编辑导读：在一些信号很差的地方，手机没网了却依然还能支付，这是什么原理呢？本文将从四个方面对这个问题展开分析，希望对你有帮助。

现在生活已经离不开微信/支付宝电子支付，平常出去吃饭、购物只要带个手机，就可以解决一切，以致于现在已经好久没摸过真金白银了。有一次出去吃饭，排着队付钱，等着过程非常无聊，准备拔出手机来把荒野乱斗，却发现这个地方竟然连不上网。

看着手机明明信号满格，但是就是显示网络无连接，苹果手机用户痛，谁用谁知道。

由于没有网络，而我又没带钱，所以就怕付钱的时候因为手机没网，没办法使用支付宝扣款。正想着时，已经排到了我，不管三七二十一，先用下支付宝试试，实在不行就不吃了。

不过没想到，当商家用扫码抢扫描支付宝上付款码支付以后，虽然我的手机最终没有弹出支付成功的页面，但是商家端显示支付成功，并成功打印出了小票，过了一会，我的手机收到支付宝扣款短信。

因为我最近的工作对都是与微信/支付宝有关，整体支付流程还是比较清楚，但是付款码为什么能离线支付确实不是很清楚，所以研究了一番，于是有了今天的文章。

一、科普支付方式

在聊付款码离线原理之前，我们先给不熟悉支付宝/微信支付方式同学先科普一下常见的两种支付方式。

微信、支付宝线下支付常用支付方式有两种，一种是我们打开手机，主动扫描商家提供码牌，这种支付方式一般称为主扫支付（用户主动扫码）。

以支付宝为例，付款流程如图所示：

图片来自支付宝官网

第二种则是我们打开手机，展示我们的付款码，然后商家使用扫码枪等工具获取付款码完成支付，这种支付方式一般称为被扫支付（用户被扫码）。

以支付宝为例，付款流程如图所示：

图片来自支付宝官网

对于第一种方式，需要手机端 APP 扫码，然后弹窗确认付款，这种方式是没有办法在手机没有网络的情况完成支付，所以我们上文说的没有网络的情况特指付款码支付的场景。

二、付款码付款流程

在聊付款码离线支付的前提前，我们先来来看下付款码的整体流程，以超市购物为例，一次付款码的支付信息流如图所示：

参考知乎@天顺

这个过程商家后台系统是需要调用的支付宝条码支付的接口，完成支付。

「由于商家后台需要在线联网与支付宝后台通讯，所以说付款码的离线支付，指的是客户端没有的网络的情况，商家端其实必须实时联网在线。」

一次付款码接口调用流程如图所示：

来自支付宝官网

通过上面两张图，我们整体了解付款码交互流程。

付款码的技术方案其实可以分为客户端在线与离线的两种情况，下面我们来看下两种方案具体实现方式。

三、在线码方案

客户端在线码的方案，这个应该比较容易想到，只要支付宝/微信在登录的情况下，点击付款按钮，客户端调用后台系统的申请付款码接口。

后台系统受到请求之后，生成一个付款码，然后在数据库保存付款码与用户的关系，并且返回给客户端。

只要客户端在有效期内展示该付款码，就可以完成支付，否则该二维码就将会过期。

使用这种方案，相对来说比较安全，因为每次都是服务端生成码，服务端可以控制幂等，没有客户端伪造的风险的。

