Berewic Notes

Alistair Mann al@berewic.com

These are early thoughts to map the contours of the space. Actual work comes later.

Introduction

A smart contract allows for Alice to guarantee something to Bob: for example she can guarantee she's not a spammer. The particular smart contract in mind is a Hashed Time-Locked Bond, of which more below; however a Hashed Time-Locked Contract would also work. Neither are implemented in the Bitcoin wallet as yet (March 2019), although both work in the main daemon, with the latter integral to creating Lightning Network channels. Berewic's first, smaller, job is to implement the smart contract on behalf of clients until the reference wallet or other better facility does it for them.
TODO: link to formal work on HTLB, including HTLC as HTLB

For Alice to know what kind of guarantee Bob wants requires that Bob declares the details in advance. The second job of Berewic is then to assist Bob in declaring to the network what it takes to access his resource.

Next, Alice and Bob need to negotiate some of the terms of a bond, particularly what the address of the bond will be; Berewic's third job is thus to structure that negotiation for their respective owner.

A Berewic server can implement bonds because it has access to a hot wallet. The security measures envisioned lend themselves well to *ledger possession* of funds to users, subusers, & individual applications. Whoever controls those funds may wish to sell access to some part of them, and so Berewic's fourth job is to assist with that transfer.

Berewic is a model describing the actors, hardware and software; it's a protocol describing what's said between them in what sequence; and berewicd is the reference implementation.

In this document bitcoin prices are denoted thus: "฿2.5", satoshi prices thus: "30,000s"

Background

The Hashed Time-Locked Contract (HTLC) on the Bitcoin network, most widely known for its use in creating Lightning channels, is a short script defined by BIP-0199 that says "Alice can redeem the funds after timestamp X, Bob can redeem the funds at anytime as long as he has secret Y."

Assuming Bob already knows secret Y, there are circumstances where Bob would not immediately redeem all the funds: if Bob is incentivised for Alice's repeat custom, he would prefer not to take the funds. If Alice wants to redeem those funds then she's incentivised to keep Bob happy. We say:

- Bob is incentivised outside the contract to good behaviour within it

- Alice is incentivised inside the contract to good behaviour outside it.

A good example use case would be a robocall, where a computer rings up and plays a prerecorded message. Indeed many robocalls see the recipient billed for receiving it. Let's imagine Alice is Bob's daughter and she has a new telephone number. She wants to call her Dad but knows he doesn't pick up strange numbers as they're usually robocalls.

She sends her Dad 20,000s ("20,000 satoshis") along with the telephone call, good for three hours. Now Bob sees the call arrive and instantly knows: someone is guaranteeing him that he wants to receive this call. And if he disagrees for any reason at all, he has three hours to take those funds at their expense. Bob takes the call.

Bob wants Alice to stay in touch, so he declines to redeem those funds in the three hours and she gets them back: Bob behaved inside the contract. For her part, Alice refrained from selling Bob encyclopaedias: she behaved outside the contract.

Back to secret Y, it turns out no one benefits from it in this kind of transaction: it adds to the things to store, it adds to the processing required but offers nothing in return. To that end I am suggesting the Hashed Time-Locked *Bond* - HTLB - to the bitcoin devs, and use HTLB throughout this project. HTLB is a HTLC but without the secret Y, and with a very different motivation.

Bonds are also passive. While Alice could pay Bob ₿1.2 for a bond, and Bob could pay Alice ₿1.2 back if she was compliant, it has problems:

- The risk that Bob falls under a bus before he can pay Alice back;
- The risk that Bob loses access to those funds;

If Bob simply does *nothing*, Alice will get regain access to her funds.

Use cases

Bonding *qua* bonding

Conceptually, bonding can find use wherever there is:

- a venue (eg server, real estate) that manages …
- a resource (eg a URL, a particular room between particular times) and that …
- has an effective mechanism for enforcing against access (eg http basic access authentication, a lock and key)

The following examples is by no means exhaustive.

Webpages, websites Particularly where expensive processing is offered to anonymous users raising the prospect of Denial of Service, a single page, group of them or entire website can require posted bond: that DoS vector then becomes a revenue stream.

Email Two thirds of email is spam. If sending a spam costs 30,000s *per spam* spammers will run out of money faster than hitherto.

Social media Trolling occurs in large part because comment is free: by requiring a posted bond beforehand low quality contributions can be effectively penalised.

Telephone Robocalling, unsolicited faxes, and obscene calls are all discouraged by a requirement to guarantee the receiver desires to receive the call particularly in networks where victims are required to pay to receive such calls. VOIP packages such as Skype would similarly benefit given the number of unsolicited connections the author has received.

Restaurant Dine and Dash is effectively prohibited where the reservation requires bonding beforehand. Similar abuses include making off without paying public transit and taxi fares, automobile fuel charges and similar defrauding of innkeepers.

Connectivity VPN exit points might well offer free access below 1Gb per IP per month, then 30,000s for the next 1 Gb. With bonding they can reduce their freeloader problem in favour of a client pool who guarantee that they'll pay for that second gigabyte if they use it.

Group chat whether in an app such as Telegram, a listserv mailing list, a teamspeak session, a facebook group or IRC session, requiring a bond before posting would be an effective way of denying access to abusers: abuse would see their bond forfeited, eventually in total.

Property Rental As a deposit against damages, Alice guarantees Bob one month's rent. If she leaves everything in order, Bob declines to redeem any funds and Alice redeems them all.

Bail Bond Judge Bob requires Alice to post bail, good for one year, against her appearance at trial expected to be in ten months. Anxious not to visit gaol too early, Alice complies.

No-Shows GP surgeries have a problem in the UK with people booking an appointment then not turning up. Rather than take names, email addresses and inside leg measurements for nagging, they can take a 100,000s bond to appear returned if and when you actually do appear.

