A First Look at the Usability of Bitcoin Key Management

Full Version

Shayan Eskandari*, David Barrera†, Elizabeth Stobert‡, and Jeremy Clark*

*Concordia University, †ETH Zürich, ‡Carleton University

Abstract—Bitcoin users are directly or indirectly forced to deal with public key cryptography, which has a number of security and usability challenges that differ from the password-based authentication underlying most online banking services. Users must ensure that keys are simultaneously accessible, resistant to digital theft and resilient to loss. In this paper, we contribute an evaluation framework for comparing Bitcoin key management approaches, and conduct a broad usability evaluation of six representative Bitcoin clients. We find that Bitcoin shares many of the fundamental challenges of key management known from other domains, but that Bitcoin may present a unique opportunity to rethink key management for end users.

I. INTRODUCTION

In all of the excitement surrounding Bitcoin [21], it is easy to forget that the decentralized currency assumes a solution to the longstanding problem of usable public key cryptography for user authentication. Studies of the usability of key management [14]–[16], [27] have shown that there are numerous usability issues that prevent public key cryptography from being effectively leveraged by end users. Managing, controlling, and using cryptographic keys are complex tasks, and no clear solution has been proposed.

Despite the known complexity in creating and managing cryptographic keys, the Bitcoin network and software clients use such keys extensively for many operations. For example, digital signatures, which require the Bitcoin software to read private keys into memory, are used to assert ownership over a specific set of Bitcoins. Thus, managing the same coins on multiple devices (*e.g.,* a desktop and a phone) requires the corresponding private keys to be copied to and made accessible on these devices.

The consequences of losing exclusive control over an account containing monetary value connects the threat of losing a Bitcoin private key to that of losing an online banking password. However, consumers in many countries are legally protected from any liability of banking credential loss. Furthermore, most bank transactions are traceable and reversible, making it difficult to extract value from stolen banking credentials

(most techniques involve a mule [13]). Bitcoin transactions are also traceable, however they are not reversible. Stolen Bitcoins can thus not be centrally or automatically recovered. Bitcoin users typically have no legal protection against loss or theft, and while stolen Bitcoins could be traced as they change ownership,[1] several mechanisms exist for laundering Bitcoins and similar digital currencies [8], [19].

In an effort to address some of the complexities of key management, developers of Bitcoin software have created a variety of innovative technologies ranging from password-derived keys to air-gapped computers to physical printouts of private keys in the form of 2D barcodes. However, since none of these proposals have been evaluated in the Bitcoin context, it remains unclear which techniques have usability advantages.

For Bitcoin to flourish, adoption must expand beyond developers and tech-savvy enthusiasts to novice users. Expansion solidifies the need for a usable, comprehensible approach to Bitcoin. If users cannot safely manage Bitcoin keys, it may result in the users' loss of funds and/or a poor reputation for Bitcoin, both of which could dissuade further user adoption.

In this paper, we aim to investigate the usability challenges surrounding key management in Bitcoin. To do this, we survey and categorize the most prominent Bitcoin key management proposals. Next we conduct an expert usability inspection technique known as a cognitive walkthrough [30] on popular examples of each proposal. Our goal is to identify overarching usability issues as well as advantages of specific proposals, allowing us to propose design recommendations for future Bitcoin clients.

Specifically, the contributions of the paper are as follows:

- We perform a broad survey of six Bitcoin key management techniques which cover the vast majority of deployed Bitcoin software.
- Using the results from our survey, we propose an evaluation and comparison framework for Bitcoin key management techniques. The framework is based on 10 security, usability and deployability criteria, and enables direct comparison of current and future key management proposals. Using our framework we find that certain properties, such as trust in a central party enable additional beneficial properties. We also find that the disadvantages of certain properties, such as malware protection, outweigh the relative benefits.

---

[1]Public keys associated with specific Bitcoins are publicly available in the Bitcoin blockchain, but the identities of users who control those keys are not.

- We perform a cognitive walkthrough of six distinct Bitcoin clients and tools to identify usability issues while performing basic Bitcoin tasks (*e.g.,* viewing account balance, sending funds, *etc.*). We find that the metaphors and abstractions used in the surveyed clients are subject to misinterpretations, and that the clients do not do enough to support their users.

II. BACKGROUND

A. Bitcoin

Bitcoin is a cryptographic currency deployed in 2009 [21] which has reached a level of adoption unrealized by decades of previously proposed digital currencies (from 1982 [11] onward). Unlike many previous proposals, Bitcoin does not distribute digital monetary units to users. Instead, a public ledger maintains a list of every transaction[2] made by all Bitcoin users since the creation of the currency. A *transaction* in its simplest form describes the movement of some balance of the Bitcoin currency (XBT or BTC) from one or more accounts (called input addresses) into one or more accounts (called output addresses). Bitcoin addresses are indexed by the fingerprint of a public key from a digital signature scheme.[3] They are not centrally allocated or registered in any way— the addresses become active when the first transaction moving money into them is added to the ledger.

In Bitcoin, every transaction must be digitally signed using the private signing key associated with each input address in the transaction. In order to spend Bitcoin, users require access to the signing key of the account holding their Bitcoin. Thus users do not maintain any kind of units of currency; they maintain a set of keys that provide them signing authority over certain accounts recorded in the ledger.

The ledger (known as the *blockchain*) is maintained and updated by a decentralized network using a novel method to reach consensus that involves incentivizing nodes in the network with the ability to generate (known as mining) new Bitcoin and collect transaction fees. The details of the Bitcoin consensus model are not relevant to this paper, but we note that clients in the network participate in the consensus model by downloading and cryptographically verifying the integrity of the blockchain. As of writing, the Bitcoin blockchain is roughly 25 GB in size.[4]

One subtlety of Bitcoin's transaction architecture is that in order to spend Bitcoins, the entire value of unspent outputs (*i.e.,* from previous transactions) must be spent. To accommodate this, Bitcoin clients automatically spend the full amount of unspent outputs and create multiple components in the transaction: one component will send part of the unspent coins to the intended recipient, and the other component will send the remaining inputs back to the sender as *change*. It is technically possible (and some clients behave this way) to send change back to the sending address. However, to enhance anonymity, the reference client generates fresh addresses (and corresponding private keys) to receive the remaining transaction amount.

As more transactions are made, Bitcoin clients must keep track of multiple private keys for use in future transactions. Many clients prominently display a Bitcoin balance on the main screen, which represents the sum of all unspent outputs for which private keys are available.

B. Usability of Key Management

Passwords remain the most common form of user authentication [18]. Private key-based authentication is rarely used by non-experts, and is typically never used as the default configuration in applications which support this authentication method. Transport Layer Security (TLS) client-side certificates have failed to reached wide-spread deployment. Secure shell (SSH) uses passwords by default, and allows certificates.

Password managers, when configured to generate or store system-chosen random passwords, share at least one property of cryptographic keys: such passwords become something you *have* instead of *know*. However, if access to such a password is lost, online services generally offer account recovery mechanisms (*e.g.,* based on email). No such recovery mechanism exists for self-managed cryptographic keys.

