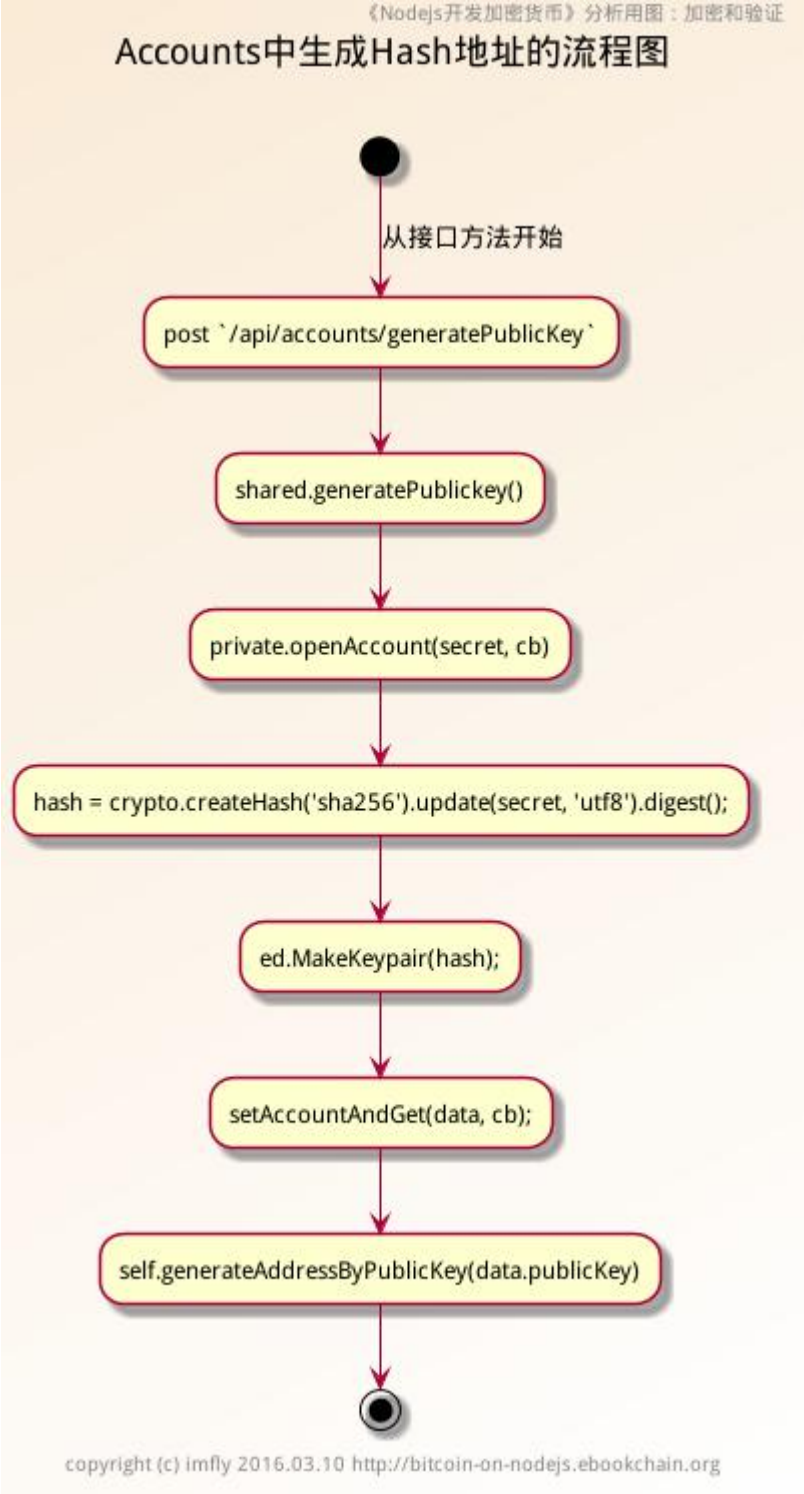
加密过程

在 Ebookcoin 世界里，Ebookcoin 把用户设定的密码生成私钥和公钥，再将公钥经过 16 进制字符串转换产生帐号 ID（类似于比特币地址）。从流程图我们可以更清楚地认识到实际的运行过程。



