Gmail **Bryan Bishop <kanzure@gmail.com>**

[bitcoin-dev] Hardware Wallet Standard

26 messages

Jonas Schnelli via bitcoin-dev <bitcoin-dev@lists.linuxfoundation.org> Tue, Aug 16, 2016 at 9:10 AM
Reply-To: Jonas Schnelli <dev@jonasschnelli.ch>, Bitcoin Protocol Discussion <bitcoin-dev@lists.linuxfoundation.org>
To: Bitcoin development mailing list <bitcoin-dev@lists.linuxfoundation.org>

Hi

Unfortunately, there is no standard in how desktop- or mobile-wallets can interact with a hardware device resulting in wallet vendors adding plugins with proprietary code for non-standardized interfaces.

I started a BIP (extreme draft, feel free to improve language, grammar and content) to address this missing part of the ecosystem.

I think it would be extremely helpful if @ledger, @trezor, @voisin/@breadwallet, @electrum, @bitpay (and more?!) would help working on a such standard.

The BIP describes two approaches how to communicate (pipe and URI-scheme) with the signing-devices app, although, in my opinion, all major platform do support the URI approach (maybe we could drop the pipe approach then).
The URI approach means that there is no need to configure the application location in order to start a inter-process(-app) communication.

Mediawiki:
https://github.com/jonasschnelli/bips/blob/8abb51f0b21b6664388f6e88f6fd642c90d25dca/bip-undef-0.mediawiki

</jonas>

---- BIP (rough early stage draft)

<pre>
  BIP: ???
  Title: Detached Signing
  Author: Jonas Schnelli <dev@jonasschnelli.ch>
  Status: Draft (early stage!)
  Type: Standards Track
  Created: 2016-08-02
</pre>

== Abstract ==

This BIP describes a way how wallet applications can decouple sensitive privatekeys from the internal keychain and interact with a signing-devices (hardware wallet, "cold" storage) over a generic interface in order to get signatures.

== Motivation ==

It seems like that the current approach for allowing signing-devices to interact with third party wallets is to build a plugin [1][2][3]. Adding plugins for each hardware wallet type will increase possible security issues and result in multiple proprietary-third-party code within the wallet application with very similar structures.

A generic interface how wallets can interact with signing-devices would result in better user experience, less critical code and simpler adaption for various signing-devices.

== Specification ==

In order to support desktop- and smartphone-wallet-applications, this BIP describes two slightly different approaches (process pipe and URI call) in how to interact with the signing-devices. If possible, the modern URI approach should be chosen.

=== Signing-Device-Controller-Application ===

To allow a generic interface while still allowing different ways how to internally communicate with the signing device itself (USB, TCP/IP, air-gapped Qr-Code scanning, etc.) a controller-application is required.

=== General signing process ===

The wallets signing process must be according the following principal:
* Wallet prepares signing-request-object including bitcoin-transaction or message together with metadata (scriptPubKey, hd-keypath of the inputs)
* Wallet passes signing-request-object to the signing-device-controller-application
* Signing-device-controller-application processes signing-request-object, eventually shows UI, user can sign or cancel
* Signing-device-controller-application sends back signing-response-object with signatures or an error
* Wallet processes signing-response-object and completes data-object creating process (example: add signatures to transaction and broadcast)

=== Desktop Process Intercommunication ===

Desktop wallets can interact with a signing device over process intercommunication (pipe) together with a signing-device-controller-application.
As specified below, the signing-request-object is a URI string passed through the pipe. The desktop wallet needs to wait (with a recommended timeout between 1 and 5 minutes) until the signing-response-object will be sent back by the signing-device-controller-application.

=== Smartphone/URI App Intercommunication ===

Smartphones and modern operating systems are trying to sandbox applications and interprocess communication (on pipe level) is mostly disallowed.
On smartphones, we must use URI-schemes.
The wallet can pass information to the signing-device-controller-application by using a predefined URI scheme.

<code>detatchedsigning://<command>?<querystring>&returnurischeme=<returnurischeme://></code>

The <code>querystring</code> must be URI encoded.
RFC 2616 does not specify a maximum length of URIs (get request). Most modern smartphone operating system allow URIs up to serval megabytes. Signing complex data-structure is therefore possible.

The <code>returnurischeme</code> must contain a URI schema where the result of the signing process should be returned to.
The returnurischeme must be populated and "opened" once the signing process has been completed (or cancled).

=== Signing Request ===

The signing request is a flexible URI-Query-String that will be used by

the Signing-device-controller-application for user confirmation as well as for creating the signature.

The URI-query-string must conform to the following format:

<code>detatchedsigning://sign?type=<bitcoin-p2pkh|bitcoin-p2sh|bitcoin-msg>&data=<raw-transaction|message>&inputscripts=<scriptPubKey-input0>,<scriptPubKey-input1>,...&inputhdkeypath=<hdkeypath0>,<hdkeypath1>,...&returnscheme=<returnurischeme></code>

type = type of the data to sign
data = raw unsigned bitcoin transaction or text-message
(optional)inputscripts = scriptPubKey(s) of the inputs in exact order
(optional)inputhdkeypath = hd-keypath of the inputs in exact order
(optional)returnscheme = a URI scheme where the response must be sent to (smartphone approach)

* inputhdkeypath or inputscripts must be provided.

=== Signing Response ===

The signing response is a flexible URI-Query-String that will be sent back to the wallet application and must contain the signatures or an error code.
The URI-query-string can be opened (smartphone approach) or will be sent back though the interprocess pipe.

<code><returnurischeme>://signresponse?errorcode=<errorcode>&signatures=<signature-input0>,<signature-input0>,...</code>

In case of ECDSA, the returned signatures must be normalized compact signatures with the size of 64bytes (128 hex chars).

