user to define an alternate location in the Registry from which to load its configuration

The API can be configured manually or through the Manager user interface.

Constructors

```
BSIDAPI()
{
// reads registry SOFTWARE\CRYPTOCard\BlackShield ID\BSIDAPI
}
BSIDAPI(string ConnectionInformation)
{
// reads registry key defined by ConnectionInformation
}
```

NOTE: By using the connection information in the constructor, an agent may specify one or more SAS servers in order to implement failover authentications. The connection information for each SAS server can be stored in its own key.

Authenticate Method

All actions related to authentication make use of a single API call:

```
Authenticate
(
   string user,
   string org,
   string ipaddress,
   string passcode,
   ref string challenge,
   ref string state
);
```

returns:

- int
  - 0 – Authentication Failed
  - 1 – Authentication Succeeded
  - 2 – Challenge
  - 3 – Server provided PIN
  - 4 – User needs to provide PIN
  - 5 – Authentication in outer window. Re-authenticate.
  - 6 – User must change their static password.
  - 7 – Static password change does not satisfy policies.
  - 8 - PIN provided doesn't meet requirements. Please provide a new PIN.

where:

- **user** –A string representing the user name of the individual who is authenticating.
- **org** – A string representing the organization to which the individual who is authenticating belongs. This should be passed as an empty string to represent the default organization.
- **ipaddress** – A string representing the IP address from which the authentication request originated. If this parameter is an empty string, SafeNet Authentication Service will attempt to detect it.
- **passcode** – A string representing the user's passcode, in one of the following formats:
  - **[PIN+OTP]** – For server-side PIN authentication
  - **[OTP]** – Token-side PIN or no PIN
  - **[PIN]** – When responding to a server-side change request for a user-changeable PIN
  - **[StaticPassword]** – User has a static password enabled or is responding to a static password change
  - **[null]** – Indicates that a challenge is required
- **challenge** – A string passed by reference that may be populated with a challenge/PIN change/outer window authentication message.
- **state** – A string passed by reference that contains a state attribute. When returning a challenge, the same state must be passed back to the server.

VerifySignature Method

Verifies the token's signature for a given hash.

```
VerifySignature
(
      string SerialNumber,
      string Hash,
      string Signature
);
```

returns:

- int
  - 0 – Signature is incorrect for the given hash.
  - 1 – Signature is correct for the given hash.

where:

- **SerialNumber** – A string representing the token's serial number.
- **Hash** – A string representing the hash value for the signature verification.
- **Signature** – A string representing the signature to be verified for the given hash.

CheckServerStatus Method

The operational status of the SAS authentication server can be monitored by using the **CheckServerStatus** method.

```
CheckServerStatus
(
      // void
);
```

returns:

- int
  - 0 – Server is down
  - 1 – Server is operational

NOTE: Use the **CheckServerStatus** method to monitor the health of SafeNet Authentication Service and to determine when to initiate failover to a secondary server.

SSL Server Certificate Validation when using API

NOTE: We strongly recommend using SSL server certificate validation.

The SSL server certificate validation is not activated by default.

Activate SSL using the ServerCertificateValidationCallback property, as in the following sample:

ServicePointManager.ServerCertificateValidationCallback = {delegate function}

For more details see:

https://msdn.microsoft.com/en-us/library/ms144153%28v=vs.110%29.aspx

https://msdn.microsoft.com/en-us/library/system.net.security.remotecertificatevalidationcallback%28v=vs.110%29.aspx

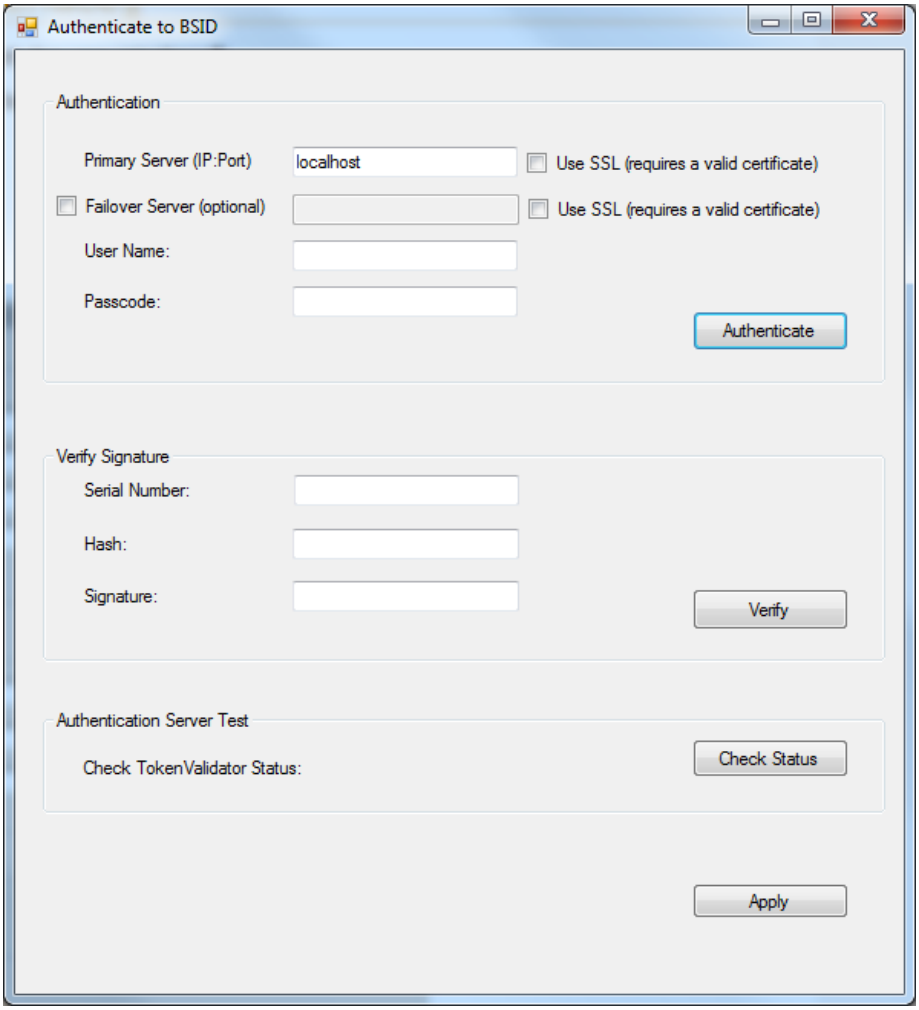
NOTE: The SSL setting will influence all SSL connections created by the application, not only SSL connections created directly by the API.

