TRIDENT™
TRusted IDENTity

TRIDENT CARD ID Integration

9 June, 2026

Table of Contents

1 CARD ID Introduction

TRIDENT Card ID Integration is built on top of the TRIDENT Middleware component, which enables secure bidirectional communication between web applications and local devices connected to a user's workstation, such as smart card readers and electronic identity cards.

The primary purpose of TRIDENT Middleware is to provide a browser-independent integration layer, allowing web applications to interact with local hardware devices without relying on browser-specific technologies.

Historically, web applications used technologies such as Java Applets and ActiveX controls to access local devices. However, modern browser security restrictions have significantly limited or completely removed support for these technologies, making them unsuitable for current web environments.

To address these limitations, TRIDENT Middleware implements a local client-server architecture in which both the client and the server run on the same workstation. The web application communicates with the local TRIDENT Middleware service through Secure WebSockets (WSS), providing a secure and standardized communication channel between the browser and local hardware resources.

This architecture enables web applications to securely access smart card functionality while maintaining compatibility across supported browsers and operating systems. It also provides a consistent integration model that abstracts the complexities of communicating directly with smart card readers and identification documents.

For the TRIDENT Card ID solution, TRIDENT Middleware acts as the bridge between the web application and the Barbados electronic identity card, allowing applications to retrieve card information, read cardholder data, obtain card images, and validate the authenticity and integrity of responses through digitally signed challenges.

2 CARD ID Prerequisites

Before integrating with the TRIDENT Card ID solution, it is necessary to ensure that the workstation environment meets the requirements described below.

2.1 Browser Compatibility

As described in the previous section, TRIDENT Middleware uses Secure WebSockets (WSS) to establish communication between the web application and the local TRIDENT Middleware service. Secure WebSockets require browser support for encrypted WebSocket communications and trusted TLS certificates.

The following browser versions are supported:

Browser	Minimum Supported Version
Internet Explorer	11
Microsoft Edge	14
Google Chrome	49
Mozilla Firefox	51

Any browser used for integration must support the Secure WebSocket specification (WSS).

2.2 Certificate Installation

The TRIDENT Middleware installer automatically installs the required certificates in the appropriate certificate stores:

- **Windows Certificate Store**, used by Internet Explorer, Microsoft Edge, and Google Chrome.
- **Mozilla Firefox Certificate Store**.

This automated process is sufficient for most installations and requires no additional configuration.

2.3 Network and Security Considerations

The communication channel between the browser and TRIDENT Middleware is established locally on the workstation using Secure WebSockets. No inbound network connectivity is required from external systems to access the local service.

Organizations should ensure that:

- Local firewall policies allow communication between the browser and the local TRIDENT Middleware service.

- Endpoint protection solutions do not block Secure WebSocket communications.
- Certificate trust stores are properly managed and maintained.

Meeting these prerequisites ensures a secure and reliable integration experience with the TRIDENT Card ID solution.

3 CARD ID Integration Architecture and JavaScript API

3.1 Overview

TRIDENT Middleware implements a local client-server architecture that enables secure communication between web applications and local hardware devices, such as smart card readers and electronic identity cards.

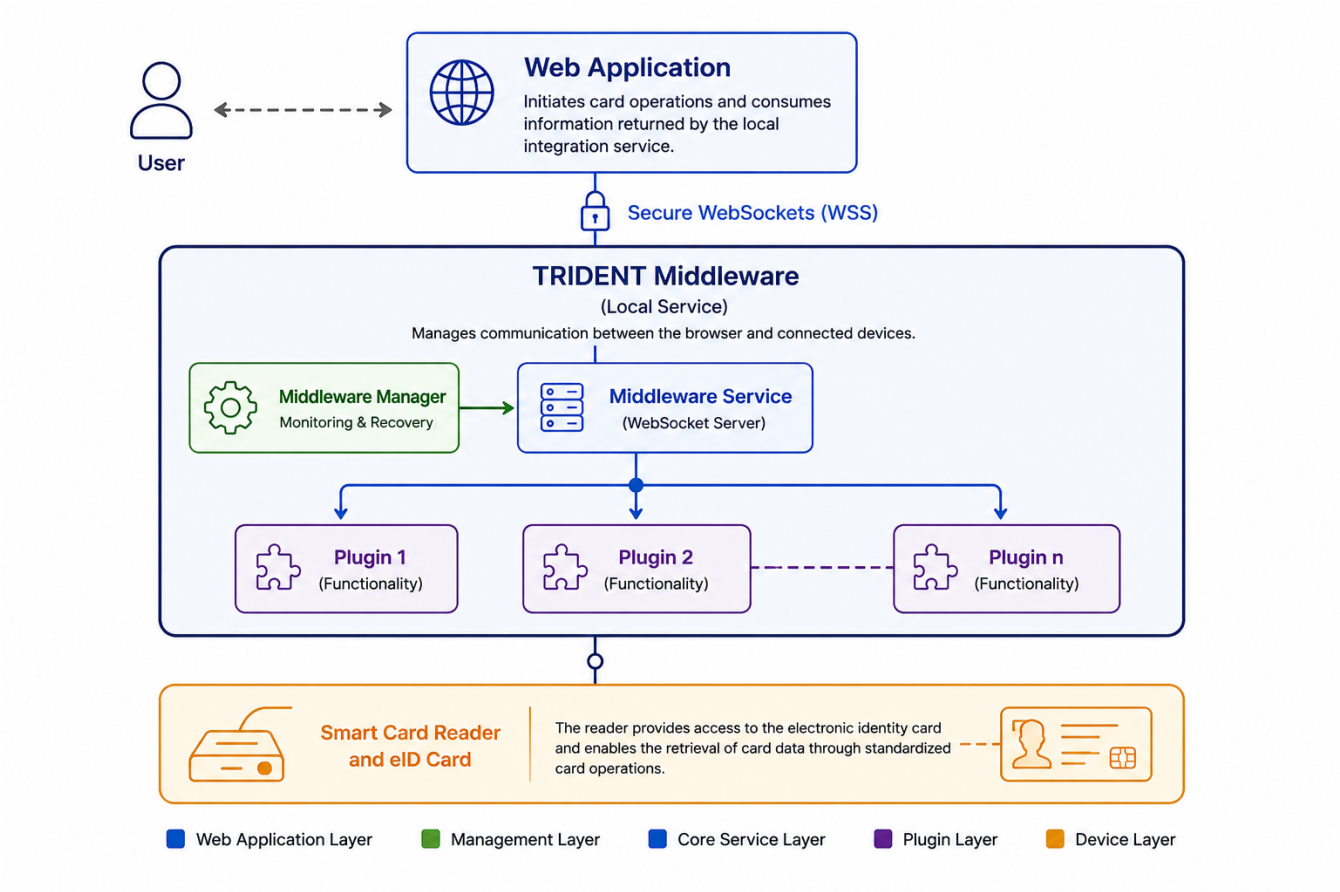
Unlike traditional server-based architectures, both the client and server components of TRIDENT Middleware run on the same workstation. Communication between the browser and the local service is established through Secure WebSockets (WSS), providing a secure and browser-independent integration mechanism.