Bonding as payment

Whether an HTLB is a bond or a payment is ultimately decided outside the contract. For example, Alice knows full well that Bob is her son but nevertheless uses her telephone to call him with a ฿0.25 bond guaranteeing he wants the call. He does - he's stony broke - and he redeems half the ฿0.25 shortly after.

Thus, adoption of crypto bonding drives enables adoption of crypto payment.

An additional advantage of bonding as payment is that if Bob above spends half then loses his phone, Alice can still redeem the other half.

Other benefits

Community moderation The automation of bonding raises the ability of communities to

moderate on their own, without the need for lower level janitors and moderators: code can be used to note complaints, raise juries, assess verdicts and effect remedy.

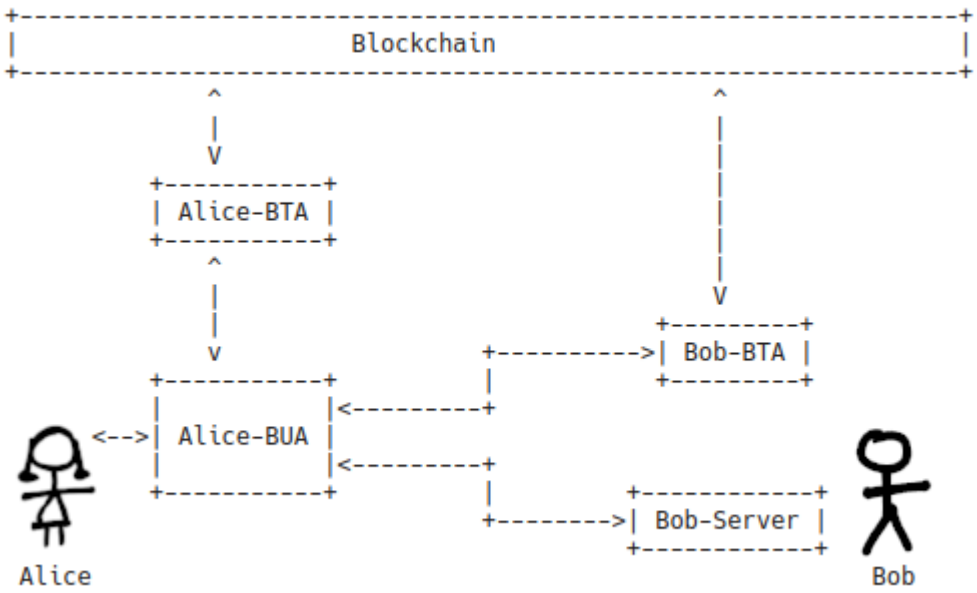
Onramping The vision for properly securing servers involved in bonding suggests server owners could serve to onramp their real life social group. Need some satoshis? Ask among your computerate friends.

Wider VPN offerings It can be easily imagined that residential users will offer VPN exit points in return for cryptocurrency.

Model

Overview

Alice wants access to a resource on Bob's server, however he wants strangers to bond their good behaviour first. She'll need to use software that's *Berewic Aware* to do so: a *Berewic User Agent* or *BUA*. (Completing the sequence manually is supported, but will be ignored for now.) For the purpose of the demopage, Alice-BUA will be your web browser, and Bob-Server will be a web server (neither *must* be, we'll show variations later). Bob doesn't want to manually handle bonding requests from strangers so he has a *Berewic Transfer Agent* or *BTA* do it instead. Alice finds it convenient to have her own BTA too, as it's easier to find apps that can use a BTA than apps that handle cryptocurrency within themselves (or through an OS call), just as it's easier to find apps that can use a SMTP server than find apps that implement an SMTP server.



Detail

- The two BTAs interact with each other only via the blockchain
    - Each BTA may communicate with more than one blockchain (not shown)
    - Each BTA may be accessible through the public Internet and/or through TOR or alternative network that supports TCP
    - A later revision might see them talk to each other directly, perhaps over a second layer network such as Lightning.

        - When you consider that if Alice is likely to continue to want to access Bob's resource (say a social media page), and Bob is likely to want her to continue to (why would you want fewer quality visitors?), there's no reason the bonds must immediately be on-chain. As Bob very likely knows others with whom Alice interacts, Bob's BTA would be a reasonable place for a Lightning node
        - TODO: The author is not at all certain Lightning yet supports bonds as imagined in this document
            - That is: Bob has the right to redeem funds from the first moment, but not the obligation to, and Alice can redeem her own funds once the bond expires. I suspect that on lightning Bob would have to actively return those funds.
    - Each BTA is envisioned as a common or garden computer running both berewicd (the front end software), and each respective wallet as a daemon.
    - Each daemon uses whatever method it needs to access its own blockchain.
    - Front end and daemon currently communicate over RPC.
    - Front end will use HTTP to communicate with each BUA
        - demopage uses HTTPS
- Bob owns the server and his BTA but otherwise has no direct role to play
    - He might get involved later to redeem Alice's bond for wrongdoing but that's not part of this protocol. It is, however, a facility him offered by the BTA: `/redemptions/N`, redeem amount/percentage/bond percentage
- Alice does have a direct input: she initiated the protocol by the connections she made, and her BUA will want to confirm with her that the details of the bond are actually acceptable to her.
    - A later revision might see her BTA also confirm instructions directly with her, perhaps with a 2FA feature such as OTP Authenticator
- Bob's server and BTA do not need to communicate directly, but do so indirectly via Alice-BUA.
    - TODO. Communicating via Alice-BUA allows her to track what's happening, and that would be reputationally better for Bob. OTOH it offers Alice one free abuse: she could make it a biggie then never forward the penalty, forgoing access in order to redeem whole value of bond at expiry. Having server communicate directly with BTA is bad, not least it requires different firewall arrangements, and a more featured Berewic-aware capability on the server. Or perhaps queue changes to be sent and have Bob send them during scheduled maintenance. Or, SHOULD be sent via Alice during service window, MAY be sent directly only after service window expires but before bond expires
    - Communicating via Alice-BUA is safe because Bob-Server and Bob-BTA both share a secret that Alice does not know, and that secret is used in a HMAC to authenticate messages sent via her BUA
        - This use of HMAC also means Alice can be sure at every step of what it's being agreed to, as everything else travels in plain text (although in a secured connection)
        - HMAC also means that a log of what was agreed to at each step can be kept by both sides so that neither side can later lie about what was agreed
        - TODO: I should like to add some signing process by which it can be established by some third party that logs were signed in the presence of a particular private key. This avoids problems with one party changing their private key then claiming not to have ever possessed the original. Alice can then publish incontrovertible evidence of Bob's malfeasance
    - Bob's server and BTA could be the same device but it's not necessary and