==== Possible error code ====

0 = no error
1 = user canceled
2 = timeout
10 = missing key identifier (missing HD keypath or input scriptpubkey)
11 = unsupported signing type
12 = could not resolve script
50 = unknown internal error

==== Examples ====

===== Simple p2pkh transaction =====
Unsigned raw transaction:
<code>0100000001fd3cd19d0fb7dbb5bff148e6d3e18bc42cc49a76ed2bfd7d760ad1d7907fd9ce0100000000ffffffff0100e1f505000000001976a9149062e542a78d4fe00dcf7cca89c24a8013c381a388ac00000000</code>
(input ced97f90d7d10a767dfd2bed769ac42cc48be1d3e648f1bfb5dbb70f9dd13cfd vout:1, output: P2PKH mtgQ54Uf3iRTc9kq18rw9SJzngvF5ryZn 1 BTC)

signing-request URI must be:
<code>detatchedsigning://sign?type=bitcoin-p2pkh&data=0100000001fd3cd19d0fb7dbb5bff148e6d3e18bc42cc49a76ed2bfd7d760ad1d7907fd9ce0100000000ffffffff0100e1f505000000001976a9149062e542a78d4fe00dcf7cca89c24a8013c381a388ac00000000&inputscripts=76a914531148ad17fdbffd4bac72d43deea6c7cf0387d088ac&inputhdkeypath=m/0'/0'/1&returnscheme=myapp</code>
The <code>inputhdkeypath</code> is optional in this case

signing-response URI must be:
<code>detatchedsigning://signresponse?error=0&signatures=<128hex-chars></code>

===== Simple a bitcoin message =====
Message: <code>Lorem ipsum dolor sit amet</code>

signing-request URI must be:
<code>detatchedsigning://sign?type=bitcoinmsg&data=Lorem+ipsum+dolor+sit+amet&inputhdkeypath=m/0'/0'/2</code>

signing-response URI must be:
<code>detatchedsigning://signresponse?error=0&signatures=<128hex-chars></code>

=== Support for multiple signing-devices ===
Must operating systems allow only one registered application per URI-scheme. To support multiple signing-devices, wallets and signing-devices can optional add support for brand based URI-schemes.

In addition to the standard URI scheme, signing-devices-controller-applications can register an additional URI scheme (with the identical request/response syntax and logic) including a brand-identifier.

Registering a brand-identifier based URI scheme without registering the default URI scheme is not allowed.

Wallets can detect if a certain brand based URI scheme is supported and therefore gives user a selection if multiple signing-devices where detected [4][5].

<code>detatchedsigning<brandid>://</code>

Supported brand-identifiers are:
* trezor
* ledger
* keepkey
* digitalbitbix

== References ==
[1] https://github.com/spesmilo/electrum/pull/1662
[2] https://github.com/spesmilo/electrum/pull/1391
[3] https://github.com/bitpay/copay/pull/3143
[4] https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIApplication_Class/
[5] https://developer.android.com/reference/android/content/pm/PackageManager.html
== Acknowledgements ==

== Copyright ==
This work is placed in the public domain.

signature.asc
1K

Pavol Rusnak via bitcoin-dev <bitcoin-dev@lists.linuxfoundation.org> Tue, Aug 16, 2016 at 9:48 AM
Reply-To: Pavol Rusnak <stick@satoshilabs.com>, Bitcoin Protocol Discussion <bitcoin-dev@lists.linuxfoundation.org>
To: Jonas Schnelli <dev@jonasschnelli.ch>, Bitcoin Protocol Discussion <bitcoin-dev@lists.linuxfoundation.org>

I think it does not make sense to try to get this standardized for

current Bitcoin transactions. They are just too complex.

What might be interesting is to have something similar for Segwit and Lightning transactions.

* TREZOR performs extended validation of the inputs, when all of prev-txs are streamed into the device and validated. Your standard does not tackle this at all and I don't think it's worthy to make this standard unnecessarily complicated.

--
Best Regards / S pozdravom,

Pavol "stick" Rusnak
SatoshiLabs.com

_____
bitcoin-dev mailing list
bitcoin-dev@lists.linuxfoundation.org
https://lists.linuxfoundation.org/mailman/listinfo/bitcoin-dev

signature.asc
1K

Jonas Schnelli via bitcoin-dev <bitcoin-dev@lists.linuxfoundation.org> Tue, Aug 16, 2016 at 10:13 AM
Reply-To: Jonas Schnelli <dev@jonasschnelli.ch>, Bitcoin Protocol Discussion <bitcoin-dev@lists.linuxfoundation.org>
To: Pavol Rusnak <stick@satoshilabs.com>, Bitcoin Protocol Discussion <bitcoin-dev@lists.linuxfoundation.org>

> I think it does not make sense to try to get this standardized for
> current Bitcoin transactions. They are just too complex.
>
> What might be interesting is to have something similar for Segwit and
> Lightning transactions.
>
> * TREZOR performs extended validation of the inputs, when all of
> prev-txs are streamed into the device and validated. Your standard does
> not tackle this at all and I don't think it's worthy to make this
> standard unnecessarily complicated.

I'm aware of this approach but I don't think this makes sense long term.
We need a better way on the protocol level to validate inputs amounts (where segwit is a first step towards this).

IMO, not having a standard for hardware wallet interfaces/communication will long term result in reducing the end user experience.

I think we should collaborate together and work out a standard.

My goal is to add hardware wallet support in Bitcoin-Core where adding proprietary code (plugin-ish) is something we don't want to do for the sake of security and compatibility.

As said, the "BIP" is very draft and I'm happy to include the input streaming as part of it (or you could add it if you want because you have more experience with it).

</jonas>

_____
bitcoin-dev mailing list

bitcoin-dev@lists.linuxfoundation.org
https://lists.linuxfoundation.org/mailman/listinfo/bitcoin-dev

signature.asc
1K

Pavol Rusnak via bitcoin-dev <bitcoin-dev@lists.linuxfoundation.org> Tue, Aug 16, 2016 at 10:21 AM
Reply-To: Pavol Rusnak <stick@satoshilabs.com>, Bitcoin Protocol Discussion <bitcoin-dev@lists.linuxfoundation.org>
To: Jonas Schnelli <dev@jonasschnelli.ch>, Bitcoin Protocol Discussion <bitcoin-dev@lists.linuxfoundation.org>

On 16/08/16 17:13, Jonas Schnelli wrote:
> I'm aware of this approach but I don't think this makes sense long term.
> We need a better way on the protocol level to validate inputs amounts
> (where segwit is a first step towards this).

So you basically rephrased what I am saying but in another words.

> I think we should collaborate together and work out a standard.

I am for it. I am just saying we should create a standard for new forms
of transactions (Segwit and maybe Lightning), not the current "ugly" ones.
[Quoted text hidden]

_____
bitcoin-dev mailing list
bitcoin-dev@lists.linuxfoundation.org
https://lists.linuxfoundation.org/mailman/listinfo/bitcoin-dev

signature.asc
1K

Jochen Hoenicke via bitcoin-dev <bitcoin-dev@lists.linuxfoundation.org> Tue, Aug 16, 2016 at 12:48 PM
Reply-To: Jochen Hoenicke <hoenicke@gmail.com>, Bitcoin Protocol Discussion <bitcoin-dev@lists.linuxfoundation.org>
To: bitcoin-dev@lists.linuxfoundation.org

Hello Jonas,

thanks for your efforts of writing the draft for the standard.

