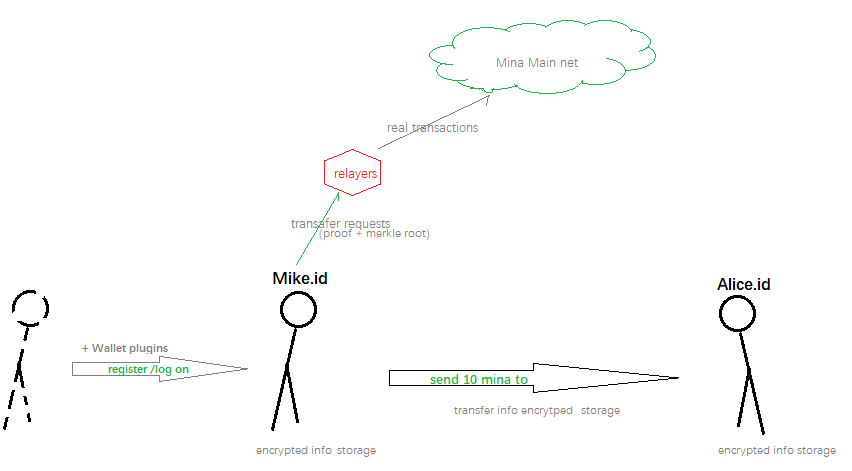
《Shadow -- A Private Account System On Mina》

As we talked before, the mixer project is an extracted component from our original design of **zkPass(Crypto Identity)** for this Builder program. Based on this, this mixer project is still based on the account model, which means everyone could register and hold a unique name in the scope.

To keep this feature and also to keep privacy, we made great effort on the design of how to maintain ‘**Deposition to mixer without name exposure**’, ‘**Transfer internally without any exposure**’, and ‘**Withdraw from mixer without name exposure**’.

So people could got a unique name in the crypto world and never worry about their asset operations to be traced in the scope.



中继者 的两种方案： **snapp代付**，或者 **第三方中继者列表**， 先mock,待后续官方讨论实现。

用户体验优化。

一个名字的hash -> 加密balance & （一个公钥，私钥密文） & hash(口令+名字)

