Version 6.18.6

PROTECTPAY® PAYER MANAGEMENT INTERFACE: SEAMLESS PAYMENT INTERFACE (SPI)

Instructions to Interface with ProPay's ProtectPay Payer Management Interface

PROPAY
A TSYS® Company

Contents

The ProtectPay Payer Management Interface: Seamless Payment Interface (SPI) is a Payer Management Interface (PMI) that allows merchants to maintain a payment page that mirrors the look and feel of their website without storing, transmitting or processing the data that their payment pages collect. The Seamless Payment Interface is based on an HTTP redirect to enable cross origin browser processes for a client system.

How to use this manual

This manual is designed to facilitate developers in building software solutions to consume the Seamless Payment Interface. It is not written for a single development platform. It provides basic information required to properly interact with the Seamless Payment Interface.

A developer should have an understanding of Hyper Text Transfer Protocol (HTTP) communication, the consuming of external Web services, Web Form POST methodology, AJAX request and Advanced Encryption Standard (AES) encryption using the Cipher Block Chaining (CBC) mode of operation, Cross Origin Resource Sharing (CORS) security standards and creating a Secure Sockets Layer (SSL) connection on the intended development platform.

While ProPay offers resources and materials that assist in creating and developing software solutions it is the responsibility of the integrating developer to design and develop his or her own software solution on the intended development platform to make use of and consume the services offered by ProPay.

Updated manuals can always be found at www.propay.com/Resources.

Additional Resources

- See ProtectPay API Manual for ProtectPay API Methods that are referenced in this manual.
- See ProtectPay API Manual Appendix A for a list of response values returned by ProtectPay.
- See ProtectPay API Manual Appendix B for a list of supported Processors, Gateways and Service Providers.
- See ProtectPay API Manual Appendix C for a list of supported Swipe Devices.

Disclaimer

ProPay provides the following documentation on an "AS IS" basis without warranty of any kind. ProPay does not represent or warrant that ProPay's website or the API will operate securely or without interruption. ProPay further disclaims any representation or warranty as to the performance or any results that may be obtained through use of the API.

Regardless of its cause, ProPay will not be liable to client for any direct, indirect, special, incidental, or consequential damages or lost profits arising out of or in connection with client's use of this documentation, even if ProPay is advised of the possibility of such damages. Please be advised that this limitation applies whether the damage is caused by the system client uses to connect to the ProPay services or by the ProPay services themselves.

1.0 Processing with the Seamless Payment Interface

The Seamless Payment Interface (SPI) is a Payer Management Interface (PMI) of the ProtectPay Application Programming Interface (API). ProtectPay ensures the payers' payment information is collected, updated, and stored in accordance with PCI standards. The SPI enables a merchant to collect sensitive payment method information by redirecting a payer's browser to post the sensitive payment method information to a ProtectPay server for processing without having it traverse the client's system. This minimizes the merchants PCI compliance requirements and limits the risk and exposure of the merchant by not handling sensitive payment information, while allowing the customer to experience the payment process on the merchant's website.

Important Concepts

- ProtectPay is not a Processor or Gateway; it is a secure collection of sensitive payment data.
- ProtectPay stores data securely for both single and recurring or subsequent payments using industry best practices.
- ProtectPay utilizes a proprietary interface to process transactions through several major gateways, processors and services providers.
- ProtectPay supports swipe transactions through integration of supported swipe devices.

Why the Seamless Payment Interface

Current web browser security standards prevent a web page from requesting resources from a domain other than the domain or origin of the current page being served (CORS standard). This restriction makes it necessary to perform a redirect in order to provide cross origin browser processes to provide a seamless payment experience to the payer.
The SPI is only needed when new payment method information must be collected. One of the SPI configuration options is to create a PaymentMethodId from the payer-entered data. Once a PaymentMethodId has been created for the specified PayerId it can be processed against using the ProtectPay API directly while maintaining minimal risk, exposure and PCI compliance scope.

