coinbin 源码解读

> coinbin 项目地址: https://github.com/coinbin/coinbin
> A Open Source Browser Based Bitcoin Wallet.
> Coinb.in supports a number of key features such as:
>
> - Offline Compressed & uncompressed Address creation.
> - Offline Multisignature Address creation.
> - Supports compressed and uncompressed keys.
> - "In browser" Key (re)generation.
> - Ability to decode transactions, redeem scripts and more offline.
> - Send and receive payments.
> - Build custom transactions offline.
> - Sign transactions offline.
> - Broadcast custom transactions.

首先看文件结构:

```
.
|--css/
|--fonts/
|--js/
|---|--aes.js
|---|--bootstrap.min.js
|---|--bootstrap-datetimepicker.js
|---|--coin.js
|---|--coinbin.js
|---|--collapse.js
|---|--crypto.min.js
|---|--crypto-sha256.js
|---|--crypto-sha256-hmac.js
|---|--ellipticcurve.js
|---|--jquery-1.9.1.min.js
|---|--jsbn.js
|---|--moment.min.js
|---|--qcode-decoder.min.js
|---|--qrcode.js
|---|--ripemd160.js
|---|--sha512.js
|---|--transition.js
```

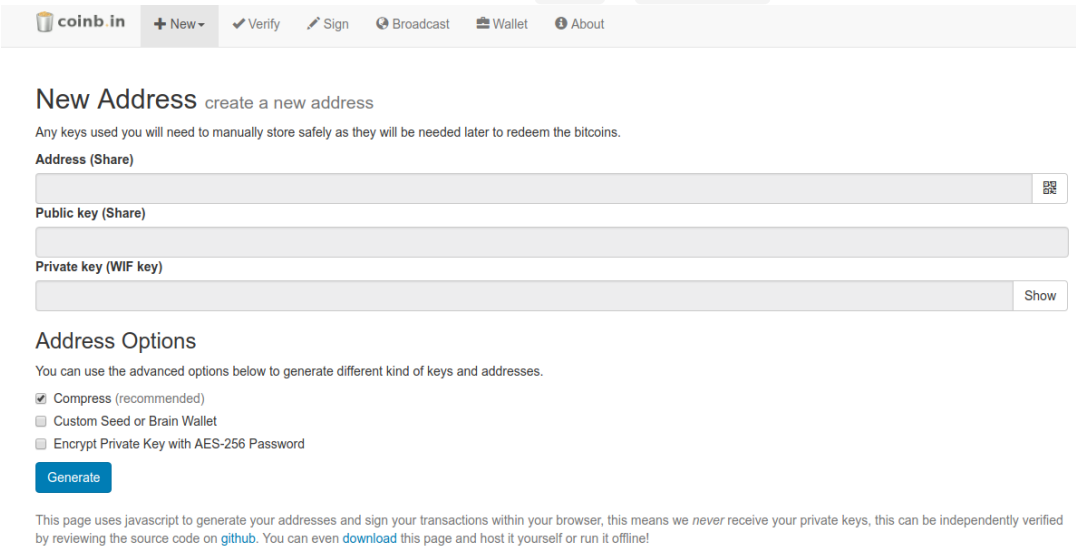
```
|--images/
|--index.html
|--LICENSE
|--README.md
|--sha1sum
```

主要文件是 index.html 和 js 文件夹中的
coin.js,coinbin.js,crypto.*.js,ripemd160.js
index.html 主要是使用 bootstrap 组件来编写, 然后在 coinbin.js 里绑定事件, 触发事件调用 coin.js 中的函数, coin.js 中又会调用其他 js 文件中的辅助函数 (加密解密 / 编码解码 / 进制转换等)

下面看主要的几个功能:

功能 1: 生成 public key/private key

首先在浏览器中打开 index.html, 选择 `New` -> `address`



右键点击 Generate 按钮, 审查元素, 获得按钮的 id 为 `newKeysBtn`

`<input type="button" class="btn btn-primary" value="Generate" id="newKeysBtn" _vimium-has-onclick-listener=""data-original-title="" title="">`
回到 coinbin.js, `ctrl + F` 查找 `newKeysBtn`, 找到点击该按钮触发的函数

```
$("#newKeysBtn").click(function(){
        coinjs.compressed = false;
        if($("#newCompressed").is(":checked")){
            coinjs.compressed = true;
        }
```

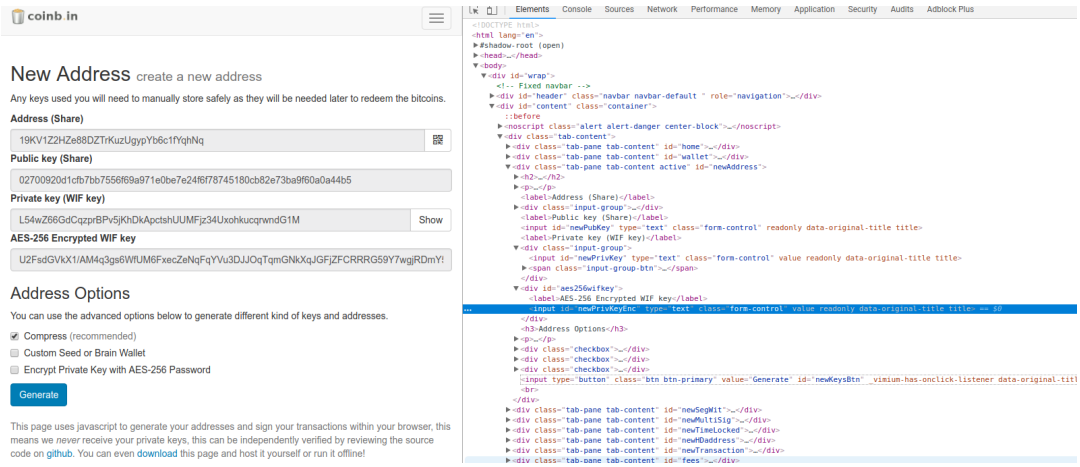
```
        var s = ($("#newBrainwallet").is(":checked")) ? $("#brai
        var coin = coinjs.newKeys(s);
        $("#newBitcoinAddress").val(coin.address);
        $("#newPubKey").val(coin.pubkey);
        $("#newPrivKey").val(coin.wif);

        /* encrypted key code */
        if((!$("#encryptKey").is(":checked")) || $("#aes256pass"
            $("#aes256passStatus").addClass("hidden");
            if($("#encryptKey").is(":checked")){
                $("#aes256wifkey").removeClass("hidden");
            }
        } else {
            $("#aes256passStatus").removeClass("hidden");
        }
        $("#newPrivKeyEnc").val(CryptoJS.AES.encrypt(coin.wif, $

    });
```