may well be better to be separate: it keeps the keys well away from the service being accessed for a start, and makes it easier to federate BTAs later.
        - Federated BTAs are two or more BTAs any one of which can answer for the server, providing a measure of resiliency
        - Alice-BTA and Bob-BTA could be the same BTA instance; there's no changes yet considered if this is so, though a very obvious one exists: Alice & Bob could transact completely off-chain.
- While the BTA may from time to time issue unique blockchain addresses to persons using it, the hot wallet utilised is the property of the BTA owner, not the person to whom the address was issued.
    - If the hot wallet contains 500,000s, the database will track that Alice owns 40,000s of them, Carol 80,000s of them.
    - Also note it may be desirable that the address issued is that of a cold wallet, with the balance being made available from a different & hot wallet.
        - This does mean Bob needs twice the funds, but also means he can publish his reserves without revealing as much as inputs are more disassociated from outputs

Types of server, types of resource

Alice is trying to access a *resource* on Bob's *server*. Both those terms should be used in their widest possible sense. That server could be:

- A computer of any description
- A telephone
- A restaurant, with the resource being a table near the window from 7pm to 8.30pm

That resource can be:

- *Addressable* such as a URL or a telephone number, where the address resolves to single endpoint. These resources can be directly covered by having the endpoint check the bond status. Here, that might be Apache httpd or the phone app.
    - The demo BTA implements the endpoint using PHP
    - Addressable resources can implement bonding in the client and daemon at clear points in the code: it's abundantly clear when a given address is in use.
- *Nonaddressable* such as an IRC channel or in game teamspeak session to which there is no public identifier. With no direct way to handle out-of-band data, inband methods can help. For IRC that's commands like `/nick`; for teamspeak that could use stegonographical techniques with the audio: while inband, Berewic aware clients would hide the transfer, legacy clients (or their users) would ignore it.
    - For more info read Alice Bonds an IRC Channel at http://berewic.com/manually-bond.html
    - A lot more work is required of client and daemon to establish if bonding is required as it's a configuration state of a channel that counts, not the address in use
    - If client or daemon does not handle bonding, bots at either end can do it by scraping the content passing through
- *Offline* such as a restaurant booking or city transit journey which is not experienced online. While there is obviously a need for the commitment to be online, the bond is provably good even in the absence of connectivity.
    - A QR code represents a convenient way of presenting it: the confirmation code is encoded as a QR which is then printed or otherwise captured as an image. As required that image is validated offline by being decoded back into a string which is then processed as normal. As the HMAC can be validated if the secret is known, connectivity is not needed.
    - A city transit journey by Alice may look like this: At the first entry gate of the day her phone bonds 200,000s that she'll pay her fare within 2 hours, and she

boards the train; she travels two stops and disembarks; the exit gates detect her entry point and calculate her bill of 10,000s, deducting it from the bond. A few hours later her phone automatically redeems the remaining 190,000s back to her credit. In cities such as London, England where there's a daily cap on such charges, Alice could guarantee that cap for up to 24 hours, with successive charges drawing down, the remainder being returned to her credit afterwards.

- *Duplexed* sees two channels used in the communication - one 'inband' carrying data, the second 'out of band' carrying control and metadata. HTTP is duplexed, as is FTP, Skype sessions, SSH etc
- *Simplexed* sees a single channel used for both data and control, with context deciding which is which. A WhatsApp group chat is simplex, as is IRC, a city transit journey and hotel reservation.

Examples

- Addressable, duplex: HTTP
  - Addressable: rfc7230 etc defines URL scheme
  - Duplex because of headers/body division
  - Headers can be extended to define and handle bonding
    - Either in the daemon httpd
    - Or server-side scripting such as PHP or Perl
- Addressable, simplex: mailman, listserv
  - Addressable because a mailing list can be addressed by its "-request" email address
  - Simplex because the contents of the subject line ("subscribe"/"unsubscribe") control behaviour
  - With no control side-channel, the protocol must be extended first, lest bonding control be mistaken for user contribution. That is, add "send bond details" et alii as a controlled subject line.
- Nonaddressable, duplex: WhatsApp group chat
  - Nonaddressable as no scheme by which a particular group chat can be referenced. Rather, users required to download a client and await an invite
  - Duplex as status changes etc are not negotiated by control verbs visible to others
- Nonaddressable, simplex: IRC
  - IRC has no URL scheme by which a single URL leads to "#bobchannel on DALnet". Rather there are multiple URLs that can get you there as well as multiple routes not involving URLs, such as configuring a client.
- Offline, duplex: a court hearing
  - Nonaddressable: there's no uniform identifier that indicates a court hearing at a particular time, and place. Further, it is not experienced online. Not withstanding that the court will have a postal address, room number etc, what works in one jurisdiction is not uniform with what works in another
  - Duplex: the control of a court hearing resides in multiple sources: a magistrate may deal in relatively simple and minor decisions as to who is or is not guilty and what justice demands, the clerks concerns themselves with what process and the law demands, the police with what security demands, that a lawyer properly represents the interests of the defence.
  - Consider that a court can require a Bail Bond, and such a transaction exactly matches a Good Behaviour Bond.
- Offline, simplex: a rental
  - Nonaddressable: there's no uniform scheme by which a particular property at a particular time may be located
  - Simplex: the lesee is entirely responsible in all aspects for the control of his side of the lease, that sums are transacted properly, that terms are adhered

to.
    - Consider that a rental deposit exactly matches a Good Behaviour Bond.