Manager User Interface

Verify authentication function with the user interface.

To open the Manger User Interface:

Run **CRYPTOCard\BlackShield ID\API\apiTestClient\bin\Release apiTestClient.exe**



Connecting to Authentication server

To connect to primary and failover server:

1. Enter the following fields:
   - Primary Server (IP:Port) - (select **Use SSL** if required)
   - Failover server (Optional) - (select **Use SSL** if required)
2. Click **Apply**.

   The **ServiceURL** and **OptionalSecondaryServiceURL** keys are updated under the following registry node:

   HKEY_LOCAL_MACHINE\SOFTWARE\CRYPTOCard\BlackShield ID\BSIDAPI

Verify Authentication

To verify Authentication:

1. Enter the following fields:
   - User Name
   - Password
2. Click Authenticate

   A message is displayed indicating if authentication succeeded or failed.

Verify Signature

To verify Signature :

1. Enter the following fields:
   - Serial Number
   - Hash
   - Signature
2. Click **Verify**

   A message is displayed indicating if Signature verification succeeded or failed.

Authentication Server Test

To test TokenValidator status:

Click **Check Status**.

A message is displayed indicating if the SAS server is running.

Use Case Scenarios

The SafeNet Authentication Service architecture supports the use of a token-side or a server-side PIN in either **QuickLog** or **Challenge-Response** mode. In addition, the application using the API must support challenges, inner/outer window authentication, and static password authentication. The following sections discuss these features in detail.

LOCALIZATION NOTE: To support localization, SafeNet Authentication Service returns only necessary data in its challenge messages. The application is required to construct a localized version of it to display to the client. For example, SafeNet Authentication Service would return only **19863257**, but the application would display, “Please respond to the challenge: 19863257”.

Basic Authentication

The communication between the application and the server uses challenge messages and states, similar to RADIUS protocol. The following scenario shows the most basic interaction between the application and server:

1. The authenticating application issues an authentication request that includes the user name, organization name, and a passcode.
2. The server responds with one of nine possible return codes. See “Authenticate Method” on page 11.

Challenge-Response Authentication

The challenge message and state attribute issued from the authenticating server are central to the concept of challenge-response authentication, outer window authentication, and server-side PIN changes. This mechanism is employed to authenticate tokens in challenge-response mode in the following manner:

1. The authenticating application issues an authentication request that includes the user name, organization name, and an empty passcode.
2. The server responds with a challenge message containing a challenge string.

   For example, “Challenge: 19863257”, and a state attribute.

   See the “LOCALIZATION NOTE” on page 16.
3. The authenticating application responds to the challenge by issuing another authentication request that includes the same user name, organization name, a response, and the state attribute.

Outer Window Authentication

User authentication through inner/outer window authentication uses challenge messages and state attributes, similar to the “Challenge-Response Authentication” scenario on page 17. In outer window authentication, users provide a match in a large look-ahead window, and respond to a follow-up challenge by providing the exact next OTP (one-time password) from their token. The follow sequence illustrates this process:

1. The authenticating application issues an authentication request that includes the user name, organization name, and a passcode.
2. The server finds a match for the provided OTP in the outer window, and then issues a challenge to the client containing an outer window authentication string, for example: “Please re-authenticate using the next OTP from your token”, and a state attribute.

   See “LOCALIZATION NOTE” on page 16.
3. The authenticating application responds to the challenge by issuing another authentication request that includes the same user name, organization name, a response, and the state attribute.

