CRYPTO GLACIER

The step-by-step protocol for storing crypto in a highly secure way

Based on the popular Glacier protocol

Version 0.94.1 RC2

Check the latest version (https://vogelito.github.io/glacierprotocol/)

1. Glacier overview

1.1. About CryptoGlacier

CryptoGlacier is a step-by-step protocol for storing crypto assets in a highly secure manner. It is based on the popular Glacier Protocol (https://glacierprotocol.org/).

There are two main differences between CryptoGlacier and Glacier:

- CryptoGlacier adds support for other cryptocurrencies
- CryptoGlacier is not intended for personal storage, but situations where various, distinct, keyholders must agree with each other to access funds

CryptoGlacier **does not address institutional security** needs such as internal controls, and transparent auditing. It **does prevent** access to funds by a single individual.

Just as Glacier, CryptoGlacier is also intended for:

- **Large amounts of money ($100,000+)**: CryptoGlacier thoroughly considers corner cases such as obscure vectors for malware infection, human error resulting in loss of funds, and so on. Even if your crypto holdings are more modest, it's worth considering using CryptoGlacier. If crypto proves successful as a technology, it will appreciate 10x (or much more) in the coming years. Security will become increasingly important if your holdings appreciate and crypto becomes a more attractive target for thieves. Consider Glacier Protocol (https://glacierprotocol.org/) if you're looking for personal storage.
- **Long-term storage**: CryptoGlacier not only considers the crypto security landscape today, but also a future world where crypto is much more valuable and attracts many more security threats.
- **Infrequently-accessed funds**: Accessing highly secure funds is cumbersome and introduces security risk through the possibility of human error, so it is best done infrequently.
- **Technically unskilled users**: Although the CryptoGlacier protocol is long, it is clear and straightforward to follow. Some technical expertise is required.

The CryptoGlacier protocol covers crypto storage, not procurement. It assumes you already possess crypto and wish to store it more securely.

If you are already familiar with crypto security concepts and are certain that you want high security, multi-signature, cold storage, you may prefer to read Trusting This Protocol and then skip to the section Choosing a Multisignature Withdrawal Policy.

Supported Cryptocurrencies

This document currently supports:

- Bitcoin

- Bitcoin Cash
- Ethereum
- Etherum-based ERC20 tokens
- Litecoin
- XRP

Trusting this protocol

Funds secured using CryptoGlacier can only be as secure as its design. As previously mentioned, CryptoGlacier is based on the Glacier protocol but has some important distinctions and at the time of writing it hasn't yet attracted widespread Expert Advisory or Community Review.

CryptoGlacier and CryptoGlacierScript, the CryptoGlacier companion software, are **open source**. The code is straightforward and well-commented to facilitate easy review for flaws or vulnerabilities. View it on Github (https://github.com/vogelito/CryptoGlacierProtocol).

All documentation and code related to this protocol is under open licenses (Creative Commons for the document, MIT license for the code), enabling others to publish their own revisions. Inferior alternatives will tend to lose popularity over time.

If you like, you may review the design document of the Glacier Protocol for details on the technical design.

Background
Glacier vs CryptoGlacier

Let's start by assessing whether Glacier or CryptoGlacier is right for you. Some advantages of CryptoGlacier include:

- **Storing multiple crypto assets**: Although Bitcoin remains the most popular cryptocurrency, today we have hundreds of crypto assets which are currently not supported by the Glacier protocol.
- **BIP39 Mnemonics**: CryptoGlacier allows you to derive keys across crypto-protocols by storing a 24-word mnemonic. This adds simplicity for key-storage and supports multiple protocols.
- **Multiple key holders**: CryptoGlacier was designed for multiple key holders and prevents a single entity from ever having unilateral access to funds. Glacier is meant for personal storage of Bitcoin where a single individual can always access funds unilaterally.
- **HD wallets for Bitcoin, Bitcoin Cash and Litecoin**: CryptoGlacier increases privacy by utilizing HD wallets instead of reusing addresses as the original Glacier Protocol does.

Self-Managed Storage vs. Managed by a third party

There is no such thing as perfect security. There are only degrees of security, and those degrees come at a cost (in time, money, convenience, etc.) So the first question is: How much security are you willing to invest in? In the last few years we've seen a rise of 3rd party institutional custody solutions for crypto assets. Most allow for multiple signatories, estate planning, etc. The pros and cons of the various 3rd party services are beyond the scope of this document. Some options are BitGo (https://bitgo.com/), Coinbase Custody (https://custody.coinbase.com/), Ledger Vault (https://www.ledger.com/vault/), and Vo1t (https://vo1t.io/).

