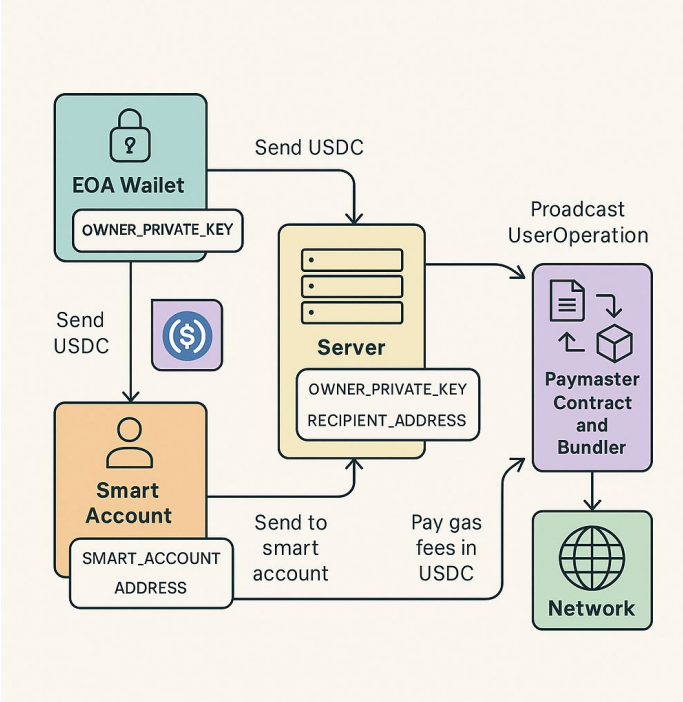
Circle Paymaster Instant USDC Gas Sponsorship: End-to-End User Guide

This AMI launches a ready-to-go Ubuntu server that pays blockchain gas fees (in USDC) on behalf of your users. You fund it once, and it handles the fees for your app or contract — making the experience gas-free for your end users!

Follow these steps for smooth setup, funding, and integration. Here is a basic flow chart.



Legend:

- **EOA Wallet:** Your MetaMask or other wallet (holds your OWNER_PRIVATE_KEY and **funds**. Be sure to fund your account).
- **Smart Account:** Special blockchain account holding USDC for paying user gas fees.
- **Server:** Your AWS EC2 instance running this AMI and the Paymaster software.
- **Paymaster/Bundler:** Circle's on-chain service that actually pays the gas with your USDC.
- **Blockchain:** The Arbitrum Sepolia network, where the transaction happens.

Important Note:
Be sure to fund your "Owner Account" (such as your MetaMask wallet) with enough Ethereum (ETH) and USDC. This funding allows the Circle Paymaster server to pay blockchain gas fees and sponsor user transactions on your behalf.

1. Launch & Access the AMI

1. **Launch** the AMI via 1-Click from AWS Marketplace.
2. Wait until the instance status is **Running** and passes all health checks.
3. **SSH** into your instance as ubuntu:

   ssh -i <your-key.pem> ubuntu@<AMI_PUBLIC_DNS>

4. **Update** system packages:

   sudo apt update && sudo apt upgrade -y

****Understanding your Keys Ledger. You will need this going forward.**

- **OWNER_PRIVATE_KEY:**
  For example in MetaMask:
  Account Details> Details> Show Private Key> Copy (Include prefix with 0x).
- **RECIPIENT_ADDRESS: (destination Wallet)**
  For example in MetaMask: Is your "Account Address".
- **SMART_ACCOUNT_ADDRESS:**
  Generate it with the script below.