SPI Processing Configurations

The SPI can be configured to perform various payment method storage and/or processing requests. These options include:

- Create a Payment Method
- Create and Authorize a Payment Method for a specified amount
- Create and Process a Payment Method for a specified amount
- Authorize a payment method for a specified amount
- Process a payment method for a specified amount
- Authorize a payment method for a specified amount and create a PaymentMethodId only if successful

1.1 Summary of Processing

Using the Seamless Payment Interface will allow a merchant to create his or her own shopping experience. Every page in the checkout process that a cardholder can see is painted by the merchant who maintains control over every bit of the flow. Here is how it works:

1. A Customer finishes shopping and clicks on a link to check out.
2. Before the customer is able to see the checkout page, the merchant should make a call to the ProtectPay API to Get a Temp Token (See ProtectPay API documentation)
3. The merchant uses this data to encrypt all of the information that the merchant can know without any input from the cardholder. (See encryption processes in this manual)
4. The merchant then paints the checkout page. Include the following on that page:
    a. The encrypted blob of data (hidden)
    b. A reference ID to the temp token (hidden)
    c. A submit button with special javascript code shown below
5. When the cardholder clicks submit, execute the special javascript code
    a. Client-side validation of the card (16 digits, valid format, all data present on page, etc.)
    b. Data is POSTed to the SPI rather than back to the merchant's server. This POST includes a return URL.
    c. Display a 'Transaction in Progress' message, spinning wheel, or whatever suits the merchant's fancy.
6. The SPI follows instructions contained in the POST to process the transaction and potentially save a payment method for future transactions. (Note, the SPI does NOT do anything to the cardholder browser experience at this point. The browser remains on the checkout page)
7. When the SPI has finished processing, it redirects the cardholder to the return URL it was given.
    a. This redirect will contain an encrypted response that should be decrypted by the merchant's system (See encryption process in this manual)
8. The merchant displays a 'success page' or 'failure page' of his or her own design.

1.2 Best Practices

- A PayerId is required when creating a PaymentMethodId. A PayerId can be created using either ProtectPay API method 4.2.1 'Create PayerId' or by using ProtectPay API method 4.7.1 'Create TempToken'. Once a PayerId is created it should be associated with the user and used in subsequent transaction requests instead of creating a new one for each transaction.
- The SPI is only required when creating a PaymentMethodId, or processing payment method information without wanting to store it. Once a PaymentMethodId is created, subsequent transactions should be processed using the ProtectPay API directly.
- Before form POSTing the payer-entered data to the SPI, the developer should validate the card number against a Mod 10 check using the LUHN algorithm, and should verify that the card number submitted conforms to rules established for the card type selected. The developer should also validate the expiration data is not past due and the CVV entered is the correct number of integers. This should be done before the cardholder submits the request to the SPI to avoid the customer waiting for an SPI response that indicates the card number, expiration date and CVV entered were incorrect. This will improve the end-user experience by not having to re-enter the information.
- Credit card transactions can take several seconds to process. This is caused by several variables with the gateway, the processor, and the issuer. There will be a wait during which a cardholder may become impatient. ProPay recommends that developers provide cardholders with a warning against clicking the back button, or refresh on their browser or pressing the rendered 'submit' button while a payment method is processing. ProPay recommends that developers generate a control that displays such a warning during the period of time it takes to receive a response.

2.0 Testing and Certification

To improve the customer experience, ProPay requires that developers test their software solutions before receiving credentials to process live transactions. Doing so ultimately improves the end-user experience so please plan accordingly and develop a timeline that provides for testing and certification against the ProPay Integration environment. Integrating a developed software solution to the ProPay web integration requires the following steps:

1. Request API credentials from your sales representative or account manager. By involving him or her in the process, Propay can provide you with guidance about the methods required for your project's scope.
2. Design, build, and test your solution using the ProtectPay integration environment.
3. Contact your Project Manager when you believe you are ready to certify. Your PM will go over your integration with you. (This is a relatively informal process, but one that ensures you've covered all your bases.)
4. Request Production (Live) Credentials from your Project Manager.

Production URLs

The Production SPI URL: https://protectpay.propay.com/pmi/spr.aspx
The ProtectPay Production REST base URI: https://api.propay.com/protectpay
The ProtectPay Production SOAP URI: https://api.propay.com/protectpay/sps.svc
The ProtectPay Production WSDL URI: https://api.propay.com/protectpay/sps.svc?wsdl
The ProtectPay Production WSDL single file URI: https://api.propay.com/protectpay/sps.svc?singlewsdl

Test URLs

The Integration SPI URL: https://protectpaytest.propay.com/pmi/spr.aspx
The ProtectPay Integration REST base URI: https://xmltestapi.propay.com/protectpay
The ProtectPay Integration SOAP URI: https://xmltestapi.propay.com/protectpay/sps.svc
The ProtectPay Integration WSDL URI: https://xmltestapi.propay.com/protectpay/sps.svc?wsdl
The ProtectPay Integration WSDL single file URI: https://xmltestapi.propay.com/protectpay/sps.svc?singlewsdl

Live Credentials MUST be kept confidential

2.1 Troubleshooting and Technical Support

Your Project Manager acts as a technical resource during integration and will assist you with trouble shooting problems encountered while you work on your solution. In an effort to make this possible, you should be prepared to provide the following information when you encounter a problem during integration:

1. Timestamp of the incident (specify time zone)
2. URI Requests are being made to
3. HTTP Method being used
4. XML/SOAP/JSON data passed to the URI
5. XML/SOAP/REST/HTTP Response received.

