CRYPTOCARD
experts in authentication

.Net Strong Authentication API

for

BLACKSHIELD SERVER   BLACKSHIELD CLOUD

Powerful Authentication Management for Service Providers and Enterprises

Authentication Service Delivery Made EASY™

Contact Information

CRYPTOCard's technical support specialists can provide assistance when planning and implementing CRYPTOCard in your network. In addition to aiding in the selection of the appropriate authentication products, CRYPTOCard can suggest deployment procedures that provide a smooth, simple transition from existing access control systems and a satisfying experience for network users. We can also help you leverage your existing network equipment and systems to maximize your return on investment.

CRYPTOCard works closely with channel partners to offer worldwide Technical Support services. If you purchased this product through a CRYPTOCard channel partner, please contact your partner directly for support needs.

To contact CRYPTOCard directly:

United Kingdom	**North America**
2430 The Quadrant, Aztec West, Almondsbury, Bristol, BS32 4AQ, U.K.	600-340 March Road, Kanata, Ontario, Canada K2K 2E4
Phone: +44 870 7077 700	Phone: +1 613 599 2441
Fax: +44 870 70770711	Fax: +1 613 599 2442
support@cryptocard.com	support@cryptocard.com

For information about obtaining a support contract, see our Support Web page at http://www.cryptocard.com

Authentication Service Delivery Platform Compatibility

BLACKSHIELD CLOUD BLACKSHIELD SERVER

Publication History

Date	Changes	Version
2009.01.26	Document created	1.0
2009.07.01	Copyright dates updated	1.1
2009.10.16	Minor updates	1.2

Overview

This document outlines the BlackShield ID authentication API that allows agents to support all the functionality that is required to interact with the BlackShield authentication server. BlackShield agents are essentially third party applications which have plug-in code embedded in order to allow the collection of user names and one time passwords to be passed to the BlackShield server to be verified.

Specification Overview

BlackShield ID Authentication API

The BlackShield ID authentication API is represented by a single C# class `BSIDAPI`. This class has a default constructor which loads its configuration information from a default registry location, and also an alternate constructor to allow the user to define an alternate location in the registry from which to load its configuration.

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 BlackShield servers in order to implement failover authentications. The connection information for each BlackShield server can be stored in its own key.

Authenticate Method

All authentication related actions 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 currently should be passed as an empty string to represent the default organization.
- ipaddress – a string representing the IP address from which the authentication request came. If this parameter is an empty string, the BlackShield server will attempt to detect the IP from which the authentication request came from.
- passcode – a string representing the user's passcode. This may take form of either:
    - [PIN+OTP] – for server-side PIN authentication.
    - [OTP] – token side PINs or no PIN.
    - [PIN] – when responding to a server-side user changeable PIN change request.
    - [StaticPassword] – user has a static password enabled or is respondng to a static password change.
    - Finally, this parameter may also be set to null to indicate 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 may be populated with a state attribute. When returning a challenge, the same state should 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 hash provided.
    - 1 – Signature is correct for hash provided.

where:

- SerialNumber – a string representing the token's serial number.
- Hash – a string representing the hash value to verify.
- Signature – a string representing the signature to verify for the provided hash.

CheckServerStatus Method

The operational status of the BlackShield authentication server can be monitored through the use of the CheckServerStatus method.

```
CheckServerStatus

(

      // void

);
```

returns:

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

Note: The CheckServerStatus method is useful to monitor the health of a BlackShield server for the purpose of initiating failover to a secondary server.

Use Case Scenarios

The BlackShield ID architecture supports the concept of a token-side and server-side PIN used in either QuickLog or Challenge/Response mode. In addition, agents must support challenges, inner/outer window authentication, and static password authentication. The following sections discuss these features in more detail.

Basic Authentication

The communication between a BlackShield agent and server is somewhat similar to that used in the RADIUS protocol, in that the concepts of challenge messages and states are used. The following scenario shows the most basic interaction between the agent and server:

1. Authenticating agent issues an authentication request with username, organization name, and a passcode.
2. Server responds with one of nine possible return codes as outlined in Authenticate Method on page 6.

Challenge-Response

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

1. Authenticating agent issues an authentication request with username, organization name, and a blank passcode.
2. Server responds with a challenge message containing a challenge string i.e.: "c*hallenge: 19863257"*, and a state attribute.
3. Authenticating agent responds to the challenge by issuing another authentication request with their username, organization name, a response, and the state attribute.

NOTE: In order to support localization, the BlackShield server returns only necessary data in its challenge messages, and the agent is required to construct a localized version of it to display to the client. For example, the BlackShield server would return only "19863257", and the BlackShield agent would display "Please respond to the challenge: 19863257".

Outer Window Authentication

Also harnessing the concept of challenge messages and maintaining state, is the authenticating a user through inner/outer window authentication. Outer window authentication allows a user to authenticate by providing a match in a large look ahead window; however, they must also be able to respond to a follow up challenge which asks them to provide the exact next OTP from their token. The follow sequence illustrates this functionality:

1. Authenticating agent issues an authentication request with username, organization name, and a passcode.
2. The server finds a match for the provided OTP in the outer window, therefore it issues a challenge to the client containing an outer window authentication string, i.e.: *"Please re-authenticate using the next OTP from your token",* and a state attribute.
3. Authenticating agent responds to the challenge by issuing another authentication request with their username, organization name, a response, and the state attribute.

