MagikPay Payout API – Full Integration Guide (A → Z)

1. API Endpoint

POST https://api.magikpay.io/transaction/payout

2. Required Request Fields

Field	Type	Required	Description
apiKey	String	✅	Your API key issued by MagikPay
timestamp	String	✅	Current UNIX timestamp (milliseconds)
beneAccNum	String	✅	Beneficiary account number
beneIfscCode	String	✅	Beneficiary IFSC code
beneName	String	✅	Beneficiary full name
txnPaymode	String	✅	Payment mode (`IMPS`, `NEFT`, `RTGS`)
txnAmount	String	✅	Amount in decimal format (e.g., `"10.00"`)
orderid	String	✅	Your unique transaction/order ID
signature	String	✅	Base64 Ed25519 signature

3. Signature Generation

Canonical String Format:
apiKey|timestamp|beneAccNum|beneIfscCode|beneName|txnPaymode|txnAmount|orderid

Steps:
1. Generate Ed25519 private key (PKCS#8 format).
2. Extract 32-byte seed.
3. Sign canonical string with Ed25519.
4. Encode signature in Base64.
5. Send signature in request.

4. Integration Examples

◇ Node.js

```
import nacl from "tweetnacl";
import { Buffer } from "buffer";
```

```
const encoder = new TextEncoder();

function buildCanonical(payload) {
  return [
    payload.apiKey,
    payload.timestamp,
    payload.beneAccNum,
    payload.beneIfscCode,
    payload.beneName,
    payload.txnPaymode,
    payload.txnAmount,
    payload.orderid,
  ].join("|");
}

function signPayload(payload, seed32Hex) {
  const seed32 = Buffer.from(seed32Hex, "hex"); // 32-byte hex seed
  const { secretKey } = nacl.sign.keyPair.fromSeed(seed32);

  const canonical = buildCanonical(payload);
  const signature = nacl.sign.detached(encoder.encode(canonical), secretKey);

  return Buffer.from(signature).toString("base64");
}

// Example usage:
const payload = {
  apiKey: "a2b1...f79",
  timestamp: Date.now().toString(),
  beneAccNum: "00000000000000",
  beneIfscCode: "KK000000000",
  beneName: "John Doe",
  txnPaymode: "IMPS",
  txnAmount: "10.00",
  orderid: "ORD12345"
};

payload.signature = signPayload(payload, "YOUR_32BYTE_SEED_HEX");

console.log(payload);
```

◇ Python

```
import base64, time
from nacl.signing import SigningKey

def build_canonical(payload):
    return "|".join([
        payload.get("apiKey",""),
        payload.get("timestamp",""),
        payload.get("beneAccNum",""),
        payload.get("beneIfscCode",""),
        payload.get("beneName",""),
        payload.get("txnPaymode",""),
```

```
        payload.get("txnAmount",""),
        payload.get("orderid","")
    ])

seed32 = bytes.fromhex("YOUR_32BYTE_SEED_HEX")
signing_key = SigningKey(seed32)

payload = {
    "apiKey": "a2b1...f79",
    "timestamp": str(int(time.time() * 1000)),
    "beneAccNum": "00000000000000",
    "beneIfscCode": "KK000000000",
    "beneName": "John Doe",
    "txnPaymode": "IMPS",
    "txnAmount": "10.00",
    "orderid": "ORD12345"
}

canonical = build_canonical(payload).encode()
signature = signing_key.sign(canonical).signature
payload["signature"] = base64.b64encode(signature).decode()

print(payload)
```

◇ PHP

```
<?php
$payload = [
  "apiKey" => "a2b1...f79",
  "timestamp" => round(microtime(true) * 1000),
  "beneAccNum" => "00000000000000",
  "beneIfscCode" => "KK000000000",
  "beneName" => "John Doe",
  "txnPaymode" => "IMPS",
  "txnAmount" => "10.00",
  "orderid" => "ORD12345"
];

$canonical = implode("|", [
  $payload["apiKey"],
  $payload["timestamp"],
  $payload["beneAccNum"],
  $payload["beneIfscCode"],
  $payload["beneName"],
  $payload["txnPaymode"],
  $payload["txnAmount"],
  $payload["orderid"]
]);

$seed = hex2bin("YOUR_32BYTE_SEED_HEX");
$keypair = sodium_crypto_sign_seed_keypair($seed);
$signature = sodium_crypto_sign_detached($canonical, $keypair);

$payload["signature"] = base64_encode($signature);
```

```
echo json_encode($payload, JSON_PRETTY_PRINT);
```

Java

```
import java.nio.charset.StandardCharsets;
import java.util.Base64;
import org.bouncycastle.crypto.signers.Ed25519Signer;
import org.bouncycastle.crypto.params.Ed25519PrivateKeyParameters;

public class MagikPaySign {
    public static void main(String[] args) {
        String canonical = String.join("|",
            "a2b1...f79", "1756272093598", "",
            "00000000000000", "KK000000000", "John Doe", "IMPS", "10.00",
"ORD12345"
        );

        byte[] seed = hexStringToByteArray("YOUR_32BYTE_SEED_HEX");
        Ed25519PrivateKeyParameters privKey = new
Ed25519PrivateKeyParameters(seed, 0);

        Ed25519Signer signer = new Ed25519Signer();
        signer.init(true, privKey);
        signer.update(canonical.getBytes(StandardCharsets.UTF_8), 0,
canonical.length());

        byte[] signature = signer.generateSignature();
        String signatureBase64 = Base64.getEncoder().encodeToString(signature);

        System.out.println("Signature: " + signatureBase64);
    }

    private static byte[] hexStringToByteArray(String s) {
        int len = s.length();
        byte[] data = new byte[len / 2];
        for (int i = 0; i < len; i += 2) {
            data[i / 2] = (byte) ((Character.digit(s.charAt(i), 16) << 4)
                                 + Character.digit(s.charAt(i+1), 16));
        }
        return data;
    }
}
```

cURL

```
curl -X POST https://api.magikpay.io/transaction/payout \
  -H "Content-Type: application/json" \
  -d '{
    "apiKey": "a2b1...f79",
    "timestamp": "1756272093598",
    "beneAccNum": "00000000000000",
    "beneIfscCode": "KK000000000",
    "beneName": "John Doe",
    "txnPaymode": "IMPS",
    "txnAmount": "10.00",
```

```
    "orderid": "ORD12345",
    "signature": "BASE64_SIGNATURE"
  }'
```

5. Response Format

```
{
  "result": 1,
  "message": "Bank Transfer has Successful",
  "data": [
    {
      "order_id": "ORD12345",
      "tnx_id": "TXN987654321",
      "beneName": "John Doe",
      "beneIfscCode": "KK000000000",
      "beneAccNum": "00000000000000",
      "status": "SUCCESS",
      "utr": "UTR20250101001",
      "txnAmount": "10.00",
      "txnPaymode": "IMPS",
      "charge": "5",
      "tds": 0,
      "gst": 0,
      "txn_date": "2025-08-28 15:45:00"
    }
  ]
}
```

6. Key Generation

Use Binance's Asymmetric Key Generator to generate Ed25519 key pairs:

https://github.com/binance/asymmetric-key-generator/releases/download/v1.1.2/AsymmetricKeyGenerator-setup-v1.1.2.exe