我们着重了解我们是如何将用户设定的密码生成私钥和公钥，再如何将公钥转换产生账号 ID（类似于比特币地址）。从流程图我们可以很清楚地了解到，我们首先需要将用户设定的密码以 post 形式传递到 api/accounts/generatePublicKey 这个 url 之中，也即通过 account.js 的 372 行"post /generatePublicKey": "generatePublickey"实现,来调用 shared.generatePublicKey(),如图

```
368         router.map(shared, {
369                 "post /open": "open",
370                 "get /getBalance": "getBalance",
371                 "get /getPublicKey": "getPublickey",
372                 "post /generatePublicKey": "generatePublickey",
373                 "get /delegates": "getDelegates",
374                 "get /delegates/fee": "getDelegatesFee",
375                 "put /delegates": "addDelegates",
376                 "get /username/get": "getUsername",
377                 "get /username/fee": "getUsernameFee",
378                 "put /username": "addUsername",
379                 "get /": "getAccount"
380         });
```

之前我们谈过 map 这个函数，这是一个地址映射函数，通过请求地址调用函数，我们回过头来再讲明这个函数，也说明一下 get 和 post 的区别。

地址映射方法。接受两个对象作为参数：

- root: 定义了所要开放 Api 的逻辑函数;
- config: 定义了路由和 root 定义的函数的对应关系。

```
// 3 行
function map(root, config) {
    var router = this;
    Object.keys(config).forEach(function (params) {
        var route = params.split(" ");//route 是一个数组!
        if (route.length != 2 || ["post", "get", "put"].indexOf(route[0]) == -1)
       {        throw Error("wrong map config");       }

        router[route[0]](route[1], function (req, res, next) {
           //req.query:一个对象，包含以键值对存放的查询字符串参数（通常称为 GET 请求参数） 。req.body:一个对象，包含 POST 请求参数。这样命名是因为 POST 请求参数在 REQUEST 正文中传递，而不像查询字符串在 URL 中传递。所以若请求方法是‘get’方式的话，就使用 req.query 来获取查询字符串参数，若请求方式为‘post’或者‘put’，就使用 req.body 来获取请求参数。
            root[config[params]]({"body": route[0] == "get" ? req.query : req.body}, function (err, response) {
                if (err) {
                    res.json({"success": false, "error": err});
                } else {
                    return res.json(extend({}, {"success": true}, response));
                }
            });
        });
    });
}

//比如"get /": "getPeers"，方法如下：
         router.get('/',function(req,res,next))
         {
                  root.getPeers(..)
         }
```

这里需要了解一下 get 和 post 的一些异同：

- GET 请求通过 URL（请求行）提交数据，在 URL 中可以看到所传参数。POST 通过“请求体”传递数据，参数不会在 url 中显示。

- GET 请求返回的内容可以被浏览器缓存起来。而每次提交的 POST，浏览器在你按下 F5 的时候会跳出确认框，浏览器不会缓存 POST 请求返回的内容。

- GET 提交，请求的数据会附在 URL 之后（就是把数据放置在 HTTP 协议头中），以?分割 URL 和传输数据，多个参数用&连接；例如：login.action?name=hyddd&password=idontknow&verify=%E4%BD%A0%E5%A5%BD。如果数据是英文字母/数字，原样发送，如果是空格，转换为+，如果是中文/其他字符，则直接把字符串用 BASE64 加密，得出如：%E4%BD%A0%E5%A5%BD，其中%XX 中的 XX 为该符号以 16 进制表示的 ASCII。