Despite all the best preparations, planning and testing there are occasions where errors can occur when transitioning from the testing systems to the live environment. Providing less information may result in a delay to any technical support questions regarding the Application Programming Interface. The ProPay Technical Support team can only assist in the troubleshooting of the API and not a client's software solution when undesired effects occur in a client's software solution when consuming the ProtectPay API.

Limitations based on a supported gateway

ProtectPay is works with multiple gateways over which ProPay has no control. As such there are instances where a gateway may return an error with a transaction passed to it from ProtectPay. These errors are indicated by the 200 series in Appendix.2. If a transaction request returns a 200 series error ProPay technical support can only troubleshoot that the MerchantProfileId is setup properly according to the specifications found in Appendix B, and upon request, provide the raw request to and response from the gateway.

Should a client require additional troubleshooting they should contact the Processor Gateway for an explanation of their specific failure. ProPay Technical Support cannot troubleshoot non ProPay merchant account issues.

3.0 Technical Integration

Secure Sockets Layer (SSL):

ProPay recognizes the importance of handling financial transactions in a secure manner and ensures that ProtectPay offers the best transmission security available. ProPay ensures that ProtectPay API request information is transmitted using the latest Secure Sockets Layer (SSL) encryption practices. SSL creates a secure connection between client and server over which encrypted information is sent. ProPay hosts the SSL certificate for this connection type. Each ProtectPay API method request, regardless of the interface, will negotiate an SSL connection automatically over port 443.

Cross Origin Resource Sharing HTTP Headers

ProPay has added the following HTTP Headers to the response from the SPI prior to its redirecting the cardholder browser:

- Access-Control-Allow-Origin:*
- Access-Control-Allow-Methods:GET,POST,HEAD

This makes it possible for a developer to receive a response from the SPI and perform checks against it before the browser is directed to the client-side results page. Handling this optional feature is fairly challenging, and should only be considered by more experienced developers.

Authentication and API Methods required to use the SPI

The Seamless Payment Interface uses a single-use working key known as a TempToken for authentication. This requires that ProtectPay API methods are called prior to a merchant painting his or her checkout page:

- ProtectPay API Method 4.2.1 'Create PayerId' (PayerId may be created in same call as 'Create TempToken').
- ProtectPay API Method 4.7.1 'Create TempToken'

Temp Tokens are built into the SettingsCipher parameter submitted to the SPI.

TempToken must be kept confidential.

3.1 Data Encryption and Decryption

The ProtectPay Seamless Payment interface requires that much of the data submitted is encrypted. This is in addition to SSL encryption that exists for all of the parameters. The purpose of this extra encryption is NOT to secure the data in transit (we trust SSL/TLS for that, and in fact the most sensitive data is not encrypted by anything else.) Instead, this extra encryption is used to establish non-repudiation for the transaction, and to protect SPI users from the potential that a response from the SPI might be 'spoofed.'

The type of data to be encrypted into a single SettingsCipher value includes all information that can be known by the processor without any input from his or her customer. This is important, because it is not appropriate that merchants handle cardholder data if using a ProtectPay Payer Management Interface. Encryption requires server-side coding, and cardholder data should not touch the merchant's server.

Encryption Process

Encrypt the Key-Value Pair string using the following method:

1. UTF-8 encode the TempToken string and generate an MD5 hash of it.
2. UTF-8 encode the Key-Value Pair string and encrypt using AES-128 encryption using Cipher Block Chaining (CBC) mode.
    a. Set both the key and initialization vector (IV) equal to result from step 1.
3. Base64-encode the result of step 2.

This encrypted value is known as the 'SettingsCipher' and will be form POSTed to the SPI along with the cardholder information. The SPI will process the request and redirect the cardholder's browser to a response page set by the 'returnURL' parameter and form POST the response known as the 'ResponseCipher'.

