# 🔒 BlockReceipt.ai – Audit & Action Plan: Receipt→Encrypt→Mint Integration

## 1. Current Code Assessment

- \*\*uploadReceipt.ts\*\* only handles file storage and responds with success; no processing after upload.  
- \*\*autoProcessReceipt.ts\*\* implements OCR, encryption, and task queueing, but is separate from upload route and uses mock IPFS (no pin to actual IPFS).  
- \*\*TACo encryption\*\* is available via `encryptLineItems` in `tpre.service.ts`, but not consistently invoked in the upload workflow.  
- \*\*IPFS pinning\*\* is mocked in `ipfsService.ts`; no real metadata URIs are used in minting.  
- \*\*Minting endpoint\*\* (`routes/nfts.ts`) might still reference mock services and not called from upload.  
- \*\*Frontend\*\* does not pass `recipientPublicKey` or `walletAddress` into upload; gallery component is not auto-rendered on main page.

## 2. Required Workflow Integration

Unify and streamline into a single `/api/upload-and-mint` route that:  
1. \*\*Requires\*\* `walletAddress` and `recipientPublicKey`.  
2. \*\*Extracts\*\* receipt data via OCR.  
3. \*\*Encrypts\*\* metadata via TACo PRE.  
4. \*\*Pins\*\* encrypted JSON to IPFS (use real Pinata/NFT.Storage).  
5. \*\*Mints\*\* NFT to `walletAddress` with `metadataUri` pointing to pinned JSON.  
6. \*\*Stores\*\* encrypted payload in DB via `metadataService`.  
7. \*\*Returns\*\* `tokenId`, `metadataUri`, and `txHash` in response.

## 3. Code Changes

### 3.1 New Unified Route

File: `server/routes/uploadAndMint.ts`

```ts  
import express from 'express';  
import multer from 'multer';  
import fs from 'fs';  
import path from 'path';  
import { extractReceiptData } from '../services/ocrService';  
import { encryptLineItems } from '../services/tpre.service';  
import { pinJSON } from '../services/ipfsService';  
import { metadataService } from '../services/metadataService';  
import ERC1155\_ABI from '../abi/BlockReceiptCollection.json';  
import { ethers } from 'ethers';  
  
const router = express.Router();  
const upload = multer({ storage: multer.memoryStorage() }).single('receipt');  
  
router.post('/upload-and-mint', upload, async (req, res) => {  
 const { walletAddress, recipientPublicKey } = req.body;  
 if (!walletAddress || !recipientPublicKey) {  
 return res.status(400).json({ success:false, message:'Require walletAddress and recipientPublicKey' });  
 }  
 try {  
 // 1. OCR  
 const receiptData = await extractReceiptData(req.file.buffer);  
  
 // 2. TACo Encryption  
 const encrypted = await encryptLineItems(recipientPublicKey, receiptData.items);  
  
 // 3. IPFS Pin  
 const { cid, url: metadataUri } = await pinJSON({  
 name: `Receipt from ${receiptData.merchant}`,  
 description: `Encrypted receipt NFT minted for ${walletAddress}`,  
 encrypted  
 });  
  
 // 4. Mint NFT  
 const provider = new ethers.providers.JsonRpcProvider(process.env.POLYGON\_RPC\_URL);  
 const devWallet = new ethers.Wallet(process.env.PRIVATE\_KEY, provider);  
 const contract = new ethers.Contract(process.env.CONTRACT\_ADDRESS, ERC1155\_ABI, devWallet);  
 const tokenId = Date.now();  
 const tx = await contract.mint(walletAddress, tokenId);  
 await tx.wait();  
  
 // 5. Store encrypted metadata off-chain  
 await metadataService.storeEncryptedMetadata(  
 tokenId.toString(),  
 walletAddress,  
 JSON.stringify(encrypted),  
 { merchant: receiptData.merchant, date: receiptData.date, total: receiptData.total }  
 );  
  
 // 6. Respond  
 return res.json({ success:true, tokenId, metadataUri, txHash: tx.hash });  
 } catch (error: any) {  
 console.error('uploadAndMint error:', error);  
 return res.status(500).json({ success:false, message: error.message });  
 }  
});  
  
export default router;  
```

### 3.2 Frontend Integration

a) Pass required fields in upload component:

```tsx  
// In ReceiptUpload.jsx  
const handleChange = async (e) => {  
 const file = e.target.files[0];  
 const form = new FormData();  
 form.append('receipt', file);  
 form.append('walletAddress', walletAddress);  
 form.append('recipientPublicKey', recipientPublicKey); // fetched on login  
 const res = await fetch('/api/upload-and-mint', { method:'POST', body: form });  
 const data = await res.json();  
 setMintResult(data);  
};  
```

b) Render NFT Gallery on main page:

```tsx  
// In App.tsx or MainLayout.tsx  
import ReceiptGallery from './components/ReceiptGallery';  
function App() {  
 const { address } = useWallet();  
 return (  
 <div>  
 <ReceiptUpload />  
 <NFTSelection />  
 <ReceiptGallery walletAddress={address} />  
 </div>  
 );  
}  
```

## 4. Next Steps for Replit

1. Add `uploadAndMint.ts` route to `server/routes` and register in `app.ts`.  
2. Remove old `uploadReceipt` and `autoProcessReceipt` endpoints.  
3. Ensure `tpre.service.ts`, `ipfsService.ts`, and `metadataService.ts` are imported.  
4. Update frontend `ReceiptUpload` to call `/api/upload-and-mint`.  
5. Test full flow: connect wallet → upload → auto-encrypt → mint → gallery shows NFT.

## 5. Conclusion

These changes will unify your flow, enforce encryption, pin metadata, and enable real on-chain NFT minting in one cohesive endpoint. Drop this doc into Replit to implement the final integration for a secure, seamless user experience.