1. **Configure Environment Variables.** *See Key Ledger below for explanations:*

Edit the **.env** file to provide your keys & addresses:

cd /opt/circle-paymaster

sudo nano .env

Replace **only** these three lines:

- OWNER_PRIVATE_KEY=YOUR_PRIVATE_KEY_HERE
- RECIPIENT_ADDRESS=YOUR_RECIPIENT_ADDRESS_HERE
- SMART_ACCOUNT_ADDRESS=YOUR_SMART_ACCOUNT_HERE

The other two are preset for Arbitrum Sepolia:

PAYMASTER_V07_ADDRESS=0x31BE08D380A21fc740883c0BC434FcFc88740b58
USDC_ADDRESS=0x75faf114eafb1BDbe2F0316DF893fd58CE46AA4d

Save (**Ctrl+O, Enter**), exit (**Ctrl+X**), then secure:

sudo chmod 600 /opt/circle-paymaster/.env

Get Your SMART_ACCOUNT_ADDRESS

Create and run this script in **/opt/circle-paymaster**

Make sure **OWNER_PRIVATE_KEY** is set in **.env** first

Run:

nano generate-account.mjs

Paste:

```
import 'dotenv/config';
import { createPublicClient, http } from 'viem';
import { arbitrumSepolia } from 'viem/chains';
import { privateKeyToAccount } from 'viem/accounts';
import { toCircleSmartAccount } from '@circle-fin/modular-wallets-core';

(async () => {
  const client = createPublicClient({ chain: arbitrumSepolia, transport: http() });
  const owner = privateKeyToAccount('0x' + process.env.OWNER_PRIVATE_KEY.replace(/^0x/,
''));
  const account = await toCircleSmartAccount({ client, owner });
  console.log('SMART_ACCOUNT_ADDRESS=' + account.address);
})();
```

Save and exit.

Run:

node generate-account.mjs

Copy the output and put it into your .env above.

Start & Verify the Paymaster Service

1. **Manual end-to-end**:

   cd /opt/circle-paymaster

   node index.js

   On success, you'll see:

   UserOperation hash: 0x...
   Tx hash: 0x...

2. **Enable auto-start**:

   sudo systemctl daemon-reload

sudo systemctl enable circle-paymaster.service

sudo systemctl start circle-paymaster.service

3. **Monitor logs**:

   sudo journalctl -u circle-paymaster.service -f

Look for balance checks and sponsored transaction output.

Fund Your Server (Testnet) (For example)

ETH: Request testnet ETH at:
https://faucets.chain.link/arbitrum-sepolia

USDC: Request testnet USDC at:
https://faucet.circle.com/
(Select Arbitrum Sepolia, use your EOA address.)

Transfer at least 1 USDC from your EOA (wallet) to your **SMART_ACCOUNT_ADDRESS** (from above) using MetaMask.

Check the balance:

node check-balance.js

You should see: USDC balance: 1 (or more)

---

5. 24/7 Uptime & Maintenance

- **Always-on**: Keep the instance running (no auto-stop).
- **Auto-restart**: *Restart=on-failure* ensures the service recovers.
- **Monitoring**: Use CloudWatch alarms on instance health and logs.

Helpful Resources

- **Paymaster Overview**: https://developers.circle.com/stablecoins/paymaster-overview
- **Paymaster Addresses & Docs**: https://developers.circle.com/stablecoins/paymaster-addresses

---

You're now ready to provide **gas-free**, USDC-sponsored blockchain transactions via this Ubuntu AMI!

AWS Data

- Data Encryption Configuration: This solution does not encrypt data within the running instance.
- User Credentials are stored: /root/.ssh/authorized_keys & /home/ubuntu/.ssh/authorized_keys
- Monitor the health:
    - Navigate to your Amazon EC2 console and verify that you're in the correct region.
    - Choose Instance and select your launched instance.
    - Select the server to display your metadata page and choose the Status checks tab at the bottom of the page to review if your status checks passed or failed.

Extra Information: (Optional)

Allocate Elastic IP

To ensure that your instance **keeps its IP during restarts** that might happen, configure an Elastic IP. From the EC2 console:

1. Select ELASTIC IPs.
2. Click on the ALLOCATE ELASTIC IP ADDRESS.
3. Select the default (Amazon pool of IPv4 addresses) and click on ALLOCATE.
4. From the ACTIONS pull down, select ASSOCIATE ELASTIC IP ADDRESS.
5. In the box that comes up, note down the Elastic IP Address, which will be needed when you configure your DNS.
6. In the search box under INSTANCE, click and find your INSTANCE ID and then click ASSOCIATE.
7. Your instance now has an elastic IP associated with it.
8. For additional help: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/elastic-ip-addresses-eip.html