PIN Authentication

SafeNet Authentication Service supports several PIN types:

- No PIN
- Fixed PIN (token-side PIN validation)
- User-changeable PIN (token-side PIN validation)
- Fixed PIN stored on server
- User-changeable PIN stored on server
- Server-changeable PIN stored on server

SafeNet Authentication Service's authentication mechanism supports incoming passcodes in the following formats:

- [PIN+OTP]
- [OTP]
- [NEWPIN]
- [StaticPassword]
- [null] - empty passcode to request a challenge

PINs stored on the server can be user- or server-changeable. To accommodate this, leverage the challenge framework in the following manner:

User-Changeable PIN Stored on Server

1. The authenticating application issues an authentication request that includes the user name, organization name, and a passcode.
2. The server finds a match for the provided OTP and determines that the PIN must be changed.
3. The server issues a challenge to the client containing a PIN change string; for example: "Your PIN has expired. Please enter a new PIN" and a state attribute.

   See the "LOCALIZATION NOTE" on page 16.
4. The authenticating application responds to the challenge by returning a new PIN and the state attribute.

Server-Changeable PIN Stored on Server

1. The authenticating application issues an authentication request that includes the user name, organization name, and a passcode.
2. The server finds a match for the provided OTP and determines that the PIN must be changed.
3. The server issues a challenge to the client containing a PIN change string (for example, "Your new PIN is 628. Please re-authenticate using this new PIN and your next passcode") and a state attribute.

   See the "LOCALIZATION NOTE" on page 16.
4. The authenticating application responds to the challenge by issuing another authentication request that includes the user name, organization name, the new PIN and OTP, and the state attribute.

Static Password Authentication

SafeNet Authentication Service offers the option of static password authentication, including enabling the user to change the password. The challenge-response architecture can be used in the following manner:

1. The authenticating application issues an authentication request that includes the user name, organization name, and a static password.
2. If the user is not required to change the password and it is correct, the server returns access-accept.
3. If the user is required to change the password, a challenge message is issued to the client, for example: “Your password has expired. Please enter a new password” and a state attribute.

   See the “LOCALIZATION NOTE” on page 16.
4. If a challenge message has been issued in step 3, the authenticating application responds to the challenge by issuing an authentication request that includes the user name, organization name, the new static password, and the state attribute.

Agent Key Files

The SafeNet Authentication Service API uses an encrypted key file to secure communication with the server. The key file is loaded and registered with the agents, and a matching key is registered with the authentication server.

A sample key file (**default.bsidkey**) is installed for evaluation purposes. This file is publicly distributed, and should be used for evaluation purposes only. It is strongly recommended that you generate your own key file for your production environment.

To use an encrypted key file, load a key file, and register its certificate as instructed below.

Load Key File

1. Open the SafeNet Authentication Server **COMMS** tab and download an agent key file from the **Authentication Agent Settings** section.
2. Copy downloaded file to **[INSTALLDIR]\KeyFile**, where **[INSTALLDIR]** is the installation directory of this API.

NOTE: We strongly recommend placing the file in a system protected folder accessible only by an account with management (read) privileges and by the account that initiates the API call.

The default location is in the Program Files folder, which is not read protected. This means that the location must be changed to one that is a protected location that cannot be read by non-administrative accounts.

Register Key File Certificate

1. Open the Registry Editor, and expand the Registry to My Computer > HKEY_LOCAL_MACHINE > SOFTWARE > CRYPTOCARD > BLACKSHIELD ID > BSIDAPI.

> **NOTE:** If a custom Registry key location passes the connection information to the **BSIDAPI** constructor, expand the Registry to the custom Registry key instead, and update its **EncryptionKeyFile** information.

2. Edit the **EncryptionKeyFile** value so that it contains the fully qualified path to the agent key file that was loaded in “Load Key File” on page 19, and then save the changes.

Logging

If unexpected behavior occurs, the SAS API logs an error message to the Windows Event Viewer’s Application log. The source appears as **BSIDAPI**.

API Example

The **apiTestClient** directory includes an example of how to do the following:

- Add a reference to the **BSIDAPI.dll**.
- Use the **BSIDAPI** class to define connection information.
- Call the **Authenticate** and **CheckServerStatus** methods.

Deployment on Additional Computers

To deploy your completed application on an additional computer:

1. Copy the Registry key **My Computer > HKEY_LOCAL_MACHINE > SOFTWARE > CRYPTOCARD > BLACKSHIELD ID > BSIDAPI** from the computer running the completed application.

> **NOTE:** If a custom Registry key location passes the connection information to the BSIDAPI constructor, copy the custom Registry key instead.

2. Deploy and install an agent key file. See “Agent Key Files” on page 19.
3. For **BSIDAPI.dll** to run correctly, ensure that the following are installed:
   - NET Framework version 2.0
   - Microsoft Visual C++ 2008 Redistributable Package
   - CryptoCOM.dll

NOTE: Deploy **CryptoCOM.dll** to the **System32** folder.

Microsoft .Net API Sample

A Microsoft .Net API sample is supplied in one of the following .rar files:

- APIx32
- APIx64