However, all 3rd party storage services still come with some notable risks which self-managed storage does not have:

1. **Identity spoofing**: Your account on the service could be hacked (including through methods such as identity theft, where someone convinces the service they are you).
2. **Network exposure**: 3rd party services still need to transmit security-critical information over the Internet, which creates an opportunity for that information to be stolen. In contrast, self-managed storage can be done with no network exposure.
3. **Under constant attack**: 3rd party services can be hacked by attackers from anywhere in the world. People know these services store lots of funds, which makes them much larger targets. If there's a flaw in their security, it's more likely to be found and exploited.
4. **Internal theft**: They have to protect against internal theft from a large group of employees & contractors.
5. **Intentional seizure**: They have the ability (whether of their own volition, or under pressure from governments) to seize your funds. There is historical precedent for this, even if funds are not suspected of criminal involvement. In 2010, Cyprus unilaterally seized many bank depositors' funds (https://www.theguardian.com/world/2013/mar/25/cyprus-bailout-deal-eu-closes-bank) to cope with an economic crisis. In 1933, the US abruptly demanded citizens surrender almost all gold they owned to the government (https://en.wikipedia.org/wiki/Executive_Order_6102). Regardless of how one views the political desirability of these particular decisions, there is precedent for governments taking such an action, and one cannot necessarily predict the reasons they might do so in the future. Furthermore, Bitcoin still operates in a political and legal gray zone, which increases these political risks.

Some 3rd party services have insurance to cover losses, although that insurance doesn't protect against all of these scenarios, and often has limits on the amount insured.

These risks are not theoretical. Many online services have lost customers' funds (and not reimbursed them), including Mt. Gox (https://www.bloomberg.com/news/articles/2014-02-28/mt-gox-exchange-files-for-bankruptcy), Bitfinex (http://www.bbc.com/news/technology-37009319), and many more.

Many people use online or hybrid solutions to store sizeable amounts of money. We recommend self-managed storage for large holdings, but ultimately it's a personal decision based on your risk tolerance and costs you're willing to pay (in money and time) for security.

CryptoGlacier focuses exclusively on storage managed by different entities where not a single entity has unilateral control over the funds.

CryptoGlacier vs. Hardware Wallets

Many people who choose self-managed storage (as opposed to an online storage service) use "hardware wallets" such as the Trezor (https://trezor.io/), Ledger (https://www.ledgerwallet.com/), and KeepKey (https://www.keepkey.com/) to store their crypto. These products can be setup in a way where multi-signature is required to move funds. While these are great products that provide strong security, CryptoGlacier is intended to offer an even higher level of protection than today's hardware wallets can provide.

The primary security consideration is that all hardware wallets today operate via a physical USB link to a regular computer. While they employ extensive safeguards to prevent any sensitive data (such as private keys) from being transmitted over this connection, it's possible that an undiscovered vulnerability could be exploited by malware to steal private keys from the device.

For details on this and other security considerations, see the "No Hardware Wallets" section of the design document. As with online multisig vaults, many people do use hardware wallets to store sizeable amounts of money. We personally recommend CryptoGlacier for large holdings, but ultimately it's a personal decision based on your risk tolerance and costs you're willing to pay (in money and time) for security.

1.2. Key concepts

Private Key

Your currency balances are effectively stored in crypto blockchains – global decentralized ledgers. You can imagine a locked box with all of your funds sitting inside of it. This box is unlocked with a piece of information known as "private key". (The boxes we'll be creating require multiple private keys to unlock; see the section "Multisignature Security" below.)

Unlike a password, a private key is not meant for you to remember. It's a long string of gibberish. The private key is what you need to keep secure. If anyone gets it, they can take your money. Unlike traditional financial instruments, there is no recourse. There is no company that is liable, because blockchains are decentralized systems not run by any person or entity. And no law enforcement agency is likely to investigate your case.

The protocol makes use of BIP39 (https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki) and BIP32 (https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki). This will allow us to deterministically derive keys across blockchains from a 24-word phrase. This 24-word phrase is more commonly known as a mnemonic phrase, mnemonic recovery phrase or mnemonic seed. It's a list of words which stores all the information needed to recover your private keys across protocols.