在该函数中, 调用了 coinjs.newKeys(s), 参数 s 在勾选 `brainwallet` 时有效, barinwallet 就相当于我们平时的密码, 在脑中想一个密码, 系统根据这个密码生成一个帐号, 如果不提供密码, 会直接生成帐号和密码 (公钥 / 私钥), 此处参数 s 为 null.
再看 coinjs 中的 newKey 函数,

```
/* generate a private and public keypair, with address and WIF a
    coinjs.newKeys = function(input){
        var privkey = (input) ? Crypto.SHA256(input) : this.newP
        var pubkey = this.newPubkey(privkey);
        return {
            'privkey': privkey,
            'pubkey': pubkey,
            'address': this.pubkey2address(pubkey),
            'wif': this.privkey2wif(privkey),
            'compressed': this.compressed
        };
    }
```

在这里, `input` 值为 `null`, 所以先调用 `newPrivatekey` 函数产生一个私钥, 然后调用 `newPubkey` 函数把私钥作为参数传进去, 得到公钥, 调用 `pubkey2address`, 传入 `pubkey` 得到钱包地址, 调用 `privkey2wif`, 传入 `privkey` 得到 `WIF` 地址 (Wallet Import Format), 把 (公钥 / 私钥 / 钱包地址 / wif 地址 / 是否压缩) 作为对象属性, 返回一个对象, 然后在 coinbin.js 中将钱包地址 / 公钥 / WIF 地址填到对应的文本框中

```
$("#newBitcoinAddress").val(coin.address);
$("#newPubKey").val(coin.pubkey);
$("#newPrivKey").val(coin.wif);
```



值得注意的是, 还有一个隐藏文本框, id 为 encryptKey, 只有在勾选上 `Encrypt Private Key with AES-256 Password` 的时候才会显示出来, 并且需要填写加密用的密码, 文本框的内容是使用密码加密之后的 WIF 地址.
现在来研究 newPrivKey,newPubKey,pubkey2address,privkey2wif 这几个函数

`newPrivKey`

```
/* generate a new random private key */
    coinjs.newPrivkey = function(){
        var x = window.location;
        x += (window.screen.height * window.screen.width * windo
        x += coinjs.random(64);
        x += (window.screen.availHeight * window.screen.availWid
        x += navigator.language;
        x += window.history.length;
        x += coinjs.random(64);
        x += navigator.userAgent;
        x += 'coinb.in';
        x += (Crypto.util.randomBytes(64)).join("");
        x += x.length;
        var dateObj = new Date();
        x += dateObj.getTimezoneOffset();
        x += coinjs.random(64);
        x += (document.getElementById("entropybucket")) ? docume
        x += x+''+x;
        var r = x;
        for(i=0;i<(x).length/25;i++){
            r = Crypto.SHA256(r.concat(x));
```

```
        }
        var checkrBigInt = new BigInteger(r);
        var orderBigInt = new BigInteger("fffffffffffffffffffff
        while (checkrBigInt.compareTo(orderBigInt) >= 0 || check
            r = Crypto.SHA256(r.concat(x));
            checkrBigInt = new BigInteger(r);
        }
        return r;
    }
```

有没有觉得稀里糊涂的? 感觉上是随机获取各种字符串, 再通过一定的密码学变换, 最后通过 hash256 得到结果, 看了源码也不太知道到底干了些什么, 于是打开浏览器的控制台, 运行一下

```
priv_key = coinjs.newPrivKey()
console.log(priv_key)
console.log(typeof priv_key)
```

输出

`19bcd3173cb7480356ba16d47c6d6e2ce04c9ea9d358ba39b58a4bbce8b92a44`
`string`

一共 64 位十六进制字符

也就是说上面这个函数是用于随机生成 64 位十六进制字符

更为科学的解释 (参考链接):

> 生成密钥的第一步也是最重要的一步， 是要找到足够安全的熵源， 即随机性来源。 生成一个比特币私钥在本质上与 “在 1 到 2^256 之间选一个数字” 无异。 只要选取的结果是不可预测或不可重复的， 那么选取数字的具体方法并不重要。 比特币软件使用操作系统底层的随机数生成器来产生 256 位的熵（ 随机性 ）。 通常情况下， 操作系统随机数生成器由人工的随机源进行初始化， 也可能需要通过几秒钟内不停晃动鼠标等方式进行初始化。 对于真正的偏执狂， 可以使用掷骰子的方法， 并用铅笔和纸记录。
>
> 私钥可以是 1 和 n-1 之间的任何数字， 其中 n 是一个常数（ n=1.158*1077， 略小于 2^256 ）， 并由比特币所使用的椭圆曲线的阶所定义。要生成这样的一个私钥， 我们随机选择一个 256 位的数字， 并检查它是否小于 n-1。 从编程的角度来看， 一般是通过在一个密码学安全的随机源中取出一长串随机字节， 对其使用 SHA256 哈希算法进

> 行运算， 这样就可以方便地产生一个 256 位的数字。 如果运算结果小于 n-1， 我们就有了一个合适的私钥。 否则， 我们就用另一个随机数再重复一次。

`newPubkey`

```
/* generate a public key from a private key */
    coinjs.newPubkey = function(hash){
        var privateKeyBigInt = BigInteger.fromByteArrayUnsigned(
        var curve = EllipticCurve.getSECCurveByName("secp256k1")

        var curvePt = curve.getG().multiply(privateKeyBigInt);
        var x = curvePt.getX().toBigInteger();
        var y = curvePt.getY().toBigInteger();

        var publicKeyBytes = EllipticCurve.integerToBytes(x, 32)
        publicKeyBytes = publicKeyBytes.concat(EllipticCurve.int
        publicKeyBytes.unshift(0x04);

        if(coinjs.compressed==true){
            var publicKeyBytesCompressed = EllipticCurve.integer
            if (y.isEven()){
                publicKeyBytesCompressed.unshift(0x02)
            } else {
                publicKeyBytesCompressed.unshift(0x03)
            }
            return Crypto.util.bytesToHex(publicKeyBytesCompress
        } else {
            return Crypto.util.bytesToHex(publicKeyBytes);
        }
    }
```