First, this only describes detached signing. A wallet also needs to
connect with a hardware wallet at some time to learn the xpubs
controlled by the hardware. Do you plan to have this in a separate
standard or should this also be included here? Basically one needs one
operation: get xpub for an HD path.

From a first read over the specification I found the following points
missing, that a fully checking hardware wallet needs to know:

- the amount spent by each input (necessary for segwit).
- the full serialized input transactions (without witness informations)
to prove that the amount really matches (this is not necessary for segwit)
- the position of the change output and its HD Path (to verify that it
really is a change output).
- For multisig change addresses, there are more extensive checks
necessary: All inputs must be multisig addresses signed with public
keys derived from the same set of xpubs as the change address and use
the same "m of n" scheme. So for multisig inputs and multisig change
address the standard should allow to give the parent xpubs of the other
public keys and their derivation paths.

It is also a bit ambiguous what the "inputscript" is especially for p2sh transactions. Is this always the scriptPubKey of the transaction output that is spent by this input? For p2wsh nested in BIP16 p2sh transactions there are three scripts

witness: 0 <signature1> <1 <pubkey1> <pubkey2> 2 CHECKMULTISIG>
scriptSig: <0 <32-byte-hash>>
(0x220020{32-byte-hash})
scriptPubKey: HASH160 <20-byte-hash> EQUAL
(0xA914{20-byte-hash}87)
(quoted from BIP-141).

In principle one could put witness and scriptSig (with "OP_FALSE" in places of the signatures) in the raw transaction and make inputscript always the scriptPubKey of the corresponding output. Then one also doesn't need to distinguish between p2pkh or p2sh or p2wpkh or "p2wpkh nested in bip16 p2sh" transactions.

Regards,
Jochen

signature.asc
1K

Luke Dashjr via bitcoin-dev <bitcoin-dev@lists.linuxfoundation.org> Tue, Aug 16, 2016 at 2:22 PM
Reply-To: Luke Dashjr <luke@dashjr.org>, Bitcoin Protocol Discussion <bitcoin-dev@lists.linuxfoundation.org>
To: bitcoin-dev@lists.linuxfoundation.org, Jonas Schnelli <dev@jonasschnelli.ch>
Cc: Pavol Rusnak via bitcoin-dev <bitcoin-dev@lists.linuxfoundation.org>

On Tuesday, August 16, 2016 2:10:04 PM Jonas Schnelli via bitcoin-dev wrote:
> The BIP describes two approaches how to communicate (pipe and
> URI-scheme) with the signing-devices app, although, in my opinion, all
> major platform do support the URI approach (maybe we could drop the pipe
> approach then).

IMO it's kindof ugly to abuse URIs for communication. Stdio pipes are pretty universally supported, why not just use those?

On the other hand, no matter how the plugin is implemented, it's still a security risk, and requires installation (which the user might not have access for). It would be best if the hardware protocol were standardised, so the user doesn't need a plugin of *any* sort... I notice some hardware wallets have begun to implement (or reuse) Trezor's interface, so that would seem a good place to start?

Luke

Aiqin Li via bitcoin-dev <bitcoin-dev@lists.linuxfoundation.org> Tue, Aug 16, 2016 at 6:36 PM
Reply-To: Aiqin Li <liaiqin.lanzhou@gmail.com>, Bitcoin Protocol Discussion <bitcoin-dev@lists.linuxfoundation.org>
To: Jonas Schnelli <dev@jonasschnelli.ch>, Bitcoin Protocol Discussion <bitcoin-dev@lists.linuxfoundation.org>

Out of curiosity, what is the technical reason a normal ECC-enabled smart-card cannot be used for the hardware signing component of a wallet app? (Since if it can, its standardization must have been discussed.)

Debian wiki gives a list of such cards with related opensource software to access them.

Regards
[Quoted text hidden]

Peter Todd via bitcoin-dev <bitcoin-dev@lists.linuxfoundation.org> Tue, Aug 16, 2016 at 7:14 PM
Reply-To: Peter Todd <pete@petertodd.org>, Bitcoin Protocol Discussion <bitcoin-dev@lists.linuxfoundation.org>
To: Aiqin Li <liaiqin.lanzhou@gmail.com>, Bitcoin Protocol Discussion <bitcoin-dev@lists.linuxfoundation.org>

On Wed, Aug 17, 2016 at 09:36:02AM +1000, Aiqin Li via bitcoin-dev wrote:
> Out of curiosity, what is the technical reason a normal ECC-enabled
> smart-card cannot be used for the hardware signing component of a wallet
> app? (Since if it can, its standardization must have been discussed.)
>
> Debian wiki gives a list of such cards with related opensource software to
> access them.

I'm not aware of any ECC-enabled smart-cards that can sign the specific curve that Bitcoin uses, not to mention the fact that those smartcards generally only speak higher level protocols than raw signature generation, precluding the signing of bitcoin transactions.

The other serious problem - and this is a problem with smartcards in general anyway - is that without Bitcoin-specific logic you're just signing blindly; we recently saw the problems with that with the Bitfinex/BitGo hack. And even then, without a screen most of the hardware wallets in are still just signing blindly, with at best hard-to-use limits on maximum funds moved per-transaction. Also note how even hardware wallets with a screen, like Trezor, aren't yet able to authenticate who you are paying.

--
https://petertodd.org 'peter'[:-1]@petertodd.org

_____
bitcoin-dev mailing list
bitcoin-dev@lists.linuxfoundation.org
https://lists.linuxfoundation.org/mailman/listinfo/bitcoin-dev

---

signature.asc
1K

Thomas Daede via bitcoin-dev <bitcoin-dev@lists.linuxfoundation.org> Tue, Aug 16, 2016 at 7:03 PM
Reply-To: Thomas Daede <bztdlinux@gmail.com>, Bitcoin Protocol Discussion <bitcoin-dev@lists.linuxfoundation.org>
To: bitcoin-dev@lists.linuxfoundation.org

On 08/16/2016 12:22 PM, Luke Dashjr via bitcoin-dev wrote:
> It would be best if the hardware protocol were standardised, so the user
> doesn't need a plugin of *any* sort... I notice some hardware wallets have
> begun to implement (or reuse) Trezor's interface, so that would seem a good
> place to start?