The use of public key systems by non-experts that is closest to Bitcoin is arguably encrypted/authenticated email, in particular Pretty Good Privacy (PGP) and its open-source alternatives (*i.e.,* GPG and OpenPGP). Beginning with *Why Johnny Can't Encrypt* [31], the usability of public key technology has been well-studied from a usability perspective [14]–[16], [27]. The findings of this literature are diverse but relevant observations include the following: (1) the metaphor and terminology behind public and private keys is confusing; (2) it is difficult to correctly obtain other users' public keys; (3) key migration between devices is difficult. This literature tends to focus primarily on encryption and not signatures, but we find some overlap to the work presented here.

III. BITCOIN KEY MANAGEMENT APPROACHES

Before turning to a detailed usability evaluation, we evaluate from a systems perspective each category of tool for managing Bitcoin private keys. We highlight security and deployability issues, and note relevant details of the Bitcoin protocol that create complexities and potential discrepancies with users' mental models.

A. Keys in Local Storage

One way in which Bitcoin software manages several private keys is by storing these keys on the device's local storage, typically in a file or database in a pre-configured file system path. When a new transaction is created, the Bitcoin client can read the keys and immediately (possibly without any further user input) broadcast the transaction over the network. The reference Bitcoin client (Bitcoin Core), as well as certain mobile wallets (*e.g.,* Android Bitcoin Wallet) use this approach, storing private keys in a file (referred to as a *wallet*) inside the user's home or application directory.

---

[2]Technically, a transaction specifies a short script that encodes how the balance can be claimed as the input to some future transaction.

[3]Elliptic Curve Digital Signature Algorithm (ECDSA) [29].

[4]Due to the large size of the blockchain, full download is infeasible for thin clients running on mobile devices, as well as some desktop clients. These clients connect to a semi-trusted node and only request transactions relevant to keys in their wallet. This technique, known as Simplified Payment Verification (SPV), eliminates the need to download and verify the entire blockchain but, when implemented incorrectly, can create privacy risks [17].

Storing keys in a locally accessible file has several advantages. First, there is no additional cognitive load on users, since only the software must access the file. Second, a practically unlimited number of keys can be stored on disk due to the small size of keys. Third, the Bitcoin software can automatically generate keys and create transactions without additional input or actions from the user.

Storing keys locally also creates several threats, which the user must consider. For example, the file storing private keys can be read by any application with access to the user's application folder. Malware authors may be particularly interested in exploiting this key management approach, since access to the local file results in the adversary gaining immediate access to the victim's funds. One of the first examples of private key-stealing malware was discovered by Symantec in 2011 [28], with many other similar malware examples following suit.

Users must be cautious to not inadvertently share their Bitcoin application folder (*e.g.,* through peer-to-peer file sharing networks, off-site backups or on a shared network drive). Physical theft, especially in the case of portable computers or smartphones must also be considered. Similar to the storage of other sensitive files, threats to digital preservation [2] should be taken into account. Examples include general equipment failure due to natural disasters and electrical failures; acts of war; mistaken erasure (*e.g.,* formatting the wrong drive or deleting the wrong folder); bit rot (*i.e.,* undetected storage failure); and possibly others. If storing private keys for a long period of time (*e.g.,* a trust fund or long-term savings), users must also preserve a specification of the file format to ensure the keys can continue to be read.

The reference Bitcoin client pre-generates keys in a batch of 100 (these keys are known as the keypool). When a transaction is made, the next available key is selected from the keypool for receiving change. The keypool is then periodically refilled with a new batch of keys as necessary. This *key churn* requires users to periodically create new backups of their key storage file to ensure that new keypool keys are stored.

B. Password-protected (Encrypted) Wallets

Certain Bitcoin clients allow a locally stored wallet file to be encrypted with a key derived from a user-chosen password or passphrase. Password-protected wallets appear to address only *physical* theft of the underlying storage device, requiring brute-force of the password if the file containing private keys is stolen. Password protection seems less useful in the case of *digital* theft; if malware can be installed on to the device storing the wallet, it is reasonable to assume a keystroke-logging module would be present, limiting or nullifying the benefits of the password protection.

Password-protected wallets share the advantages and disadvantages of non-encrypted wallets (see Section III-A), with a few subtle differences. Password-protected wallets trade recoverability and usability for the mitigation of physical theft. If the password is forgotten, users lose the balance of their password-protected wallet since no mechanism exists for recovery[5]. For day-to-day use, users must unlock the wallet by entering their password when new transactions are made.

Fig. 1. Bitcoin paper wallet generated using https://bitcoinpaperwallet.com. The printout is designed to be folded such that the private key (right) remains hidden while the public component (left) remains visible.

Password-protected wallets may mislead the user to believe that the password itself provides access to their funds regardless of the location of the device storing the wallet, as would be congruent with a traditional mental model for web-based online banking. Users may be surprised to discover that they cannot access their funds at a new device by simply entering their encryption password; the wallet file must also be transferred to the new device.

C. Offline Storage of Keys

To further protect Bitcoin private keys from malware-based threats, wallets can be stored offline on some form of portable media, such as a USB thumbdrive. Keeping keys offline enables the use of traditional physical security techniques (*e.g.,* storing the drive in a fire-proof safe) to protect the wallet. However, offline storage has the drawback of making the wallet inaccessible for immediate use by software, preventing users from spending funds unless the offline storage media is nearby. As expected, offline storage can be used for backup, but all copies of the wallet must be kept offline for the full benefits of theft-protection to be realized. Prior to offline storage (wallet creation) and after storage (future transactions), the wallet will be exposed on a computational device, potentially to malware.

An interesting case of offline key storage is paper wallets (see Figure 1) where private keys are printed onto paper typically in the form of a 2D barcode (*e.g.,* a QR code) or as a long sequence of characters. Barcodes facilitate reading the key back into a Bitcoin client by, for example, scanning the code with a smartphone camera. Securing a paper wallet is similar to securing cash, which most users should be comfortable with. However, funds can be stolen from a paper wallet by simply observing the QR code (*e.g.,* on live television[6]), which is not possible with physical money. Thus transporting a paper wallet securely requires that the printed contents remain unobservable at all times. Users must remember that a paper wallet does not contain the funds itself, but rather enables signing authority over a set of Bitcoins. For example, if a paper wallet is discarded after funds are spent, the paper wallet still provides access to any future funds that may be sent to that address.[7]

As with any long-term storage, users must preserve software capable of decoding the QR code in the event that the paper wallet generation service is unavailable when attempting to

---

[5]Of course, exhaustive search of the password space is theoretically possible, and is available as a service: http://www.walletrecoveryservices.com

[6]"A Bloomberg TV Host Gifted Bitcoin On Air And It Immediately Got Stolen," *Business Insider*, 10/23/2013.

[7]"Five Ways to Lose Money with Bitcoin Change Addresses," *Bitzuma* (Blog), 17/03/2014.

reload keys onto a device. As of writing, many Bitcoin clients as well as offline storage solution use a common "wallet import format", which involves manipulating an ECDSA private key by performing cryptographic hashes, adding a checksum for integrity, and encoding the resulting string into Base58.[8]

D. Air-gapped Key Storage

In offline storage, we assume the device or media holding private keys cannot perform computations such as creating digital signatures. We distinguish this type of storage from air-gapped storage, where wallets are stored on a secondary device that generates, signs, and exports transactions, but this secondary device is never connected to a network. When spending Bitcoins using an air-gapped device, a transaction is created from the air-gapped device and the resulting signed output transported (usually through portable media) to an Internet-enabled device for transmission onto the Bitcoin network.