Decryption Process

The 'ResponseCipher' is encrypted using the same process and TempToken used to encrypt the 'SettingsCipher'.

1. Base64 decode the response cipher.
2. UTF-8 encode the same TempToken used to encrypt and generate an MD5 hash of it.
3. Decrypt the result of step 1 using AES-128 decryption using Cipher Block Chaining (CBC) mode.
4. Set both the Key and Initialization Vector (IV) equal to result from step 2.

Message Padding

AES 128 Encryption using Cipher Block Chaining requires the size of the message must be a multiple of the cipher block size. In this instance the block size is the same size as the MD5 Hash of the TempToken which is 16 bytes. Due to the variable nature of the Key-Value Pair that is to be encrypted, padding may need to be added in order to ensure the resulting message to be encrypted is a multiple of 16 bytes. If the string is padded in order to be encrypted the decrypted response will need to have any added padding removed before being converted back to a readable string.

Example Key-Value Pair String before Encryption and Base64 encoding:

AuthToken=1f25d31c-e8fe-4d68-be73-f7b439bfa0a329e90de6-4e93-4374-8633-22cef77467f5
&PayerID=2833955147881261
&Amount=10.00
&CurrencyCode=USD

&ProcessMethod=Capture
&PaymentMethodStorageOption=None
&InvoiceNumber=Invoice123
&Comment1=comment1
&Comment2=comment2
&echo=echotest
&ReturnURL=https://il01addproc.propay.com:443/Return.aspx
&ProfileId=3351
&PaymentProcessType=CreditCard
&StandardEntryClassCode=
&DisplayMessage=True
&Protected=False

*The AuthToken value in the Key-Value pair string is set to the created TempToken and this value is used to encrypt.

Example Key-Value Pair String after Encryption and Base 64 encoding:

WD7n54SPFT4Pa/GdLy5Pg8rKnArxQkVQr+plCmj3Nc+vz8JZ0ugsKiFmiPw5roHKEjV7vaff1k+SG3Sxs1L9yfnnE1uLi/AVP4O1H/vpK+MOfPVFczXQ9TCPYnDT
w+r/A7c6nwUOnbEsO+xF++k0cuqEMGzaQxNV3kJfsGMegBvlzXH56jzZ39/S+p4g3PGbQ7ZP6K/bkF9URyBq2+gaDuEVWt1AF3v69CX7VVy45TTnU/zhCd8
PFLMh83lc0UJp0ZTIM60rMZOCJbGhccSZ6hujW0d4bz5qocpFxVA9lapSilrnKsFGp3a6njOMsFHgZznKgXaEAJmT59M30Uk+ml4uhKuj9Tx8n2DW6b3UVhqIvi
DXn4sXeQ1LXuOTskQJroBQzqrj9RYw/Dw7q2a2ubwr3GYVhq2fI1tZ2ohfFju4j9wRJ33tIIfs5OB0gP8R46Z2JYrWLNPPIh9ZGczrUM7sFpIBepsyKlSPnw43zZek+3L
N/+Sr3nmFnxO4sQ1ZasuvxQ1L4auL6LJg1anBZcWkkNXkcFqRLaZ6LlF506t5hjl2xK3Lp8K4z4JJJ7i3

3.2 JavaScript handling of cardholder data

The SPI provides its data security benefit because cardholder data never traverses a merchant's own website. Instead, it is POSTed directly to a secure interface hosted by ProPay. (See section 1.1: Summary of Processing) That said, it is perfectly acceptable for client-side code, executed on the cardholder's browser, to handle the card prior to its submission via POST. There are several validation steps that you should take with your JavaScript (or similar) code:

- Make sure the card is of an appropriate length.
- Make sure the card starts with the correct digit based on card type identified.
- Make sure the card number passes a LUHN check
- Make sure none of the other fields (such as cardholder name) contains sensitive card data. If a cardholder includes this data in one of the fields you paint, and the POST made by his or her browser contains it, ProPay will return an error response. You would be well served to handle the information up front in order to create a good experience.
- Display a spinning wheel, "Do Not Press Back Button" or similar message for the customer to view prior to the redirect that the SPI initiates.
- Disable the browser's back button.
- Disable a second click of the submit button you served up.

4.0 Request Interface

4.1 Required Parameters

Element Name	Type	Max	Required	Notes
CID	Int32	-	Required	The 'CredentialId' of the Temp Token used to encrypt the settings cipher
SettingsCipher	string	-	Required	The SettingsCipher is the encrypted value of the parameters that cannot be changed by the cardholder See 3.2.3 Required Encrypted Parameters

Additionally, either ACH or Credit Card optional parameter group must be included.