I also agree with this - the user experience would be a lot better without the need to install custom adapter software, especially for the desktop case.

There could be two layers to the specification - the raw messages that need to be passed, and the transport mechanism to pass them (USB HID, QR code, audio...). For the most common case (USB), both layers could be

defined, and other transports could be added later. This split already exists in the draft specification, though it's not very clear (URIs include return URIs that don't make sense for a pipe, for example).

The existing URI scheme, while allowing disambiguate by manufacturer, provides no way to to enumerate available manufacturers or enabled wallets. This means that the "driver" would have to include a GUI to select this. Also, passing return URIs seems rather fragile - are there any other examples of protocols that use URIs for bidirectional IPC?

Thomas

[Quoted text hidden]

Thomas Kerin via bitcoin-dev <bitcoin-dev@lists.linuxfoundation.org> Tue, Aug 16, 2016 at 7:25 PM
Reply-To: Thomas Kerin <me@thomaskerin.io>, Bitcoin Protocol Discussion <bitcoin-dev@lists.linuxfoundation.org>
To: bitcoin-dev@lists.linuxfoundation.org

Hi all,

Thanks again Jonas for starting this!

I worked on a similar proposal a while back (never posted), approaching the same problem as if a merchant's website accepted xpubs/public keys, created multi-signature addresses, and wanted the user to easily sign offline instead of using some javascript code / using Core's debug console / coinb.in

Happily the procedure is largely the same, though I would echo Jochen's point that there needs to be a way to request an xpub/public key.

The redeemScript and witnessScript are also required fields for full validation & signing a transaction input if it's P2SH, or just the witnessScript if it's bare V0_P2WSH

Since the output amounts are required, so maybe instead provide serialized TxOut's? or Utxo's i.e: [txid, vout, amount, scriptPubKey].

The protocol ought to be as stateless as possible - it can't be assumed whether the redeemScript and other details will ever be saved on the device - so perhaps provide the redeemScript + witnessScript as the final fields on the Utxo structure above.

I do think it enables two important choices for bitcoin users:

* it might be preferable to provide your own xpub vs generating a brand new HD key to potentially lose.

* you could leverage the services provided by [random example] GreenAddress without necessarily having to rely on signing code provided by them, and so end up only having to trust only one ECDSA implementation when interacting with a wide number of services

All the best

Thomas

[Quoted text hidden]

> [Quoted text hidden]
>
> _____
> bitcoin-dev mailing list
> bitcoin-dev@lists.linuxfoundation.org
> https://lists.linuxfoundation.org/mailman/listinfo/bitcoin-dev

[Quoted text hidden]

> [Quoted text hidden]
> [Quoted text hidden]

_____
bitcoin-dev mailing list
bitcoin-dev@lists.linuxfoundation.org
https://lists.linuxfoundation.org/mailman/listinfo/bitcoin-dev

signature.asc
1K

Jonas Schnelli via bitcoin-dev <bitcoin-dev@lists.linuxfoundation.org> Wed, Aug 17, 2016 at 2:24 AM
Reply-To: Jonas Schnelli <dev@jonasschnelli.ch>, Bitcoin Protocol Discussion <bitcoin-dev@lists.linuxfoundation.org>
To: bitcoin-dev@lists.linuxfoundation.org

Hi all

Thanks for the response.

Jochen's points:
==========
Indeed. There are some missing points and I'd like to work this into the BIP. Thanks for bringing this up.

Along with a support for wallet-creation with a xpub from the signing device, we might also want to support loading multiple pubkeys into a keypool from the device (in case someone likes to use hardened derivation at all levels). I guess this would not be over-complex to achieve.

Luke's points:
==========

USB / Plugin/Driver problematic

I don't think it would be wise to set Trezors USB communication (hardware interface) as "the standard". A) A USB stack/interaction in wallets should be avoided IMO. B) This approach won't work for some platforms (like iOS) due to technical and legal restrictions.

In my opinion, each hardware wallet has to provide custom software in any case. We don't want to standardize how a hardware wallet has to do backups, recovers, firmware upgrade, etc. and if we agree on that, then hardware wallets must provide an application (mostly Chrome extensions today) to implement theses processes.

Also diversity at the hardware interface will reduce centralized risks for weak security/vulnerabilities.

The proposed URI scheme approach does not require any sorts of libraries/dependencies. USB HID can be a problem for cross platform desktop wallets as well as it won't work of one of the major mobile platform (iOS). USB HID interaction can be restricted or disabled in non superuser setups where I'm not aware of any restriction on URI-Scheme level.

URI scheme instead of stdio/pipe

The URI scheme is not ugly. Its a modern way – implemented in almost all platforms – how applications can interact with each other while not directly knowing each other. Registering a URI scheme like "bitcoin://" has some concrete advantages over just piping through stdio.

Also, the stdio/piping approach does not work for mobile platforms (where the URI scheme works).

The URI scheme does not require any sorts of wallet app level configuration (where the stdio/pipe approach would require to configure some details about the used hardware wallet).

Thomase D.'s points:
==========
Standardizing to many layers of the interaction stack (including the hardware interaction) will very likely result in vendors not sticking to the standard.

I agree, the URI scheme has some fragility, but at a level where we can handle it and with the advantage of abstracting the used brand/device for privacy and security reasons.

> The existing URI scheme, while allowing disambiguate by manufacturer, provides no way to to enumerate available manufacturers or enabled wallets.

Most operating systems allow to check if a certain URL-Scheme is supported (registered), this would allow at least to check for known major vendors (like trezor, etc.) which should solve most multi-hardware-wallet use-cases.

The URI return scheme does work fine and with the correct set timeouts it should result in a neat user experience.
It's the proposed way of application intercommunication in Apple iOS [1] and Google Android [2].

Conclusion:
==========
* Non of the points convinced me that there is a better alternative to the proposed URI scheme interaction (please tell me if I'm stubborn).
* Also, we should move the end users UX in the center of the problems-to-solve (and not overweight the ideal code-/API-/hardware-interaction-design while ignoring the end user experience).
* We should try to not over-standardize the interaction with the device itself to allow flexibility on the hardware wallet vendor side.