Protocol

The protocol sees four phases: *Setup* during which the need for bonding is detected and an initial request for more details is made. *Negotiation* follows allowing both sides to reach agreement about the bond. In the *Commitment* Alice funds the bond and waits for Bob-BTA to see it. In the final *Service* phase she receives access to the resource. Redeeming of the bond by either party is outside the scope of this protocol: they are standard transactions.

On Hash use

The BTA needs to separate out that Alice is visiting and is charged at one rate, vs Carol who's charged at a different rate. The BTA doesn't have to know anyone's actual identity to operate - that's a problem for Bob-Server. Nor does the BTA need to know the hostname for Bob's server (that's DNS' problem), nor which resource is being addressed (that's Bob's problem), just that any given user for a given resource and a given server is consistently handled. As usernames, hostnames and resources can be of varying lengths & descriptions, it makes more sense to store identifiers instead - and hashes are a perfect way to do this, as is Bob manually inventing identifiers.

Note that for the our purposes here, a Berewic resource can refer to one or more resources on Bob-Server if Bob so chooses. Indeed, multiple different Bob-servers can reuse the same identifier with Berewic if he so chooses. Hashes are a great way to accomplish this.

Also note that using identifiers makes for good privacy controls: if the BTA is compromised the use of identifiers means that it cannot be determined who has bonded, where they bonded, and for what they bonded. Bob is of course free to use personally identifying data if he wishes.

On HMAC use

The BTA needs to hear details from the Server, report results back to it, and inform Alice what's going on. This can be accomplished by two network connections: BTA to Alice and Alice to Server. Dropping the connection direct from BTA to Server reduces the things that can go wrong, not just with network conditions but with ancillary work such as firewall maintenance.

The obvious problem giving Alice information to hand on is that she might change it to her benefit, or be otherwise mischievous, and that's where the HMAC (Hash-based Message Authentication Code) comes in. The BTA and Server both know a secret, but Alice does not. When plain text information is passed between them, a HMAC code is generated to include all the information Alice can see plus the secret, and is sent along with it. If Alice changes the information she can see, then HMAC code becomes wrong and the other end will refuse to use it. If she strips out the HMAC altogether, the other end will again refuse to use it.

Setup phase

1. Alice uses a BUA called Alice-BUA to access Bob-Server.

    a. The protocol does not address how this happens: it could be over TCP/IP but could equally be over the PSTN or IP over Avian Carrier
2. Bob-Server, recognising that Alice-BUA has not supplied a bond for her connection, declines to allow access. A description of how this might work for an HTTP resource follows this section
3. Bob-Server directs Alice-BUA to Bob-BTA and the URI *CR*. A typical redirect will look like:
   `10,https://bobs-bta.mpsvr.com:8443/proposal/4025061200627151c0c2b7b80d7af47b3b5c8bd2/e8ebaa9cb957844658dd0bcea2aeae6ffb1e2349?idv1=bf1f8ecf&ratev1=normal&hmacv1=c78db3121a8ed993bb9dfe63cdd12a348032f5e4`
   Where:
    a. `10` indicates the preference Bob has that Alice use that particular BTA, with lower values being more preferable
    b. The comma terminates the preference value
    c. `https://bobs-bta.mpsvr.com:8443` describes the protocol, hostname and port where the BTA can be found
    d. `/proposal` This is the start of value *CR* and indicates to the BTA what kind of request is coming
    e. `/4025061200627151c0c2b7b80d7af47b3b5c8bd2` An identifier for the Bob-Server that initiated the request. This is a hash of `https://berewic.com:8443`
    f. `/e8ebaa9cb957844658dd0bcea2aeae6ffb1e2349` An identifier for the resource. This is a hash of `/covered-resource`, the URI of the demopage
    g. `?idv1=bf1f8ecf&ratev1=normal&hmacv1=c78db3121a8ed993bb9dfe63cdd12a348032f5e4` The query string carries additional info for Bob-BTA from Bob-Server
        i. `idv1` An identifier for a particular user. Again a hash, for the demopage a crc32 hash of the user's IP Address.
        ii. `ratev1` An order from Bob-Server as to what value of bond should be required. Here it says 'normal' (whatever that might mean) but it could equally be for a certain amount. One might use "zero" to specifically say this user be allowed to connect without bonding; that is, be immediately given a confirmation code without any exchange of cryptocurrency. We might do this rather than check Alice credentials each time because
            1. It's likely cheaper than processing credentials or otherwise repeatedly identifying Alice
            2. The resource, indeed site, may not require credentials, indeed they may not be available such as with a telephone call
            3. The eventual server may be too low powered to manage credentials (note that it's not necessarily so that the site Alice visited for this link was her final destination)
        iii. `hmacv1` An HMAC to authenticate this request.
4. Alice-BUA recognises that direction for what it is and constructs a new connection from itself to Bob-BTA and makes a request for CR. This does happen over HTTP
5. Bob-BTA understands from CR: that Bob-Server created the request, what URI was desired, which user desired it and other facts, including that the HMAC authenticates the request really did come from Bob-Server.
6. If the request specifies Alice is to be recognised as bonded without transfer of cryptocurrency, skip straight to the Service phase.