Required Encrypted Parameters (These values are used to build the SettingsCipher)

Parameter	Type	Max	Notes
AuthToken	string	-	Returned by 'Get a Temp Token' API call as "TempToken"
PayerID	long	-	Returned by ProtectPay API Method 'Create PayerId' as "ExternalAccountId" Returned by ProtectPay API Method 'Create TempToken' as "PayerId"
PaymentProcessType	String	-	Valid values are: ▪ ACH ▪ CreditCard
ProcessMethod	String	-	(see below)
PaymentMethodStorageOption	String	-	(see below)
CurrencyCode	String	3	ISO 4217 standard 3 character currency code
Amount	long	-	The value representing the amount the for which the transaction should be processed *This amount must include a decimal point followed by two digits
InvoiceNumber	string	50	Recommended Transaction descriptor-only passed if supported by your gateway *ProPay rejects transactions as duplicate when the same card is charged for the same amount with the same invoice number, including blank invoices, in a 60 second period.
ReturnURL	string	-	Fully Qualified URL to direct client browser to redirect to when response is received *The URL cannot contain query string parameters, an error will be returned indicating an invalid SettingsCipher

Configuring Functionality

The values shown above, PaymentMethodStorageOption, and ProcessMethod are used to define the action the SPI will perform. Consider the following possible combinations:

Desired behavior	PaymentMethodStorageOption	ProcessMethod
Use SPI only to store a payment method	Always	None
Use SPI to Process a payment without storing the payment method	None	Capture
Use SPI to Authorize a payment without storing the payment method	None	AuthOnly
Use SPI to Process and store a payment method	Always	Capture

Use SPI to Authorize and store a payment method	Always	AuthOnly
Use SPI to attempt a payment and store only if successful	OnSuccess	Capture
Use SPI to attempt an authorization and store only if successful	OnSuccess	AuthOnly

4.2 Credit Card Parameters (non-encrypted)

Element Name	Type	Max	Required	Notes
CardHolderName	string	50	Required	The name on the card
PaymentTypeId	string	-	Required	Valid values are: ▪ Visa ▪ MasterCard ▪ AMEX ▪ Discover ▪ DinersClub ▪ JCB
CardNumber	string	16	Required	*You should perform your own validation of card numbers lengths which are generally 15 or 16 digits
ExpMonth	Int32	2	Required	Month portion of credit card expiration date expressed in a 2 digit format
ExpYear	Int32	4	Required	Year portion of credit card expiration date expressed in a 2 digit format
CVV	string	4	Optional	Card security code ProtectPay will NOT store this value
Address1	string	50	Optional	Payer's address line 1; if your gateway supports AVS this value will be passed for AVS
Address2	string	50	Optional	Payer's address line 2; if your gateway supports AVS this value will be passed for AVS
Address3	string	50	Optional	Payer's address line 3; if your gateway supports AVS this value will be passed for AVS
City	string	25	Optional	Payer's address city; if your gateway supports AVS this value will be passed for AVS
State	string	25	Optional	Payer's address state; if your gateway supports AVS this value will be passed for AVS
PostalCode	string	10	Optional	Payer's address postal code; if your gateway supports AVS this value will be passed for AVS
Country	string	25	Optional	Payer's address country; if your gateway supports AVS this value will be passed for AVS *ISO 3166 standard 3 character country codes

4.3 ACH Parameters

Encrypted Values

Element Name	Type	Max	Required	Notes
StandardEntryClassCode	String	-	Required	Standard Entry Class Code required for ACH Payment Processing Valid values are: ▪ PPD ▪ CCD ▪ WEB ▪ TEL

Non-Encrypted Values

Element Name	Type	Max	Required	Notes
BankName	string	-	Optional	The name of the financial institution *Recommended this be collected
RoutingNumber	string	-	Required	The routing number of the financial institution
Bank CountryCode	string	3	Required	The country of the financial institution *ISO 3166 standard 3 character country codes
NameOnBankAccount	string	50	Optional	The primary name on the account
Bank AccountNumber	string		Required	The account number at the financial institution
BankAccountType	string	-	Required	Valid values are: ▪ Checking ▪ Savings
StandardEntryClassCode	string	-	Required	Valid values are: ▪ PPD ▪ CCD ▪ WEB ▪ TEL ▪ IAT