另外即使需要对付款码规则调整，比如付款码位数增加一位，我们只要调整服务端代码即可，客户端都无需升级。

「不过这种方案缺点也比较明显，客户端必须实时在线联网，没有网络则无法获取付款码。」

另外，现在有一些智能设备也开始支持支付宝支付，这些设备中很大一部分是没有联网的功能（比如小米手环四），那这种情况是没办法使用在线码方案。

四、离线码方案

说起离线码大家可能比较陌生，但是实际上你如果仔细观察，其实很多场景都用到了离线码。

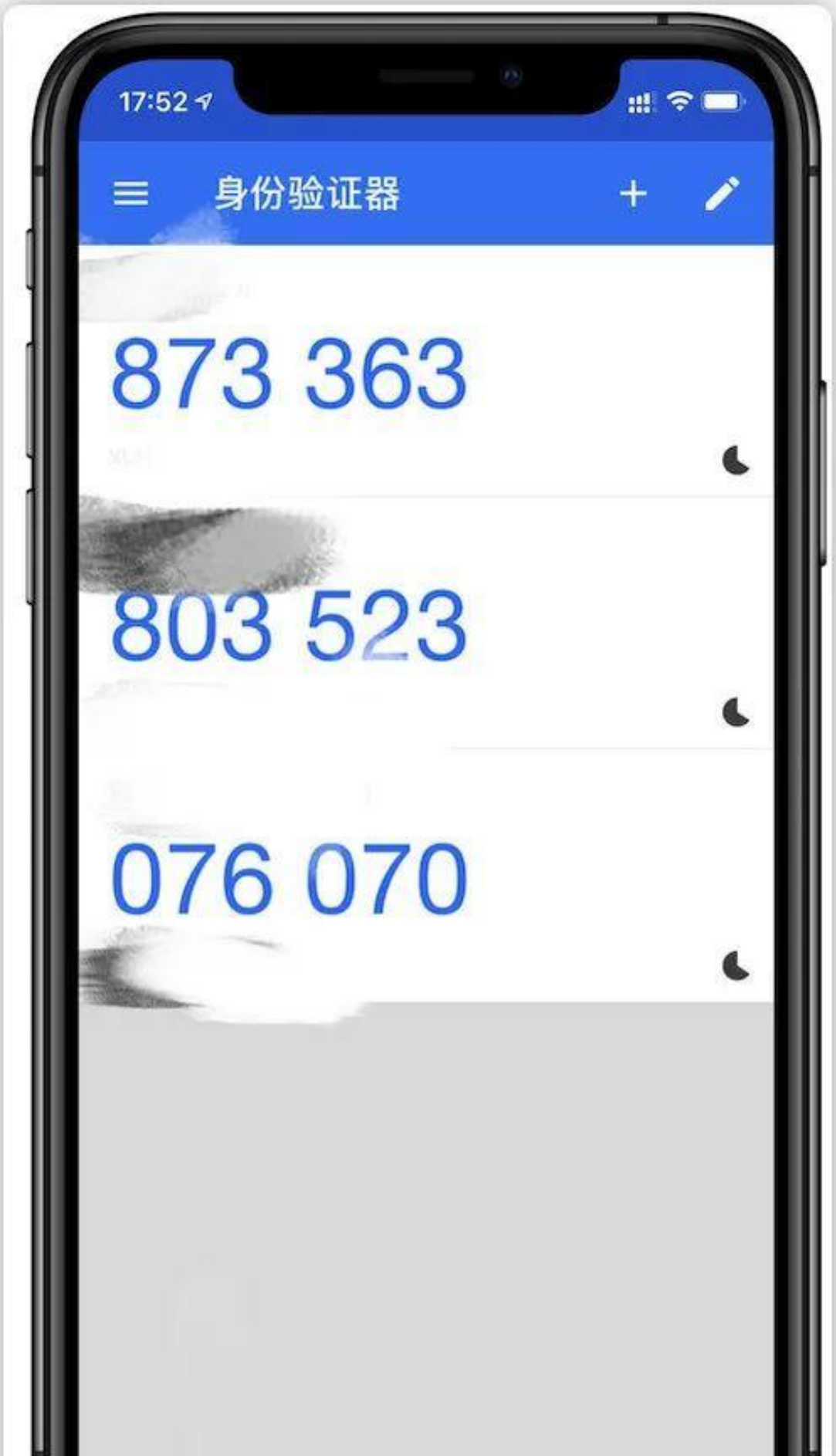
比如说以前去黑网吧玩梦幻西游的时候，账号总是被盗。

没办法，花了一笔重资买了一个网易将军令，每次登录的时候，除了输入用户名与密码以外，还需要输入动态口令。从此账号就很少被盗了。

又比如说每次网易支付的时候，我们除了输入银行卡密码以外，还需要输入网银盾上动态码，这样才能完成支付。

画外音：这里又要吐槽一下，网银盾以前真的超难用，动不动就驱动不兼容。还记得当初用网银充值黄钻，搞了一下午都没有成功！

当然上面这些可能已经是老古董了，很多人都可能没用过，现在比较流行是「手机验证器APP」，比如 「Google Authenticator」 等。


17:52
身份验证器
873 363
803 523
076 070

这种令牌器，动态产生一次性口令（「OTP, One-time Password」），可以防止密码被盗用引发的安全风险。

其实付款码离线方案技术原型就是基于这种方案，所以下面我们就基于 Google Authenticator，来了解一下这其中的原理。

1. 动态口令技术原理

首先如果我们需要使用 「Google Authenticator」，我们需要在网站上开启二次验证功能，以 Google 账号为例，在设置两步验证的地方可以找到如下设置：

当我们点击设置，将会弹出一个二维码，然后使用 「Google Authenticator」 APP 扫码绑定。

当我们绑定之后， 「Google Authenticator」 APP 将会展示动态码。

我们来解析一下这个二维码，对应下面这个字符串：

otpauth://totp/Google%3Ayourname@gmail.com?secret=xxxx&issuer=Google

上面的字符串中，最重要就是这一串密钥 secret，这个是一个经过 「BASE32」 编码之后的字符串，真正使用时需要将其使用「BASE32」 解码，处理伪码如下：

original_secret=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxsecret=BASE32_DECODE(TO_UPPERCASE(REMOVE_SPACES(original_secret)))

「这个密钥客户端与服务端将会同时保存一份，两端将会同样的算法计算，以此用来比较动态码的正确性。」