Offline Key Storage ("Cold Storage")

You don't want to store your private key on any computer that's connected to the Internet ("hot storage"), because that exposes it to more hacking attempts. There are viruses out there that search computers for private keys and steal them (thereby stealing your money).

One way to protect against this is by encrypting your private key, so even if a thief steals it, they can't read it. This helps, but is not foolproof. For example, a thief might install keylogger malware (https://en.wikipedia.org/wiki/Keystroke_logging) so that they steal your password too.

Online keys are inherently exposed to hackers. You therefore need to make sure your private key stays offline ("cold storage") at all times.

Paper Key Storage

Because the mnemonic phrase is a relatively small piece of information, it can be stored on paper as easily as it can be stored on a computer. And when it comes to key storage, paper has various advantages compared to computers: It's always offline (no chance of accidentally connecting it to the Internet!), it's easy & cheap to make multiple copies for backups, and it's not susceptible to mechanical failure.

1.3. Multi-signature security

Central to our security protocol is a technique called "multisignature security." You'll need a quick primer on this topic to understand the CryptoGlacier protocol.

Regular Private Keys are Risky

Remember that anybody with access to your private key can access your funds. And if you lose your private key, you cannot access your money; it is lost forever. There is no mechanism for reversal, and nobody to appeal to.

This makes it difficult to keep funds highly secure. For example, you might store a private key on paper in a safe deposit box at a bank, and feel fairly safe. But even this is not the most robust solution. The box could be destroyed in a disaster, or be robbed (perhaps via identity theft), or intentionally seized (http://abcnews.go.com/GMA/story?id=4832471).

You can try to mitigate these risks by storing the key yourself, perhaps in a fireproof home safe (as opposed to a bank). But this introduces new risks. A determined thief (perhaps a professional who brings safe-drilling tools on their burglary jobs, or who somehow got wind of the fact that you have a $100,000 slip of paper sitting in a safe) might break into the safe and steal the wallet.

Or a major natural disaster might prevent you from returning home for an extended period, during which time your safe is looted.

What is Multisignature Security?

To address these issues, most cryptocurrency protocols provide a way to secure funds with a set of private keys, such that some of the keys (but not necessarily all) are required to withdraw funds. For example, you might secure your funds with 3 keys but only need any 2 of those keys to withdraw funds. (This example is known as a "2-of-3" withdrawal policy.)

The keys are generated and controlled by different entities in different locations so someone who gets access to one key will not automatically have access to the others. Key custodians are called "signatories."

This approach of using multiple keys is known as "multisignature security." The "signature" part of "multisignature" comes from the process of using a private key to access funds, which is referred to as "signing a transaction." Multisignature security is analogous to a bank requiring signatures from multiple people (for example, any 2 of a company's 3 designated officers) to access funds in an account.

How Does Multisignature Security Help?

Multisignature security protects against the following scenarios:

- **Theft**: Even if somebody physically breaks into a safe, any one key is not enough to steal the money.
- **Loss**: If a key is destroyed or simply misplaced, you can recover your money using the remaining keys.
- **Key-man risk**: By having multiple signatories you significantly reduce the risk of loss of funds in case a signatory dies or becomes incapacitated. In the case of duress, a single-signatory is unable to access funds.
- **Unilateral access**: With multisignature security, signatories with a key will not be able to move funds (unless they steal additional key(s), or collude with additional signatories). This allows for more institutional decision-making.

Choosing a Multisignature Withdrawal Policy

An M-of-N Multisignature Withdrawal Policy will provide a way to enforce multi-person control over fund access. N signatories will generate a key each so that M < N of them (M of N) are needed in order to access funds, but no smaller group up to M -1 can do so.

For example, in a 1-of-2 setup, there would be two signatories and any one of them would be able to spend funds. In a 2-of-3 setup, there would be three signatories and at least two will be required to spend funds.

By choosing M < N, you give up control but gain redundancy in the event of key loss.

Depending on your use case, you will need to optimize choosing your M-of-N policy. You will need to select a policy before beginning the protocol.

Signatory responsibilities

Each signatory is responsible for securing their key in a safe deposit box. Signatories should make legal arrangements in advance so their key can be accessed in case of death or incapacitation. CryptoGlacier doesn't provide specific procedures to ensure this.