An air gap improves theft-resistance by never directly using a private key on an Internet-connected device. However, air gapped devices are capable of actually executing malware if infected. Malware may jump the air gap by infecting the portable media used to export signed transactions.

While not literally an air gap, hardware security modules (HSMs) emulate the properties of an air gap by isolating the key material from the host device, and only exposing the ability to sign transactions. Bitcoin-specific HSMs are under active development at the time of writing and a few have been recently released (*e.g.,* Trezor[9]).

Note that the consequences of obtaining access to the private keys are not much different from accessing a transaction-signing oracle for the wallet—both allow the current balance of Bitcoin to be stolen. However, future funds may be protected if access to the signing oracle is non-persistent.

E. Password-derived Keys

Thus far, all key management solutions have required users to maintain cryptographic keys. The remaining two solutions enable users to access their Bitcoin with a password instead.

The first approach is to derive cryptographic keys from a user-chosen password (*e.g.,* using PBKDF2 [26], manipulating the output to produce a valid Bitcoin private key). The disadvantage of using this approach directly is that only one resulting keypair is created, requiring the user to select a new (different) password for a new keypair.

A more robust approach is described in the Bitcoin Improvement Proposal 32 [24], and is known as a Hierarchical Deterministic (HD) Wallet. HD wallets deterministically derive a set of private keys from a master secret (a randomly chosen passphrase). These keys can derive new private keys. The deterministic nature allows the password holder to view the balance, as well as spend the funds, of any sub-account derived from the password. However, if the private key on one of the sub-accounts is compromised, only the funds sent to that sub-key (or sub-keys derived from it) may be stolen.

Password-derived wallets are targeted at loss-prevention and simpler cross-device access. The challenges of preserving access to a digital file are no longer necessary as long as the wallet can be re-generated from a memorized password. The primary drawback of a password-derived wallet is that weak user-chosen passwords can be found through unthrottled exhaustive search since a fingerprint of the associated public key will be in the global public ledger if the account holds any amount of Bitcoin. Rainbow tables [23] for password-derived keys have been developed.[10] Finally, it remains unclear whether memorization poses an advantage over maintaining a digital file when preventing loss—a forgotten password will orphan all funds in the account.

F. Hosted Wallets

A final approach to key management is to host user accounts on a third-party web service. In this case, the service maintains possession of the private keys. Hosted wallet web services provide the user with access to transactional functionalities through standard web authentication mechanisms, such as a password or two-factor authentication, and may also offer password recovery mechanisms. Bitcoin smartphone applications that act as clients to hosted wallets benefit from reduced application complexity (*i.e.,* no need to perform cryptographic operations on the device) and brick and mortar bank-like user interfaces. Currency exchange services that allow Bitcoin to be exchanged with fiat currency effectively provide this service, as do web services deployed specifically to host wallets.

It is natural to expect hosted wallet services will become primary targets of attack since these services typically hold large amounts of Bitcoin. Offloading the task of key management to a third-party requires users to assume the risk that the service could be breached and funds lost, in exchange for a traditional online banking-style user experience.

As a counter-measure to theft, hosted wallet providers often keep only a small float of their holdings online (called *hot storage*) and store the majority of their holdings offline in *cold storage*. This has the drawback of causing delays in transactions for users if the hot storage amount is exhausted. Hosted wallet services may also allow audits, where they cryptographically prove possession of sufficient Bitcoin to match their liabilities.

Another approach that falls under the hosted wallet category is a hybrid hosted wallet. Hybrid wallets use client side encryption (typically in Javascript) to encrypt all private keys and sensitive data. The web service is then only used for broadcasting transactions to the network and for displaying the user's balance (which requires inspecting the entire blockchain).

IV. EVALUATION FRAMEWORK

In this section, we systematize the major category-wide issues we have uncovered in describing the various key management approaches used by Bitcoin clients. We present an evaluation framework based on 10 criteria as shown in Table I and discussed in the following subsections. This framework both summarizes the advantages and disadvantages of the

---

[8]Base58 avoids the use of characters such as "0, O, I, and l" which may look visually similar, and also avoids punctuation characters which may trigger software (*e.g.,* e-mail clients) to perform line breaks.

[9]http://www.bitcointrezor.com

[10]D. Martyn. "Bitcoin 'Brainwallets' and why they are a bad idea," *Insecurety Research* (sic) (Blog), 26 Mar 2013.

Category	Example	Malware Resistant	Key(s) Kept Offline	No Trusted Third Party	Resistant to Physical Theft	Resistant to Physical Observation	Resilient to Password Loss	Resilient to Key Churn	Immediate Access to Funds	No New User Software	Cross-device Portability
Keys in Local Storage	Bitcoin Core			●		●	●	●	●		
Password-protected Wallets	MultiBit		○	●	○	●	○	●	●		
Offline Storage	Bitaddress	○	●	●			●				●
Air-gapped Storage	Armory	○	●	●		●	●	●			
Password-derived Keys	Brainwallet		●	●	○		○	●	●	●	●
Hosted Wallet (Hot)	`Coinbase.com`						●	●	●	●	●
Hosted Wallet (Cold)			○				●	●		●	●
Hosted Wallet (Hybrid)	Blockchain.info		○	○			●	●	●	●	●
Cash		●	●	●		●	●	●	●	●	●
Online Banking							●	●	●	●	●

TABLE I.   A COMPARISON OF KEY MANAGEMENT TECHNIQUES FOR BITCOIN (CONTRASTED WITH TRADITIONAL FINANCIAL SERVICES). ● INDICATES THE CATEGORY OF CLIENT IS AWARDED THE BENEFIT IN THE CORRESPONDING COLUMN. ○ PARTIALLY AWARDS THE BENEFIT. DETAILS PROVIDED INLINE.

various approaches we have evaluated, while also providing a benchmark for evaluating future key management proposals. The framework is adapted from a similar framework for evaluating password replacement schemes [7].

A. Evaluation Criteria

We briefly enumerate the criteria used to evaluate each proposal in the framework below.

Malware Resistant. Malware designed to steal Bitcoin wallets and related passwords has been observed in the wild. Wallets that are not stored on an Internet-connected device, or devices capable of performing computations are considered malware resistant (●), unless creating a transaction involves transferring to a computational device (○).

Key Stored Offline. For archival storage of infrequently used keys, keys not directly accessible from an Internet-connected device—either due to being offline (●) or online but password-protected (○)—are preferable.

No Trusted Third Party. All Bitcoin key management tools are trusted to a certain extent. This criteria considers the absence of a persistent trusted third party (●) that maintains direct signing authority over a user's Bitcoin.

Resistant to Physical Theft. If the cryptographic keys are stored on some media or device that can be physically stolen, we do not consider the tool to be resistant to physical theft. Within our framework, the only tools meeting this requirement rely on a human memorized password being necessary for key recovery. These are awarded (○) since passwords tend to be weak and may not adequately resist unthrottled guessing.

Resistant to Physical Observation. Physical observation, such as observing key strokes or capturing QR codes with a camera, may result in access to a user's Bitcoin account.

Resilient to Password Loss. If passwords are used (○), the loss of a password could result in some Bitcoin becoming unrecoverable if it is a necessary authentication factor in obtaining access to the signing key. For solutions where funds are held by third parties, these entities could provide a password recovery/reset mechanism (●).