The architecture is designed to abstract the complexity of interacting with local devices while providing a consistent programming model for web applications.

The main components of the solution are:

- Web Application
- TRIDENT Middleware JavaScript Library
- Local TRIDENT Middleware Service
- Plugin Framework
- Smart Card Reader
- Electronic Identity Card

The following diagram illustrates the high-level architecture of the solution.



3.2 Server Component

The TRIDENT Middleware service runs continuously in the background, waiting for requests from web applications.

When the service starts, it attempts to bind to one of the configured communication ports. Multiple ports are supported to reduce the likelihood of conflicts with other applications running on the workstation.

To ensure high availability, TRIDENT Middleware includes an automatic monitoring process called **HubManager**. If the local service stops unexpectedly, HubManager automatically restarts it.

If HubManager is not running, the service can also be started directly from the browser using the operating system protocol handler:

```
iscert://
```

The server is composed of a collection of modules, referred to as **plugins**, each responsible for implementing a specific set of business or device-related operations.

Every request received through the Secure WebSocket interface is routed to the corresponding plugin for processing.

3.3 Request Processing Model

Requests exchanged between the browser and TRIDENT Middleware are represented as JSON structures.

Each request contains:

- A module identifier, which determines the plugin that will process the request.
- A parameters object containing the input values required by the plugin.

Conceptually, requests follow the structure:

```
{
  "module": "<plugin-name>",
  "parameters": {
    ...
  }
}
```

The exact structure of the parameters object depends on the plugin and operation being invoked.

3.4 Client Component

Web applications communicate with TRIDENT Middleware through a JavaScript library called TRIDENT Middleware.js.

This library is provided as part of the integration package and abstracts all communication with the local service.

To use the library, include it in the web application using a standard HTML script tag:

```
<script type="text/javascript" src="js/tridentHub.js"></script>
```

Once loaded, the library exposes a set of methods for:

- Launching the local service
- Executing plugin operations
- Handling responses
- Managing connection failures
- Building requests programmatically

3.5 JavaScript API

3.5.1 Launching the Service

The following method starts HubManager and initializes the local service if it is not already running:

```
hub.launch();
```

If an instance of HubManager is already active, no additional action is performed.

3.5.2 Executing Plugin Operations

Plugin operations can be invoked using the `hub.call()` method.

```
hub.call(module, params, handler, port)
```

3.5.2.1 Parameters

Parameter	Type	Description
module	String	Plugin identifier to execute
params	JSON	Input parameters passed to the plugin
handler	Function	Callback function that receives the response
port	Array[int]	List of ports used to establish the WebSocket connection

3.5.2.2 Example

```
hub.call(
    "sign-p7-win",
    {
        content: "Message for Bob, signed by Alice"
    },
    function(response) {
        if(response.code === 0) {
            alert(response.result);
        }
    },
    [4322, 4123, 4567]
);
```

3.5.3 Request Builder API

As an alternative to directly invoking `hub.call()`, developers can use the Request Builder API.

The builder provides a fluent interface for constructing requests.

```
var builder = hub.caller(module);
```

3.5.3.1 Supported Operations

3.5.3.1.1 param()

Adds a single parameter to the request.

```
builder.param(key, value);
```

3.5.3.1.2 value()

Sets the entire parameter object.

```
builder.value(jsonObject);
```

3.5.3.1.3 port()

Defines the list of ports used for communication.

```
builder.port([4322, 4123, 4567]);
```

3.5.3.1.4 call()

Executes the request.

```
builder.call(handler);
```

3.5.3.1.5 onFail()

Registers a callback invoked if repeated connection attempts fail.

```
builder.onFail(function() {
    // Handle failure
});
```

3.5.3.2 Builder Example

```
var args = JSON.parse($("#txtName").val());

hub.caller(module)
    .value(args)
    .onFail(function() {
        console.log("Unable to connect to TRIDENT Middleware");
    })
    .call(function(response) {
        console.log(response);
    });
```

The Request Builder API is recommended for complex integrations where requests are dynamically constructed or additional error handling is required.

4 CARD ID Response Signing and Validation

4.1 Overview

To ensure the authenticity and integrity of responses returned by TRIDENT Middleware, the platform includes a response signing mechanism based on cryptographic challenges and digital signatures.

This functionality allows integrated applications to verify that responses were generated by a trusted instance of TRIDENT Middleware and that the returned data has not been modified during transmission.

Each response may include a digitally signed challenge that can be independently validated by the consuming application.

4.2 Objectives

The response signing mechanism has the following objectives:

4.2.1 Authenticity

Provide a mechanism that enables applications to verify that responses originate from a legitimate TRIDENT Middleware instance.

4.2.2 Integrity

Ensure that response data has not been altered after being generated by TRIDENT Middleware.

4.2.3 Security

Use a dedicated signing certificate to guarantee that only responses generated by an authorized TRIDENT Middleware installation are accepted by integrated applications.

4.3 Technical Description

4.3.1 Protected Configuration

The signing process relies on cryptographic material stored in a protected keystore managed by TRIDENT Middleware. Configuration properties associated with the signing process may be encrypted and decrypted internally by the component as required.

4.3.2 Challenge Generation

Whenever TRIDENT Middleware receives a request, a challenge may be generated using one of two mechanisms.

4.3.2.1 Plugin-Defined Challenge

If the request includes the parameter `challenge_parameters`, the plugin may generate a custom challenge to be signed.

To support this functionality, the plugin must implement the `IChallenge` interface defined in the Plugin Definition library.

The structure and content of the generated challenge depend on the plugin implementation. Refer to the corresponding **"TRIDENT Middleware Response Signing"** section for the specific plugin being used.

If `challenge_parameters` are provided but the selected plugin does not implement the `IChallenge` interface, a random challenge will be generated automatically.

4.3.2.2 Default Challenge Generation

If no `challenge_parameters` are provided, TRIDENT Middleware generates a default challenge.

The default challenge is constructed as follows:

1. Generate a random 32-byte value using `SecureRandom`.
2. Encode the random value using Base64.
3. Generate a timestamp using the format:

```
yyyy-MM-dd'T'HH:mm:ss
```

4. Concatenate the timestamp and the Base64-encoded random value.
5. Convert the resulting string to UTF-8 bytes.
6. Encode the final byte array using Base64.

This process guarantees that every challenge is unique, unpredictable, and suitable for authentication and verification purposes.

4.4 Challenge Signing

The generated challenge is digitally signed using an internal signing certificate managed by TRIDENT Middleware.

The following signature algorithm is used:

```
SHA256withRSA
```

Signature format:

```
PKCS#1
```

The signing certificate and private key are loaded from the keystore managed by TRIDENT Middleware.