The most fail-safe way to ensure a signatory's agent will have access to a signatory's safe deposit box is to check with the bank. Standard estate planning legal documents should allow a signatory's agent to access the box upon a signatory's death or incapacitation. But banks can be fussy and sometimes prefer their own forms.

Choosing signatories

Consider the following when choosing signatories:

- **Availability**: If your signatory lives in a rural area, there may not be many vaults or safe deposit boxes that are practical to get to.
- **Privacy**: Signatories will have the ability to see account balances.
- **Signatory collusion**: Although possessing one key won't allow a signatory to access your funds, signatories might collude with each other to steal funds.
- **Signatory reliability**: A signatory may fail to store the key securely, or they may lose it.
- **Geography risk**: Signatories should be located in different physical locations to reduce the risk of loss of funds due to events that could affect wide geographical areas, like natural disasters or power losses which could potentially affect all your signatories' capability to sign transactions.
- **Jurisdiction risk**: Signatories should be located in different jurisdictions to reduce the risk of a coordinated fund seizure attack.
- **Signatory safety**: Giving your signatories custody of a valuable key may expose them to the risk of targeted physical theft.
- **Kidnapping risk**: If your signatories anticipate traveling in high-crime areas with kidnapping risk (http://www.nytimes.com/2012/05/03/business/kidnapping-becomes-a-growing-travel-risk.html), funds will be at greater risk because a signatory will have the ability to access them remotely (by contacting other signatories and asking for their keys). Financially-motivated kidnapping hinges on a signatory's ability to access funds to give to the kidnappers. If a signatory is literally unable to access additional funds (because there are duress protocols in place or signatories do not know of each other or don't have a way of contacting each other), kidnappers will have no incentive to hold a signatory.

1.4. Attack surface and failure points

This list describes the attack surface and other failure points for CryptoGlacier. We include only attacks and failures limited in scope to specific coins. Attacks and failures related to the broader crypto ecosystem as a whole (newly discovered cryptographic flaws, critical Bitcoin protocol security or scalability failures, etc.) are not included as most are equally likely to impact the value of all crypto whether or not they are secured with CryptoGlacier.

This list assumes no security measures from Extend CryptoGlacier security are implemented.

Most attacks require the presence of malware, either in or near the quarantined environment. We'll therefore inventory two layers of CryptoGlacier's attack surface:

- Ways in which a malware infection might occur
- Ways in which a critical failure might happen (possibly, but not necessarily, due to a malware infection)

Malware infection vectors

- Software
  - OS/App software has malware (i.e. malicious code) built into official distributions. In particular, CryptoGlacier relies on the following packages and their dependencies NOT to distribute malicious code:
    - Ubuntu desktop
    - zbar-tools (via Ubuntu Package archive)
    - qrencode (via Ubuntu Package archive)
    - libappindicator1 (via Ubuntu Package archive)
    - libindicator7 (via Ubuntu Package archive)
    - Electrum
    - Electrum-LTC
    - Electron-Cash
    - Gnosis' MultiSigWallet
    - nodejs (via Nodesource)
    - Several node packages (and their dependencies) are required for CryptoGlacierScript (via npm):
      - argparse
      - bip39
      - bitcoinjs-lib
      - enquirer
      - ethereumjs-tx
      - ethereumjs-wallet
      - js-sha256
      - lodash.clonedeep
      - ripple-bip32
      - ripple-keypairs
      - ripple-lib
      - ripple-sign-keypairs
      - wallet-address-validator

- Malware on Setup Computer infects Setup USB software AND malware on Setup USB infects Quarantined USB software AND checksum verifications produces false positives
  - Checksum false positives could happen because:
    - Malware might interfere with the verification process (or the display of its results).
    - The checksum verification software could be compromised.
      - Verifying the integrity of GnuPG requires one have access to a trusted installation of GnuPG, but many CryptoGlacier users won't have that. CryptoGlacier currently recommends users simply trust the version of GnuPG they download.
- Malware on Setup Computer infects OS/App USB software AFTER checksum verification produces a true positive (i.e. before/during copying of software to the USB, or during USB ejection)
- Firmware
  - Malware on Setup Computer infects Setup Boot USB firmware AND malware on Setup Boot USB infects Quarantined Boot/App USB
  - Laptop or USB firmware has malware in the shrinkwrapped package
- Hardware
  - Laptop or USB hardware has "malware" in the shrinkwrapped package. e.g. a USB JTAG exploit (http://www.itnews.com.au/news/intel-debugger-interface-open-to-hacking-via-usb-446889) or chip-level backdoors (such as this rootkit (https://www.wired.com/2016/06/demonically-clever-backdoor-hides-inside-computer-chip/)). "Malware" usually refers to software, but we're using it here more broadly to mean "computing technology which undermines the integrity of the computing environment in which it resides."