Negotiation phase

Content is transferred throughout as JSON

1. Bob-BTA supplies to Alice-BUA a list of templates reflecting bonds Bob-BTA is prepared to accept. The pretty-printed JSON would look something like this:

```
{
  "version": "0.1",
  "timestamp": 1551033502,
  "0": {
    "version": "0.1",
    "idv1": "bf1f8ecf",
    "type": "bond",
    "value": {
      "currency": "btc",
      "value": "0.0004"
    },
    "network": {
      "networkname": "testnet",
      "buyer-address": "",
      "p2sh-address": "",
      "seller-address": "2Mv7JjZLNrueGqyWcxisBL3jHWKxSpkYbsu"
    },
    "min-timeout": {
      "minblocktime": 1551038902
    }
  }
}
```

   The version and timestamp of the lists, plus the templates themselves starting from template #0. In the example above the template describes a bitcoin bond over testnet. `Idv1` is a reference to the particular user from Bob-Server, perhaps a hash of that user's UID. It's not necessary, and very likely not desirable, that a BTA have personally identifying information. The type allows for later expansion. `Value` specifies how much of which currency. Facts about the network are added although note `buyer-address` and `p2sh-address` are both empty. The `seller-address`, used by Bob as the address to which to redeem funds in the face of wrongdoing is known, so is filled in. The network name is important: bitcoins could be transferred over mainnet, testnet or regtest. With traditional fiat we might indicate the LINK network, SWIFT, or "*" for "I don't mind how it arrives." Last the template indicates how long the bond should be good for - this will be the Bond Window.
2. Alice-BUA connects to Alice-BTA and supplies the list just received.
3. Alice-BTA filters and reorders the list according to the means available to it, for instance, removing Bond templates where the currency is not recognised or for which funds are insufficient.
4. Alice-BUA then communicates with Alice herself as to the need to make a bond for what she's trying to do, and the recommendations it received from Alice-BTA. The protocol makes no proscription against what form this communication takes: email, app, website or other.
5. Alice approves the bond.
6. Alice-BUA sends the approved bond to Alice-BTA to have a redeem address added, known as Alice-RA.
7. On receiving it back Alice-BUA sends the template with Alice-RA back to Bob-BTA for the P2SH address. The P2SH address is the address on the blockchain where the bond funds live until redeemed: the actual address depends entirely on the facts of the bond so cannot be determined before those facts are negotiated.
8. Bob-BTA double checks the template and adds the P2SH address it calculates to the

template before returning it.
9. Alice-BUA supplies the completed template back to Alice-BTA for effecting.
10. Alice-BTA double-checks that the template and ensures the P2SH addresses added is the also the P2SH address it itself would calculate.

At this point, all parties have agreed to the bond, but neither has yet committed to it.

Commitment phase

1. Alice-BTA has enough data now to commit to the bond, and does so, informing Alice-BUA of the transaction id.
2. Alice-BUA now polls Bob-BTA every 30 seconds as to the status from its side.
    a. Alice-BUA polls status from Bob-BTA/bonds/<P2SH Address>
    b. Polling too often leaves Bob-BTA free to issue a cached or 420 Enhance Your Calm response
    c. If Bob-BTA does not recognise the P2SH address, it issues a `404 not found`
        i. A 404 response despite agreeing to the bond above suggests a very bad break, or malfeasance on either Bob's or Alice's part. In theory Bob is incentivised not to let this happen through his desire for Alice's continuing patronage. He could be trying to take the funds without providing service. Alice might have forged the templates to Bob's disadvantage and this represents the next step in her fraud. TODO - what to do? Elsewhere: proving who said what.
        ii. Could it be an idea to precede this stage with a confirmation step looking for other than a 404. The response being verifiable as coming from Bob? If 404d Alice could publish the fraud outside of the protocol.
    d. If it recognises it but has not seen it on the blockchain, it issues `202 Accepted`. Per the HTTP RFC this is correct for when processing is underway but the results are not likely known until later.
    e. If it has seen it but there are insufficient funds to honour the bond, it issues `402 payment required`. This might be because multiple transactions fund the bond, but too few of them have yet been mined.
    f. If everything is in order, proceed to next phase

Service phase

Both parties have now committed. This phase can be repeated at any point

1. At each request Bob-BTA determines if it agrees the P2SH address is still funded sufficiently, or that there are other reasons to permit the request.
    a. If not, Bob-BTA returns a `402 Payment Required` response.
    b. If it is, Bob-BTA returns `200 OK` as well as a new confirmation code - *CC* - to Alice-BUA.
2. CC is constructed using data from CR as well as with a new HMAC and looks like: `idv1=52676381&bta=78f7&amount=0.0004&locktime=1548464532&mtime=1548461575045848&hmacv1=48d6bf813687f3b9d9b1fa5a2300d0ee3c571530`
    a. `idv1` matches the `idv1` in the template negotiation above
    b. `bta` is an identifier for the particular instance handling the request. This allows for each instance to use a different secret
    c. `amount` identifies how much of the currency actually remains at the address. That it can be greater than expected may be used to proportionally increase the service window. That it may be less than expected may reflect a penalty for wrongdoing or service charge.
    d. `locktime` identifies when the bond is good until. Like `amount` this could be

used to proportionally increase the service window
    e. `mtime` is the timestamp this CC was created, down to the microsecond
    f. `hmacv1` authenticates the CC
3. Alice-BUA then returns to Bob-Server and repeats the original request this time including CC.
4. Bob-Server, sharing a secret with Bob-BTA, can reconstruct what CC should be, and by doing so confirm Alice has properly bonded her good behaviour and so allow access to the resource.

