```
const express=require("express");
const request=require("request");
const NodeRSA=require("node-rsa");
const crypto=require("crypto");
const blocks=require("./blocks.js")

let curBlock=blocks.makeTestBlock();

const internal=express();
const external=express();

let keys=new NodeRSA({b: 512})
const fingerprint=crypto.createHash("sha256").update(keys.exportKey("public")).digest("base64")
console.log("Generated public and private keys. Public key fingerprint: "+fingerprint)

/**
* Emulates Java's String->hashCode to a certain degree. Will create a number that is based off of the string passed to it.
* @returns Number
*/
function numerify(str)
{
	let num=0;
	let buf=Buffer.from(str);
	for(let i=0;i<str.length;i++){
		num+=buf[i]*61^i
	}
	return num;
}
```

```
/**
* For use in mining cards. Will produce a hash that is based off of the 3 key constraints in order to meet card privacy, security, and entropy requirements
* @param cardNonce The value to use as a nonce when calculating the card
* @returns String that is base64 encoded and has the hash value of the card. Could have returned a buffer, but for simplicity everything is being base64'd.
*/
function hashCard(cardNonce)
{
	return crypto.createHash("sha512").update(fingerprint+curBlock.prevHash+cardNonce).digest().toString("base64");
}

/**
* Will mine until it finds a valid card.
* @returns Object with keys hash, salt, and cardID
*/
function drawCard()
{
	let salt=0;
	while(!hashCard(salt).startsWith("CRD")) salt+=Math.random();//add a random number every time to make it less easy to guess the salt. Later, we'll make this crypto.randomBytes. Otherwise, it would take the same amount of time to find the card because the salt would be the lowest incremental number that works
	return {hash:hashCard(salt),salt:salt,cardID:numerify(hashCard(salt))%52}
}
```

```
/**
* For use in debugging and in programs that depend on this.
* @param cardID Number between 0 and 51 that represents a card.
* @returns String that is a human-readable representation of the card
*/
function getCardString(cardID)
{
    if(cardID<0 || cardID>51) throw "cardID "+cardID+" out of bounds!";

    let str;
    if(cardID%13==0) str="Ace of "
    else if(cardID%13<10) str=(cardID%13)+1+" of ";
    else if(cardID%13==10) str="Jack of "
    else if(cardID%13==11) str="Queen of "
    else if(cardID%13==12) str="King of "

    switch(Math.floor(cardID/13))
    {
        case 0:
            str+="Spades"
            break;
        case 1:
            str+="Clovers"
            break;
        case 2:
            str+="Hearts"
            break;
        case 3:
            str+="Diamonds"
            break;
        default:
            throw "cardID "+cardID+" above bounds, something is wrong"
            break;
    }
    return str;
}
```

```
let card=drawCard()

console.log("Drew card "+getCardString(card.cardID)+" with hash: "+card.hash+" and salt: "+card.salt)
```