4.5 Response Structure

After a plugin generates its response, the TRIDENT Middleware signing service enriches the JSON response by adding a new object called `signatureInfo`.

This object contains all information required to validate the authenticity of the response.

Example:

```
{
  ...
  "signatureInfo": {
      "signature": "Ar9kOANb44syFSFmOTPv.....",
      "certificate": "MIIERTCCAi2gAwIBAgIBKjANBg.....",
      "challenge": "MjAyNC0xMS0xOVQxMDo1MToyM2RwMn....",
      "algorithm": "SHA256withRSA",
      "timestamp": "2024-11-19 10:51:23.984"
  }
  ...
}
```

4.5.1 signatureInfo Fields

Field	Description
signature	Base64-encoded digital signature
certificate	Base64-encoded X.509 signing certificate
challenge	Challenge value that was signed
algorithm	Signature algorithm used
timestamp	Signature generation timestamp

4.6 Signature Validation

Integrated applications should validate the response signature before trusting any returned data.

The validation process consists of:

1. Extract the `certificate` value from the response.
2. Decode the Base64 certificate.
3. Obtain the public key from the X.509 certificate.
4. Decode the Base64 signature.
5. Verify the signature against the returned challenge.
6. Accept the response only if signature verification succeeds.

The public key used for verification is obtained directly from the `certificate` field included in the response.

4.6.1 Java Validation Example

```
public static boolean validateSignature(
        String challenge,
        String signatureBase64,
        String certificateBase64,
        String algorithm)
        throws NoSuchAlgorithmException,
               InvalidKeyException,
               SignatureException,
               CertificateException,
               UnsupportedEncodingException {

    byte[] decodedCert = Base64.getDecoder().decode(certificateBase64);

    CertificateFactory certFactory =
            CertificateFactory.getInstance("X.509");

    X509Certificate certificate =
            (X509Certificate) certFactory.generateCertificate(
                    new ByteArrayInputStream(decodedCert));

    PublicKey publicKey = certificate.getPublicKey();

    byte[] signatureBytes =
            Base64.getDecoder().decode(signatureBase64);

    byte[] messageBytes =
            challenge.getBytes("UTF-8");

    Signature signature =
            Signature.getInstance(algorithm);

    signature.initVerify(publicKey);
```

```
    signature.update(messageBytes);

    return signature.verify(signatureBytes);
}
```

If the verification succeeds, the application can trust that:

- The response was generated by TRIDENT Middleware.
- The response contents were not modified after signing.
- The signing certificate was successfully validated.

4.7 Error Handling

The following error codes may be returned during challenge generation, signing, or validation operations.

Error Code	Description
-1	Invalid request or invocation parameters
1000	Unexpected processing error
1001	Keystore loading error
1002	Digital signature processing error
1003	JSON processing error
1004	Invalid challenge parameters
1005	Error obtaining challenge data
1006	Signing certificate not found

4.7.1 Error Code 1001 – Keystore Loading Error

Possible causes:

- Corrupted keystore file
- Incorrect keystore password
- File permission issues
- Unsupported cryptographic algorithm
- Invalid certificates stored in the keystore

4.7.2 Error Code 1002 – Digital Signature Error

Possible causes:

- Private key cannot be recovered
- Invalid signing key
- Incorrect data format
- Unsupported signature algorithm
- Security provider issues

4.7.3 Error Code 1003 – JSON Processing Error

Possible causes:

- Malformed JSON
- Serialization or deserialization errors
- Incompatible data types
- Memory limitations when processing large JSON documents

4.7.4 Error Code 1004 – Invalid Challenge Parameters

Possible causes:

- Unsupported values in `challenge_parameters`
- Malformed `challenge_parameters` array

4.7.5 Error Code 1005 – Challenge Data Retrieval Error

Possible causes:

- Failure while obtaining challenge-related data from a device or reader

4.7.6 Error Code 1006 – Certificate Not Found

Possible causes:

- No certificate associated with the configured signing key was found in the keystore

4.8 Updating the Response Signing Certificate

4.8.1 Overview

The response signing process requires access to a private key stored in a keystore. This private key is associated with a public key and an X.509 certificate used to sign response challenges.

To simplify certificate replacement and key rotation procedures, TRIDENT Middleware provides the **KeystoreUpdater** utility.

4.8.2 KeystoreUpdater Directory Structure

The `KeystoreUpdater` directory contains:

Item	Description
artifacts	Supporting binaries and utilities
ChallengeKeyStore.jks	Signing keystore
challenge-keystore.properties	Configuration properties
UpdaterHubIsc.bat	Update script
Documentation	Update instructions

4.8.3 Update Procedure

4.8.3.1 Step 1 – Replace the Keystore

Replace the existing `ChallengeKeyStore.jks` file with the new keystore.

Ensure that:

- The new keystore file is named `ChallengeKeyStore.jks`
- The keystore contains the appropriate private key
- The corresponding signing certificate is present

4.8.3.2 Step 2 – Update Configuration Properties

Open:

```
challenge-keystore.properties
```

Update the configuration values according to the new keystore settings.

Save the file after making the required changes.

4.8.3.3 Step 3 – Execute the Update Script

Run:

```
UpdaterHubIsc.bat
```

The script can be executed by double-clicking the file or from the command line.

Wait for the process to complete successfully.

4.8.4 Important Notes

- Ensure you have sufficient permissions to modify files in the installation directory.
- Always create a backup of the original files before starting the update process.
- If any issues occur, review the generated logs for additional diagnostic information.
- After completing the update, validate the response signing functionality before returning the system to production use.

5 CARD ID Barbados eID Card Operations

5.1 Overview

The NID Reader plugin provides the functionality required to communicate with the Barbados Electronic Identity Card (eID) through supported smart card readers.

The plugin encapsulates all APDU command exchanges required to interact with the card and retrieve data stored within the chip. The implementation follows the technical specifications defined by the Government of Barbados electronic identity card profile.

Throughout this document, the terms card, electronic identity card, and Barbados eID are used interchangeably to refer to the Barbados government-issued electronic identity document.

5.1.1 Module Information

Property	Value
Module Name	`nid-reader`
JAR File	`plugin.nid-reader-trident.isc`
Service Class	`uy.isaportal.iscert.NidReaderPlugin`

5.2 Supported Operations

All operations require a smart card reader connected to the workstation.

Certain operations also require the user to enter the card PIN before protected data can be accessed.

The plugin supports the following operations:

Operation	Description
CHIP_ID	Retrieve the chip identifier
PERSONAL_INFO	Retrieve personal information
IMAGE	Retrieve the cardholder image
ALL	Retrieve all available card data

WAIT_FOR_INSERT	Wait for card insertion
DETECTED_DEVICES	Detect connected smart card readers

5.3 CHIP_ID Operation

5.3.1 Description

Returns the unique chip identifier of the inserted Barbados eID card.