4.4 Optional Parameters (encrypted)

SessionId	string		Session id for ThreatMetrix
InputIpAddress	string		Input IP Address for AmexEnhancedauth and ThreatMetrix
ShippingAddress1	string		Shipping Address1 for AmexEnhancedauth and ThreatMetrix
ShippingAddress2	string		Shipping Address2 for AmexEnhancedauth and ThreatMetrix
ShippingCity	string		Shipping City for AmexEnhancedauth and ThreatMetrix
ShippingState	string		Shipping State for AmexEnhancedauth and ThreatMetrix
ShippingZip	string		Shipping Zip for AmexEnhancedauth and ThreatMetrix
ShippingCountry	string		Shipping Country for AmexEnhancedauth and ThreatMetrix
ShippingFirstName	string		Shipping First Name for AmexEnhancedauth and ThreatMetrix
ShippingLastName	string		Shipping Last Name for AmexEnhancedauth and ThreatMetrix
ShippingPhoneNumber	string		Shipping Phone Number for AmexEnhancedauth and ThreatMetrix
ShippingMethod			Shipping Method for AmexEnhancedauth
CUA1	string		This is optional ThreatMetrix parameter CustomAttribute1
CUA2	string		This is optional ThreatMetrix parameter CustomAttribute2
CUA3	string		This is optional ThreatMetrix parameter CustomAttribute3
CUA4	string		This is optional ThreatMetrix parameter CustomAttribute4
CUA5	string		This is optional ThreatMetrix parameter CustomAttribute5
CUA6	string		This is optional ThreatMetrix parameter CustomAttribute6
CUA7	string		This is optional ThreatMetrix parameter CustomAttribute7
CUA8	string		This is optional ThreatMetrix parameter CustomAttribute8
CUA9	string		This is optional ThreatMetrix parameter CustomAttribute9
CUA10	string		This is optional ThreatMetrix parameter CustomAttribute10
CA1	string		This is optional ThreatMetrix parameter ConditionalAttribute1
CA2	string		This is optional ThreatMetrix parameter ConditionalAttribute2
CA3	string		This is optional ThreatMetrix parameter ConditionalAttribute3
CA4	string		This is optional ThreatMetrix parameter ConditionalAttribute4
CA5	string		This is optional ThreatMetrix parameter ConditionalAttribute5
CA6	string		This is optional ThreatMetrix parameter ConditionalAttribute6
CA7	string		This is optional ThreatMetrix parameter ConditionalAttribute7
CA8	string		This is optional ThreatMetrix parameter ConditionalAttribute8
CA9	string		This is optional ThreatMetrix parameter ConditionalAttribute9
CA10	string		This is optional ThreatMetrix parameter ConditionalAttribute10

CA11	string		This is optional ThreatMetrix parameter ConditionalAttribute11
CA12	string		This is optional ThreatMetrix parameter ConditionalAttribute12
CA13	string		This is optional ThreatMetrix parameter ConditionalAttribute13
CA14	string		This is optional ThreatMetrix parameter ConditionalAttribute14
CA15	string		This is optional ThreatMetrix parameter ConditionalAttribute15
CA16	string		This is optional ThreatMetrix parameter ConditionalAttribute16
CA17	string		This is optional ThreatMetrix parameter ConditionalAttribute17
CA18	string		This is optional ThreatMetrix parameter ConditionalAttribute18
CA19	string		This is optional ThreatMetrix parameter ConditionalAttribute19
CA20	string		This is optional ThreatMetrix parameter ConditionalAttribute20
CreditCardNumberHash	string		CreditCard Number Hash for ThreatMetrix
SocialSecurityNumberHash	string		SocialSecurityNumber Hash for ThreatMetrix
ACHAccountHash	string		ACHAccount Hash for ThreatMetrix
DriversLicenseHash	string		DriversLicenseHash for ThreatMetrix
Policy	string		Policy for ThreatMetrix
IsThreatMetrix	bool		True if ThreatMetrix Fraud detection is required, else false
IsAmexEnhancedAuth	bool		True if Amex enhanced Auth is required , else false
ProfileId	long	-	Used to specify merchant account when your biller ID has access to multiple merchant accounts *If your account is set to point to multiples this value is required
Comment1	string	128	Transaction descriptor. Only passed if supported by your gateway *The following characters cannot be passed: “?”	”&”	” =”
Comment2	string	128	Transaction descriptor. Only passed if supported by your gateway *The following characters cannot be passed: “?”	”&”	” =”
echo	string	-	*Optional value that is not passed to gateway and is returned in the response *The following characters cannot be passed: “&”	”=”
Protected	bool	-	True or False required if storing the payment method *Indicates whether a stored payment method can be deleted by the payer

Using Fraud Detection

The Seamless Payment Interface has been supports multiple Fraud Detection Provider options. Elements listed above are also supported by the ProtectPay API and while all are optional parameters, use of a fraud solution makes some values required as a group. Each Fraud Provider has a unique set of required and optional variables. Please view Fraud solutions documentation for further details.