Failure scenarios

Electronic failures

- Exfiltration of critically sensitive data (e.g. private keys)
  - A Quarantined Computer leaks critically sensitive data over a side channel (https://en.wikipedia.org/wiki/Side-channel_attack) (possibly due to malware) AND complementary malware on a (networked or attacker-controlled) device in range steals the data
    - Visual side channel (does not require malware on the quarantined computer, since sensitive data is displayed on the screen as part of the protocol). If the protocol is followed, the attack surface here should be narrow, as users are instructed to block all visual side channels. However, at a minimum, they are using their smartphone for reading QR codes, and that has a camera on it.

- - - Acoustic side channel, if inadequately blocked (i.e. insufficient sound blockage or masking noise). See example (https://www.wired.com/2016/06/clever-attack-uses-sound-computers-fan-steal-data/).
    - Radio side channel (example 1 (https://www.usenix.org/legacy/event/sec09/tech/full_papers/vuagnoux.pdf) , example 2 (http://cyber.bgu.ac.il/content/how-leak-sensitive-data-isolated-computer-air-gap-near-mobile-phone-airhopper) , example 3 (https://www.wired.com/2015/06/radio-bug-can-steal-laptop-crypto-keys-fits-inside-pita/))
    - Seismic side channel (example (https://www.cc.gatech.edu/fac/traynor/papers/traynor-ccs11.pdf))
    - Thermal side channel (example (http://cyber.bgu.ac.il/blog/bitwhisper-heat-air-gap))
    - Magnetic side channel (example (http://fc15.ifca.ai/preproceedings/paper_14.pdf))
  - Malware on a Quarantined Computer exfiltrates critically sensitive data via QR codes AND cooperating malware on the QR reading device steals the data. The risk of this scenario is negligible; unless the attacker simultaneously compromised every major smartphone QR reader with cooperating malware, any manipulation of QR codes would be quickly detected by people using non-compromised QR reader software, leading to widespread awareness and isolation of the threat. This makes it a very unattractive attack vector.
  - Critically sensitive data is leaked (intentionally or otherwise) as part of the payload of valid data (e.g. if the nonce used for a transaction signature contains bits of the private key)
- Undetected generation of flawed sensitive data. (Requires compatible malware present on BOTH quarantined environments)
  - Private key creation is compromised to make keys easily guessable
  - Transaction creation is compromised to use output addresses belonging to an attacker, AND cooperating malware on a networked computer sends the malicious transaction before the manual address verification is done)

Physical failures

- M paper keys are stolen by an attacker
- (N-M+1) paper keys are lost or destroyed, making it impossible to achieve M signatories
- An attacker with physical line-of-sight to the laptop takes a photo of the screen while sensitive data is displayed
- Malware on the quarantined machines writes sensitive data to persistent media (USB or laptop hard drive) AND the hardware is physically stolen afterward

CryptoGlacier protocol failures

- CryptoGlacier hosting (i.e. DNS, Github, website hosting, etc.) is compromised to inject weaknesses into the protocol documentation or CryptoGlacierScript

- Protocol delivery is compromised (e.g. with a man-in-the-middle attack on the user's computer or network) to deliver or display a weakened version of the protocol documentation or software
- Protocol hardcopy is compromised (e.g. by malware to alter the user's hardcopy as it is printed)
- A flaw in CryptoGlacierScript causes sensitive data to be leaked or flawed
- Human error during protocol execution
- Design failure in the protocol misses or inadequately addresses a risk

For potential man-in-the-middle vulnerabilities, we mitigate this by signing a checksum of the CryptoGlacier document itself, and including steps in the protocol for users to verify the signature and checksum. But this is not foolproof:

An attacker could remove the self-verification procedure from the protocol document, and many users would not notice.