Resilient to Key Churn. Assuming the client sends change from transactions to a newly created change addresses, a tool is resilient to key churn if it can maintain access to the funds even after exhausting the initial keypool (●). Tools not awarded this benefit are not guaranteed to maintain persistent access to new change addresses, and any balance sent to these addresses may be lost.

Immediate Access. Key management mechanisms that maintain direct access to the wallet enable Bitcoin to be transacted immediately (●). We award this benefit to techniques that require a user to enter a password. We omit the benefit for techniques that require data to be obtained from external storage medium or secondary device.

No New User Software. Some approaches require users to install new software on their system, for which the user may not have suitable permission, or software may not be developed for their specific platform (*e.g.,* some mobile platforms). By contrast, some tools can be executed from widely available software such as any standards-compliant web browser (●).

Cross-Device Portability. A key management technique is cross-device portable (●) if it allows easy sharing of the a Bitcoin address across multiple devices with minimal configuration or usability issues due to complexities like key churn.

B. Discussion

Table I demonstrates that key management approaches provide varying levels of security and convenience, with no single approach being obviously superior to others. One possible takeaway from our evaluation and comparison is that users can benefit heavily by offloading key management to a trusted party (*e.g.,* hosted wallets). The lower right side of the chart focuses on usability properties that are already present in traditional financial services (*i.e.,* resilient to password loss, no new software, cross-device portability). These properties are difficult to obtain if users independently manage their keys

through one of the local storage techniques. Of course, the disadvantage of trusting a third party is that Bitcoin funds are now bound by a contractual agreement between users and the hosted wallet provider, negating one of the primary features of Bitcoin: a fully decentralized currency. Users in countries lacking regulatory maturity for digital currencies should exercise caution when trusting a third party with large amounts of Bitcoin.

Based on our analysis, users can be given the concrete advice of treating digital currency much like they would treat fiat currency: keeping small amounts in ready-to-spend form (*e.g.,* local storage or online hosted walled) mimicking cash, and keeping larger sums in more difficult to access but more secure storage (*e.g.,* air-gapped or offline storage) mimicking a savings account or trust fund. Barber *et al.* [3] suggest the use of "super wallets" where users essentially run their own personal bank. A super-wallet keeps keys across multiple devices and requires all (or a subset using a threshold scheme) to be present to transfer funds to sub-wallets. Pre-configured transfers of small amounts can be authorized to move funds to sub-wallets that can be used for day-to-day spending. While the idea of super-wallets is intuitive, the implementation of such a scheme could introduce high levels of complexity.

V. USABILITY EVALUATION OF BITCOIN CLIENTS

A. Methodology

We used a series of cognitive walkthroughs [30] to evaluate the usability of six Bitcoin clients. Cognitive walkthrough is a form of expert evaluation where an expert (or group of experts) steps through the design to evaluate aspects of its usability. The focus of the walkthrough is on the novice user and emphasizes *learnability*. At each step, the evaluators ask three questions: Will the user see what to do? Will the user see how to do it? And once it is done, will the user know if they have performed the correct action?

We chose to use cognitive walkthroughs for several reasons. First, it allowed us to choose and compare standard tasks on disparate tools, and gave us easily compared insight into the common problems and successes of different Bitcoin clients. The cognitive walkthrough also allowed us to keep the focus on the novice user. The goal of our evaluation was to uncover problems specific to key management within Bitcoin software rather than to evaluate the usability of the clients themselves.

For our cognitive walkthrough, we defined a set of core tasks involving key management that a typical user needs to perform. We compared the results of each walkthrough against a standard set of evaluation guidelines, combining aspects of an heuristic evaluation [22] with the walkthrough in order to interpret our results.

Each of the following four tasks was independently performed by 2 experts to evaluate each tool:

T1 Configure a new Bitcoin address and obtain its balance. This task involves launching the Bitcoin client (or logging into one if hosted online) for the first time. After a new address has been generated (either explicitly or transparently in the background), the user should be confident that the address' balance is XBT 0.00000000. The user should also be able to find their receiving Bitcoin address.

T2 Spend Bitcoin. Send some amount of Bitcoin to an arbitrary (but valid) Bitcoin address. This task requires the user to create a new transaction, entering relevant information such as recipient, amount, *etc.*

T3 Spend Bitcoin from the same address as above, but on a secondary device. This task may require copying private keys to the secondary device, entering passwords on multiple devices, or logging in to a hosted wallet provider on a different browser.

T4 Recover from the loss of the main credential. In the case of locally stored keys, this task involves restoring a file from backup. Otherwise this task involves recovering from password loss.

Since the focus of our walkthrough was on configuration and learnability, we used a set of heuristics first developed for a usability evaluation of Tor [12]. We chose to use these guidelines because like the anonymity software, successfully managing Bitcoin involves the application of complex cryptographic knowledge in an everyday activity. The set of guidelines, from [12], are:

G1 Users should be aware of the steps they have to perform to complete a core task.

G2 Users should be able to determine how to perform these steps.

G3 Users should know when they have successfully completed a core task.

G4 Users should be able to recognize, diagnose, and recover from non-critical errors.

G5 Users should not make dangerous errors from which they cannot recover.

G6 Users should be comfortable with the terminology used in any interface dialogues or documentation.

G7 Users should be sufficiently comfortable with the interface to continue using it.

G8 Users should be aware of the application's status at all times.

B. Evaluated Clients

Real-world evaluation of the general approaches detailed in Section III is difficult. Thus, we select six distinct Bitcoin clients or utilities that implement the key management approaches described. For the purposes of our usability evaluation, each client was evaluated in its default configuration on OS X unless otherwise stated.

Keys in Local Storage. The reference Bitcoin client, Bitcoin Core [5], is a cross-platform client that stores keys locally (optionally encrypted with a password). Bitcoin Core is the first recommended client on the bitcoin.org website.

Password-protected (Encrypted) Wallet. We use the Multi-Bit [20] client (also recommended on bitcoin.org) since it provides a more convenient way to encrypt with a user-chosen password.

Offline Storage. We use paper wallets as offline storage. While paper wallets can be as simple as printing private keys on to paper, we select the paper wallet creation website Bitaddress.

org [25]. Bitaddress allows users to generate new randomized keys in their web browsers, and then print QR encoded keys.

Air-gapped Storage. We select the Bitcoin Armory [1] client which includes functionality for creating an offline wallet that can be used to sign and export transactions.

Password-derived Keys. One of the simplest ways to create a password-derived key is on the Brainwallet [9] website. The site allows users to enter a passphrase which is converted into a private key.

Hosted Wallets. We use Blockchain.info [6] as our hosted wallet provider. As of writing, Blockchain.info advertises the management of over 2.5 million user wallets.

VI. RESULTS

The following is the full details of our walkthroughs, which expands on the shorter version presented in the conference version of this paper.

A. Keys in Local Storage (Bitcoin Core)

We begin with an evaluation of Bitcoin Core, the original Bitcoin wallet client, which uses locally-stored keys. We assume the user has downloaded and installed the Bitcoin Core client (it has a straight forward wizard installation procedure).

T1: Configure. Bitcoin Core transparently generates a new set of addresses on first run, but shows no notification to the user that this has occurred (fails G3). The receiving address can be found under the *Receive coins* tab, but this could be easily confused with the *Addresses* tab which contains a contact list of other user addresses (fails G2).