Note: browsers have differing limits imposed upon the number of characters allowed in query-string submission. As such it is highly recommended that, if using a Fraud Detection solution with the SPI, the developer submits via Form POST rather than query string.

5.0 Response Handling
5.1 Decrypting and Parsing the Response Cipher

Response Cipher – Raw Data

Element Name	Value
ResponseCipher	irvN4mdV6jA0wmsSVq8yHv3F+2frLchvQpTuJj1r8lBMmYhP8ZJ5TmVGbdm1vPm91UEW3m89IgfgM5R+HjF8jwzskTX4sRExSbDv3szDdwRyUnAT9neieDJ DXzdCmG6/+FhxIN/Lai0Dg5s7lWfGsl+xNmsTr/4yQ//btMl1u6AM+Jyi2+tBwGPgjMgrG5hjnqpbKwsd5k7yELDdCQHLzkjFagKjYMyfXgaRlHX7rNBpiQSK qmhESZm/XrktySfOrf80jFQtUZq0iSHxVr83W55/QGhH0TFs+avcP4yVJPuwCju2bH0Kkd6m3QtclygPM29mc2xlylxl/8SB3bgJ8ESwWJCImlvqXJa1rcSO8y 5UYkn+QL5cI9iuAaEhGjlwoC3q/q6F9kQrSlnV4ExymTio0VLFUSN4Sx6GjV5E2lTT25ypspkp05FvQPZnp+8S

Response Cipher – Decrypted and Parsed

Name	Type	Notes
Action	string	"Complete" indicates the request was completed "Err" indicates one or multiple errors with the transaction request
ErrCode	String	Numeric Code returned only when Action=Err. This indicates a problem with the SPI Request *Multiple ErrCode values may be returned. Example: ErrCode1 =, ErrCode2 =, …
ErrMsg	String	Text detail returned only when there Action=Err. This indicates a problem with the SPI Request *Multiple ErrCode values may be returned. Example: ErrMsg1 =, ErrMsg2 =, …
echo	String	*Returned only if submitted
ProcessResult	String	*Result of transaction processing request
ProcessResultResultCode	String	Result Code of transaction request *Not returned if there was an error processing the transaction request
ProcessResultResultMessage	String	Result text description of transaction request *Not returned if there was an error processing the transaction request or if the ProcessResultResultCode=00
ProcErrCode	string	Processing Error Code for transaction Processing *Only returned if there was an error processing the transaction request
ProcErrMsg	string	Processing Error text description for transaction *Only returned if there was an error processing the transaction request
StoreErrCode	string	Storage Error Code for transaction Processing *Only returned if there was an error storing the payment method
StoreErrMsg	string	Storage Error text description for transaction *Only returned if there was an error storing the payment method
ProcessResultAuthorizationCode	string	The auth code supplied by the issuing bank *Only returned on a successful transaction
ProcessResultCvvCode	string	The issuer CVV response *Only returned if supplied *ProtectPay WILL NOT store the CVV code of a Credit Card Payment Method
ProcessResultAVSCode	string	AVS response produced by gateway *Only returned if AVS information is supplied and AVS is supported by your gateway
PayerId	string	Id of the payer that is the owner of the payment method

PaymentMethodId	string	*Only returned if a payment method was stored as defined by PaymentMethodStorageOption
CardholderName	string	*Returned only for credit card transactions
ObfuscatedAccountNumber	string	*Returned only for credit card transactions obfuscated for security
ExpireDate	string	*Returned only for credit card transactions
Address1	string	*Returned only for credit card transaction.
Address2	string	*Returned only for credit card transactions
Address3	string	*Returned only for credit card transactions
City	string	*Returned only for credit card transactions
State	string	*Returned only for credit card transactions
PostalCode	string	*Returned only for credit card transactions
Country	string	*Returned only for credit card transactions
BankName	string	*Returned only for ACH transactions
ProcessResultTransactionHistoryID	string	Unique transaction number assigned by ProtectPay
ProcessResultTransactionId	string	Transaction number assigned by processor (Gateway)
GrossAmt	string	Gross amount of transaction repeated back to you
NetAmt	string	Net amount of transaction after fees charged; *ProPay Gateway Only
PerTransFee	string	Per transaction fee ; *ProPay Gateway Only
Rate	string	Percentage fee ; *ProPay Gateway Only
GrossAmtLessNetAmt	string	Total of fees; *ProPay Gateway Only

*Not all values are returned. See the individual notes for each response value.