- An attacker could compromise the key-pair and create a fraudulent signature (although this is exceedingly unlikely, due to Keybase's key verification systems)
- The protocol document does begin with document self-verification on one Setup Computer. However, it doesn't guide the user through self-verification on the second Setup Computer. Nor does it have them re-verify the document when they first boot into Ubuntu on the Setup Computers to create the Quarantined Boot USBs. If the portion of the protocol document related to creating the Quarantined Boot USBs were compromised between the initial self-validation & the later re-validation (when creating the Quarantined App USBs), the user would probably not notice, even without a forged signature.
- Protocol hardcopy is compromised (e.g. by malware to alter the user's hardcopy as it is printed)
- A flaw in CryptoGlacierScript causes sensitive data to be leaked or flawed
- Human error during protocol execution
- Design failure in the protocol misses or inadequately addresses a risk

Gnosis MultiSigWallet Failure

The Gnosis MultiSigWallet is a smart contract deployed on the Ethereum network. Albeit being used to store significant funds, there is no guarantee that a flaw with the smart contract will not be found in the future. Funds in smart contracts have previously been compromised. Example one (https://www.coindesk.com/30-million-ether-reported-stolen-parity-wallet-breach), two (https://medium.com/swlh/the-story-of-the-dao-its-history-and-consequences-71e6a8a551ee), and three (https://mashable.com/2017/11/08/ethereum-parity-bug/) should serve as cautionary tales.

2. Before you start

2.1. Protocol overview

This section establishes a basic understanding of the CryptoGlacier protocol in order to facilitate its execution. For more background on the protocol's design, see the Glacier design document.

As previously described, the CryptoGlacier protocol is based on the Glacier protocol. It's aim is to put crypto funds in cold storage, using multisignature security, with keys generated and accessible by distinct signatories stored only on paper.

Eternally Quarantined Hardware

The bulk of the CryptoGlacier protocol consists of ways to safeguard against theft of private keys due to malware infection. To accomplish this, CryptoGlacier uses eternally quarantined hardware.

Quarantined hardware means we drastically limit the ways in which a piece of hardware interfaces with the outside world in order to prevent the transmission of sensitive data (e.g. private keys) or harmful data (e.g. malware). We consider all interfaces – network, USB, printer, and so on – because any of them might be used to transmit malware or private keys.

Eternally quarantined hardware means we use factory-new hardware for this purpose (to minimize risk of prior malware infection), and never lift the quarantine. The quarantine is permanent because any malware infection which does somehow get through the quarantine might wait indefinitely for an opportunity to use an available interface (e.g. the Internet, if a quarantined laptop is later used to access the web). Eternal quarantining renders the hardware essentially useless for anything else but executing this protocol.

Parallel Hardware Stacks

There is a class of attacks which rely not on stealing your sensitive data (e.g. private keys), but in subverting the process of generating your sensitive data so it can be more easily guessed by a third party. We call this "flawed data."

For example, a variant of the Trojan.Coinbitclip (https://www.symantec.com/security-center/writeup/2016-020216-4204-99) attack which replaces keys displayed on your screen (or keys stored in your clipboard) with insecure keys.

Because we are generating our data in eternally quarantined environments, any malware infection attempting this is unlikely to have come from your other computers – it would likely have already been present when the quarantined system arrived from the manufacturer. For example, the Lenovo rootkit or this Dell firmware malware infection.

The way to defeat these attacks is to detect them before we actually use the flawed data. We can detect such an attack by making sure each signatory replicates the entire data generation process on two sets of eternally quarantined hardware, from different manufacturers. If the process generates identical data on both sets of hardware, we can be highly confident the data is not flawed because it would have to be an identical attack present on both sets of hardware, factory-new from different manufacturers. This is exceptionally unlikely.

Electrum (and forks), Gnosis MultiSigWallet and CryptoGlacierScript

The original Glacier Protocol relies on Bitcoin Core (https://bitcoincore.org/) for all cryptographic and financial operations because "its open source code is the most trustworthy… due to its track record of securing large amounts of money for many years, and the high degree of code review scrutiny it has received." Unfortunately Bitcoin Core only supports Bitcoin transactions and it lacks support for BIP39 mnemonics.

CryptoGlacier relies on Electrum (https://electrum.org) for Bitcoin transactions, Electrum-LTC (https://electrum-ltc.org) (a fork of Electrum) for Litecoin transactions, Electron-Cash (https://electroncash.org/) for Bitcoin Cash transactions and Gnosis MultiSignatureWallet (https://github.com/gnosis/MultiSigWallet) for Ethereum and ERC20 transactions. Although none of these projects have received the amount of code review scrutiny that Bitcoin Core has received, they are all popular open-source projects with significant contributions and code reviews.