这是使用了椭圆曲线加密算法, 具体的介绍见 这里
另外一篇简单介绍椭圆曲线加密算法:

> 作者：知乎用户
> 链接：https://www.zhihu.com/question/26662683/answer/325511510
> 来源：知乎
> 著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。
>
> 椭圆曲线加密

考虑 K=kG ，其中 K、G 为椭圆曲线 Ep(a,b) 上的点，n 为 G 的阶（nG=O∞），k 为小于 n 的整数。则给定 k 和 G，根据加法法则，计算 K 很容易, 但反过来，给定 K 和 G，求 k 就非常困难。因为实际使用中的 ECC 原则上把 p 取得相当大，n 也相当大，要把 n 个解点逐一算出来列成上表是不可能的。这就是椭圆曲线加密算法的数学依据. 点 G 称为基点（base point）k（k<n）为私有密钥（privte key）K 为公开密钥（public key)

ECC 保密通信算法

- 1.Alice 选定一条椭圆曲线 E，并取椭圆曲线上一点作为基点 G 假设选定 E29(4,20)，基点 G(13,23) , 基点 G 的阶数 n=37
- 2.Alice 选择一个私有密钥 k（k<n），并生成公开密钥 K=kG 比如 25, K= kG = 25G = (14,6）
- 3.Alice 将 E 和点 K、G 传给 Bob
- 4.Bob 收到信息后，将待传输的明文编码到上的一点 M（编码方法略），并产生一个随机整数 r（r<n,n 为 G 的阶数） 假设 r=6 要加密的信息为 3, 因为 M 也要在 E29(4,20) 所以 M=(3,28)
- 5.Bob 计算点 C1=M+rK 和 C2=rG C1= M+6K= M+6*25*G=M+2G= (3,28)+(27,27)=(6,12) C2=6G=(5,7)
- 6.Bob 将 C1、C2 传给 Alice7.Alice 收到信息后，计算 C1-kC2，结果就应该是点 M C1-kC2 =(6,12)-25C2 =(6,12)-25*6G = (6,12)-2G =(6,12)-(27,27) =(6,12)+(27,2) =(3,28) 数学原来上能解密是因为: C1-kC2=M+rK-krG=M+rkG-krG-M

比特币系统选用的 secp256k1 中，参数为
p = 0xFFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFC2F = 2^256 – 2^32 – 2^9 – 2^8 – 2^7 – 2^6 – 2^4 – 1
a = 0，
b = 7
G= (0x79BE667EF9DCBBAC55A06295CE870B07029BFCDB2DCE28D959F2815B16F81798, 0x483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8)
n = 0xFFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE BAAEDCE6 AF48A03B BFD25E8C D0364141
h = 01

该函数作用:
Take the corresponding public key generated with it (65 bytes, 1 byte 0x04, 32 bytes corresponding to X coordinate, 32 bytes corresponding to Y coordinate)

`pubkey2address`

```
/* provide a public key and return address */
    coinjs.pubkey2address = function(h, byte){
        var r = ripemd160(Crypto.SHA256(Crypto.util.hexToBytes(h
        r.unshift(byte || coinjs.pub);
        var hash = Crypto.SHA256(Crypto.SHA256(r, {asBytes: true
        var checksum = hash.slice(0, 4);
        return coinjs.base58encode(r.concat(checksum));
    }
```

该函数作用:
1. Perform SHA-256 hashing on the public key
2. Perform RIPEMD-160 hashing on the result of SHA-256
3. Add version byte in front of RIPEMD-160 hash (0x00 for Main Network)
(note that below steps are the Base58Check encoding, which has multiple library options available implementing it)

`privkey2wif`

```
/* provide a privkey and return an WIF  */
    coinjs.privkey2wif = function(h){
        var r = Crypto.util.hexToBytes(h);
        if(coinjs.compressed==true){
            r.push(0x01);
        }
        r.unshift(coinjs.priv);
        var hash = Crypto.SHA256(Crypto.SHA256(r, {asBytes: true
        var checksum = hash.slice(0, 4);
        return coinjs.base58encode(r.concat(checksum));
    }
```

WIF 全称 (Wallet Import Format), 作用是导出导入密钥, 方便备份迁移.
Note: The WIF (Wallet Import Format) includes information about the network and if the associated public key is compressed or uncompressed (thus the same bitcoin address will be generated by using this format).

功能 2: 多方签名

查找多方签名核心代码步骤同上,

代码如下

```
coinjs.pubkeys2MultisigAddress = function(pubkeys, required) {
  var s = coinjs.script();
  s.writeOp(81 + (required*1) - 1); //OP_1
  for (var i = 0; i < pubkeys.length; ++i) {
    s.writeBytes(Crypto.util.hexToBytes(pubkeys[i]));
  }
  s.writeOp(81 + pubkeys.length - 1); //OP_1
  s.writeOp(174); //OP_CHECKMULTISIG
  var x = ripemd160(Crypto.SHA256(s.buffer, {asBytes: true}), {a
  x.unshift(coinjs.multisig);
  var r = x;
  r = Crypto.SHA256(Crypto.SHA256(r, {asBytes: true}), {asBytes:
  var checksum = r.slice(0,4);
  var redeemScript = Crypto.util.bytesToHex(s.buffer);
  var address = coinjs.base58encode(x.concat(checksum));

  if(s.buffer.length> 520){ // too large
    address = 'invalid';
    redeemScript = 'invalid';
  }

  return {'address':address, 'redeemScript':redeemScript, 'size'
}
```

多方签名函数生成两个地址, P2SH 地址和 REDEEM SCRIPT, 前者用于收款, 后者用于提款
一个有效的 REDEEM SCRIPT 地址, 格式为:< A pubkey>< B pubkey>< C pubkey>
前一个 代表需要两个私钥才能使用该多方签名地址的资金, 代表创建这个多方签名地址一共需要有三个公钥
`81 + (required*1) - 1` 这句中, 如果 required = 1, 那么就是运算结果是 81, 也就是 < OP_1>, 这是由 Script 语言规定的, 之前都说比特币中有非图灵完备的脚本语言, 即是指 Script 语言, https://en.bitcoin.it/wiki/Script 维基百科中有 Script 的介绍. 表示需要一个人签名就可以用多方签名钱包里的钱, 如果 required = 2, 运算结果是 82, 也就是 < OP_2>, 同理. 由于 Script 语言中只定义了 < OP_1 > 到 < OP_16>, 所以最多只能 15 个人创建一个钱包