Berewic access of a HTTP resource

1. TODO: For HTTP, service refusal from Bob-Server should not be understood in terms of the http status code, but in terms of the content delivered OR a separate 'bond status code', modelled here after the HTTP status code.
    a. Perhaps the HTTP status should match the bond status IF the original request asserted a bond of any kind. Reasoning: HTTP credentials do this with `401` if credentials bad even if the same resource would 200 without credentials
    b. With the former, `200 OK` responses occur if bonded, not bonded, or with expiring/expired bond: you are getting what your status allows.
    c. With the latter, as well as the HTTP status line, a second header is added `berewic-status: 200 OK` might be used to inform the BUA that the server remains happy with the bond.
        i. `200 OK 1550785723 0` The bond is good until timestamp provided, and hasn't been changed since second timestamp provided. Alice-BUA is in charge of warning Alice should the second timestamp be unexpectedly changed.
            1. Bob-Server is not obliged to accept a given CC until the first timestamp provided, see `306` response below
        ii. `306 Get New Code` While the confirmation code provided seems valid, Bob-Server is insisting a new confirmation code be sought from Bob-BTA. This might be because Alice has been penalised and so may no longer have sufficient funds to bond her connection. Bob-Server MAY elect to require a new code at any time. This code might also come with a new URI to visit that includes the instruction to penalise Alice, or upgrade her to white-listed and so exempted from being bonded, etc
            1. While she could refuse to visit the link and so leave her bond unpenalised, without doing so she's now prohibited from accessing the resource because of this bond status code. Bob could alternatively directly inform his BTA of the penalty.
            2. If she does visit the link the HMAC ensures she can't downgrade her fine, or change it to upgrade her to whitelisted.
        iii. `402 Payment Required` while bond is in date, it no longer has sufficient funds to honour commitments. Bob-Server knows this from reading the confirmation code.
        iv. `418 Expired` the bond is no longer good for access. That is, while the Service Window has passed, the Bond Window may still be open.
2. For HTTP, this redirection is indicated by the presence of one or more headers `berewic-transfer-agent:` followed by
    a. a preference as a string represented integer
    b. a comma terminating the preference
    c. a URL for one BTA consisting of:
        i. `https://`

ii. The hostname of the BTA
iii. Optional colon and port, otherwise defaulting to :443
iv. CR the covered resource
    1. /bond
    2. /<Bob's server identifier> eg, an MD5 hash of the server hostname.
    3. /<Bob's resource identifier> eg a hash
    4. ?id=<Alice identifier> eg a hash
    5. &<note>=<value> one or more optional instructions from the server, for instance rate=low might inform the BTA to ask for a lower than usual bond
    6. &hmac=<hash>

d. A user agent SHOULD honour the lowest preference integers first particularly as the higher integers may cost both parties more

BTA Users and Accounts

The first priority is safety, the second is adversity management

A user is a person who owns one or more accounts. Ownership of the account is synonymous with knowing the credentials necessary to operate it. Every user has a parent user, other than the owner of the device. A user has a share of the bitcoins or other cryptocurrency in the parent user's control, except the owner, who has custody of all bitcoins or other cryptocurrency in all wallets on the device. This does mean that if you are not the owner, your funds of the device are mere ledger entries: all are greatly recommended to keep the bare minimum of funds on a BTA: you can probably expect that if an account transfers thousands of BTC to it, the entire device will suddenly vanish offline!

This structure of users and subusers allows, for instance, Bob to fund his BTA to 1,000,000s and sell the right to 800,000s to Carol, who sells the right to 300,000s to her Husband Dave and their Daughter Erin, and they in turn sell half each to their neighbour, Frank.

An account comes in two forms: admin and normal. An admin account is what the owner can use to administer his holdings but does not have great rights to spend funds. That is something reserved to normal accounts.

Take the email client Thunderbird for example. Given the admin credentials it cannot directly transact. Rather it uses those credentials to obtain more minor credentials that do allow it to transact, but are more heavily limited. The Admin credentials SHOULD be discarded such that all changes require checking with a human.

If the credentials are stolen from Thunderbird, the thief is now very strictly limited in how much can be stolen & it will be clear where to start looking for the culprit.

If the original credentials are stolen, withdrawal is limited to that which matches the least of the subaccounts: thus if Safari may access 2% and 5,000s, the original credentials can withdraw the least being 1% (from Thunderbird) and 5,000s (from Safari)

- It may make more sense to require a deposit/total-withdrawal subaccount with

appropriate extra security
- It would also make sense to have a 2FA option such that any or some uses of the BTA requires a 2FA code.
- It would be mandatory to ask for password and 2FA code even if they are not set up for that subaccount: attackers should not benefit from knowing if they are not in use.

A withdrawal to another wallet address would mean a straightforward transaction (or, it could be “lined up” to occur in one larger go, that is, scheduled to be sent on Friday, with the one quarter signed off each of Monday, Tuesday, Wednesday & Thursday.) Each signing off would be another opportunity to reveal what’s going on to the true owner.

A withdrawal to another account on the same machine would happen instantly IF the funds are being credited to a subaccount OR the funds belong to the owner (on the grounds of not otherwise being spoken for.)

Subaccounts can themselves have subsubaccounts such that Bob can dedicate 1,000,000s to an account for Carol, and she can dedicate 300,000s and 400,000s of it to accounts for Dave and Erin respectively. Carol can not withdraw their funds except by first withdrawing from their accounts back to hers - and she doesn’t have that permission.

Empty Accounts can be opened anonymously. Obviously any received funds would come externally rather than from the Owner.