CryptoGlacier also utilizes CryptoGlacierScript, a software program that automates much of the manual work involved in executing the protocol. It is also used for all cryptographic and financial operations for XRP transactions. CryptoGlacierScript's open source code (https://github.com/vogelito/CryptoGlacierProtocol) aims to be straightforward and be extensively commented to facilitate easy review for flaws or vulnerabilities.

Protocol Output

The end result of the CryptoGlacier protocol is two sets of paper information packets per signatory. A private packet with the mnemonic phrase (which will be created by each signatory independently) and a packet which will be shared among signatories (which will be created by a designated signatory and distributed to the rest).

Each signatory will have a **private and never to be shared** package containing:

- A **24-word seed mnemonic** based on BIP39 – these 24 words will be used to derive the private keys that will provide one of the signatures needed to move your funds.

There will also be a package that will need to be shared with the other signatories containing:

- 3xN **Master Public Keys** – each is an alphanumeric string used in the multisignature protocol of Bitcoin, Litecoin, and Bitcoin Cash.
- N **Ethereum account addresses** – each is a hex string designating the Ethereum account to be used as a signatory when setting up the Gnosis MultiSignatureWallet.
- N **Ripple addresses** – each is an alphanumeric string designating the XRP account to be used as a signatory when setting up the Ripple Multi-Signing account.
- The **Ethereum Multisig Smart Contract Address** – a hex string designating the Ethereum contract deployed using the Gnosis MultiSignatureWallet.
- The **XRP Multisign Account** – an alphanumeric string designating the Multi-Signing XRP account.

Technical details: The CryptoGlacier protocol reuses Ethereum and Ripple addresses but uses HD wallets for Bitcoin, Bitcoin Cash, and Litecoin.

Protocol Cost

The CryptoGlacier protocol requires over $600 in equipment per key, and approximately 12 hours of work divided in two different sessions to perform an initial cold storage deposit. This excludes time for:

- Obtaining equipment
- Printing documents
- Downloading files
- Physically storing the resulting keys

Subsequent deposits and withdrawals re-use the same equipment and take a fraction of the time.

No Formal Support

As a free, volunteer-developed community project, there is no formal support channel for CryptoGlacier should you encounter any issues. However, you may be able to ask advice from Glacier's community members on their Gitter chat room (https://gitter.im/glacierprotocol/Lobby) or other Bitcoin or crypto community forums.

Privacy Considerations

Because blockchains are public, the way you route and store funds has privacy implications. For example, any person to whom you give your cold storage address for Ripple or Ethereum (because, for example, they're sending you funds which you want to keep in cold storage) can see your total cold storage balance. This is easy to do with many free services (e.g. Etherscan (https://etherscan.io/)).

This is true not just of individuals, but entities. That is, any online wallet service which you use to send funds to cold storage can see your cold storage balance, and may deduce that it belongs to you. They may, of course, also choose to share this information with others.

If this is a concern for you, the easiest way to keep your cold storage balance private from a particular entity is to route the payment through one (or more) intermediary addresses before sending it to your cold storage address, with a few transactions going to each intermediate address. This does not provide perfect privacy, but each intermediate address provides increasing levels of obfuscation and uncertainty.

CryptoGlacier utilizes HD wallets for Bitcoin, Litecoin, and Bitcoin Cash, thereby improving privacy over the original Glacier protocol.

If privacy is very important to you, you might consider using a service like Shapeshift (https://shapeshift.io/#/coins) to exchange your Bitcoins for a more anonymous cryptocurrency, such as Monero (http://monero.org/), and then exchange them back to Bitcoins. However, this will cost you fees, and more importantly, it requires you trust the operator of the exchange service not to steal or lose your funds.

This guide (https://bitcoinnewsmagazine.com/how-to-use-monero-to-anonymize-bitcoin/) gives additional detail about how to increase Bitcoin anonymity using Monero & Tor.

