SuchPrivacy

Challenge files:

- `hint.json`
- `main.js`
- `package.json`
  Category: `Crypto`

Write up

With the given script we got a json files containing the 5 previous `Math.random()` results.
`Math.random` in nodejs use the xorshift128+ cipher that is well explained here
With few internet research we can find that z3 can reverse the node.js Math.random

My solving script:

```
const { CURVE, Point } = require('@noble/secp256k1');
const { BigNumber } = require('ethers');
const { keccak256, arrayify, getAddress, hexDataSlice } =
require('ethers/lib/utils');
const { init } = require('z3-solver');
const { readFileSync, writeFileSync } = require('node:fs');
const { createHash } = require('node:crypto')



const bufferToU64 = (a) => a.readBigUInt64LE(0);

const bufferToDouble = (a) => a.readDoubleLE(0);

const u64ToBuffer = (a) => {
    const buf = Buffer.alloc(8);
    buf.writeBigInt64LE(a, 0);
    return buf;
};

const doubleToBuffer = (a) => {
    const buf = Buffer.alloc(8);
    buf.writeDoubleLE(a, 0);
    return buf;
};

const getValue = (a, m) => m.get(a).value();
```

```
const getDeclsValues = (model)=>
  model.decls().reduce((a, c) => {
    a[c.name()] = getValue(c, model);
    return a;
}, {});

(async () => {
    const RAND_SUITE = Object.freeze(JSON.parse(readFileSync('hint.json',
{encoding: 'utf-8'})).reverse());
    const api = await init();
    const { Solver, BitVec } = api.Context('main');
    const solver = new Solver();

    console.info('Random suite to break', [...RAND_SUITE].reverse());

    let [seState0, seState1] = BitVec.consts('seState0 seState1', 64);

    RAND_SUITE.forEach((rand) => {
        let seS1 = seState0;
        let seS0 = seState1;

        seState0 = seS0;
        seS1 = seS1.xor(seS1.shl(23));
        seS1 = seS1.xor(seS1.lshr(17));
        seS1 = seS1.xor(seS0);
        seS1 = seS1.xor(seS0.lshr(26));
        seState1 = seS1;

        const doubleBuf = doubleToBuffer(rand + 1);
        const u64 = bufferToU64(doubleBuf);

        solver.add(seState0.lshr(12).eq(BitVec.val(u64 & ((1n << 52n) - 1n), 64)));
    });

    console.info('Begin solving');
    while ((await solver.check()) !== 'unsat') {
        const model = solver.model();
        const states = getDeclsValues(model);
        const u64 = (states.seState0 >> 12n) | 0x3ff0000000000000n;
        const prediction = bufferToDouble(u64ToBuffer(u64)) - 1;
        console.log({
            states,
            prediction,
        });
```

```
        const seedKey = arrayify(BigNumber.from(keccak256(Math.floor(prediction *
Number.MAX_SAFE_INTEGER))).add(BigNumber.from('0x260026002600260026002600').mul(0x2
600n)).mod(CURVE.n));
        let newPoint = Point.fromPrivateKey(seedKey);
        for (let i = 1; ; i++) {
            newPoint = newPoint.add(Point.BASE);
            const newAddress = hexDataSlice(keccak256(hexDataSlice('0x' +
newPoint.toHex(), 1)), 12);
            if (newAddress.startsWith('0x2600')) {
                writeFileSync('flag.json', JSON.stringify({
                    privateKey: BigNumber.from(seedKey).add(i).toHexString(),
                    publicKey: getAddress(newAddress)
                }, null, '\t\v\n\r\f'), {encoding: 'utf-8'});
                const flag =
createHash("md5").update(readFileSync('flag.json')).digest("hex");
                console.info(`The flag is PWNME{${flag}}`)
                process.exit();
            }
      }
    }
    console.error('NO SOLUTION');
})();
```

That give the flag: `PWNME{85cf83b0a54bdc2a5f0eaf57be3994ef}`