我们以客户端为例，生成一个动态码，首先我们需要经过一个签名函数，这里 **Google Authenticator **采用的 「HMAC-SHA1」，这是一种基于哈希的消息验证码，可以用比较安全的单向哈希函数（如 SHA1）来产生签名。

签名函数伪码如下：

hmac=SHA1(secret+SHA1(secret+input))

上面函数中的，input 使用当前时间整除 30 的值。

input=CURRENT_UNIX_TIME()/30

这里时间就充当一个动态变参，这样可以源源不断产生动态码。

「另外这里整除 30，是为了赋予验证码一个 30 秒的有效期。」

这样对于用户输入来讲，可以有充足时间准备输入这个动态码，另外一点客户端与服务端可能存在时间偏差，30 秒的间隔可以很大概率的屏蔽这种差异。

画外音：这个有效时间其实很考量，如果比较长，安全性就差。

如果比较短，用户体验就很差，不容易输入准备。

经过 「HMAC-SHA1」 签名函数以后，我们得到一个长度为 40 的字符串，我们还需要将其转化为 6 位数字，方便用户输入。处理的伪码如下：

four_bytes=hmac[LAST_BYTE(hmac):LAST_BYTE(hmac)+4]large_integer=INT(four_bytes)small_integer=large_integer%1,000,000

完整的算法伪码如下：

original_secret=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxsecret=BASE32_DECODE(TO_UPPERCASE(REMOVE_SPACES(original_secret)))input=CURRENT_UNIX_TIME()/30hmac=SHA1(secret+SHA1(secret+input))four_bytes=hmac[LAST_BYTE(hmac):LAST_BYTE(hmac)+4]large_integer=INT(four_bytes)small_integer=large_integer%1,000,000

当客户端将动态码上传给服务端，服务端查询数据库获取到用户对应的密钥，然后使用同样的算法进行处理生成一个动态码，最后比较客户端上传动态码与服务端生成是否一致。

2. 付款码离线方案

上面我们了解了动态口令的实现方案，付款码生成原理其实也大致如此。

不过付款码离线方案采用动态密钥的方式(「全局唯一」)，定时请求服务端更换密钥，以此保证更高的安全性。

另外在一次性动态口令方案，需要双方基于同样的秘钥，所以服务端需要明确知道这「背后正确用户」。以上面的登录场景为例，登录过程输入用户名,服务端就可以根据这个在数据库中查询相应的密钥。

但是在付款码的支付场景中，支付过程仅仅传递一个付款码，就可以向相应的用户扣款。不用想，这个付款码这串数字一定包含相应的用户信息。

所以付款码的相应的算法相比动态码会更加复杂，这样才可以有效保证安全性。

看到这里，不知道你们是否急切想了解这套算法那?