计算 redeemScript 地址:

在数组中放入 81 + (required*1) - 1
再放入参与方的 pubKeys
再放入 81 + pubkeys.length - 1
再放入
结果得到:< pubKey A>< pubKey B>< pubKey C>

计算 address(钱包地址):

进行 ripemd160(SHA256(redeemScript 地址)) 运算
添加表示多方签名的前缀 0x05 得到钱包地址 addressTmp
计算 checksum:
对钱包地址 addressTmp 进行两次 SHA256 运算
取前四位作为 checksum
将钱包地址 addressTmp 加上 checksum 并进行 Base58 编码作为最终钱包地址
存在疑问: 要不要连接上摘要再 Base58 编码? coinbin 中要, 但文章中 (http://www.soroushjp.com/2014/12/20/bitcoin-multisig-the-hard-way-understanding-raw-multisignature-bitcoin-transactions/, 和 https://zhuanlan.zhihu.com/p/30949559) 不要

功能 3: 从个人钱包转账到多方签名钱包

输入: input_transaction_id,privateKey,destination,amount
输出 raw transaction
首先说一下背景知识

> 三、交易过程
> 下面，我把整个流程串起来，看看比特币如何完成一笔交易。
> 一笔交易就是一个地址的比特币，转移到另一个地址。由于比特币的交易记录全部都是公开的，哪个地址拥有多少比特币，都是可以查到的。因此，支付方是否拥有足够的比特币，完成这笔交易，这是可以轻易验证的。
> 问题出在怎么防止其他人，冒用你的名义申报交易。举例来说，有人申报了一笔交易：地址 A 向地址 B 支付 10 个比特币。我怎么知道这个申报是真的，申报人就是地址 A 的主人？
> 比特币协议规定，申报交易的时候，除了交易金额，转出比特币的一方还必须提供以下数据。

> 上一笔交易的 Hash（你从哪里得到这些比特币）
> 本次交易双方的地址
> 支付方的公钥
> 支付方的私钥生成的数字签名
> 验证这笔交易是否属实，需要三步。
> 第一步，找到上一笔交易，确认支付方的比特币来源。
> 第二步，算出支付方公钥的指纹，确认与支付方的地址一致，从而保证公钥属实。
> 第三步，使用公钥去解开数字签名，保证私钥属实。
> 经过上面三步，就可以认定这笔交易是真实的。
> 引用自: http://www.ruanyifeng.com/blog/2018/01/bitcoin-tutorial.html

如何生成 raw transaction

Description		Hex Bytes
Version byte		01000000
Input count		01
Previous tx hash (reversed)		acc6fb9ec2c3884d3a12a89e7078c83853d9b7912281cefb14bac00a2737d33a
Output index		00000000
scriptSig length of 138 bytes		8a
scriptSig	Push 71 bytes to stack	47
	<signature>	304402204e63d034c6074f17e9c5f8766bc7b5468a0dce5b69578bd08554e8f21434c58e0220763c6966f47c39068c8dcd3f3dbd8e2a4ea13ac9e9c899ca1fbc00e2558cbb8b01
	Push 65 bytes to stack	41
	<pubKey>	0431393af9984375830971ab5d3094c6a7d02db3568b2b06212a7090094549701bbb9e84d9477451acc42638963635899ce91bacb451a1bb6da73ddfbcf596bddf
Sequence		ffffffff

No. of outputs		01
Amount of 65600 in LittleEndian		4000010000000000
scriptPubKey length of 23 bytes		17
scriptPubKey	OP_HASH160	a9
	Push 20 bytes to stack	14
	redeemScriptHash	1a8b0026343166625c7475f01e48b5ede8c0252e
	OP_EQUAL	87
locktime		00000000

< redeemScriptHash>

```
# Makes a transaction from the inputs
# outputs is a list of [redemptionSatoshis, outputScript]

def makeRawTransaction(outputTransactionHash, sourceIndex, scrip
    def makeOutput(data):
        redemptionSatoshis, outputScript = data
        return (struct.pack("<Q", redemptionSatoshis).encode('he
        '%02x' % len(outputScript.decode('hex')) + outputScript)
    formattedOutputs = ''.join(map(makeOutput, outputs))
    return (
        "01000000" + # 4 bytes version
        "01" + # varint for number of inputs
        outputTransactionHash.decode('hex')[::-1].encode('hex')
        struct.pack('<L', sourceIndex).encode('hex') +
        '%02x' % len(scriptSig.decode('hex')) + scriptSig +
        "ffffffff" + # sequence
        "%02x" % len(outputs) + # number of outputs
        formattedOutputs +
        "00000000" # lockTime
        )
```

makeSignedTransaction

```
def makeSignedTransaction(privateKey, outputTransactionHash, sou
    myTxn_forSig = (makeRawTransaction(outputTransactionHash, so
         + "01000000") # hash code

    s256 = hashlib.sha256(hashlib.sha256(myTxn_forSig.decode('he
    sk = ecdsa.SigningKey.from_string(privateKey.decode('hex'),
    sig = sk.sign_digest(s256, sigencode=ecdsa.util.sigencode_de
    pubKey = keyUtils.privateKeyToPublicKey(privateKey)
    scriptSig = utils.varstr(sig).encode('hex') + utils.varstr(p
    signed_txn = makeRawTransaction(outputTransactionHash, sourc
    verifyTxnSignature(signed_txn)
    return signed_txn
```