5.3.1.1 Input Parameters

Parameter	Type	Required	Description
operation	String	Yes	Must be set to `CHIP_ID`

5.3.1.2 Response Parameters

Field	Type	Description
code	Integer	Response status code
message	String	Response message
success	Boolean	Operation result
operation	String	Executed operation
data.chipId	String	Card chip identifier
signatureInfo	JSON	Signed challenge information

5.3.1.3 Example Request

```
{
  "operation": "CHIP_ID"
}
```

5.3.1.4 Successful Response

Returns the card chip identifier together with the signed response metadata.

5.4 PERSONAL_INFO Operation

5.4.1 Description

Retrieves personal information stored on the Barbados eID card.

The information is obtained by processing and mapping values stored in:

- EF21
- DG1 (MRZ)

The user must enter the card PIN before the information can be accessed.

If the PIN is incorrect, an error response is returned together with the number of remaining attempts before the card becomes blocked.

5.4.1.1 Input Parameters

Parameter	Type	Required	Description
operation	String	Yes	Must be set to `PERSONAL_INFO`

5.4.1.2 Returned Personal Data

Field	Source
chipId	Card Chip Identifier
docNumber	EF21 - DOCUMENT_NUMBER
nationalRegistrationNumber	EF21 - NATIONAL_REGISTRATION_NUMBER
givenName	EF21 - GIVEN_NAMES
lastName	EF21 - LAST_NAME
issuedDate	EF21 - DATE_OF_ISSUE
expirationDate	EF21 - DATE_OF_EXPIRY

issuingCountry	EF21 - NATIONALITY
address2	EF21 - ADDRESS2
phone	EF21 - PHONE_HOME
cellphone	EF21 - PHONE_CELL
email	EF21 - EMAIL
sex	EF21 - SEX
dateOfBirth	DG1 MRZ - DATE_OF_BIRTH

5.4.1.3 PIN Validation

When an invalid PIN is entered:

```
{
  "code": 314,
  "message": "Wrong pin, attempts left: 2",
  "success": false,
  "data": {
    "attemptsLeft": 2
  }
}
```

5.5 IMAGE Operation

5.5.1 Description

Retrieves the facial image stored on the Barbados eID card.

The image is obtained by processing the contents of DG2 and is returned as a Base64-encoded string.

A valid card PIN is required.

5.5.1.1 Input Parameters

Parameter	Type	Required

operation	String	Yes

Value:

```
IMAGE
```

5.5.1.2 Response Parameters

Field	Type	Description
data.personInformation.image	Base64 String	Cardholder image

5.5.1.3 Image Format

The returned image can be rendered by decoding the Base64 content and displaying it using standard image libraries or browser APIs.

5.5.1.4 Invalid PIN Response

Same as the PERSONAL_INFO operation.

5.6 ALL Operation

5.6.1 Description

Retrieves all information available on the card, including:

- Personal information
- Cardholder image

The operation combines the data returned by:

- PERSONAL_INFO
- IMAGE

Information is obtained from:

- EF21
- DG1
- DG2

A valid PIN is required.

5.6.1.1 Input Parameters


```json
{
  "operation": "ALL"
}
```


5.6.1.2 Response

The response contains the complete set of personal attributes together with the Base64-encoded image.

{
  "code": 0,
  "message": "Success",
  "success": true,
  "data": {
    "personInformation": {
      "chipId": "0346790C2E6DA069",
      "docNumber": "008108081",
      "nationalRegistrationNumber": "0212050231",
      "givenName": "TEST",
      "lastName": "TEST",
      "issuedDate": "2023-03-09",
      "expirationDate": "2028-01-05",
      "issuingCountry": "BRB",
      "nationality": "BRB",
      "address2": "ABBEVILLE GDNS",
      "phone": "",
      "cellphone": "",
      "email": "",
      "dateOfBirth": "2002-12-05",
      "sex": "M",
      "image": "/9j/4AAQSkZJRgABAQEAYABgAAD/
2wBDAAoHBwgHBgoICAgLCgoLDhgQDg0NDh0VFhEYIx8lJCIfIiEmKzcvJik0KSEiMEExNDk7P
j4+JS5ESUM8SDc9Pjv/
2wBDAQoLCw4NDhwQEBw7KCIoOzs7Ozs7Ozs7Ozs7Ozs7Ozs7Ozs7Ozs7Ozs7Ozs7Ozs7Oz
s7Ozs7Ozs7Ozs7Ozs7Ozv/wAARCAEsASADASIAAhEBAxEB/
8QAGwAAAQUBAQAAAAAAAAAAAAAAAAQIDBAUGAAf/
xAA9EAABBAECBAUCBQMEAAQHAAABAAIDEQQSIQUxQVEGEyJhcTKBI0KRobEUUsEVJNH
hJTOS8ENiY3JzwvH/xAAaAQADAQEBAQAAAAAAAAAAAAAAAgMBBAUG/
8QAJBEBAQACAgICAwEBAQEAAAAAAAECEQMhEjEEQRMyUSIzBXH/
2gAMAwEAAhEDEQA/
APXjzQRKCAC5FBABBFBAcgUUEByCKCA5cuXIAIrlyA5BFBAcgTQsnZEmgkBpf6nbdh2QBEgI
toJHskunAfo0m+y6S9msPqPU9FGnmEMoiYQ0Nbrcb3Ky1sh85BbethaBuSSuOXCBetovuaW
eyeKOyHuA9JbzcQaaPb3VVNxLDa/
eMyHrvf6lJc9HmG20GdAbqRu3W9k82aN4tr2u+Da89PGBEbjDQL2t1J6LxW1pIdyPuHD+Fk5
G3jb5csvh+J8eQgPAbX9p5LQYuXHkx6mvaR7FPMpSXGxIKSlIFMUlApSCASUEopKA5FBFABci
ggJpQRKCA5BFBABBcgSgOXWhaCAKC5cgORQRQHLly5AcmxKwnmEzm5LIIjqdVhZPP8RR4