Possible Error Responses

ErrCode=301&ErrMsg= Invalid CID
- The TempToken has expired
- The CID is an invalid CID
- The SPI did not get the CID value from the request

ErrCode=301&ErrMsg= Invalid SettingsCipher
- The SPI was able to acquire the CID and the SettingsCipher is improperly encrypted
- The SPI was able to acquire the CID and the SettingsCipher is improperly encoded in the request
- The SPI was able to acquire the CID However the SPI is unable to get the SettingsCipher value from the request

ErrCode=348&ErrMsg= Invalid SettingsCipherLength
- Query string exceeded max length of characters allowed. Reduce the number of characters submitted, use a form POST, or switch to the Hosted Payment Page.

5.2 SPI Transitional Response html

The following html is returned if there were no errors in submitting the payment method details to the SPI. The clients browser will read interpret the HTML and execute the Script if the request was form POSTed. If the request was submitted via AJAX instead of a form POST the request the response is identical without the script being executed.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<!—(c)2016-ProPay -->
<html xmlns="http://www.w3.org/1999/xhtml" >
        <head>
    <title></title>
  </head>
        <body>
          <form method="post" action="spr.aspx" id="form1">
            <div class="aspNetHidden">
                        <!--The SPI supports View States -->
                <input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE" value="" />
            </div>
            <div class="aspNetHidden">
                        <!--The SPI supports Event Validation -->
                <input type="hidden" name="__EVENTVALIDATION" id="__EVENTVALIDATION" value="" />
            </div>
            <div></div>
                <!--The ResponseCipher prior to be Form POSTed to the ReturnURL -->
                <input name="ResponseCipher" type="hidden" id="ResponseCipher" value="" />
    </form>
        </body>
</html>
<script type="text/javascript">
  //The action will be the returnURL in the SettingsCipher
        document.forms[0].action='https://il01addproc.propay.com/Return.aspx';
        document.forms[0].submit();
</script>
```

Sample Transitional Response html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
        <head>
    <title></title>
  </head>
        <body>
          <form method="post" action="spr.aspx" id="form1">
            <div class="aspNetHidden">
                <input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE" value="" />
            </div>
            <div class="aspNetHidden">
                <input type="hidden" name="__EVENTVALIDATION" id="__EVENTVALIDATION" value="
/wEdAAKvVXD1oYELeveMr0vHCmYPexh68czOUCZr7Ag7DJmz7Z+a1vpqAoNCsL3bMp63kqR0V0mUiq3TIlVw+e5c39X2" />
            </div>
```

```
                <div></div>
                        <input name="ResponseCipher" type="hidden" id="ResponseCipher"
value="irvN4mdV6jA0wmsSVq8yHv3F+2frLchvQpTuJj1r8lBMmYhP8ZJ5TmVGbdm1vPm91UEW3m89IgfgM5R+HjF8jwzskTX4sRExSbDv3szDdwRyUnAT9neieDJDXzdCmG6/+FhxlN/Lai0Dg5s7IWfGs
l+xNmsTr/4yQ//btMl1u6AM+Jyi2+tBwGPgjMgrG5hjnqpbKwsd5k7yELDdCQHLzkjFagKjYMyfXgaRlHX7rNBpiQSKqmhESZm/XrktySfOrf80jFQtUZq0iSHxVr83W55/QGhH0TFs+avcP4yVJPuwCju2bH0
Kkd6m3QtclygPM29mc2xlylxl/8SB3bgJ8ESwWJCImlvqXJa1rcSO8y5UYkn+QL5cI9iuAaEhGjlwoC3q/q6F9kQrSlnV4ExymTio0VLFUSN4Sx6GjV5E2lTT25ypspkp05FvQPZnp+8S" />
        </form>
                </body>
</html>
<script type="text/javascript">
                document.forms[0].action='https://il01addproc.propay.com/Return.aspx';
                document.forms[0].submit();
</script>
```