[1] https://developer.apple.com/library/ios/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/Inter-AppCommunication/Inter-AppCommunication.html
[2] https://developer.android.com/training/basics/intents/sending.html

</jonas>

signature.asc
1K

Nicolas Bacca via bitcoin-dev <bitcoin-dev@lists.linuxfoundation.org> Wed, Aug 17, 2016 at 2:27 AM
Reply-To: Nicolas Bacca <nicolas@ledger.fr>, Bitcoin Protocol Discussion <bitcoin-dev@lists.linuxfoundation.org>
To: Bitcoin Protocol Discussion <bitcoin-dev@lists.linuxfoundation.org>

On Wed, Aug 17, 2016 at 2:14 AM, Peter Todd via bitcoin-dev <bitcoin-dev@lists.linuxfoundation.org> wrote:

> I'm not aware of any ECC-enabled smart-cards that can sign the specific curve
> that Bitcoin uses, not to mention the fact that those smartcards generally only
> speak higher level protocols than raw signature generation, precluding the
> signing of bitcoin transactions.

any Java Card supporting ECC can sign on user supplied Weierstrass curve parameters - you can find a good shopping list at http://www.fi.muni.cz/~xsvenda/jcsupport.html (look for ALG_ECDSA_SHA256 on javacard.crypto.signature). The NXP JCOP platform (found in Yubico Neo) is a popular choice, and then you can add your own custom logic for validation.

--
Nicolas Bacca | CTO, Ledger

Nicolas Bacca via bitcoin-dev <bitcoin-dev@lists.linuxfoundation.org> Wed, Aug 17, 2016 at 2:40 AM
Reply-To: Nicolas Bacca <nicolas@ledger.fr>, Bitcoin Protocol Discussion <bitcoin-dev@lists.linuxfoundation.org>
To: Bitcoin Protocol Discussion <bitcoin-dev@lists.linuxfoundation.org>

On Wed, Aug 17, 2016 at 9:24 AM, Jonas Schnelli via bitcoin-dev <bitcoin-dev@lists.linuxfoundation.org> wrote:

> Conclusion:
> ==========
> * Non of the points convinced me that there is a better alternative to
> the proposed URI scheme interaction (please tell me if I'm stubborn).

I'd also agree with this - and it's convenient to test against simulators / mocks.

--
Nicolas Bacca | CTO, Ledger

Dana L. Coe via bitcoin-dev <bitcoin-dev@lists.linuxfoundation.org> Wed, Aug 17, 2016 at 5:13 AM
Reply-To: "Dana L. Coe" <dana.coe@bitlox.com>, Bitcoin Protocol Discussion <bitcoin-dev@lists.linuxfoundation.org>
To: Jonas Schnelli <dev@jonasschnelli.ch>, Bitcoin Protocol Discussion <bitcoin-dev@lists.linuxfoundation.org>

> On Aug 17, 2016, at 15:24, Jonas Schnelli via bitcoin-dev <bitcoin-dev@lists.linuxfoundation.org> wrote:
>
> URI scheme instead of stdio/pipe
> ---
> The URI scheme is not ugly. Its a modern way – implemented in almost all
> platforms – how applications can interact with each other while not
> directly knowing each other. Registering a URI scheme like "bitcoin://"
> has some concrete advantages over just piping through stdio.
>

> Also, the stdio/piping approach does not work for mobile platforms
> (where the URI scheme works).
>
> The URI scheme does not require any sorts of wallet app level
> configuration (where the stdio/pipe approach would require to configure
> some details about the used hardware wallet).

Hi everybody, just thought I'd throw my opinion in here.

The URI scheme is a nice idea, but this ignores the fact that hardware wallet vendors do most of the work on talking between the computer/mobile and the wallet on a lower level of communication. In the case of BitLox, the base protocol is Google's ProtoBuf. The commands and transaction data is in a "schema" which is then encoded in different methods accessible via ProtoBuf (depending on the data being sent). The advantages of this protocol is that it can be implemented on a wide variety of platforms. (but that's a whole 'nother discussion)

The URI would be handled waaaaay up in the specific application (such as the mytrezor wallet software or the various standalone wallets) - nowhere near the actual hardware communications layer.

Best regards,
Dana
BitLox

signature.asc
1K

Jonas Schnelli via bitcoin-dev <bitcoin-dev@lists.linuxfoundation.org> Wed, Aug 17, 2016 at 6:34 AM
Reply-To: Jonas Schnelli <dev@jonasschnelli.ch>, Bitcoin Protocol Discussion <bitcoin-dev@lists.linuxfoundation.org>
To: "Dana L. Coe" <dana.coe@bitlox.com>, Bitcoin Protocol Discussion <bitcoin-dev@lists.linuxfoundation.org>

Hi Dana

>> The URI scheme does not require any sorts of wallet app level
>> configuration (where the stdio/pipe approach would require to configure
>> some details about the used hardware wallet).
>
> Hi everybody, just thought I'd throw my opinion in here.
>
> The URI scheme is a nice idea, but this ignores the fact that hardware wallet vendors do most of the work on talking between the computer/mobile and the wallet on a lower level of communication. In the case of BitLox, the base protocol is Google's ProtoBuf. The commands and transaction data is in a "schema" which is then encoded in different methods accessible via ProtoBuf (depending on the data being sent). The advantages of this protocol is that it can be implemented on a wide variety of platforms. (but that's a whole 'nother discussion)
>
> The URI would be handled waaaaay up in the specific application (such as the mytrezor wallet software or the various standalone wallets) - nowhere near the actual hardware communications layer.

This is maybe a question of the scope.
The BIP I'm proposing would make a clear interface cut between wallet-with-unsigned-transaction and a signing-device (and maybe between wallet-requires-pubkey, signing-device generate some pubkeys [or non-hardened xpub]).

The detached-signing proposal does not duplicate work. It just moves the current plugin design into a separate application. Plugins in security and privacy critical wallet software is something that should probably be avoided.

It's intentional at a high level to allow maximum flexibility at the hardware interaction layer.

Your protobuf example is a good use-case. You could implement your custom processes behind the URI scheme (which is probably way more efficient then writing a couple of wallet plugins where you – at the end – mostly don't control the deployment and the source-code).

Defining a standard on the hardware interaction layer is possible, but a fairly different approach.

