Lyra Payment Gateway

UPI Static QR Integration

Document version 1.0

HISTORY OF THE DOCUMENT

Version	Date	Comment
1.0	February 2021	Initial release for review set mid in orderInfo

TECHNICAL SUPPORT

For technical inquiries or support, you can reach us from Monday to Friday, 9am to 6pm

by phone:	+91 (022) 33864910 / 911
by email:	*support.pg.in@lyra.com*
from the Merchant Back Office:	(Menu: **Help** > **Contact support**)

For any support request, please provide your shop ID (8-digit number).

1. UPI Static QR Code

UPI Static QR Code provides a smooth checkout experience as it automatically launches the preferred UPI mobile app during payment.

Compared to the dynamic QR Code, the static QR does not encode any order information nor any transaction amount. It encodes a static reference to the merchant account and as such may be printed by the merchant and displayed at his/her desk counter.

In order to create a Static QR code, a URL needs to be generated using minimal specifications as directed by NPCI:

Parameter name	Description	Mandatory	Value
pa	Payee VPA	M	Merchant vpa, **to be shared by Lyra**
pn	Payee Name	M	Merchant name
tr	Transaction ref id	O	Static order reference, as per merchant. Optional for static QR code
mode	Transaction mode	M	01 = QR code
orgid	ID origination	M	000000

Sample URL:

upi://pay?pa=lyra.p000999.p0001234@bankname&pn=SuperRetailStore&
&mode=01&orgid=000000

Once the URL is created it can be encoded into a QR code by any suitable library.

Sample QR code:

2. Webhook notification

The merchant registers a webhook url to receive HTTP POST notifications from Lyra when the UPI payment is complete.

The webhook url must be publicly accessible.

2.1 Webhook content

The body of the webhook request contains a charge resource which holds payment information. The most important fields are described below.

Parameter name	Description	Value
uuid	Unique Charge ID	32 alphanumeric characters, e.g. 27019964b16b46d5b13795aaecff18ff
date	Charge creation date	e.g. 2021-01-08T10:10:21.329+00:00
status	Charge status	**PAID**
orderId	Order Id	randomly generated by Lyra
orderInfo	Order additional info	contains the **MID** used for the payment
currency	Currency	INR
paid	Amount paid by the customer, in paise	In paise, e.g. 56380 -> 563.80 INR
transactions	Array of payment transaction information	contains a single UPI transaction

Example of charge resource :


```
{
    "uuid": "67c6a38023554c639cf54f9e19fd747c",
    "date": "2021-02-04T17:25:30.000+00:00",
    "expiryDate": "2021-02-05T17:25:30.000+00:00",
    "status": "PAID",
    "orderId": "583508",
    "currency": "INR",
    "amount": 41095,
    "paid": 41095,
    "due": 41095,
    "refunded": 0,
    "customer": {
        "uid": null,
        "name": null,
        "phone": null,
        "email": "test@lyra-network.com",
        "address": null,
        "city": null,
        "state": null,
        "zip": null,
        "country": null
    },
    "orderInfo": "mid=p0001234",
```

```
    "attempts": 1,
    "testMode": true,
    "dropReason": null,
    "paymentLink": "https://api.in.lyra.com/charge/67c6a38023554c639cf54f9e19fd747c",
    "transactions": [
        {
            "payment_option": "UPI",
            "status": "ACCEPTED",
            "date": "2021-02-04T17:25:39.000+00:00"
        }
    ]
}
```


The merchant shall do the following:

1. check that the **status** is PAID
2. extract the amount from the **paid** field
3. extract the merchant vpa from the **orderInfo** field

2.2 Webhook security

Since the webhook url is a public url it must be protected against malicious use or data tampering.

The request is **signed** following the *Digest Headers Drat* and the *HTTP Message Signature draft* from the IETF (Internet Engineering Task Force).

References:

HTTP Signature: https://tools.ietf.org/html/draft-ietf-httpbis-message-signatures-01
HTTP Digest Headers: https://tools.ietf.org/html/draft-ietf-httpbis-digest-headers-04
HTTP Semantics: https://tools.ietf.org/html/draft-ietf-httpbis-semantics-12

Basically a digest is computed from the request body and the digest is signed with the shop key. The digest and the signature are placed in HTTP headers.

e.g.

```
POST /
HTTP Headers
content-type: 'application/json'
digest: 'SHA-256=/U+c6wqyNUmaDzlT6MxMHDE+w1FRyiCAvAqsljnv8Jw='
signature: 'v1=:3/bA+uT86y1hQVI1beH6txZGIWrCBNTeOIwfU9aF1no=:'
signature-input: 'v1=(*created content-type digest); alg=hmac-sha256;
keyid="44247028.test"; created=1610717375'
HTTP Body
{"uuid":"7f4b634a78054183b91fb06ade5549cd","date":"Jan 15, 2021 6:59:34
PM","expiryDate":"Jan 16, 2021 6:59:34
PM","status":"PAID","orderId":"fv9sfjzw","currency":"INR","amount":836991,"paid":836991,"d
ue":836991,"refunded":0,"customer":{"uid":"customer1234","name":"Payzen
```

```
Customer","phone":"2554562523","email":"emailId@emailId.com"},"attempts":1,"testMode":true
}
```


The merchant must cross-check the digest and the signature to guarantee the integrity of the data.

Failure to check the signature will expose the merchant to data tampering and/or fraudulent use of the webhook.

2.3 Cross-check the request digest

1. Compute the SHA-256 digest from the request body
2. Convert the digest to BASE64 character string
3. Compare the result with the value of the **Digest** HTTP header. It should match.

Pseudo-code

```
val checksum = sha256(request.body)
request.headers.Digest = 'SHA-256=' + base64(checksum)
```

2.4 Cross-check the request signature

1. Extract the **keyId** value from the **signature-input** HTTP Header. It indicates the shop id and the test environment which was used to sign the request, e.g. keyId="**44247028.test**" for test transaction on shop 44247028 and keyId="**44247028.prod**" for live transaction.
2. Extract the **created** value from the **signature-input** HTTP Header. It indicates the UNIX timestamp which was used to sign the request, e.g. created=**1610717375**
3. Extract the value of the **content-type** HTTP Header, e.g. **application/json**
4. Take the digest value computed in the previous step or from the **digest** HTTP header.
5. Concatenate the timestamp, the content type and the digest with the exact format detailed in the pseudo-code below
6. Sign the concatenated string with HMAC-SHA-256 algorithm and the shop key. Make sure to use the correct test or production shop key.
7. Convert the result to BASE64 character string
8. Compare the result with the value of the **signature** HTTP header. It should match.

Pseudo-code

```
val shopId = request.headers['signature-input'].keyId
val timestamp = request.headers['signature-input'].created
val contentType = request.headers['content-type']
val digest = request.headers['digest']

val toSign = '*created: ' + timestamp + '\n'
             + 'content-type: ' + contentType + '\n'
```

```
             + 'digest: ' + digest
val signature = hmacSha256(toSign, shop.key)

request.headers['Signature'] = 'v1=:' + base64(signature) + ':'
```

2.5 Signature kit

A kit is readily available to verify the signature in different languages (javascript, typescript, PHP, etc.). Please get in touch with our support team.