  而 POST 提交：把提交的数据放置在是 HTTP 包的包体中。

因此，GET 提交的数据会在地址栏中显示出来，而 POST 提交，地址栏不会改变。而在这里我们再看流程图第一个调用："post /generatePublicKey": "generatePublickey",之所以要使用 post 提交，是为了更加安全，而不能以 get 方式请求，这样会暴露在 url 上，这些比较不安全。我们需要将用户的密码 secret 以 post 的方式请求，这样 secret 的内容会被放在请求体传递数据，比较安全。然后很自然地通过地址映射，调用了 shared.generatePublickey（）方法，接下来逐个分析被调用的函数。

```
// modules/accounts 628 行
//检查用户的 secret 是否符合规则
shared.generatePublickey = function (req, cb) {
    var body = req.body;
    library.scheme.validate(body, {
        ...
        required: ["secret"]
    }, function (err) {
        ...
    // 644 行
    //将用户的密码传进 openAccount 函数，下一步准备生成密文和密钥对
        privated.openAccount(body.secret, function (err, account) {
            ...
            cb(err, {
                publicKey: publicKey
            });
        });
    });
};
```

```
// 447 行
//通过 crypto 模块和 Ed25519 组件生成密钥对
privated.openAccount = function (secret, cb) {
    //传进来的公钥是 utf-8 形式，我们需要先创建一个 Hash 实例，用 update 接受 utf-8 形式的公钥，再调用 digest（）方法获得密文，digest（）默认返回的是 2 进制的数据
    var hash = crypto.createHash('sha256').update(secret, 'utf8').digest();
    var keypair = ed.MakeKeypair(hash);//生成密钥对

    //将密钥对的公钥转化为十六进制形式
    self.setAccountAndGet({publicKey: keypair.publicKey.toString('hex')}, cb);
};

// 482 行
Accounts.prototype.setAccountAndGet = function (data, cb) {
    //对于||来说，假设 a||b，只要 a 为真，b 就不用操作，因为 a||b 已经为真，而 a 若为假，b 还是需要操作的。而对于&&来说，假设 a&&b，若 a 为假，b 就不需要被操作了，而 a 为真，b 还是要操作的。在 nodejs 这本书中我们看到很多的代码，用&&和||分割两个语句，这里我们需要多注意一下。
    var address = data.address || null;

    if (address === null) {
        if (data.publicKey) {
      // 486 行
            address = self.generateAddressByPublicKey(data.publicKey);
          ...
        }
    }
    ...
  // 494 行
    library.logic.account.set(address, data, function (err) {
        ...
    });
};
```

```
// modules/accounts 455 行

//对公钥进行再一次加密，并做一次十六进制处理，生成地址
Accounts.prototype.generateAddressByPublicKey = function (publicKey) {

   //传进来的公钥是十六进制形式，我们需要先创建一个 Hash 实例，用 update 接受十六进制形式的公钥，再调用 digest（）方法获得密文，digest（）默认返回的是 2 进制的数据
    var publicKeyHash = crypto.createHash('sha256').update(publicKey, 'hex').digest();

    var temp = new Buffer(8);//在 nodejs 中，对于二进制数据并没有提供一个很好的支持。然后在 nodejs 中需要处理像 TCP 流或文件流时，必须要处理二进制数据。因此在 node.js 中，定义了一个 Buffer 类，该类用来创建一个专门存放二进制数据的缓存区。

    for (var i = 0; i < 8; i++) {
        temp[i] = publicKeyHash[7 - i];
    }

    var address = bignum.fromBuffer(temp).toString() + 'L';
    if (!address) {
            throw Error("wrong publicKey " + publicKey);
    }
    return address;
};
```

这个地方的公钥十六进制转换还是有点奇怪，这个转换的操作还不是特别清楚，等我们看完全部代码以后再回来看这个部分吧。这部分只要逻辑明白了，了解一点 get、post 的区别，运用封装好的 crypto 模块就能够快速将用户设定的密码生成密文，再使用 Ed25519 组件生成密钥对，再将公钥经过转换产生账号 ID（类似于比特币地址）。现在回头看流程图，一目了然。