7jI2U0HUxrSdkty0aY7a+TJiaQC8boidpG1HrQO68u4j4vnynEgbN+kFx2VcfE3ETX47mjlQS+Z/
B66/Ka2GSXU2wNt1neLZoEEjw+jK4MbfXb+OaxsHi3Oja5vmNcHcwb3/
dRuI8dmy2NaTTQSdPTdLcttmOlhxbjjdseDaKMcr+o9SVSP4tKW03S0AcqVfPOZCXd002S/
uiQ209vEpg51Eb8zXNceKS3u1vxSrNZs90t7qfY5JtF2tv8AVGEt1xtBr8uyteHcbnxXtmgkBLfy
uPNZN7rdzTrJC2L0Dkeqy4tmX9eycC8Tw8SAimAilrodir9eG4ebJBpk1b30XpHhfxMzNjbjzua
HAUDypbMvqkyx+41K5FBUTJQSiggAiuXIALkUEBMKCJQQAQKKBQAKSiUEAFy5BAFcuXIArl
y5AdyUTNz4sOEvcfhOZOQyCMl2/
svNPFPiF0878eGU6Rs43zS5ZaNjjsrxD4odO98cL7PI1yCycksspsuNJt8pc81R+Bso80xB0CzSm
t6Pu5/UB90kuPTdIx8WWd1kfsrbG4SSAXJblIfHG5K5scjt6ATogf2tXsfDo2Vtf2UhuJGPyhS/
KrOFmv6Vzvy0k/6e/wCpoIWp/pY/7R+iIxY/7R+iPyt/
Cx8mDIPVRBUd4kjFObsea3DsOJwosCrszhTXA6WCk2PN/
SZcH8Zf8u3JKD72q1KycB0ROkH4UQsczdXllc9xs9nw8lo9IIHYqwwct0ErXxvojuqcTFpA3Cl48
7XEB1D3RZ0JXsnhnjcfFMBjHuAnjFOb391dryLgvE5eF57JGk7cxfML1bCy483FZPGbDha3G76
Lnjrs8glIJ0wRXLkAEEVyAllBEoIAIFFAoBJQKJQQCUVy5AcuXLkAUiWQRs1HkEXkNaSRyWT8
R8dOLE5rHkPrYDkB3WW6NjN1F8V+JBjxuhheDI6wQN9I/
wCV5tPM6RxdRFnqn87OfPM57nlxJJs9VWSZBJ2Uu72t1Jo6ZCDQ3Kk4mF5smo72efdM4GJ
JkPshaPHxWwtArdJnlpXjw33RxsZsbRspYockkCjslgErltdUmh1lKa7fcoaT0Sg0oMcBB6ogUUk
ApYFrGlNaVxiDktgpOht9lgVWZw8PBOnn7LO52A5hsDkt0Yw5pB3voqnOwtTXABW487KlyY
TKMUYrFHn/
AJSWjSa3sKflY5hmeCKB2UVw1W4bOHNdcu3BljpMiybiBP1MofZeh+C+JkRtxnu9Dt22vMY3
eqitTwPLMEMMjSba4hZl13DY/
wCpqvWUE3jTCeBkjTYItOKk7c+tOXLly0OQRQQEs80ETzQQAKBRQKASUlKKCAC5cuQHIrkO
SAg8VyxjYjzuPSSSvJOPcVMzpLOnWbI7DoFu/F2aYuHygHeQV/
heQcSndLKW3YU73VseoQ+V0rjWzUuCAyStaOaZj5hnVW/
DIQ7ILq5LMrqGxm6t8HHbDE1tbjmpzQux4DQUkRUFx29u6TUMhpTjW0N0oiuqANBLo0LYB
2R00bASA4WlNkBRppeyLR0SbsghKBorBs6wWf2T7WVvSjtd1tSQ8U0oada2t0zPEHdOaksIIC
EgB2WxlZLjmFpBe0LNklj/
AIW84lCJIXNIWJzIfLmcOxXVxZfTk58fsztqBHJXXDHk4sjG82u1BUjDfpKtOFS+XMfsq5zcQwu
q9O8K5vm4vlE7haJYLw1kHFzjASfUQAt212poKzivWmc2OrsVy5FVRBBFBAS+qCJQQAQKKB
QCSglFJKACK5cgOQO4pFA7BAeY+PMzTkGIu+nV/
heak6pienNbDxxJr4vlu8yw11D7rGE0xzup5KeK1O479WTutVwHH1tLzyLisjinS+
+pK2WKHwcPZDF9bxZ9gp8v8V4f6n5fE48UaYxqcOZPJVGRx2c/
QCD7JT8Vl6pnE+1pt8cI2EYPypTxi98qjjjWVd6ifYhPQ8dnG0jAUgtZy8sJsxRHm2k+8f4nrL+r
WHivmEW0D7qe2cOI5LPNY0H0lWOM5xAvmpZSL42/
a5abo9EXPDRdpGO0llUkZbXBjgB0SHKGbE0/
UnBxTHYac8LNvZI1x57pp0Tj0tUmMTudbCHj2EygZQVPh4lhZGzZW791gGY5Bs7lSoMR93Zr2
W+GJPPJtMqBk0ZMbgQsNxyExZG/
cq5xZpICKsEcr6qF4hezJxxMNng04e63CayHJfLFnR9VqZjO0y33FKEDblKi2cuquKNfw6enQZ
AO7CGv+F6NjS+ZEx5/MLXlXC5ADR5PH7r0Tw/
kedgMaRuzYqeHWWleTvHa3RXLlZzAgUSgUBKK5cea5ABAorkAkoIoFABciggOSJiWxOLRZpL
QeR5bieVIDw/xmX/6hkOI+uVxB+Fk5TTAAtL4snE2dMAKAlf/
ACspkO9XNJitl0ncIxjlZ7IxuOZW5la2GK2gaiKv2Wb8IY4Pm5BHsCrrLldqNcgufku8tOrhx1jsxLI
dWlrdch6dB8qPI17Rc+Q2P2ao2ZnSQ3FECHu+pyr8mKSNjZHuLwTzu6W44Fy5P4sXPiP05Rv3
SPNe36qcO4VTqDXbM1drUqceU5phJutwNwqeMJM6sA8HcKyw3agCqfGJkbZaQVc8NiJAFK
GfTo4+1/
gNLgnc2ICIu5GlJ4bjCgu4nCfLczuFzy9r2MdlTHWVCOQ261X8bp7ijHY0h1HYnYdSoMTJJmOk
D2xtaL2XXjj05M8u02KZgI9Lv/
SVY4k0LyGteA7tyWcblzxvtkxpWWLxZpaGZmK2Vp5Pb6XD4Ka4Fx5J9tEWBwGsA+6q+LQPE
DqFtI52l42awuDGSF8TvoLubfYp+c+bAW81Kbxva11lOmRafUpTCoz2+XM5p6Gk+w7fZdjgXP