</jonas>

signature.asc
1K

Marek Palatinus via bitcoin-dev <bitcoin-dev@lists.linuxfoundation.org> Wed, Aug 17, 2016 at 12:06 PM
Reply-To: Marek Palatinus <marek@palatinus.cz>, Bitcoin Protocol Discussion <bitcoin-dev@lists.linuxfoundation.org>
To: Jonas Schnelli <dev@jonasschnelli.ch>, Bitcoin Protocol Discussion <bitcoin-dev@lists.linuxfoundation.org>

Hi,

I fundamentally disagree with the concept of driving signing workflow by the wallet software. Wallet software does not know in advance all data necessary for the signer to do the job. As Jochen mentioned above, Segwit vs Non-segwit use cases are a good example, but there may be many.

Currently the TREZOR protocol works like device is a server and wallet is a client calling methods on it. It's like: "Sign this for me, please", "Ok, give me this information", "Here it is", "Now I need this another piece".... "There is the signature". Wallet does not know in advance what will go next, and it is for sake of simplicity. I'm quite happy with the protocol so far.

Considering the difference in between current hardware, I really don't think it is possible to find any minimal URI-based API good enough for communicating with all vendors. What I see more likely is some 3rd party libraries (JS, C++, Python, ...) defining high-level API and implementing hardware-specific protocols and transports as plugins. That way vendors are not limited by strict standard and application developers and services can integrate wide range of hardware wallets easily. However, this can be done already and we do not need any standardization process (yet).

slush

[Quoted text hidden]
> [Quoted text hidden]
> [Quoted text hidden]

Bryan Bishop <kanzure@gmail.com> Wed, Aug 17, 2016 at 1:36 PM
To: Peter Todd <pete@petertodd.org>, Bitcoin Protocol Discussion <bitcoin-dev@lists.linuxfoundation.org>, Bryan Bishop <kanzure@gmail.com>

On Tue, Aug 16, 2016 at 7:14 PM, Peter Todd via bitcoin-dev <bitcoin-dev@lists.linuxfoundation.org> wrote:

> The other serious problem - and this is a problem with smartcards in general anyway - is that without Bitcoin-specific logic you're just signing blindly; we recently saw the problems with that with the Bitfinex/BitGo hack. And even then, without a screen most of the hardware wallets in are still just signing blindly, with at best hard-to-use limits on maximum funds moved per-transaction. Also note how even hardware wallets with a screen, like Trezor, aren't yet able to authenticate who you are paying.

"Welcome to my threat model."

In multisig scenarios, there must be a different "trust root" for each key. For example, storing two private keys next to each other on the same web server is broken because if one key is compromised it is infinitely trivial to compromise the second key. Using multiple web servers is also broken if the two servers are controlled by the same AWS keys or same "help me get my servers back" support email request to whatever single sign-on service is used. In some cases, it can be better to write software such that transaction data is served at a particular location, and another security-critical step is responsible for downloading that data from the first machine, rather than the first computer directly pushing (with authentication credentials in place for the attacker to compromise) the data to the second computer.

I recommend using hardware security modules (HSMs). It's important to have a public, reviewed bitcoin standard for hardware wallets, especially HSMs. I expect this is something that the entire industry has a tremendous interest in following and contributing to, which could even lead to additional resources contributed (or at the very least, more detailed requirements) towards libconsensus work.

Instead of signing any bitcoin transaction that the hardware wallet is given, the hardware should be responsible for running bitcoin validation rules and business logic, which I recommend for everyone, not only businesses. Without running business logic and bitcoin validation rules, the actual bitcoin history on the blockchain could be a very different reality from what the hardware thinks is happening. Using a different out-of-band communication channel, the hardware could query for information from another database in another trust root, which would be useful for business logic to validate against.

As for a screen, I consider that somewhat limited because you only get text output (and I don't know if I can reasonably suggest QR codes here). With a screen, you are limited to text output, which can compromise privacy of the device's operations and info about the wallet owner. An alternative would be to have a dedicated port that is responsibly only for sending out data encrypted to the key of the wallet owner, to report information such as whatever the hardware's transaction planner has decided, or to report about the state of the device, state of the bitcoin validation rules, or any accounting details, etc. Additionally, even a signed transaction should be encrypted to the key of the device owner because a signed transaction can be harmless as long as the owner still has the ability to control whether the signed transaction is broadcasted to the network. It's "separation of concerns" for transaction signing and decrypting a signed transaction should be unrelated and uncoupled.

Also I am eager to see what the community proposes regarding signed and authenticated payment requests.

((insert here general promotional statement regarding the value of reusable checklists used during every signing ritual ceremony))

- Bryan
http://heybryan.org/
1 512 203 0507

Jonas Schnelli via bitcoin-dev <bitcoin-dev@lists.linuxfoundation.org> Thu, Aug 18, 2016 at 1:54 AM
Reply-To: Jonas Schnelli <dev@jonasschnelli.ch>, Bitcoin Protocol Discussion <bitcoin-dev@lists.linuxfoundation.org>
To: Marek Palatinus <marek@palatinus.cz>, Bitcoin Protocol Discussion <bitcoin-dev@lists.linuxfoundation.org>

Hi

> I fundamentally disagree with the concept of driving signing workflow by
> the wallet software. Wallet software does not know in advance all data
> necessary for the signer to do the job. As Jochen mentioned above,
> Segwit vs Non-segwit use cases are a good example, but there may be many.

I think this is easily solvable. The required data to verify and sign a
(standard) bitcoin transaction (including P2WSH multi-sig) is manageable.

IMO what a signing devices requires in order to sign a (standard)

transaction:
-> serialized tx
-> serialized tx of the inputs
-> scriptPubKey of the inputs
-> inputs redeem-Scripts
-> input amounts
-> position of the change output any maybe its keypath
-> cosigners pubkeys for inputs and changeaddress

This seems to be manageable for a 1 round communication?
Or do I miss something?

> Currently the TREZOR protocol works like device is a server and wallet
> is a client calling methods on it. It's like: "Sign this for me,
> please", "Ok, give me this information", "Here it is", "Now I need this
> another piece".... "There is the signature". Wallet does not know in
> advance what will go next, and it is for sake of simplicity. I'm quite
> happy with the protocol so far.

I think multiple rounds would still be possible with a clever design.
Although I could imaging that >95% of the users transaction would
require only a single "shot".

Whats the benefits of the multiple rounds communication? Would a single
round result in to many data transported?

Passing a 300kb chunk (assuming a large transaction) over a URI scheme
requires a couple of milliseconds on standard Smartphones or PCs.