只有知道口令，才可以转账&提款。

一：用户注册

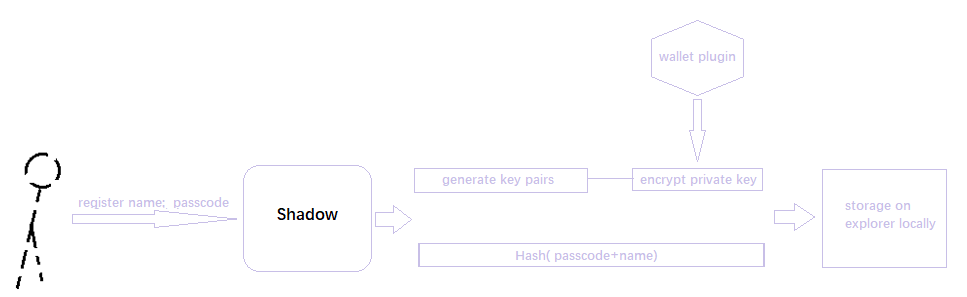
**首先要求用户钱包里有小部分mina，**

用户注册一个名字，并本地生成一把公私钥，此公私钥用来加解密用户账户信息和交易信息。

采用外部钱包来加密 上面本地生成的私钥。

同时用户提供一个口令，以hash(口令+名字)保存。

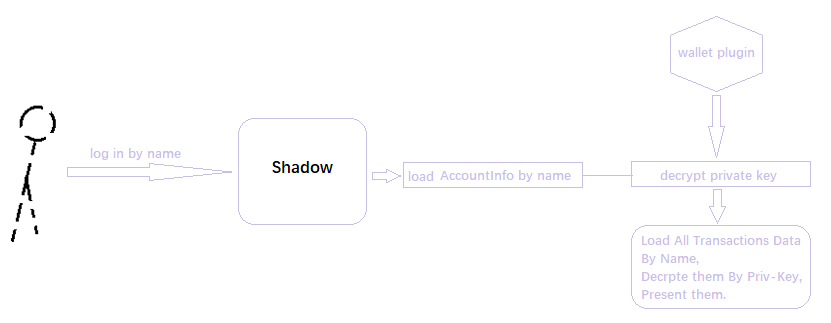
// TODO 在公开数据保存时，采用外部钱包私钥还是对称密钥加密更加好？



二：用户登录

在当前浏览器的钱包插件的同一个钱包下，用户提供名字即可登录。

浏览器通过名字搜索出本地的保存的所属信息(公钥 和 加密后私钥A 和 hash(口令+名字))，并显示在页面。



三：用户充值

Ipfs保存了映射<hash(名字\_**私钥A的片段?**), encrypted balance>

用户通过钱包插件充值到snapp地址，

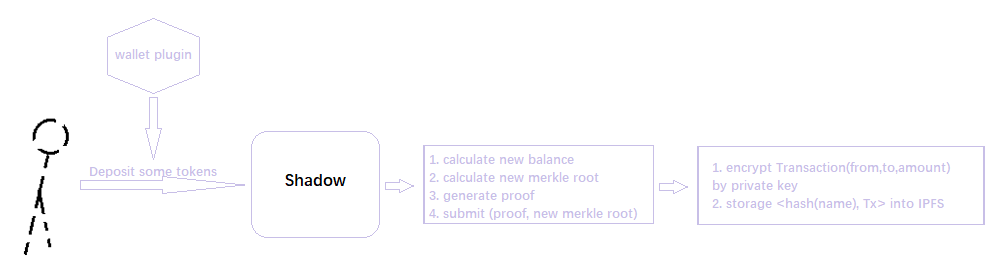
浏览器从mekletree上获取到加密的balance，并通过和链上merkle root 对比来确定获取的balance是否在合约对应的mekle tree上。

解密后的balance + 充值金额得到新balance，并基于之重新计算新的merkle root.

将新的balance，用 公钥A 加密后保存到ipfs.

提供交易(proof, 新merkle root) 到链上，验证通过即更新merkle root.

// 少用户操作时，很容易追踪某个hash名字的累计余额，待讨论。



四：用户提现 **-- 中继者**

Ipfs保存了映射<hash(名字\_**私钥A的片段?**), encrypted balance>

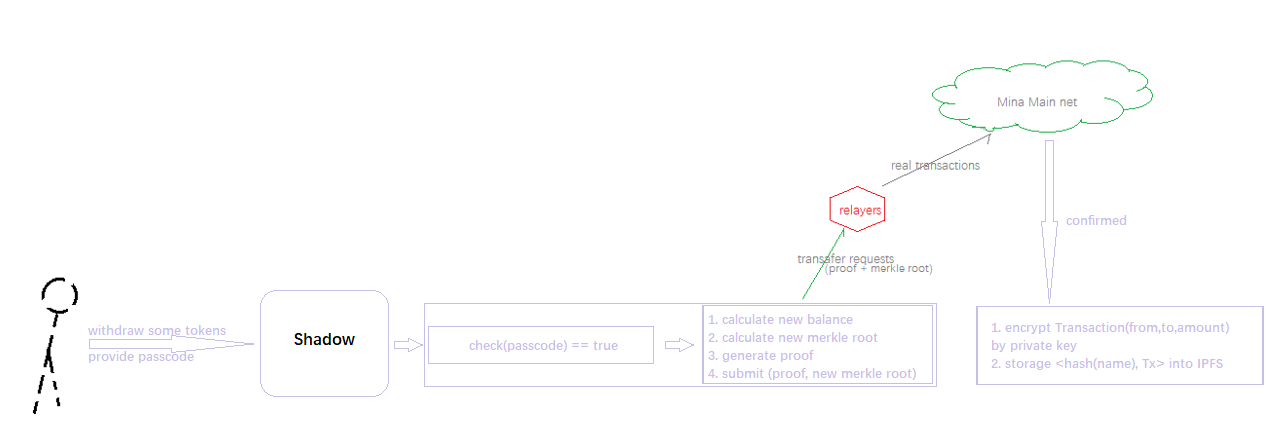
要求用户提供口令原文，并计算hash(口令+名字)后是否等同于existing的hash(口令+名字)

浏览器从mekletree上获取到加密的balance，并通过和链上merkle root 对比来确定获取的balance是否在合约对应的mekle tree上。

解密后的balance - 充值金额得到新balance，并基于之重新计算新的merkle root.

将新的balance，用 公钥A 加密后保存到ipfs. 提供交易(proof, 新merkle root) 到链上，验证通过即更新merkle root. 同时从合约中转一笔款到指定地址。

提现过程中，生成票据<当前用户名字hash, 提现金额>，然后公钥加密之，链下保存。



五：用户间内部转账 **-- 中继者**

先要求用户提供口令原文，并计算hash(口令+名字)后是否等同于existing的hash(口令+名字)

浏览器从mekletree上获取到加密的balance，并通过和链上merkle root 对比来确定获取的balance是否在合约对应的mekle tree上。

解密后的balance – 转账金额得到新balance，并基于之重新计算新的merkle root.

将新的balance，用 公钥A 加密后保存到ipfs. 提供交易(proof, 新merkle root) 到链上，验证通过即更新merkle root. 同时从合约中转一笔款到指定地址。

用户准备好接收人的公钥，生成转账记录<from, to, amount>, 并分别用自己的公钥和对方的公钥各自加密此转账记录（即得到两条）。自己公钥加密的记录标记为已经消费状态，而对方公钥加密的记录放入pending mapping中(接收者的名字hash, List<对方公钥加密的记录>)。

当接收者登录时，根据名字hash获取所有的pending接收记录，解密后，更新balance（提交到链上验证等流程同上）.

这样就双方都保存了 所有历史转账记录。