D5NwAa3W78J5X4skJP1Cx7rzvBfpeB7rYeHp/
K4nAQdnbFRy6y2tO8bHoCK4LldygUkpRSUBKQRKCA5BFcUAEkpSSgAuXLkByD6Mbr5Uikv+
h3wgPCfF2g8SnDDt5jv5WRmdqkK1fikBvEZwN/xHb/
AHWUlAEingvm2nheLy+Eg1u4kqbksBTfh9lcIh+FOlxy7e1x5X/Vd+E/
zFNNis3pvNQ345ArTYPQhXrseuabdA0cwnmZLxqWPHc0elrW/
DU4McjnasnRgGmtS4sQu3cFtzZONFgxS5wJbSusOAMoJtkbWOAAU2JtusBRuW1scdLvhjvU
B8KXnQa2nbdQuGXrHZW04v4ISHy9sJxjALpNZbao5MNzQdLefOtl6Hk4kctgjdUGbw50Ti5rb
b8KuPJrpLLjl7ZAYQa8epzaO1ttTooGGAQNY43zdW5VuzGY47hPx4TGmwKVfyJfhU+Pw2WC
TU5pLHcu4ViY3CNT/JNAUdkXQExk0p5Z7Ux4/
GMTxGPRmHar3SYzsFL42zTlt+FCjOwXbhd4uDOayqdjO0yt+VqOFy6MiCQdH0spCae0+60W
DJTYtwPWCp8inG9Vjdrja7uAUtRsJ2rDiPdgUhVnpzXquQKK5axIKC481yA5cuXIAFBKQQCUE
qkEAEiZwZA955BpKWonE5vJwpDdek/
xaGx4f4kcJM+d4Nh8jiD8lZWXZ61HGbkLpCd9RtZmUev7qWC2bfeGyDwiC/7VZuO6p/
DEgdwtjf7bH7q4Isriz/
avR4/1hqRoKaMVqRo3RLaKTZ9I7ccA2QlloaluI5JmZ3p2W+xQY65QOpVm2o2gdVWYrmRM
dO82eTUoZwmcSx7XAbbHkm0yNDgzaXDsr90QkxQ8EEgLIYc9jnurbG4m6Jwa8+krNGym/
RU0wbLSZkAkBSeMkDy8mP6XbH5UWDIvYpdAmbCA9QFFNsjIICsQQ9oXeQ1xqt0bBmKHvu
pL8ZvknvSDY3MKcL6YQs2K868RnTxDT2CrozyUvxBJr4xJR5UFCYd6Xp4frHk8n71MhP4jflX2
KaMI7PVBBvK35V9jH8aMDuk5FON6dwSbzuGx927KwWc8N5Aje6Bxq7FHuFo03Hd4pcuOsn
LkUFRI+uXLkBy5cuQHLl1oIAFJSkEAFS+KJPK4TO/UBUZ59zsP5KuVk/
HGQ2LhbxY1Oc0LLdQ2M3XleW4yCdl7gWs9PzJ5K+c4HKkPR2ypsxmmQhSxXyjTeFZgYdH3
WlWK8My+XPR5FbUbgFc3LNZOzhu8XboOFlKpJcaUVzbzSjyAu2Tr3WaCWxoq1sZUCYlsDon
XvdfdU2PwqbHzvPDy1vSuq08jW9QmxGZDQCpM9EuOzmCx0urS/
S7TsfdU48N8TZnuyXyvcA6w7UbV7jAQSgBaGBwdFpcAQVnno9m/
aujZLLw5sTuYNlRGh8b9J5haZkLKIA2ULOwA4a2cxzU9t2iY8t0CpzG2QVAijLXUVYRj0rKKU7l
ah5k3lxucFPcB5e6zviHI/
p+HyOvetluM3dFyuptgsqbz8yWX+5xXM5qOzc7qRHzXqTp5N7u0zH+sK+wwTOzbkQqLH3df
wrzh4/
Gb+qlmtxtZhF0WQ97SQQQVsIZWzRNkbyIWKwpSMjXf5QCtJwicaTD7khJxXV0bnx3NrRFC1
y6XGkFBE80EAFy5JJQCrQtC11oDkFy5AJkcGtJPRedePMku8iK93XIR/H8LfZjtMR99h8rzLxY/
+p4yWN3qmDf7V/lTzvSvHO9sdJs8ntZUDNbrLiOfRW08Q/
GIGwdXxzVXkWQlxVvovgZc3I1D8pF+1rewnVE0rDcAdoznR6NQlaW/r/
3S22E7VisPLZR5p2vwXrR92zVHkdvQT7j6aUdwr7rndQNFfKUD7qO+cNvfqktyIqsvAHyt0Eg7
8lIxGguondVUmfe0ewTbMx7HWHmym8aza8lheJgWjqrXD16Be5HRZyDPc4U5xvoU/
FxJ7H22Ug+6y4UzUCQsdvdJ3WHDmqfE45G6o8mrP5k/
PlsieA19tPJJZYxImiGqx1Rh3FFMsyRIznuE9GPV7FK07LuxYrxtk+XisjB3c5bGV1ModF5341n8z
iUcIP0Nsq3DN5xDmusKom87HVPRgkpqMHRunoudL0HnpmN9Su8E6XB3vSpccU5qt8R1aK5
OUslsOmjwJQJ6J2qlqOHR6cgsLiK3FHmsbjOtwK07ZiBHlB21C/4/yp4dU/
J3GhtG0gG0V1OFLPNJKUeaSUACguK5ABcuXIDly5cgIPEHVHfINNrzHifr44XNOo+YTt0oL0zij
SY3m9g26XmMtt4w03sZXUR3Us1uNRzNBxpybvVd+9qmyRX6LROjOmZrSC10Zcb7g/
8Aaz0lObvzWQ9DhTmN4jF5ji1pNEjmFuMGRxa8PrVqJNcvssBF6Z4ybrUFvg0Q4+PJ6reKJPX
qCk5ZuKcN1klEpmYWAAuEmyS95I2XI7lVlse70tJ5qM3hMwdqdO9w7K3DQ5xtOXp+FSZaJ4
7Vw4W1/OVw+6cj4GNWoZDk+54uwaQbkOYasEe6byPJCv8AQRbHnKko/
lBpWeJ4exNIfI9xB2vUdv3VcM9woKyxMtpaBrRafo/J4fw3MLWOkD+hBKq8rgPEsTTIczzGg/
TXIfK0UEwAskJ+R4ljIIsKfmSzakwHvDgHHorqN1AUq10Aim1DkpbJKANqeV22H53ARm15bx
yf+r45kSA2A7SPst7xriH9Lw+WX+0be685ijdNMXEFxcf1JXT8ee65Pk3qYl6NEQcRsjDuUvNcxj
hCwhwYN3DqUiL0tJXW5E2EgvodB+qtYvSY/
ZVOLu8+wVlE+9IU6pPS4x3VNTiavdaXCcJMEsdyBr7LKQvqcErTcKfbXwg3rb6SO6l9q300nDZ
TNgRPO5qj9jSlqv4RI10cjGXQdqF9j/