> Considering the difference in between current hardware, I really don't
> think it is possible to find any minimal URI-based API good enough for
> communicating with all vendors. What I see more likely is some 3rd party
> libraries (JS, C++, Python, ...) defining high-level API and
> implementing hardware-specific protocols and transports as plugins. That
> way vendors are not limited by strict standard and application
> developers and services can integrate wide range of hardware wallets
> easily. However, this can be done already and we do not need any
> standardization process (yet).

The URI-based API allows transmitting data of multiple megabytes while
there is no need for...
* dependencies of any form (library, etc.)
* library support for a particular language
* platform that supports the dependencies of the library (like USBHID,
not supported by iOS)

Can you elaborate what benefits you would get from the library approach
and how the library API would be different form the proposed URI-scheme?

How would the library approach work on mobile platforms? Would USB be
the only supported hardware communication layer?

Thanks
--
</jonas>

_____
bitcoin-dev mailing list
bitcoin-dev@lists.linuxfoundation.org
https://lists.linuxfoundation.org/mailman/listinfo/bitcoin-dev

signature.asc
1K

Marek Palatinus via bitcoin-dev <bitcoin-dev@lists.linuxfoundation.org> Thu, Aug 18, 2016 at 4:15 AM
Reply-To: Marek Palatinus <marek@palatinus.cz>, Bitcoin Protocol Discussion <bitcoin-dev@lists.linuxfoundation.org>
To: Jonas Schnelli <dev@jonasschnelli.ch>
Cc: Bitcoin Protocol Discussion <bitcoin-dev@lists.linuxfoundation.org>

> Can you elaborate what benefits you would get from the library approach
> and how the library API would be different form the proposed URI-scheme?

The main benefit is that you don't need "standard" to solve problem, but use natural tools in given environment and programming stack. Build a "standard" on top of URI protocol is a huge limitation, which does not give any advantage.

We already see issues with dead simple "bitcoin uri" standard, it barely works in most of bitcoin apps. Think of vague definitions of parameters or ability to send payment requests over it. HW API would be complicated by an order of magnitude and I have serious concerns that it will be helpful for anything. So why complicate things.

> How would the library approach work on mobile platforms? Would USB be
> the only supported hardware communication layer?

Interprocess communication/libraries/dependencies on Android are not bound to specific transport anyhow. Such library could be used by any android app, and the library would implement proper transports for various supported vendors. USB for Trezor, NFC for something different etc. If the point is "make life of app developers easier", let's do this and do not define artifical "standards".

slush

[Quoted text hidden]

_____
bitcoin-dev mailing list
bitcoin-dev@lists.linuxfoundation.org
https://lists.linuxfoundation.org/mailman/listinfo/bitcoin-dev

---

Jonas Schnelli via bitcoin-dev <bitcoin-dev@lists.linuxfoundation.org> Thu, Aug 18, 2016 at 4:35 AM
Reply-To: Jonas Schnelli <dev@jonasschnelli.ch>, Bitcoin Protocol Discussion <bitcoin-dev@lists.linuxfoundation.org>
To: Marek Palatinus <marek@palatinus.cz>
Cc: Bitcoin Protocol Discussion <bitcoin-dev@lists.linuxfoundation.org>

Hi

> The main benefit is that you don't need "standard" to solve problem, but
> use natural tools in given environment and programming stack. Build a
> "standard" on top of URI protocol is a huge limitation, which does not
> give any advantage.

Standards can help an ecosystem to grow, can help to sustain a good user experience.

The hardware wallet vendors have used "natural tools" and look where we are. We have *native* plugins in Electrum, Copay, etc. for different hardware wallets. Mostly the plugins are in the code base of the wallet, which makes it – in theory – impossible to change from the perspective of the hardware wallet vendor (there is no control of the deployment if there are bugs in the plugins code).
The plugins functions overlap significant.

I think this is a bad design for security critical applications.

What I want as hardware wallet user:
* I'd like to have a trusted application (layer) where I'm sure I'm

using software provided through my hardware wallet vendor.

What I want as hardware wallet vendor:
* I'd like to be able to provide and update a software layer (app) to my customer with the ability to provide code signatures and security updates anytime. I do want to control the user experience.

> We already see issues with dead simple "bitcoin uri" standard, it barely
> works in most of bitcoin apps. Think of vague definitions of parameters
> or ability to send payment requests over it. HW API would be complicated
> by an order of magnitude and I have serious concerns that it will be
> helpful for anything. So why complicate things.

As far as I know most bitcoin wallets do support the bitcoin:// URI scheme quite well.
I agree that BIP70 is a mess (including the bitcoin:// additions).

The proposed URI scheme would be completely different. The only similarity is using the URI scheme as transport layer (which is the proposed long term inter-app communication layer by Apple and Google).

>> How would the library approach work on mobile platforms? Would USB be
> the only supported hardware communication layer?
>
> Interprocess communication/libraries/dependencies on Android are not
> bound to specific transport anyhow. Such library could be used by any
> android app, and the library would implement proper transports for
> various supported vendors. USB for Trezor, NFC for something different
> etc. If the point is "make life of app developers easier", let's do this
> and do not define artifical "standards".

So you propose having one library that would support multiple vendors? What if new vendors add a new transport layer (lets assume NFC or Bluetooth), wouldn't that result in every possible consumer of that library (all wallets) need to update before the new vendors transport layer could be used, resulting in a huge deployment process probably require many month until it can be used?

What if there is a critical security issue in the library? How would the deployment plan looks like?

I really think we should remove the "hardware communication layer" from wallets and move it towards the hardware vendor app.

What about iOS? Should we just leave that platform unsupported with hardware wallets?

</jonas>

_____
bitcoin-dev mailing list
bitcoin-dev@lists.linuxfoundation.org
https://lists.linuxfoundation.org/mailman/listinfo/bitcoin-dev

signature.asc
1K

Marek Palatinus via bitcoin-dev <bitcoin-dev@lists.linuxfoundation.org> Thu, Aug 18, 2016 at 4:43 AM
Reply-To: Marek Palatinus <marek@palatinus.cz>, Bitcoin Protocol Discussion <bitcoin-dev@lists.linuxfoundation.org>
To: Jonas Schnelli <dev@jonasschnelli.ch>

Cc: Bitcoin Protocol Discussion <bitcoin-dev@lists.linuxfoundation.org>

> On Thu, Aug 18, 2016 at 11:35 AM, Jonas Schnelli <dev@jonasschnelli.ch> wrote:
> > I agree that BIP70 is a mess (including the bitcoin:// additions). The proposed URI scheme would be completely different.