然后将 raw transaction 广播到网络中, 被矿工收集打包成 block, 经过确认, 写入区块链中

功能 4: 从多方签名钱包转账到个人钱包

输入: input_tx,amount,destination,privateKey[], redeemScript
输出:
raw transaction

传入 destination 的是个人钱包地址, scriptPubKey 中的 pubKeyHash 是 publickey

Description	Hex Bytes
Version byte	01000000
Input count	01
Previous tx hash (reversed)	3dcd7d87904c9cb7f4b79f36b5a03f96e2e729284c09856238d5353e1182b002
Output index	00000000
scriptSig length of 349 bytes	fd5d01

scriptSig	OP_0	00
	Push 71 bytes	47
	<sig A>	30440220762ce7bca626942975bfd5b130ed3470b9f538eb2ac120c2043b445709369628022051d73c80328b543f744aa64b7e9ebefa7ade3e5c716eab4a09b408d2c307ccd701
	Push 72 bytes	48
	<sig C>	3045022100abf740b58d79cab000f8b0d328c2fff7eb88933971d1b63f8b99e89ca3f2dae602203354770db3cc2623349c87dea7a50cee1f78753141a5052b2d58aeb592bcf50f01
	OP_PUSHDATA1	4c
	Push 201 bytes	c9
	<redeemScript>	524104a882d414e478039cd5b52a92ffb13dd5e6bd4515497439dffd691a0f12af9575fa349b5694ed3155b136f09e63975a1700c9f4d4df849323dac06cf3bd6458cd41046ce31db9bdd543e72fe3039a1f1c047dab87037c36a669ff90e28da1848f640de68c2fe913d363a51154a0c62d7adea1b822d05035077418267b1a1379790187410411ffd36c70776538d079fbae117dc38effafb33304af83ce4894589747aee1ef992f63280567f52f5ba870678b4ab4ff6c8ea600bd217870a8b4f1f09f3a8e8353ae
Sequence		ffffffff

No. of outputs		01
Amount of 55600 in LittleEndian		30d9000000000000
scriptPubKey length of 25 bytes		19
scriptPubKey	OP_DUP	76
	OP_HASH160	a9
	Push 20 bytes	14
	<pubKeyHash>	569076ba39fc4ff6a2291d9ea9196d8c08f9c7ab
	OP_EQUALVERIFY	88
	OP_CHECKSIG	ac
locktime		00000000

辅助加密函数算法如下:

```
(typeof Crypto=="undefined"||!Crypto.util)&&function(){
  var f=window.Crypto={},
    l=f.util={rotl:function(b,a){return b<<a|b>>>32-a},
      rotr:function(b,a){return b<<32-a|b>>>a},
```

```
  endian:function(b){
    if(b.constructor==Number)
      return l.rotl(b,8)&16711935|l.rotl(b,24)&4278255360;
    for(var a=0;a<b.length;a++)
      b[a]=l.endian(b[a]);
    return b},
    randomBytes:function(b){
      for(var a=[];b>0;b--)
        a.push(Math.floor(Math.random()*256));
      return a},
    bytesToWords:function(b){
      for(var a=[],c=0,d=0;c<b.length;c++,d+=8)
        a[d>>>5]|=(b[c]&255)<<24-d%32;
      return a},
    wordsToBytes:function(b){
      for(var a=[],c=0;c<b.length*32;c+=8)
        a.push(b[c>>>5]>>>24-c%32&255);
      return a},
    bytesToHex:function(b){
      for(var a=[],c=0;c<b.length;c++)
        a.push((b[c]>>>4).toString(16)),a.push((b[c]&15).toS
      return a.join("")},
    hexToBytes:function(b){
      for(var a=[],c=0;c<b.length;c+=2)
        a.push(parseInt(b.substr(c,2),16));
      return a},
    bytesToBase64:function(b){
      for(var a=[],c=0;c<b.length;c+=3)
        for(var d=b[c]<<16|b[c+1]<<8|b[c+2],q=0;q<4;q++)
          c*8+q*6<=b.length*8?a.push("ABCDEFGHIJKLMNOPQRSTUV
          return a.join("")},
    base64ToBytes:function(b){
      for(var b=b.replace(/[^A-Z0-9+\/]/ig,""),a=[],c=0,d=0;
        d!=0&&a.push(("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijk
        return a}
},
f=f.charenc={};
f.UTF8={stringToBytes:function(b){
  return i.stringToBytes(unescape(encodeURIComponent(b)))},
  bytesToString:function(b){
    return decodeURIComponent(escape(i.bytesToString(b)))}};
var i=f.Binary={stringToBytes:function(b){
  for(var a=[],c=0;c<b.length;c++)
    a.push(b.charCodeAt(c)&255);
    return a},
              bytesToString:function(b){
                for(var a=[],c=0;c<b.length;c++)
                  a.push(String.fromCharCode(b[c]));
                return a.join("")}}};
```

```
(function(){
  var f=Crypto,
  l=f.util,
  i=f.charenc,
  b=i.UTF8,
  a=i.Binary,
  c=[1116352408,1899447441,3049323471,3921009573,961987163,15089
2820302411,3259730800,3345764771,3516065817,3600352804,409457190
d=f.SHA256=function(b,c){
  var e=l.wordsToBytes(d._sha256(b));
    return c&&c.asBytes?e:c&&c.asString?a.bytesToString(e):l.byt
    d._sha256=function(a){
      a.constructor==String&&(a=b.stringToBytes(a));
      var d=l.bytesToWords(a),e=a.length*8,a=[1779033703,3144134
1013904242,2773480762,1359893119,2600822924,528734635,1541459225
d[e>>5]|=128<<24-e%32;
d[(e+64>>9<<4)+15]=e;
for(s=0;s<d.length;s+=16){
  e=a[0];
  m=a[1];
  n=a[2];
  i=a[3];
  h=a[4];
  o=a[5];
  p=a[6];
  r=a[7];
  for(g=0;g<64;g++){
    g<16?f[g]=d[g+s]:(k=f[g-15],j=f[g-2],f[g]=((k<<25|k>>>7)^(k<
    j=e&m^e&n^m&n;
    var t=(e<<30|e>>>2)^(e<<19|e>>>13)^(e<<10|e>>>22);
    k=(r>>>0)+((h<<26|h>>>6)^(h<<21|h>>>11)^(h<<7|h>>>25))+(h&o^
    j=t+j;
    r=p;
    p=o;
    o=h;
    h=i+k>>>0;
    i=n;
    n=m;
    m=e;
    e=k+j>>>0}
      a[0]+=e;
    a[1]+=m;
    a[2]+=n;
    a[3]+=i;
    a[4]+=h;
    a[5]+=o;
    a[6]+=p;
    a[7]+=r}
```

```
    return a};
    d._blocksize=16;d._digestsize=32})();
```

附录:

参考链接:

https://arstechnica.com/tech-policy/2017/12/how-bitcoin-works/

https://zhuanlan.zhihu.com/p/30949559

http://www.righto.com/2014/02/bitcoin-mining-hard-way-algorithms.html

https://en.bitcoin.it/wiki/Script

https://bitcoin.stackexchange.com/questions/3374/how-to-redeem-a-basic-tx

http://www.righto.com/2014/02/bitcoins-hard-way-using-raw-bitcoin.html

http://www.soroushjp.com/2014/11/21/helpful-bash-scripts-for-working-with-byte-arrays-and-hex-in-bitcoin/

https://bitcoin.org/en/developer-guide#verifying-payment

http://www.soroushjp.com/2014/12/20/bitcoin-multisig-the-hard-way-understanding-raw-multisignature-bitcoin-transactions/

测试:

先生成三个地址:
address:
1CZz3Xtq2H4qHFjMzvrXXgtFrz2L1mwLou
pubKey:
03fb8ebbd4ad9e8e5b8194e2e85c4c11f2105837b3c9d4f1c13ab20d41f8809554
privKey WIF:
L2BgBMGo3KFXMzShBj3gSGoT8o7Nhs5dVQqY6iWFJ9NrQEzwH1r1

address:
1A1Q2Y2TAFNXpcMXLwWpqSax4JpT4iuNY9
pubKey:

03f45a6244114e32c7139a74ba14777214b321edbe563f77ca8c413f748d79b8c2
privKey wIF:
L4CtJR61u7oFuNwsVobmNcQdDoYoWqqw9HfVbyqKc5KRECZUPikC

address:
1M4wVzYuKwcQ3xiCkWvYSjefL5sxvt91Hm
pubKey:
026c7c9262452b8d5a350844ea6d0d08e9c581ee55356d280e7e7930845efd8302
privKey WIF:
KwEQST2n8g47ZKyWwgQZY5cLRPWJimFHxvz7skhxAxD2R3mMyEQH

2 of 3
address:
3FEuy5DpfQJU2qHM5zT95ig66iSigKTadk
redeemScript:
522103fb8ebbd4ad9e8e5b8194e2e85c4c11f2105837b3c9d4f1c13ab20d41f88095542103f45a6244114e32c7139a74ba14777214b321edbe563f77ca8c413f748d79b8c221026c7c9262452b8d5a350844ea6d0d08e9c581ee55356d280e7e7930845efd830253ae
如果每次输入的公钥不变, 公钥输入顺序不变, 那么结果生成的地址和 redeemScript 也不变.
下面查看 redeemScript:
52 代表
21 代表 The next opcode bytes is data to be pushed onto the stack
03fb8ebbd4ad9e8e5b8194e2e85c4c11f2105837b3c9d4f1c13ab20d41f8809554 代表第一个公钥
21
03f45a6244114e32c7139a74ba14777214b321edbe563f77ca8c413f748d79b8c2 代表第二个公钥
21
026c7c9262452b8d5a350844ea6d0d08e9c581ee55356d280e7e7930845efd8302 代表第三个公钥
53 代表
ae 代表

130 个
04a882d414e478039cd5b52a92ffb13dd5e6bd4515497439dffd691a0f12af9575fa349b5694ed3155b136f09e63975a1700c9f4d4df849323dac06cf3bd6458cd

66 个
026c7c9262452b8d5a350844ea6d0d08e9c581ee55356d280e7e7930845efd8302
为什么 文章 中是 130 个而 coinbin 网站则是 66 个呢? 为什么不是以 04 开头呢? 查看源码, 得知是因为在 coinbin 的网站上默认勾选了 compress, 所以会将 130 个压缩成 66 个

```
if(coinjs.compressed==true){
            var publicKeyBytesCompressed = EllipticCurve.integer
            if (y.isEven()){
                publicKeyBytesCompressed.unshift(0x02)
            } else {
                publicKeyBytesCompressed.unshift(0x03)
            }
            return Crypto.util.bytesToHex(publicKeyBytesCompress
        } else {
            return Crypto.util.bytesToHex(publicKeyBytes);
        }
```

取消掉压缩重新生成的, 又是 130 位.
0470916f0ec91cdbfef7f26e05bc3eef7c94a7268bcac4b868fd6815efbde32d6d5c91064999086dff56343c7c49898c1a635481677059bda3e3fa3696ca69df21

重新来过:
address:
1BgzrXYXmTpEJikGBvdKAX6HzJudCy9m4E
pubKey:
049ef619e2ce931cec27c5bbc5541a42115cd2b138841296415955701 81c2904473f3c79c916bbcb1b0ba017175be37220ed511ca64945a4fbcb411057a4133dd9
privKey WIF:
5JTC6REoGqPm5KGW7LRBVshJ9z2Ett79vQfvwdhLdsqk2QQ1BUq

address:
17oHxZPA3JU8A15wZsh8qLAvm7mPx4Sfci
pubKey:
04e363b88f7b6493ee073c189fe9912242707fcbcf56e25385a6bc207694cd925fb0f03cccf28b5ee7342aacb1f65e9eb9d472c8bb1ff73f542bbbed798648968c
privKey WIF:
5HyvHLJ8hvMMnsE8HDNCKu99hmd5p3UKtSvQfKrRcZc5X4bB9d4

address:
1AZXX6qk3TGEBgzwVKNXHdNPG5fyoXAkTS
pubKey:
04d0e6cf82b9146b03285833b8ff5137e52adc92c87dde5f2ede80cfee83ddd9d9bf67566a3cb84b23cb9ef89b750de2e0316f7de72aebf4b16d2f85a62cf58900
privKey WIF:
5KiukcG3VwXKN4vzbyBaNDCrhfA3L8ytfWSdRndzdyw16mZhgAb

2 of 3
address:
37LYRCZ2kSe3GLxiDGcEw2vWA2vGRp7ujz
redeemScript:
5241049ef619e2ce931cec27c5bbc5541a42115cd2b138841296415955701 81c2904473f3c79c916bbcb1b0ba017175be37220ed511ca64945a4fbcb411 057a4133dd94104e363b88f7b6493ee073c189fe9912242707fcbcf56e2538 5a6bc207694cd925fb0f03cccf28b5ee7342aacb1f65e9eb9d472c8bb1ff73f5 42bbbed798648968c4104d0e6cf82b9146b03285833b8ff5137e52adc92c87 dde5f2ede80cfee83ddd9d9bf67566a3cb84b23cb9ef89b750de2e0316f7de7 2aebf4b16d2f85a62cf5890053ae

这些都是十六进制
52 代表
41 转化为十进制 65, 代表把接下来的 65byte 放入栈, 刚好一个公钥长度为 130, 每两个公钥字符对应一个 byte
剩下的都和之前一样, 就不重复了.