3asKXVPTiy9pZ5pJSzzSaWlIIQpLpdSAQQgl0hSASuRpdSAi50YfC7bm0heV8RHk8UdpaB5eQ
baOtr117dTSO4XlnjLHOJxQy7/icz0FbKecU46o5Tp80A1pe5tV0v/
mln9DRIWSbUDXz0Wgz42xzPZFZE0YcCB15lUOQ52qSQCyaII6JYrUFxqiB12XoJPmeHMTIP1
eQ9wPev+FgZWVC1wPMnktlw15y/
Ax0G3YrnarPQgorZey8XIE0LXg8+acf9OyosDKdjy0T6DzCu2yNe2wuXKartxu4AFC7XGzzKda
B2StGo2kp0R8RduLTJx5D0Kto4m8yE82NhINLdjSh/
ppCdmlSseOZpHMK3bEwONtCkx48ZogI8hIaxi4NF2poeQ1FkLWdAjI0AKdMjyHUUjXtW6Mm
170qji/E/6SHyonAzSWGj/
K2Td0y2SbVPiTPOTkNw43ehv1OvqqqEeREZyaP0sH+UwZdeQaOo9b6nunMh1FsV3QBPzzX
fhj4zTzs8vK7NO9VHre6XdNASGbF56IXuLKomnYp9Lj3CsonbtN2KVVjmmn4VlF6CwHk4Wkp
4tAdL2nuLWg4bLpljeNgCAVnXGgw9AKVxwyW2FpPuAo1ZruElscxjHW27ctjY/
Yq3pUXC6GRqfRBcHNo8tiP8q+XTj6cec7STzXUiSLQsd0xApdSNjuu2QApJIS7C7ZAN0upLQ2
QCKVB4l4IOI4z3BoPp3v8A9/C0NIFoc0g8ijTZdPIDwvKmZE/
Q4SwyBrh81W3Y0szxjh03Dcl0c8T4yOhHReleLOH5nDp2Z+PG6SJlh2nqy7o/
fqsn4t8Yw8b4ScJ2E8S6w5spI2pLqRSW1muCYH+qSSYzAdRA0EnkOW/
6ruGZc+CzIwzsJQWPHwqZrnsk2Jb0v2UvBBDrPUpMvSmMu1rGN1Px5iwAE7KHAwkBSQCNl
zV2YrKGeypTJBSpmuLeSfZkadnbKdikqzMvYpyOXSOaqzkN7ojJrawjRtrMZA181NgyKHPks/
8A1Iu7TjM8MO52R4jyacTBwBCRNMA3mqE8XjYL1BVPFfFQjaWQEOf/
AAsnHlaXLkxxi44vxmLBiNm3nZrRzJWRzsuV+uWYkyycv/
lHZRGzS5eR58zy6Q8vZNSSunn0N3HIbrrw45i4uTluVS8NhIBJ3O5JSmDzpSO55oxAsa8g2GN
pGGMgWOyqkS4jcVQFpm/UnJ6YA2+fNMtIokLWVOgd6fkKxa7/AMsjsqmE7/
ZWbD+GL7paaLdrtWOAenKlYcLmY0MlO9GnC1W4HrglB5htj9Qn8N2gvZdC1Kqytdi5DYoNT
m3RIaR03B/wtVC7XCx3doKwsDwYHtdzoOathwZ4fh1VEFUwy30ly4/
axJ3K60kn1H5XKznKtdaSmsjMx8RmvImZGB/
c6llumyW3UP2kTZEWPGZJpGsYOrjSzGf44xYZDFhxf1Dh+bVQCyfiHxXJlMdJO6mM5MbyCjlz
Yzqd128XweTLvPqNvL414PFmMxnSyevk/T6U9J4v4DFE6R3Eo6aLIFk/
pS8In4zLNkGcm3AUxt/
SmRmzyytBcBZHRPj5faXJjxb1ht7dB46xcx3+0wMqSMGvMdpa39yncrxa1jPwMf1d3nksliyluJ
E0UKaBQSZHly5sufL6Wx+Pj9l8Z43n8RYY5Mh4jPNjTQP/
ACspkY7XElwV7M21Akju7CnM7b2t4SdRRPwGusaBafxcFodZCnujATkTK6J7ndMmE2bZCByF
JWik+G2EC3ZS2rpHc32SHDZSS1NvZtt1W7ZpDfYOybLnjqU89pCaJtPC02ZJOSQ+V+km6TjjQ
UOV+x3TxO0xkZD3ekHZQy3178hzT0m5QjZu1lfURZ7BViGXZ+BobizSmw4MJHxYH8lM4lse2
QitPqCfe4SY82k+lwDdv/uv/
Cae8DS1v5Y6o90yVTGgjGcG8y4BTmxhsLTQugCokbKa0F31C6/9/
Cl0BCDyND9Vhorsg6nknpajN2CfyLAcdvqofZMX6STsmKkQup9+ys4XExA9lVQdT2CsMYh7d
P7rK2Lfhcp890dgW0iypcTvxuyqcSUx5Mb2j5Fq0eC2c132/
wAKdVi5ZLogY7bZpWr4JlhrYwTtI0X7FYpr7hduOS0fA3tl8qNxq21f8LMbqtzm42B+opjJzMf
DjL8iZkYHcqh434tgxNcOKdUgJBcRsFh8ris+fI6WWUyOI5notz55Op2r8f4GfJ3n1Go4t41fqfF
gNAA5PPMrIZeflZ85M0rnEnfdQcnIc3Zrrs803DK4tO5vuVy5Z5Zd17PFwcfFNYxKdNT9DNmg
dOqo+P5NYzWXWp+9qwa4Ek2Tao/
EJtsbRfO903FP9xP5mWuHJA118rtZ2IO4TTz6WG9yN1wcvQfOPR+C5gy+GwvvfSAflTiVj/
CedofJjPdsSC0fyteDYXncmPjk9Hiy8sYQ9qjSsGkqU5MyAFpSKoJZuNvui0UnS3bkkAey3bNFt
b1QcEpvJErDGaTbtiQnyLSHgEUVu2Ic26ivFBS5LBIUR4sp4nTL7pRXsNb7b9VLc0EclFlcGbGgq
RLJGc0Dpe/VIbJfmSXyBAKTLMPMoC6SWn/ZjmNTwFaIZHoATgyNuqaHfNO/
7KSSH6iBRoJzBcC50brIcxzfk0f+k2z05BvkQFpFjsAwh24aAdvZTJARFde4UNgp1Hc6gCn86TS
AwEihZPssMrcp48zQN62+/X903IKAHskNJfJqN7myuJMkp7WnIeiNMP2U7CeBq7hyrzswAA/
VupeManc08yVlbFlXlZAIA0ndp/hWTXuk9RHNux9wqtxjMLCCQ/ewpuPMXRC3fSb/
AOUlUixjk1RHfcilfcKyDGIng7CtlnQa1EHmLAVngSkwMc3ahRU70rO1TxSbXmT2P/iO/
lQPMGg8/lSOID/f5FdZHfyq8PIJs/