To retrieve the account balance, Bitcoin Core must be online and the user must wait until a full copy of the blockchain has been downloaded. Except for a small status indicator on the bottom-right side of the window that shows a small red cross in-between two black windows, there are no other messages to show the user that the application should be online. Due to the size of the blockchain, this may take hours to days to complete. A status bar displaying "Synchronizing with network" shows the progress of the blockchain download (achieves G8), but the terminology may be too technical for novice users, With a mouse over the icon, it says '0 active connections to bitcoin network' which is likely unfamiliar language that does not help resolve the error (fails G4 and G6). Once the blockchain has been downloaded, the balance is displayed on the *Overview* tab (achieves G3).

T2: Spend. Spending Bitcoin is straightforward since the keys are readily available to the Bitcoin Core client. Users spend Bitcoin by navigating to the *Spend* tab (achieves G1 and G2). Since our focus is on key management, we do not evaluate the actual completion of transactions (which may have additional usability issues). We focus on ensuring the key is available to the software tool (which is not so straightforward with *e.g.,* offline storage).

T3: Spend from Secondary Device. Installing Bitcoin Core on a secondary device creates a new set of keys. Users may not understand that the keys must be copied to the secondary device (fails G1), and if so, what file must be copied (fails G2). The correct procedure is to back up the `wallet.dat` with the 'backup wallet...' option in the 'File' tab of the first installation and chose a directory to save the `wallet.dat`. Next the user must securely transfer this file to the secondary device, and no guidance is provided on how to do this (fails G2) or the dangers of transferring it through an insecure mechanism (fails G5).

Assuming the user has transferred `wallet.dat` to the secondary device, she could try looking for import options in the newly installed wallet client, or drag and drop the `wallet.dat` into the client, but she would fail to do so as no import option exists. The documentation is inadequate here as well—there is actually nothing in the help menu except a debug window that is for advance user to tweak the application (fails G2 and G6)!

The only mechanism to activate the wallet on a secondary file is to actually overwrite `wallet.dat` on the secondary device with `wallet.dat` from the first. It is unlikely any novice user would be able to complete this step. It is actually even difficult to find the path to copy `wallet.dat` to on the new device—this could be possible by searching the local file system for `wallet.dat`, which might not succeed due to non-searchable system reserved folders or not knowing the exact file name (spotlight does not return any result for `wallet.dat`). More likely, the user will search online.[11] On OS X, the path is `/Users/User/Library/Application Support/Bitcoin/wallet.dat`.

The next step is to replace the new `wallet.dat` with the one from the primary device. It should be noted that the name of the file should be exactly `wallet.dat` for the Bitcoin Core to be able to read the file. Some of errors that the user might encounter during this procedure are:

- The user might copy `wallet.dat` from the primary device wallet client path instead of the one exported through the back up option. This could cause a corrupted `wallet.dat` that is not readable by the secondary device's Bitcoin Core. This is due to Bitcoin Core's procedure to lock `wallet.dat` while it is in use. The error is recoverable by repeating the process correctly (fails G4).

- User should wait for the Bitcoin Core on the secondary device, to download and sync the Blockchain from the P2P network to be able to authorize a transaction.

- On the secondary device, the final balance might be wrong and there would be the need to resynchronize and rescan the blockchain to have the correct final balance (fails G3).

Finally, this process must be repeated if either client exhausts their keypool. If both do, there is no way to merge the new keys in the keypool, and replacing one `wallet.dat` with the other will lead to unrecoverable funds (fails G5).

We note that replacing the key file may require a re-scan of the blockchain to display the correct balance (fails G3).

---

[11] https://en.bitcoin.it/wiki/Data_directory

T4: Recovery. If only one device is used, there is no way to recover from loss of the key file (*e.g.,* due to a disk failure, file corruption, or loss of the device itself; fails G5). If the user backed-up the key file, the process for recovering from loss is equivalent to that of T3 above.

B. Password-protected Wallets (MultiBit)

Although it is possible to encrypt the `wallet.dat` with a password in Bitcoin Core, it is not the default option nor is there any cue to do so. Instead we evaluate the MultiBit client, where one of the recommended first steps is to password protect the wallet file. MultiBit is a popular client in particular for its use of SPV[12] for lightweight blockchain validation that can complete within minutes instead of, relative to Bitcoin Core, days.

T1: Configure. On first run, a welcome page contains an explanation of common tasks that can be performed with MultiBit—where the send, request and transaction tabs are and how to password protect the wallet file (achieves G1 and G2). The client provides help options for other functionalities with direct and non-technical guides (achieves G6).

MultiBit automatically generates a new receiving address on first run, but does not notify the user (fails G3). Reading the newly generated address requires navigation to the *Request* tab, which displays "Your address" as well as a copy of the QR code of address.(partially achieves G2).

The interface shows the status of the program (online, offline, or out of sync) on the bottom left status bar, the balance of the user's wallet on the upper left, and the latest price of bitcoin on the upper right of the window (achieves G8). The interface seems to minimize jargon and technical vocabulary (achieves G6). As it is mentioned on the *welcome page* of the application, every option in MultiBit has the ability to show help tips by hovering the mouse over that option (achieves G6 and G7).

The displayed balance is not necessarily current until synchronization is completed, however there is no direct cue in the balance area indicating this (achieves G8).

T2: Spend. The user must navigate to the tab labeled *Send*, as instructed on the welcome screen (achieves G1 and G2). If the client is not synced, the send button is disabled (achieves G4). If it is synced, the user fills out the transaction details, destination address and amount and clicks send. The client prompts the user for the decryption password (achieves G2). An incorrect password displays the error 'The wallet password is incorrect' but otherwise allows immediate and unlimited additional attempts. Entering the correct password authorizes the transaction (achieves G3).

T3: Spend from Secondary Device. On the primary device, the user must navigate to the *Options* menu, and select *Export private keys* under tools (fails G1 and G2). The interface displays a wizard requesting an export password as well as a file system path for the exported file to be saved. If the user attempts to save the exported file without password, a warning is displayed in red: 'Anyone who can read your export file

can spend your bitcoin' (achieves G5). By having a password-protected file exported, the user can securely copy the file to the secondary device with some protection against interception. After clicking to export, wallet file is saved in the given path and the client checks that the file is readable (achieves G4 and G5).

On the secondary device, the user must select *Import private keys* from the *Options* menu. After selecting the previously exported file, the wizard confirms the completion of the import (achieves G4) and the balance is updated to reflect the newly imported keys. The user can proceed to create a new transaction as in T2. MultiBit sends change to the originating address, so keypool churn is not an issue.

T4: Recovery.. As with Bitcoin Core, recovery is not possible if no backup of the wallet file was made. Creating a backup and importing it follows the same procedure as T3. As expected, both the password and the backed up wallet are necessary for recovery.

C. Air-gapped Key Storage (Armory)

Bitcoin Armory is an advanced bitcoin wallet that allows the wallet to be stored and managed on an offline device, while supporting the execution of a transaction through an online device. Armory is also used on the online device to obtain the blockchain and broadcast the transaction created on the offline device. It is possible to use some other online application to implement the airgap, however this is the recommend method and the one we will consider.