redeemScript 转钱包地址 address 基本上是固定的了, 不会有问题

转账部分:
怎么从 inputTX 获取相关知识? 还是默认不管是否符合公钥, 未花费等条件了?
raw transaction 是:
0100000001acc6fb9ec2c3884d3a12a89e7078c83853d9b7912281cefb14bac00a2737d33a000000008a47304402204e63d034c6074f17e9c5f8766bc7b5468a0dce5b69578bd08554e8f21434c58e0220763c6966f47c39068c8dcd3f3dbd8e2a4ea13ac9e9c899ca1fbc00e2558cbb8b01410431393af9984375830971ab5d3094c6a7d02db3568b2b06212a7090094549701bbb9e84d9477451acc42638963635899ce91bacb451a1bb6da73ddfbcf596bddfffffffff01400001000000000017a9141a8b0026343166625c7475f01e48b5ede8c0252e8700000000

01000000 代表版本号,
01 代表 Input count
Previous tx hash (reversed)
acc6fb9ec2c3884d3a12a89e7078c83853d9b7912281cefb14bac00a2737d33a
Output index 00000000
scriptSig length of 138 bytes 8a
276 位即 138*2, 两位作为一个 byte
47304402204e63d034c6074f17e9c5f8766bc7b5468a0dce5b69578bd08554e8f21434c58e0220763c6966f47c39068c8dcd3f3dbd8e2a4ea13ac9e9c899ca1fbc00e2558cbb8b01410431393af9984375830971ab5d3094c6a7d02db3568b2b06212a7090094549701bbb9e84d9477451acc42638963635899ce91bacb451a1bb6da73ddfbcf596bddf
签名部分:
47 代表十进制的 71
接下来是 142 位签名, 71bytes
304402204e63d034c6074f17e9c5f8766bc7b5468a0dce5b69578bd08554e8f21434c58e0220763c6966f47c39068c8dcd3f3dbd8e2a4ea13ac9e9c899ca1fbc00e2558cbb8b01
这个签名是怎么产生的?
41 代表十进制的 65,
再接着公钥, 但这是谁的公钥??(猜想: 这是个人转账到公有地址钱包, 所以公钥是该个人钱包的, 签名也是该个人使用私钥产生的, 因为转账时使用到了私钥, 所以最后的问题是: 怎么使用私钥进行签名, 解答: 使用 ECDSA 签名)
Sequence ffffffff
No. of outputs 01
Amount of 65600 in LittleEndian 4000010000000000
scriptPubKey length of 23 bytes 17
进入 scriptPubKey 部分:
OP_HASH160 a9
Push 20 bytes to stack 14
redeemScriptHash 1a8b0026343166625c7475f01e48b5ede8c0252e
OP_EQUAL 87
这个 redeemScriptHash 又是谁的? 怎么产生的?
locktime 00000000

查找资料:
scriptPubKey refers to Public Key of?
Whoever you're sending the bitcoins to.(但不是只有对方的钱包地址

address 吗? 只有钱包地址是不可能推出公钥的, 这里是个人转账到个人, 所以 scriptPubKey 中也有 pubKey, 是接收转账者的个人公钥)

Where does the Input Script come from?
It is composed of two pieces - your public key, and a valid signature corresponding to that public key. So, you make it.

the scriptSig in Input Script refers to the Public Key and Signature of whom?
You.

When I am sending Bitcoins to a Bitcoin Address, the input Script is automatically generated by what?
Your client, after looking at your wallet

Even though the wiki gives details about Input Script and output Script, I was unable to correlate it to a real time scenario where one person sends BTC to another.

There are two parts to a bitcoin transaction - the part where you prove that you own some bitcoins, and the part where you designate someone else to be able to prove they own some bitcoins.

So, the Public Key in scriptPubKey differs from the Public Key in scriptSig?

scriptPubKey is very poorly named, because when you're sending bitcoins to an address, you only know the 160-bit hash of the public key. A scriptPubKey looks like this:

OP_DUP OP_HASH160 06f1b670791f9256bffc898f474271c22f4bb949 OP_EQUALVERIFY OP_CHECKSIG
The 06f1b6... is how your client specifies who it's sending to. The parts around it are scripts that tell the bitcoin client how to validate the transaction.

Also, we only have the Bitcoin Address of the person we intend to send the Bitcoins to. How do we get the Public Key of the recipient?

We don't need to. We can tell the network the 160 bit hash, and the person you're sending to can show the public key to the network when they want to spend their bitcoins.

The script contains two components, a signature and a public key. The public key must match the hash given in the script of the redeemed output.

The public key is used to verify the redeemers signature, which is the second component. More precisely, the second component is an ECDSA signature over a hash of a simplified version of the transaction. It, combined with the public key, proves the transaction was created by the real owner of the address in question. Various flags define how the transaction is simplified and can be used to create different types of payment.

A Bitcoin address is only a hash, so the sender can't provide a full public key in scriptPubKey. When redeeming coins that have been sent to a Bitcoin address, the recipient provides both the signature and the public key. The script verifies that the provided public key does hash to the hash in scriptPubKey, and then it also checks the signature against the public key.

see this

Multisignature transactions
Funds are sitting in one or more multisignature transaction outputs, and it is time to gather signatures and spend them.

Assumption: you know the multisignature outputs' {txid, outputNumber, amount}.

Create a raw transaction to spend, using createrawtransaction.
Use signrawtransaction to add your signatures (after unlocking the wallet, if necessary).
Give the transaction to the other person(s) to sign.
You or they submit the transaction to the network using sendrawtransaction.
You must be careful to include an appropriate transaction fee, or the sendrawtransaction method is likely to fail (either immediately or, worse, the transaction will never confirm).

生成 raw transaction 的时候上面两个 (公钥和 redeemScriptHash) 是不用填的, 等到签名 (sign) 的时候才填写, 之前说过, redeemScript 是付款 (即签名, 签上自己的名字代表确认这笔交易, 即付款) 时用的, 传入 redeemScript, 经过 ripemd160(sha256()) 运算, 得到的 hash, 别人可以根据公钥确认签名, 从而确认这笔交易.