NOTE: Please see the localization note in Challenge-Response on page 9.

PIN Styles

A number of PIN styles in BlackShield ID are supported:

- No PIN.
- Fixed PIN. (token side PIN validation)
- User-changeable PINs. (token side PIN validation)
- Stored on server, fixed PIN.
- Stored on server, user-changeable PIN.
- Stored on server, server-changeable PIN.

The authentication mechanism in BlackShield supports incoming passcodes in the following format [PIN+OTP], [OTP], [NEWPIN], [STATICPASSWORD] and an empty passcode to request a challenge.

In the case of PINs which are stored on the server, yet are user or server changeable, the challenge framework needs to be leveraged again in the following manner:

Stored on server, user-changeable

1. Authenticating agent issues an authentication request with username, organization name, and a passcode.
2. The server finds a match for the provided OTP however it determines that it is necessary for the user to change their PIN, therefore it issues a challenge to the client containing a PIN change string, i.e.: *"Your PIN has expired. Please enter a new PIN:"* and a state attribute.
3. The authenticating agent responds to the challenge by returning the new PIN, and the state attribute.

NOTE: Please see the localization note in section 3.2.2.

Stored on server, server-changeable

1. Authenticating agent issues an authentication request with username, organization name, and a passcode.
2. The server finds a match for the provided OTP however it determines that it is necessary for the user to change their PIN, therefore it issues a challenge to the client containing a PIN change string, i.e.: *"Your new PIN is 628. Please re-authenticate using this new PIN and your next passcode"* and a state attribute.
3. Authenticating agent responds to the challenge by issuing another authentication request with their username, organization name, the new PIN and OTP, and the state attribute.

NOTE: Please see the localization note in Challenge-Response on page 9.

Static Password Authentication

BlackShield ID also offers the option of static password authentication, with the ability for the user to change their password. The challenge response architecture can be used to facilitate this functionality, in the following manner:

1. Authenticating agent issues an authentication request with username, organization name, and a static passcode.
2. If the user is not required to change their static password and it is correct, the server returns access-accept. If the user is required to change their static password, a challenge message will be issued to client, i.e.: *"Your password has expired. Please enter a new password:"* and a state attribute.
3. The authenticating agent responds to the challenge by issuing an access-request message with their username, organization, the new static password, and the state attribute.

NOTE: Please see the localization note in Challenge-Response on page 9.

3.3 Agent Key Files

The BlackShield ID API uses an encrypted key file to secure communication with the server. To accomplish this, a key file is loaded and registered with agents, and then a matching key is registered with the authentication server.

A sample key file (default.bsidkey) has been installed for evaluation purposes; however, it is strongly recommended that you generate your own key file for a production environment, as the sample file is publicly distributed. The section below will describe the steps required to use your own key file.

Load Key File

1. Download an agent key file in the Agent Settings section of the System Tab.
2. Using Windows Explorer change your current working directory to the ***KeyFile*** directory by typing ***"[INSTALLDIR]\KeyFile\"*** in the address bar where ***[INSTALLDIR]*** represents the install directory of this API.
3. Copy and paste the agent key file obtained from step number one above.

Register Key File

1. It is necessary to register the certificate that has been loaded above. To register, first open up a command window (Start->Run) and type ***regedit***, followed by clicking on ***OK***.
2. Expand ***My Computer***, ***HKEY_LOCAL_MACHINE***, ***SOFTWARE***, ***CRYPTOCARD***, ***BLACKSHIELD ID, BSIDAPI***[1]
3. Double click **EncryptionKeyFile**
4. In the text box, enter the fully qualified path to the agent key file that was loaded above. Then click ***OK***.

[1] In the event that you are using a custom registry key location as passed to the BSIDAPI constructor, you should update key file information of that registry key instead.

Logging

If an unexpected behavior occurs, the BlackShield ID API will log an error message to the Windows Event Viewer's Application log. The source will appear as **BSIDAPI**.

API Example

Please see the apiTestClient directory for an example of how to add a reference to the BSIDAPI.dll, and use the BSIDAPI class to define connection information, and call the Authenticate and CheckServerStatus methods.

Deployment

Should you wish to deploy your completed application to another computer, you should keep in mind the following configuration needed to support the BlackShield ID API:

1. Deploy the registry keys as defined by:
   `SOFTWARE\CRYPTOCard\BlackShield ID\BSIDAPI`

   Or if you are using custom registry key(s) to define connection information as passed to the BSIDAPI constructor, you should deploy those registry key(s).

2. An agent key file should also be deployed and installed using the instructions in 3.3 Agent Key Files on page 12.

3. Finally, BSIDAPI.dll depends on the .NET Framework version 2.0, Microsoft Visual C++ 2008 Redistributable Package and CryptoCOM.dll. For best results, deploy CryptoCOM.dll to the System32 folder.