T1: Configure. The user begins by installing Bitcoin Armory on the offline computer. On the start, the welcome page offers the option to 'Import Existing Wallet' and 'Create Your First Wallet!' (achieves G2). The user creates her wallet, with passphrase-protection being a mandatory option. Armory asks to verify the passphrase and warns the user not to forget her passphrase (achieves G5). After this step, a backup window pops up with the options to print a paper wallet or save a digital backup of the wallet, and also warns the user if he decides not to backup his wallet (achieves G5). After proceeding, the user must click on 'Receive Bitcoins' to prompt the client to generate the bitcoin address in the wallet file (fails G3,G4). By contrast, most clients do this step automatically on launch.

In order to see the balance of the account, the device must be online and synced (fails G2). Thus the user must use the online device, not the offline device, to check her balance. Users can click on the 'Offline Transaction' button, which offers a short documentation of the steps to be taken to sign a transaction and in doing so, explains the offline/online distinction relevant to checking a balance. Within the 'Wallet' window, there is an option to 'Create Watching-Only Copy.' The language is difficult (fails G6): this option allows a copy of only the bitcoin addresses to be exported, not the private keys, for use on the online computer to display an updated balance for each address. The exported file can be copied to the online computer.

We assume the user has installed Armory on the online computer. It should be noted that Armory only works side-by-side with Bitcoin Core and uses Bitcoin Core to synchronize and read the downloaded blockchain (fails G1). A pop-up

---

[12]Simplified Payment Verification**??**

window will alert the user when 'the blockchain' has been downloaded (partially achieves G3). Armory displays a 'Connected' cue in green in the bottom-right when it connects to Bitcoin Core. Upon launching Armory, the user should click on 'Import Existing Wallet' and she is prompted to import either a digital backup or watch-only wallet. She should chose the watch-only back up file that has been copied from the offline computer. After the application is done syncing, the balance is displayed on the main window under 'available wallets' (achieves G8).

T2 & T3: Spend. With an air gap, the distinction between primary and secondary devices is less clear given that the basic setup itself includes two devices: one online and one offline, but authorization of transactions uses the offline device. To authorize a transaction, the user may begin from Armory on the online or offline device (may not fully achieve G2). On the either device, the user should click on 'Offline Transactions' in the main window which displays a very detailed description of the steps involved (achieves G1, G2, and G6). On the online computer, the user clicks the option: *Create New Offline Transaction*. The user will be asked to enter the transaction details to generate an unsigned transaction as a file. The user must transfer this file to the offline computer. As mentioned in this step's documentation, the unsigned transaction data has no private data (the exact data will ultimately be added to the public blockchain) and no harm can be done by an attacker who captures this file (achieves G5) other than learning the transaction is being prepared.

On the offline computer, the user clicks on *Offline Transactions* and then *Sign Offline Transaction* which prompts the user for the unsigned transaction data file. Armory asks the user to review all the transaction information, such as the amount and the receiving addresses (achieves G5). By clicking on the *sign* button signed transaction data can be saved to a file. Text at the top of the window describes the current state of the file (signed) and what must be done (move to online device) to complete the transaction (achieves G1 and G2).

The signed file should be transferred to the online computer and be loaded through the same offline transaction window. When a signed transaction is detected, the *Broadcast* button becomes clickable. By clicking on broadcast, the user can once more review transaction details, and receive confirmation that the Bitcoins have been sent (achieves G3 and G8).

T4: Recovery. Like Bitcoin Core and MultiBit, Armory requires a backup of the wallet to have been made. Without this backup, recovery is impossible. Armory encourages backups at many stages (achieves G1 and G2). Armory provides many prompts for the user to back up her wallet keys. At the time of creating the wallet, there are multiple windows and alerts conveying the importance of a back up, with options for digital and paper copies. Even if user decides not to back up her wallet at this stage, she is provided a persistent 'Backup This Wallet' option (achieves G4). In the backup window, there are a number of options to back up: a digital copy, paper copy, and others. By clicking on the 'Make Paper Backup' for example, the paper backup is shown to the user containing a root key that consist of 18 sets of 4-characters words and a QR code. To restore the paper wallet backup, on the main page, the user can click on 'Import or Restore Wallet' and select 'Single-sheet

Backup' option. She will be prompted to input the Root Key from the paper wallet. The 'Digital Backup' option provides an unencrypted version of the wallet file that can be securely stored on portable media. Recovering from a lost wallet with a digital backup involves selecting 'Import Digital Backup or watch-only wallet' from the 'Import or Restore Wallet' window as explained in core task 1. Armory also enables the user to test the backups to ensure there is no error in the backup file (achieves G5) through the same import procedure.

D. Offline Storage (Bitaddress)

There are different methods to use for offline storage of a bitcoin wallet. For our evaluation, we consider the use of a paper wallet. Specifically we use the Bitaddress web-service, a popular Bitcoin paper wallet generator. Many paper wallet generators exist, however Bitaddress, at the time of writing, is the first search result for 'bitcoin paper wallet' on Google.

T1: Configure. Upon visiting the bitaddress.org, the user is asked to move the mouse or enter random characters in a text box to generate a high-entropy random seed to be used to generate a private key associated with the Bitcoin address (achieves G1 and G2). Once enough entropy has been collected, the site redirects the user to a page that shows the receiving Bitcoin address and it's associated private key (achieves G3). The public key (Bitcoin address) is labeled *Share* in green text and the private key *Secret* in red (helping achieve G5). In general Bitaddress uses non-expert terminology and simple instructions (achieves G6). To ensure the web service does not retain a copy of the users' key (the generation is done client-side in Javascript), the user should complete the process offline.

After printing, the user has a bitcoin receiving address and, as mentioned in the documentation, the balance can be checked through a Bitcoin Block Explorer[13], such as blockchain.info. The user uses this site to search for her bitcoin address and checks her balance. Although it is documented that the private key must remain secret, a user may inadvertently expose the private key by placing the paper wallet where it can be observed or by searching the website for the private key instead of the public key.

T2 & T3: Spend. Since the keys are printed on paper, there is no difference between authorizing from a primary or secondary device so we collapse the analysis of core tasks 2 and 3.

To send funds from a bitcoin address that has been stored on a paper wallet, as it is mentioned in the documentation, the user has to import her private key in one of the wallet clients available, such as Armory or the Blockchain.info hosted wallet discussed below. If the user inputs the private key address into a client that returns change to newly generated addresses, she must export these new addresses to a new paper wallet or she will lose the surplus when she removes the wallet from the client (fails G5). If the user fails to remove the wallet from the client, a second copy is maintained increasing her exposure to theft (but reducing her exposure to key loss).

The process to import a key from a paper wallet depends on the client. For Blockchain.info, after making an online account,

---

[13]webservice that provides access to the blockchain

the user navigates to the 'Import/Export' tab and uses the option 'Import Using Paper Wallet, Use your Webcam to scan a QR code from a paper wallet.' It is also possible to type in the private key in the 'Import Private Key' text field. After this step, the address now is hosted on the online wallet and is the same as core task 2 in the Hosted Wallet section (Section VI-F) below.

T4: Recovery. Loss of a paper wallet makes the funds unrecoverable (fails G5). Bitaddress prompts the user to acknowledge this fact (also mentioned in its short documentation) when creating a paper wallet (achieves G1).

E. Password-Derived Keys (Brainwallet)

The most popular and complete implementation of a deterministic wallet with password-derived keys, at the time of writing, is Brainwallet.