A 想要向 B 发送一笔交易, A 填写上 raw transaction,

```
{
    “txid” : “54f773a3fdf7cb3292fc76b46c97e536348b3a0715886dbfd2
```

```
    “version” : 1,
    “locktime” : 0,
    “vin” : [
    {
        "txid" : "296ea7bf981b44999d689853d17fe0ceb852a8a34e68fc
        "vout" : 0,
        "scriptSig" : {
            "asm" : "",
            "hex" : ""
            },
            "sequence" : 4294967295
        }
    ],
    “vout” : [
    {
        "value" : 0.10000000,
        "n" : 0,
        "scriptPubKey" : {
            "asm" : "OP_DUP OP_HASH160 fdc7990956642433ea75cabdc
            "hex" : "76a914fdc7990956642433ea75cabdcc0a9447c5d2b
            "reqSigs" : 1,
            "type" : "pubkeyhash",
            "addresses" : [
            "1Q8s4qDRbCbFypG5AFNR9tFC57PStkPX1x"
            ]
        }
    },
    {
        “value” : 0.09890000,
        “n” : 1,
        “scriptPubKey” : {
            “asm” : “OP_DUP OP_HASH160 d6c492056f3f99692b56967a4
            “hex” : “76a914d6c492056f3f99692b56967a42b8ad44ce76b
            “reqSigs” : 1,
            “type” : “pubkeyhash”,
            “addresses” : [
            "1Lab618UuWjLmVA1Q64tHZXcLoc4397ZX3"
            ]
        }
    }
    ]
}
```

至此，一个 “空白交易” 就构造好了，尚未使用私钥对交易进行签名，字段 scriptSig 是留空的，无签名的交易是无效的。此时的 Tx ID 并不是最终的 Tx ID，填入签名后 Tx ID 会发生变化。

形象地讲，一笔交易输出 TxOut 的脚本 scriptPubKey 主要由接收方公钥的 Hash 和指令 OP_EQUALVERIFY、OP_CHECKSIG 构成，如同给这笔转账里的比特币上了一把锁，能够解开这把锁钥匙的拥有者即 OP_HASH160 公钥的拥有者，只有用相对应的私钥才可以使用被锁住的比特币。为了打开这把锁，在下一笔交易中，拥有者需要创建一个脚本 scriptSig 提供自己的签名和公钥。节点在验证交易的时候，会首先验证该公钥的 Hash160 是否与上一笔交易输出中的 OP_HASH160 相同，即表明该公钥为接收方所拥有。为了确保是接收方本人的操作，节点接下来要验证所提供的签名是否与公钥相匹配。如果验证通过，则该交易为合法，否则便会被拒绝

redeemScriptHash 产生:
对 < PK1>< PK2>< PK3> 进行 ripemd160(sha256()) 运算得到,< PK1>< PK2>< PK3> 可以由地址进行 base58 解码, 再去掉前导 05 得到.(有无校验和?)

剩下的 scriptSig 里的 signature 是发送方的签名, pubkey 是发送方的 pubkey,signature 是怎么产生的呢?
首先构造 raw transaction, 除了 scriptSign 和 scriptPubKey, 其他区域先填充进去, 然后填充 scriptPubKey, 只需要获得地址, base58 解码, 转化为十六进制, 去掉前后缀, (hash160 一下?)
使用 ECDSA 椭圆曲线数字签名算法, 使用私钥对经过 hash 的信息 (transaction, 但有少许改动) 进行签名
构造待签名的 transaction message
先使用 the scriptPubKey of the previous Output that we are trying to spend 填充 scriptSig,
此时的 raw transaction 除了 signature 没有, 其他的都有了
我們先暫時的在 raw transaction 後面掛上 4 bytes 的 hash type code，一般的 transaction 來說都用 SIGHASH_ALL (0x00000001)，要記得因為也是用 little endian 表示所以是 01000000
然後我們把整串 raw transaction 複製一份之後拿去做兩次 SHA256
接著把兩次 SHA256 的結果連同我自己的 private key 一起送去 ECDSA 簽名，會得到一個 signature
再來把 signature 後面再掛上 1 byte 的 hash code type，在這邊一樣是 01 的 SIGHASH_ALL
最後把上一步的 signature 連同我自己的 public key 去取代這個 raw transaction 原本的 input unlocking script，記得也要符合 script language，所以 signature 和 public key 前面都需要加他們的長度

回到我們的 signature，他其實也是由 G 去乘上一個 k 所產生的一個座標，只是這個 k 並不只是 private key 了，他是由 private key 和傳進來的

message (两次 SHA256 的 raw transaction) 和一個 random number 所組成的，因此要簽名一個 transaction 還是一定要有 private key 才能產生。

问题:
比特币是基于 UTXO 模型的, 其实并没有严格意义上的账户密码, 只有钱包公私钥, 以太坊是基于账户模型的,
所谓 UTXO 模型:
以每一笔资金为中心, 记录资金从被挖矿出来那一刻一直到当前时间的流向.

```
产生 |-> 转 x 到 A -> 转 a 到 X
     |-> 转 y 到 B -> 转 b 到 Y
     |-> 转 z 到 C -> 转 c 到 Z
```

以太坊的账户模型:
以账户为中心

```
用户 A |<- 收入 x
       |-> 支出 y
       |<- 收入 z
       |-> 支出 w
```

问题是: 对方公司采用的是哪种? 又或者模型对多方签名重不重要?

附录:
http://www.righto.com/2014/02/bitcoins-hard-way-using-raw-bitcoin.html

http://www.soroushjp.com/2014/12/20/bitcoin-multisig-the-hard-way-understanding-raw-multisignature-bitcoin-transactions/

https://en.bitcoin.it/wiki/Script

https://klmoney.wordpress.com/bitcoin-dissecting-transactions-part-2-building-a-transaction-by-hand/

https://bitcoin.stackexchange.com/questions/36440/signing-a-raw-transaction-with-python-ecdsa-or-openssl?rq=1

https://bitcoin.stackexchange.com/questions/32628/redeeming-a-raw-transaction-step-by-step-example-required/32695#32695

https://bitcoin.stackexchange.com/questions/3374/how-to-redeem-a-basic-tx/5241#5241

http://lenschulwitz.com/base58

https://medium.com/@bun919tw/transaction-%E7%B0%BD%E7%AB%A0%E7%9A%84%E6%B5%81%E7%A8%8B-45b883e8082a

https://en.bitcoin.it/wiki/Protocol_documentation#Transaction_Verification

https://en.bitcoin.it/wiki/Transaction_malleability