This reminds me https://xkcd.com/927/

I have some experience with hardware wallet development and its integration and I know it's a mess. But it is too early to define such rigid standards yet. Also, TREZOR concept (device as a server and the primary source of workflow management) goes directly against your proposal of wallet software as an workflow manager. So it is clear NACK for me.

slush

_____
bitcoin-dev mailing list
bitcoin-dev@lists.linuxfoundation.org
https://lists.linuxfoundation.org/mailman/listinfo/bitcoin-dev

---

Jonas Schnelli via bitcoin-dev <bitcoin-dev@lists.linuxfoundation.org> Thu, Aug 18, 2016 at 4:49 AM
Reply-To: Jonas Schnelli <dev@jonasschnelli.ch>, Bitcoin Protocol Discussion <bitcoin-dev@lists.linuxfoundation.org>
To: Marek Palatinus <marek@palatinus.cz>
Cc: Bitcoin Protocol Discussion <bitcoin-dev@lists.linuxfoundation.org>

Hi

> I have some experience with hardware wallet development and its
> integration and I know it's a mess. But it is too early to define such
> rigid standards yet. Also, TREZOR concept (device as a server and the
> primary source of workflow management) goes directly against your
> proposal of wallet software as an workflow manager. So it is clear NACK
> for me.

The current question – as already mentioned – is we ACK to work together on a signing protocol or if we NACK this before we even have started.

I'm not saying that the draft proposal I made is the way to go, I'm happy to NACK it myself in favor of a better proposal.

I strongly recommend to work together on a standard that will have one central winner: the end user.

</jonas>

_____
bitcoin-dev mailing list
bitcoin-dev@lists.linuxfoundation.org
https://lists.linuxfoundation.org/mailman/listinfo/bitcoin-dev

---

signature.asc
1K

Nicolas Bacca via bitcoin-dev <bitcoin-dev@lists.linuxfoundation.org> Thu, Aug 18, 2016 at 5:23 AM
Reply-To: Nicolas Bacca <nicolas@ledger.fr>, Bitcoin Protocol Discussion <bitcoin-dev@lists.linuxfoundation.org>
To: Bitcoin Protocol Discussion <bitcoin-dev@lists.linuxfoundation.org>

> On Thu, Aug 18, 2016 at 11:49 AM, Jonas Schnelli via bitcoin-dev <bitcoin-dev@lists.linuxfoundation.org> wrote:

> Hi
>
> > I have some experience with hardware wallet development and its
> > integration and I know it's a mess. But it is too early to define such
> > rigid standards yet. Also, TREZOR concept (device as a server and the
> > primary source of workflow management) goes directly against your
> > proposal of wallet software as an workflow manager. So it is clear NACK
> > for me.
>
> The current question – as already mentioned – is we ACK to work together
> on a signing protocol or if we NACK this before we even have started.

ACK for Ledger. What's necessary to sign a transaction is well known, I don't see how driving any hardware wallet from the wallet itself or from a third party daemon implementing that URL scheme would make any difference, other than providing better devices interoperability, as well as easier maintenance and update paths for the wallets.

[Quoted text hidden]

Moral Agent via bitcoin-dev <bitcoin-dev@lists.linuxfoundation.org> Mon, Aug 22, 2016 at 11:50 AM
Reply-To: Moral Agent <ethan.scruples@gmail.com>, Bitcoin Protocol Discussion <bitcoin-dev@lists.linuxfoundation.org>
To: Jonas Schnelli <dev@jonasschnelli.ch>, Bitcoin Protocol Discussion <bitcoin-dev@lists.linuxfoundation.org>

It would be nice if the detached signer and the normal wallet could both verify the correctness of generated addresses before you cause coins to be sent there.

e.g. the hardware wallet could give its master public key to Bitcoin Core and you can thereafter generate your receiving addresses on Core, with the option to have the HW wallet validate them.

One of my biggest fears about using any wallet is the "whoops, cosmic ray flipped a bit while producing receiving address; SFYL!" possibility. For high value cold storage, I always generate my addresses on two independent machines using two different pieces of software. Am I nuts for doing that?

With the above scheme, you are pretty well protected from losing money if your HW wallet is defective. You could still lose it if the HW wallet was evil of course, but that strikes me as much more likely to be discovered quickly.

Thomas Kerin via bitcoin-dev <bitcoin-dev@lists.linuxfoundation.org> Wed, Aug 24, 2016 at 5:31 AM
Reply-To: Thomas Kerin <me@thomaskerin.io>, Bitcoin Protocol Discussion <bitcoin-dev@lists.linuxfoundation.org>
To: bitcoin-dev@lists.linuxfoundation.org

I want to pitch a use-case that might have been ignored in this discussion:

I don't think this protocol is only useful for hardware wallets. Technically any website that wants to request public keys/signatures and offload the responsibility for managing keys and signing to the user would also find this valuable.

I hope we can move forward with a protocol that suits both the hardware people, and the people who find signing transactions in browsers unsettling.

Maybe we the focus should move away from only servicing hardware, and asking if the motivation is better captured by "allow users pick their own ECDSA implementation, hardware or software", then working out what we need to get us there.

[Quoted text hidden]

[Quoted text hidden]
[Quoted text hidden]

signature.asc
1K

Corey Haddad via bitcoin-dev <bitcoin-dev@lists.linuxfoundation.org> Sun, Aug 28, 2016 at 6:14 PM
Reply-To: Corey Haddad <corey3@gmail.com>, Bitcoin Protocol Discussion <bitcoin-dev@lists.linuxfoundation.org>
To: Moral Agent <ethan.scruples@gmail.com>, Bitcoin Protocol Discussion <bitcoin-dev@lists.linuxfoundation.org>

One of my biggest fears about using any wallet is the "whoops, cosmic ray flipped a bit while producing receiving address; SFYL!" possibility. For high value cold storage, I always generate my addresses on two independent machines using two different pieces of software. Am I nuts for doing that?

A randomly flipped bit would be extremely unlikely to yield a valid address, however, I still think it you are wise to use independent routes to confirm that your addresses match the keys. I do the same when I generating my cold storage key pairs. I think malicious address substitution is an under appreciated attack vector.

Regarding this thread in general, would it make sense for this proposal to include standards for multi-sig wallet interoperability? A whole spectrum of attacks would be made less likely - and easy for typical users to guard against - by using wallets on separate devices AND where the wallet software was written and provided by different parties.

[Quoted text hidden]
[Quoted text hidden]
[Quoted text hidden]