T1: Configure. The Brainwallet website displays by default a pre-generated address corresponding to an empty passphrase. The passphrase input field displays "Long original sentence that does not appear in any song or literature. Never use empty passphrase. (SHA256)", but there is no corresponding documentation explaining the purpose of the passphrase or how it relates to the generated key (fails G1, G2, G6). As characters are typed, a new key is generated. User may not notice that generation of keys is happening dynamically, possibly preventing the user from noticing that the task is complete (fails G3). The user should replace the default passphrase with her own, hopefully ensuring her passphrase is not a commonly used phrase or anything that could be brute forced by an offline dictionary attack[14] as this passphrase is sufficient to access the funds stored in the resulting bitcoin address. On entering the desired passphrase, the public and private keys are displayed on the same page.

Once the address has been generated, retrieving the balance of that address requires the use of an external service, but no suggestions are provided on the site (fails G1 and G2). The interface does display a number of other fields (*e.g.,* additional encodings of the public key) which may not be meaningful to novice users (fails G6 and G7).

T2 & T3: Spend. Spending Bitcoins from a password-derived wallet requires the user to import the private key into another client. The user should experience similar usability challenges as those detailed in the Offline Storage client above.

T4: Recovery. Forgetting the password of a password-derived key leads to funds becoming unrecoverable (fails G5). Users will typically return to the same website (*i.e.,* the Brainwallet website) to extract private keys, but this may not be possible if the site is inaccessible (fails G5).

F. Hosted Wallets (Blockchain.info)

A variety of online services offer online hosted wallet clients to users. We use the popular Blockchain.info webservice for our evaluation.

---

[14]"Bitcoin Brainwallets and why they are a bad idea", *Insecurety Research* (blog), 3/26/2013.

T1: Configure. The user navigates to the Blockchain.info site and creates a new wallet by providing an email address and a (min) 10 character password (achieves G1 and G2). A message warning the user about the importance of not forgetting the password is displayed during registration (achieves G5). Next, a *Wallet Recovery Mnemonic* is shown to the user as a backup in case the password is forgotten. The balance and address are immediately displayed (achieves G3).

T2 & T3: Spend. Hosted wallets are accessible from any web browser, so creating transactions from many devices is straightforward. The user logs in to the site, clicks *Send money* (achieves G1 and G2). After filling in the required fields, the user is informed that the Bitcoins have been sent (achieves G3). Some of Blockchain.info's error messages may be too technical for novice users. For example, *No free outputs to spend* is displayed when transactions are created without sufficient funds (fails G6).

T4: Recovery. To recover from a forgotten password, a wallet recovery mnemonic may be provided on the login page. By clicking the *Recover Wallet* button, the site will ask for the mnemonic phrase and the email address send the new credentials (achieves G1 and G2). Another recovery option is to proactively make backups and import them in case recovery is needed. To do so, in the main wallet page, user has to click *Import/Export* and exporting either an encrypted or unencrypted backup. Unencrypted backups should be kept in a secure storage. There are different options for the unencrypted backup procedure that could confuse the user and might result in an unrecoverable backup (fails G5 and G6)—the back up is shown on a text field that the user has to copy and paste into a text file to be able to save it on her computer (fails G2, G3 and G7). To restore the backups, there is an 'Import Wallet' option.

VII. Discussion

A. Metaphors

Bitcoin naturally invites a metaphor to traditional currency. This metaphor is often used in the clients (*e.g.,* send coins, receive coins, wallet), but does not always support their usability. The coin metaphor fails in both of the ways that user interface metaphors traditionally fail [10]: aspects of Bitcoin transactions do not easily fit the coin metaphor, and conversely, encourages users to overextend the metaphor. Both of these lead to confusion on the part of users.

One way in which the metaphor of physical coins fails is in the sending and receiving of Bitcoin. In the physical world, the same physical token is almost always used to represent the same unit of currency (*i.e.,* giving money to a friend involves handing them the coin). However, when Bitcoins are exchanged, the private key is not transferred along with the balance. Private keys remain in possession of the sender, and can be reused and associated with new coins at a later time.

Many of the evaluated clients use the word "Send" to describe authorizing (digitally signing) a transaction, and private keys are not mentioned in any of the evaluated clients at the moment of transaction. It may appear counter-intuitive that this is a bad thing, but never mentioning the existence of keys may cause further confusion. The password-protected wallets, (*e.g.,*

Multibit) require the user to input their password, but do not clarify the reason for the password.

Addresses are another metaphor that relate to the issue of transacting. The evaluated clients use the word "Address" to refer to the public key associated with a private key held by some user. This seems to be a relatively successful metaphor: it emphasizes the public nature of the public key, and also divorces the user's perception of a relationship between the public and private keys. To momentarily extend the metaphor, a user is accustomed to the idea that they will need to share their address in order to receive an item. However, the private key is more akin to the key to their mailbox, and a user would never think that they should share their mailbox key in order to receive mail to an address.

Another pervasive metaphor in the evaluated clients is the Bitcoin "wallet", where the user's Bitcoins are stored. The wallet metaphor is deeply entrenched in the foundations of Bitcoin. The reference client, *Bitcoin Core*, stores private keys in a file named wallet.dat and the *MultiBit* client invites users to "create your first wallet!" on first launch. The hosted clients also use the metaphor; Blockchain.info prominently shows a Wallet tab, under which users are invited to "Create My Free Wallet". The wallet metaphor is descriptive for users, but fails to encompass the complexity of a user's collection of keys. In reality, the Bitcoin wallet contains private keys, but the term wallet is used to describe both the file storing the private keys, and the main interface of Bitcoin clients (as in Blockchain.info). This main interface sometimes includes a variety of other information, such as transaction history, address book, currency exchange rates, *etc.*

B. Abstractions

Abstraction and automation are complex issues for security software. Often, security is too complex to be completely automated, and the problem cases are often punted to the user (*e.g.,* in the case of TLS certificates [4]).

On first run, all of the evaluated clients transparently generate keypairs without informing the user. This behaviour continues as new transactions are made, where clients generate new addresses with no user notification (*e.g.,* for receiving change). It is unclear how well this abstraction works: while users do not need to be burdened with the knowledge of each private key, there are still situations in which a user might need to manage those keys, and the abstraction prevents users from doing so. Recovery from key loss depends on the existence of an up-to-date backup. While backup sounds like a simple task, in many of the evaluated clients, it involves finding the right menu (MultiBit), or the right file (Bitcoin Core). Some clients do prompt the user to back up their wallets (*e.g.,* Bitcoin Armory), but with the private keys so completely abstracted away, users may not even understand what they are backing up, or why. Key churn, and the consequent need for semi-regular backups complicate the issue even farther.

The abstractions made in Bitcoin clients are sometimes beneficial for users, such as in the case of displaying a user's balance. A user's Bitcoin balance is typically made up of many small amounts corresponding to many private keys. However, most of the evaluated clients abstract these balances into a single figure. This highlights a usability disadvantage of paper wallets – the user must manage these multiple balances manually, and there is no method of seeing an aggregate balance when multiple paper wallets are in use.

C. Technical Language and Content
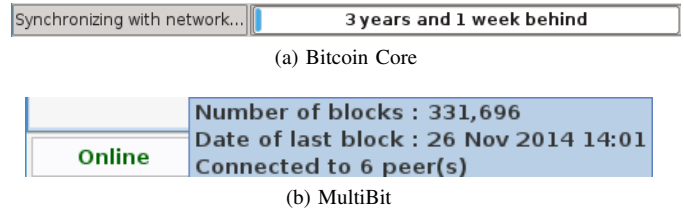

(a) Bitcoin Core

(b) MultiBit

Fig. 2. Screenshots of technical language displayed by two different clients.