这种算法岂能是我们能掌握的?

支付宝核心算法咱不知道，但是我们可以从其他人公开设计方案了解一个皮毛。

这里小黑哥给你一个知乎网友@反方向的钟回答的离线二维码实现方式，给你 look look 。

> 在翼支付的离线二维码上，方案的设计概述为：
>
> 1、 登录翼支付，服务器生成唯一token，通过加密方式(如https)传递到客户端。
>
> 2、 打开付款码时，本地生成一段含有token与当前时时间戳的哈希值，如sha1(token+UnixTimestamp)，转换为byte[]并截取指定长度后转换为int变量otp。
>
> 3、 设翼支付用户账号(手机号)为int变量id。
>
> 4、 设otp在[0,n]中，通过code=id*n+otp，即可将OTP与ID合并在同一个数字里，成为最终的条形码/二维码，并每间隔指定时间更新一次。
>
> 5、 通过商家扫码枪扫描，服务端获取了code，通过(int)(code/n)就得到了id，通过code%n就得到了otp。
>
> 6、 通过id找到token，通过当前时间戳与前后若干个相同间隔的时间戳以步骤2相同的方法生成对应的一组otps[]，用于容许客户端和服务端之间的时间差。
>
> 7、 将从客户端得到的otp与otps[]中的元素逐一比对，如有相同项，则为验证通过。

来自：https://www.zhihu.com/question/49811134/answer/135886638

3. 付款码离线码的劣势

最后我们来看下付款码离线方案的劣势：

第一，算法调整不灵活，如果相关算法较大的调整，可能需要升级客户端,并且这个期间服务端还需要兼容新老算法产生的付款码。

第二，安全性问题，正常的情况相关密钥无法被普通用户获取，但是架不住有有心之人。他们可能通过获取手机用户 Root 权限或者越狱手机，利用恶意程序获取密钥，然后随意生成付款码。

看到这一点，大家可能会担心自己的钱包安全了。不过这一点，我觉得不过过分担心，蚂蚁集团这么多大神，不是吃干饭的，他们肯定有很多措施保证支付安全。

第三数据碰撞问题，A 用户生成付款码算出来与 B 用户一致，这就 Hash 算法一样，再怎么优秀的算法，也有概率才生一样的额 Hash 值。

这就导致原本是扣用户 A 的钱，最后却扣了 B 用户。这样一来，确实很乌龙，对于 B 用户来讲，莫名其妙被扣钱了。

不过放心，这种事放到放到现在，我觉得还是比买彩票中奖低，所以这种事还是不用过分担心了。

即使真被误扣了，放心，支付宝这么大体量肯定会跟客户赔钱的。

五、最后

最后总结一下，我们平常使用付款码支付，其实原理就是商家端获取我们手机 APP 付款码（「其实就是一串数字」），然后后台调用支付宝支付接口完成扣款。

这个流程商家端后台程序必须联网在线，但是对于我们客户端来讲可以在线，也可以离线。

如果我们客户端在线，那就可以通过服务端向客户端发送付款码，这种方式更加安全，灵活，但是对于弱网环境下，体验就很差。

如果我们客户端没网，那就通过客户端通过一定算法生成付款码，服务端收到经过相关校验，确认是哪个用户，确认码有效性，并且完成扣款。这种方式，适合客户端没有网络的情况，不过相对不灵活，且安全性稍差。

嘿嘿，了解原理，有没有觉得还是挺有意思的~

下次排队付款钱,如果手机没网，不要担心尴尬，放心拿出手机付钱~

对了，看完记得一键三连，这个对我真的很重要。

六、参考

https://www.zhihu.com/question/49811134/answer/135886638

https://garbagecollected.org/2014/09/14/how-google-authenticator-works/

作者：楼下小黑哥；微信公号@程序通事，支付行业，后端技术