Users might take a percentage or fixed amount of funds travelling through subusers. This may carry on up the chain: Erin may transact 100,000s, Dave takes 1% of what Erin transacted (1,000s) because Erin has an account through him; Carol gave Dave the account and takes 10% of what he just earned (so, 100s) and Bob owning the BTA and having given Carol the initial account takes 20% of what Carol just earned - 20s. Thus, onramping becomes an ongoing revenue stream.

Resale

Carol could agree to give Bob $10 in return for 30,000s. Bob, takes the $10, creates the appropriate account on his BTA, and assigns the satoshis over. This is an *Internal Transfer*. Carol then spends 25,000s buying a doobry from Dave, who does not share the same BTA. This is an *External Transfer*.

Internal transfer

Bob is known in his family as the Go-To Source for Bitcoins - he runs a BTA with 2,000,000s on it, which he sells in 20,000s lots for $10 each. Carol knows Bob through their workplace and approaches him for satoshis.

- Carol asks Bob for 20,000s in exchange for $10.
- Bob agrees
- Carol gives Bob $10
- Bob accepts the $10
- Bob gives Carol the IP address of his BTA
- Carol visits Bob’s BTA and creates an account. She now has hostname,

@username, and password via visiting `/account`
- Carol gives Bob her @username
- Bob assigns 20,000s to her via visiting `/transfer` and assigning the satoshis over.

Time passes and Carol wishes to repeat the trade

- Carol asks Bob for 20,000s in exchange for $10.
- Bob agrees
- Carol gives Bob $10
- Bob accepts the $10
- Bob assigns 20,000s more to her via visiting `/transfer` and assigning the satoshis over.

External transfer

Carol has 40,000s and wishes to purchase a doobry for 35,000s and has received the address to which it should be sent. It is beyond the scope of the protocol to check that she received the correct address.

- Carol visits Bob's BTA
- She visits `/transfer`
- She fills in the form including the remote cryptocurrency address instead of the @username
- She submits the form and the process begins
- Assuming she has enough credit and that all checks pass, the equivalent amount of crypto is transferred from the BTA's hot wallet.
    - Having checks pass could include 2FA checks such as entering three random digits from the remote address. This would raise a significant barrier to a mitm attack whereby the address that arrives is not the address that she wanted sent.

Competitors

No one is known to be working on solving the same problems as Berewic however many are operating in the same space with considerable overlap. Research on any is yet to be done

- 0xchan. A project to create an immutable imageboard based on the Ethereum network. Noting because:
    - The project is building a separate client which I don't propose here, and they're suggesting with it the posting of 2ETH bond in order to settle fines
- Veropost.com A pay-to-read-email service originally found at Reddit

Reference Server

Idea: can a 2nd server offer a means to immediately migrate funds to a brand new address that can't be calculated on the stolen machine?

The guidinging principles for the server are:

- Minimising Loss of funds in the event of compromise
- Maximising security over UX (unlikely to be a graphical front end to use, etc. Minimised attack surface)

- Open source: GPLv3

Service design

Covered elsewhere.

Platform design

1. A base such as Raspberry Pi, HP MicroServer Gen8 (relatively small, cheap)
2. Two disks in a RAID 1 configuration (data security in event of disk failure)
    a. SSD(s) as a future options
3. Use TPM if present (OS will not spin up in the event of physical changes to the hardware)
4. Tamper evident seal on outside (visible indication of foul play)
5. Debian Linux v9 as OS, hardened kernel, minimal install
    a. selinux as a future option
6. EncFS as file system containing wallets & other important files (Substantially hinders fund theft in the event of physical theft)
    a. EncFS key on a physical presence indicator, such as an SD Card that should be removed following boot. (Key too long to reasonably memorise, removable card indicates likely physical presence of owner)
7. Ext4 as host file system (Faster for non-critical storage than EncFS)
8. IPv4 support
    a. IPv6 as a future option
9. Very heavily constraining iptables firewall
    a. Everything blocked
    b. anything opening the firewall is per protocol, per address and per process.
10. Tripwire as a future option
11. TOR support
    a. TOR Endpoint
    b. TOR node
12. Automated backup facilities
    a. Symmetric encryption of a tar ball of critical files
    b. public key encryption instead as a future option
    c. Download of same only via the service and thus subject to controls
13. Daemons separated out into different users (confines damage should one daemon be hostile)
    a. bitcoind - implements BTC
    b. litecoind - implements LTC
    c. lnd - implements lightning network - as a priority
    d. Other daemons as a future option
14. Network accessed through separate user (httpd will not have access to wallet files but instead have to use RPC)
15. Apache v2.4 as http daemon
16. PHP v7.2 as scripting platform
17. PostgreSQL v9.6 as databasing platform
18. RBAC approach to connecting users. Example: Admin account can create subaccounts but not transact; subaccounts can transact only up to individual limits.

    a. ABAC as a future option
19. RESTful API.
    a. RPC for certain functionality is a possibility
    b. Documentation

Berewic Transfer Agent API

`/account`

`GET`. I want a new account

`POST`. Here are my details. Returns `/accounts/N`

Note that using no credentials indicates an admin account is to be created. Using credentials indicates a subaccount. Admin accounts SHALL NOT have greater withdrawal privileges than the least of the subaccounts. Subaccounts ONLY will be able to retrieve deposit addresses and enter into bonds. All usernames must start with an @ symbol such that there's no ambiguity between a cryptocurrency address and a username

`/accounts`

`/accounts/N`

`GET`. Give me the status of account `N`

`PUT`. Change account `N` to these details

`DELETE`. Close account `N`. Transfer funds first!