quV9Bj6hMrt7vYpeoNiqtz2Tbi0u3390ouBbQWgG7iq3VLxxx8xjSehV1fLa/lUPHTcrXVVdFXh/
dx/O/41Wh12jaRypG913vn0vCynYmXHOCaad/
cL0TAyDLitc5+skc6XmXP4Wt8K5xkgOO8kuYdvhc3PjubdHBnq6ajVskltikQCbalae643ejOZvv
0TZbR9lKe0lNub7IYZXWlObSQd0AH2RsmXGgSTunTsmZCADumgRpDXVRnkd07IQLKhTy6e
R+LTyJ5BM6uv7qsyJbNX+6dnya5G1Akc5xtw26BXxjnzyAyBxscwd1KcwHCLhsB6goIIbYBO/
dWLA1+CHA7UqImcZ9BjhYIdVqTIxjpNQcBtQ25qJGdL9BPpcf/AOKSx2prA4/
mH2RWJwsSg3uQHfskcSfU7mgmtLRz9rTjGEyO23aCCFFzr/
qiSeYH8LDfRltBriDZOwCW2MsaO7yjE0BoIq769F0koErSNxYoApium/
8A2TzDpmDu7UzNWllAknfmnCbY2qsNQE+MmQ6e+4+VKxXhp1D5VdE8gB17tN7dlNHokEj
RbHmxXdLTRZtcCwtHTdvwVM4fMfKq/pNFVjHDci9tx7jqFLxZAJXA8nhTquNRuIEDiGQP/qO/
kqARcm+6ncScf9QyBpFeY7+VB31+y5X0knUB7b6fquYDp59UXkhnshEbYa7rG67c4eo2R91Rc
ZaHZLBYI02rwj7Kh4qbz/hqvwfs4fn/
APFWyCiAElKl3fXZJ6fC7XzxY3bSlcMznYOYzIbuGmnDuFEjO9d1x9LqWWbmmy6r1PDmZkw
slY62uFghSaWK8K8aGLKMKcgRvPod2PZbjmvP5MLjlp6XHn5Y7MvFDZN7J9wTLgkVNEbppw
op4hId0tDDD+SiTnYqZINIJ9lDlHLumjKgzOohVmTLpJrn1U7MeGH3Va1pkJcd91bFz536Mhur
1SC+yjzP1GgFMnJrSOShuqyVXFHIwRSsMY/
7MgcheygSAV7qXhvAjkjr1OComMlBxLRsd04AC3Zw7lMeYfKB60nY9v8A00sYt4JGveJDW4o
n3TGdF+LqA2J2+E1i2XtG+lwo+ys5IfNYwO36fcLDzuKR736QxxrWenbqku9T2j+1B7CMibV9U
ZLf3pJY7TJqO/VMRKld+GzatJP+E4xwYzldEX8EUlZA1sa69nG/
mxabjI9TXCw4af8AhYEiIFor7KdhFr2Px3bVu2+irYjtpkNGud9VK1Fjmyj6mndZTRPbqiOl21H9
1Ia/y5GPB2SXSw5cXmsFACnDqB3+yQGlo0OF7ekpTlcQ9XEMjbfzXfyopbvsflS84f8AiGT/
APkd/
KigWeq4n1UnUJe06HEduSRjbxnfcFOkDy3JnC3a6+6PofZwjdZviL9XEpd+VD9lpzsCspl750x6
2r/H/Z5v/p9cc/
8AqK4esoEbJTuZ+Fw359V2vAIaacEqQCgd0hwpwTrhcYQwlh0kCyOy3XhnxCMqIYmU/
wDHH0ud+cf8rBE8k9HI+KRsjHlr2GwQd1PPCZRTjzuF29ZO/
LdNPZ1ChcDy5cvhME0xBe5u5A5qcSvPs1dPTl3Nmkg7jdOd0zId6QDMxPJQslwaCSeSmS8rV
Vmkl1e6fGEyqo4hKS8XyJUdkrWRkOOylvaH5RDuSp8iNrJSB3XRI5cr9pM07KABu0wH63HSN
gmKHNSmMDW7Kkmk7dmTYNnr0SoXaZBSS421zuoSYiQ9tdVpUjSBbfc0nXAgNroN0lp/
F+QE8Bq3PdYE7AZTowp+O+2vcLOmQH7clCxT/uIvuU9hktilrrv+6ymiu4w3yc/
IafzvtRI6LqvairLxMB/XMIHNoJ/
QKpjP4g+E89EvtaRufJw4bElh2J9uiApsod+V4pK4ZK4sdGT6HCyPdCEAxOaRYa7ZK07JHqYX
Eb6aJ7lOY0gczQ4eqtvdKaAYZGnkKKjw7mj0ooan40px5gK9JKsJY2hgew+g7ju3uFWsHmsLX
b6TQPVTOHSOliMbzYALf0Smj//
ZbxGAAwAAX4IBAYMCAQGGYV1fH1pJPEJSQjAwODEwODA4MTgwMjEyMDUwMjMxPDw8
PDwwMjEyMDU2TTI4MDEwNTBCUkI8PDw8PDw8PDw8PDJURVNUPDxURVNUPEVJPDw8P
Dw8PDw8PDw8PDw8PDw="
    }
  }
}

5.7 WAIT_FOR_INSERT Operation

5.7.1 Description

Places the plugin in a waiting state until one of the following events occurs:

- A card is inserted before the timeout expires.

- The timeout period is exceeded.

The default timeout value is 30 seconds but may be customized by the application.

5.7.1.1 Input Parameters

Parameter	Type	Required	Description
operation	String	Yes	Must be `WAIT_FOR_INSERT`
timeOutSeconds	Integer	No	Timeout in seconds

5.7.1.2 Default Request

```
{
  "operation": "WAIT_FOR_INSERT"
}
```

5.7.1.3 Custom Timeout Example

```
{
  "operation": "WAIT_FOR_INSERT",
  "timeOutSeconds": 40
}
```

5.7.1.4 Successful Response

Indicates that a card was successfully inserted before the timeout expired.

5.7.1.5 Timeout Response

Error code:

```
315
```

Example message:

```
Time out for card insertion finished, cause: Time out exceeded
```

5.8 DETECTED_DEVICES Operation

5.8.1 Description

Detects and returns the list of connected smart card readers available on the workstation.

5.8.1.1 Input Parameters

Parameter	Type	Required
operation	String	Yes

Value:

```
DETECTED_DEVICES
```

5.8.1.2 Response Parameters

Field	Type	Description
data.devices	Array	List of detected readers
data.devices[].name	String	Reader name

5.8.1.3 Example Response

```
{
  "devices": [
    {
      "name": "Circle CIR115 ICC 0"
    }
  ]
}
```

If no readers are detected:

```
{
  "devices": []
}
```

5.9 Common Response Signature

All successful operations include a `signatureInfo` object containing:

Field	Description
challenge	Base64 challenge
signature	Base64 signature
certificate	Base64 X.509 certificate
algorithm	Signature algorithm
timestamp	Signature timestamp

The response signing process is described in the chapter **Response Signing and Validation**.

5.10 Response Codes

5.10.1 Success

Code	Description
0	Operation completed successfully

5.10.2 User Interaction

Code	Description
101	PIN dialog closed by the user

5.10.3 Card Detection Errors

Code	Description
301	No card reader detected

302	Multiple card readers detected
303	No card inserted

5.10.4 Card Interaction Errors

Code	Description
311	Unable to retrieve chip identifier
313	Unable to retrieve EF21 data
314	Incorrect PIN
315	Card insertion timeout exceeded

5.10.5 Input Errors

Code	Description
501	Input is not valid JSON
503	Unsupported operation value

5.10.6 Unexpected Errors

Code	Description
511	Unexpected processing error
512	Unable to parse date obtained from EF21