When performing our evaluation, we identified multiple occurrences of highly specialized or technical language used in the Bitcoin clients. These instances of technical language are confusing, particularly to novice users who are unlikely to be aware of either the jargon, and for whom the language will not help clarify the issues. The language itself highlights the complexity of the tasks associated with Bitcoin, and the difficulty of explaining them simply.

Examples of such language included messages in MultiBit and Bitcoin Core that referenced the client being "out of sync" or "synchronizing with network" (see Figure 2a) referring the process of downloading a full copy of the blockchain or retrieving relevant blocks from a trusted peer. A related message in MultiBit (Figure 2b) and Armory displayed the number of blocks that had been downloaded, as well as the number of connections to the Bitcoin network. These messages are intended to communicate that clients may benefit from faster transaction notifications when connected to more peers, but since peer connectivity is difficult for users to control, there is little benefit in communicating these ideas with the user. Similarly, the number of blocks independently has little significance to most tasks performed by an end user. We suggest that not only could this language be clarified, but that the interfaces could also streamline the amount of information that is presented to the user on every screen.

We also noticed that some clients used highly technical language when they could have used the metaphor to provide a simpler explanation to users. When attempting to authorize a new transaction on Blockchain.info with insufficient funds, the web interface displayed "no free outputs to spend". This error message is confusing, and would be more easily understood if it referred to the lack of coins instead of the lack of outputs. Similarly, essential actions such as importing or exporting keys were often buried behind advanced or debug menus.

In the evaluated clients, there were often few resources to which users could turn for help. In the cognitive walkthroughs, the answer to the question "will the user know what to do?" was almost always unclear. Interface cues and features such as tool tips, wizards, or other contextual help were almost entirely lacking. Some actions were guided (*e.g.,* Multibit's prompted backups or create your first wallet), but many actions such as obtaining the balance of a paper or password-derived address were unsupported by help or documentation.

VIII. Conclusion

Bitcoin's usability limitations, particularly those related to key management, pose challenges to its rising popularity. In our evaluation, we found that developers in the Bitcoin ecosystem are making innovative attempts at solving the decades-old problem of usable key management. While some of these techniques seem promising, we find that tasks involving key management can be mired in complex metaphors and confusing abstractions.

Further investigation is needed to better understand and address these issues. A user study would give insight into exactly how these problems are affecting users and it would be interesting to investigate how expert users are (apparently successfully) handling these challenges. Bitcoin presents a new opportunity for public key cryptography to become mainstream, and our evaluation is a first step towards achieving usable key management in decentralized cryptocurrencies.

References

[1] Armory, "Armory Secure Wallet," https://bitcoinarmory.com.
[2] M. Baker, K. Keeton, and S. Martin, "Why traditional storage systems don't help us save stuff forever," in *HotDep*, 2005.
[3] S. Barber, X. Boyen, E. Shi, and E. Uzun, "Bitter to Better — How to Make Bitcoin a Better Currency," in *Financial Cryptography*, 2012.
[4] R. Biddle, P. C. van Oorschot, A. S. Patrick, J. Sobey, and T. Whalen, "Browser interfaces and extended validation ssl certificates: An empirical study," in *CCSW*, 2009.
[5] Bitcoin Core Developers, "Bitcoin Core," https://bitcoin.org.
[6] Blockchain Team, "My Wallet Be Your Own Bank," https://blockchain.info.
[7] J. Bonneau, C. Herley, P. C. van Oorschot, and F. Stajano, "The quest to replace passwords: a framework for comparative evaluation of web authentication schemes," in *IEEE Symposium on Security and Privacy*, 2012.
[8] J. Bonneau, A. Narayanan, A. Miller, J. Clark, J. A. Kroll, and E. W. Felten, "Mixcoin: Anonymity for bitcoin with accountable mixes," in *Financial Cryptography*, 2014.
[9] brainwallet, "Brainwallet," https://brainwallet.github.io/.
[10] J. Carroll, W. Kellogg, and R. Mack, *Interface metaphors and user interface design*, 1987.
[11] D. Chaum, "Blind signatures for untraceable payments," in *CRYPTO*, 1982.
[12] J. Clark, P. C. van Oorschot, and C. Adams, "Usability of anonymous web browsing: An examination of tor interfaces and deployability," in *SOUPS*, 2007.
[13] D. Florencio and C. Herley, "Is everything we know about password stealing wrong?" *IEEE Security & Privacy*, vol. 10, no. 6, 2012.
[14] S. L. Garfinkel, D. Margrave, J. I. Schiller, E. Nordlander, and R. C. Miller, "How to make secure email easier to use," in *CHI*, 2005.
[15] S. L. Garfinkel and R. C. Miller, "Johnny 2: A user test of key continuity management with S/MIME and outlook express," in *SOUPS*, 2005.
[16] S. Gaw, E. W. Felten, and P. Fernandez-Kelly, "Secrecy, flagging, and paranoia: Adoption criteria in encrypted email," in *CHI*, 2006.
[17] A. Gervais, G. Karame, D. Gruber, and S. Capkun, "On the privacy provisions of bloom filters in lightweight bitcoin clients," in *ACSAC*. ACM, 2014.
[18] C. Herley and P. C. van Oorschot, "A Research Agenda Acknowledging the Persistence of Passwords," *IEEE Security & Privacy*, vol. 10, no. 1, pp. 28–36, 2012.
[19] I. Miers, C. Garman, M. Green, and A. D. Rubin, "Zerocoin: Anonymous Distributed E-Cash from Bitcoin," in *IEEE Symposium on Security and Privacy*, 2013.
[20] MultiBit Team, "MultiBit," https://multibit.org.
[21] S. Nakamoto, "Bitcoin: A peer-to-peer electionic cash system," Unpublished, 2008.
[22] J. Nielsen, "Finding usability problems through heuristic evaluation," in *CHI*, 1992.
[23] P. Oechslin, "Making a faster cryptanalytic time-memory trade-off," in *CRYPTO*, 2003.
[24] Pieter Wuille, "BIP32: Hierarchical Deterministic Wallets," https://github.com/genjix/bips/blob/master/bip-0032.md.
[25] Pointbiz, "JavaScript Client-Side Bitcoin Wallet Generator," https://bitaddress.org.
[26] RSA Laboratories, "PBKDF2 (Password-Based Key Derivation Function 2)," http://tools.ietf.org/html/rfc2898.
[27] S. Sheng, L. Broderick, C. A. Koranda, and J. J. Hyland, "Why Johnny still can't encrypt: Evaluating the usability of email encryption software," in *SOUPS (Poster)*, 2006.
[28] Symantec, "Infostealer.Coinbit," http://www.symantec.com/security_response/writeup.jsp?docid=2011-061615-3651-99.
[29] S. Vanstone, "Responses to NIST's Proposal," *Communications of the ACM*, vol. 35, pp. 50–52, 1992.
[30] C. Wharton, J. Rieman, C. Lewis, and P. Polson, "The cognitive walkthrough method: a practitioner's guide," in *Usability Inspection*. Wiley & Sons, 1994.
[31] A. Whitten and J. D. Tygar, "Why Johnny can't encrypt: a usability evaluation of PGP 5.0," in *USENIX Security*, 1999.