`/accounts/N/al`

`GET`. Retrieve subaccounts associated with `N`

`/proposal`

`/proposals`

`/proposals/N`

`GET`. Accept this proposal. Fills in buyer-address and connects to remote end for P2SH address. Double checks and funds the bond. Returns the txid

`/bond`

`/bonds`

`/bonds/N`

`GET`. Get me the facts of Bond with P2SH address `N`

`/redemptions`

`/redemptions/N`

`GET`. Get a form for redeeming against P2SH address `N`

`POST`. Start the redemption process, obtains `/redemptions/N/M`

`/redemptions/N/all`

`GET`. Retrieve a list of all redemptions against `N`

`/redemptions/N/M`

`GET`. Obtain the status of the `M`th redemption against P2SH address `N`

`/request-proposals`

Used for manual connections

`GET`. Retrieve a form into which BTAs can be pasted

`POST`. Upload pasted BTAs, which will be ordered, and the most preferred one connected to in order to retrieve proposals. Returns list of proposals that could be accepted in form `/proposals/N`

`/transfer`

`GET`. Get a form for transferring satoshis outright. A transfer to an address starting with @ is considered internal, otherwise it's considered external. There is a freeform field for notes.

`POST`. Start the transfer process. Returns `/transfers/N`

`/transfers/N`

`GET`. Get the status of transfer `N`

`/receive`

`GET`. Get a form requesting a receive address

`POST`. make request, returns `/received/N` where `N` is the cryptocurrency address. Here's an interesting question. If Carol sets up address N, receives 10,000s to it and is never heard from again, and then three years later - completely out of the blue - Ƀ10.0 is received to the address, what should happen? Clearly it's very likely a mistake of some sort. I notice Barclays Bank seem to transfer unclaimed funds after 15 years to an entity that uses those funds towards the UK National Lottery. Possibly just alert the owner that funds have unexpectedly arrived (== more than 1 month after last sighting of user, repeated at 2 months, 6 months, 12 months, and rolled up in a year end report?)

`/received/N`

`GET`. Get information about address `N` in the local wallet. Visible to the owner and original requestor, not otherwise.

- Requestor can see "Credited: 0.00"
- Owner can see "Balance: 0.00, Credited: 0.00"
- Other facts likely
- The act of checking `/received/N` by requestor or owner is what causes an update to happen. Don't check the address? Then it has not arrived. This relieves the BTA from constantly sweeping old, old addresses looking for new values not expected to arrive. Does this break REST? It changes something on the server - the balances etc, but is that worse than the visitor count?
- When funds arrive and address checked, balance is updated
- When Balance is updated, Credit to requestor happens

Timeline and todo list

- HTLB introduction to bitcoin-dev
- Berewicd does not yet sign transactions as that code needs special writing (cf Pieter Wuille.) until then use should be confined to testnets.
  - It seems to need a php elliptic curve library, but I've not yet looked for one
  - It seems to need the signing to be specially constructed but I've not yet looked at how that happens
  - It may be that bx can be used for this purpose but I've not yet investigated it
  - It may be that HTLBs will not be mined or propagated on mainnet because they are "non-standard" - I'm not clear on what defines those, or if that applies to the conceptual level higher, ie, that all P2SH transactions are by definition standard.
  - Direct support in the wallet negates the need for any of the above
- No effort has been made towards a UI much less a UX: berewicd is a restful service so will have a very bare bones html feel to anyone visiting directly.
  - This does allow room for UX devs to skin a service, or develop an app, and perhaps front their own BTA with same
- No proper effort has gone into defining anything about berewicd - what there is has grown organically for the proof of concept. For instance the API has just been built up

as whatever was expedient to solve the issue of the moment. Are there lessons to be learned, rationalisations to be made, formal definitions to be laid down? Absolutely
- A greenpaper, then a whitepaper, giving a proper treatment is desirable.
- An RFC document defining the berewic protocol is desirable if only for interoperability reasons
- No concerted peer review
- Analysis of real world analogues for what's suggested here, might suggest additional avenues for investigation

Glossary

Ledger possession You don't own the money, someone else owns it, and you possess a claim to it hoping they'll give it to you when you ask.

Notes without a home

Adding other cryptos. Some alts will be easier than others. If an alt uses OP_CHECKLOCKTIMEVERIFY it's likely to get HTLB support earlier than those using some other method.

Implementing Berewic with traditional fiat. There are considerable headwinds to implementation using traditional currencies:

1. It would need to be centralised, perhaps a Google Bond, Facebook Bond and maybe a Federal Reserve Bond whereby that centralised entity would accept your currency to credit your account. This would necessarily exclude large parts of the world (For instance, Google barely serves the Chinese market at all, much less offers to receive small amounts from Chinese citizens)
2. There is no standard for Bonding connections (something a Berewic RFC would look to address) and so access will be extremely fractured: one may find that in order to access a resource covered by ACME each user would need to create and fund an ACME account
3. It would be expensive. One can expect centralised entities will quickly start to gouge not just malefactors but also the innocent hosts and users.
4. It would be less reliable. Cryptocurrency bonds are non-repudiable, so both parties can be certain what's going on. With a traditional currency bond the network is less reliable by being more centralised in particular data centres (Microsoft has had +24hours of downtime in the last six months for their flagship Office 365 – http://currentlydown.com/office365.com), whereas Bitcoin has had zero downtime in ten years, so there is a non-zero chance that a fiat bond would simply not be honoured on time
5. It is censurable. It is an ideological position that merely being out of favour should not in and of itself be a cause that someone should be denied access to an economy. It's quite clear that the offence of Wrongthink is alive and well, and that deplatforming is recognised as a weapon against a person not against falsity. It therefore follows that (for example) Tumblr Bonds might well choose not to honour the bonding of connections to resources a loud section of their subscribers consider plusungood. Too few follow the Voltairean principle!

This is not to say doing bonds in cryptocurrencies are without their own problems, not least that an Indian earning $3 per day may have to bond the same as a Swiss earning $250 per day.