Lower-security Protocol Variants

If you are willing to accept lower security for lower cost, you can do so with only slight modifications:

1. **Perform this protocol using only one quarantined computer**. CryptoGlacier protocol repeats all operations on two computers per signatory to detect defects or tampering in the key generation process. However, this is costly and adds significantly to the labor required to execute the protocol. The risks it mitigates are small: that malware conducting flawed key-generation attacks found its way onto the eternally quarantined systems, or that the computer firmware was tampered with at the manufacturer to include such malware. If you are willing to accept this risk, you could skip buying the parallel hardware stack (and needing the second setup computer) and skip the process of re-generating and verifying keys & transactions on the parallel hardware stack.

2. **Use existing hardware**. An even lower-security variant is to use nothing but existing laptops you already possess, disabling all network connections during protocol execution, instead of purchasing new quarantined hardware. This fails to protect against some malware attacks, but provides additional savings in cost and effort. Such as an existing infection of a laptop's firmware (https://www.youtube.com/watch?v=sNYsfUNegEA), malware which overrides OS settings to disable wireless connectivity, or certain undiscovered vulnerabilities in the software used by the protocol.

These modifications are left as an exercise to the reader.

Out of scope

There's always more one could do to increase security. While CryptoGlacier is designed to provide strong protection for almost everyone, some situations (e.g. being the focus of a targeted attack by a sophisticated, well-resourced criminal organization) are beyond its scope.

For some additional security precautions beyond those provided in the standard protocol, see the possible improvements to CryptoGlacier.

2.2. Hardware required

Glacier has been written and tested around these specific equipment recommendations. We have used similar equipment recommendations for CryptoGlacier.

Each signatory will need the following equipment:

Eternally quarantined hardware: Set 1

- Factory-sealed computer with 2 USB ports and a camera: 2016 Dell Inspiron 11.6" (http://a.co/1E6HEQA)
- Two factory-sealed USB drives (2GB+) from the same manufacturer: SanDisk Cruzer 8GB (http://a.co/1Us66ze).

We'll be using two USB drives at the same time. If the computer has only one USB port, you'd need to use a USB hub, which is a separate piece of USB hardware subject to malware infection of its firmware.

We'll use the camera for reading QR codes.

Eternally quarantined hardware: Set 2

- Factory-sealed computer from a different manufacturer, also with 2 USB ports and a camera: Acer Aspire One Cloudbook 11" (http://a.co/1ZMSB3Y)
- Two factory-sealed USB drives (2GB+) from the same manufacturer, but a different manufacturer than the drives for Set 1: Verbatim 2GB (http://a.co/jdzEf8O)

Used/existing computing equipment

- Two computers with Internet connectivity, administrator access, at least 4GB RAM, and about 2GB of free disk space. **Each computer must be running Windows 10, macOS, or Linux.** At the time of writing there is poor support for Ubuntu in newer generation Macs (starting with 2016 models). We do not recommend you use a Mac device released after 2015.

One of these two computers should be a computer that you do not own (unless purchased brand new), or that has spent much time on your home or office network.

- Printer
- Smartphone with a working camera

Other Equipment

- Two factory-sealed USB drives (2GB+): Verbatim 2GB (http://a.co/jieluaE)
- Precision screwdrivers (http://a.co/bbvj16a), for removing WiFi cards from laptops
- Electrical tape (http://a.co/gZZiEdA)
- Casino-grade six-sided dice (http://a.co/ghbdiak). Regular dice are insufficient.
- Faraday bag (http://a.co/3wiNPLT). Used to prevent smartphone malware from stealing sensitive data using radio frequencies (https://www.usenix.org/legacy/event/sec09/tech/full_papers/vuagnoux.pdf).
- Table fan (http://a.co/98PrpMs). White noise can prevent malware on nearby devices from stealing sensitive data using sound (https://www.wired.com/2016/06/clever-attack-uses-sound-computers-fan-steal-data/).
- Home safe (http://a.co/6sRoaPv). Consider bolting it to your floor to deter theft.
- TerraSlate paper (http://a.co/7pk5fJN). Waterproof, heat resistant, and tear-resistant.
- Cardboard envelopes (http://a.co/7jUPLMR), for opacity
- Tamper-resistant seals (http://a.co/96KlsAl)

Notes

Standard software algorithms that generate random numbers, such as those used to generate Bitcoin private keys, are vulnerable to exploitation (https://bitcoin.org/en/alert/2013-08-11-android), either due to malware or algorithmic weakness (i.e. they often provide numbers that are not truly random). Dice offer something closer to true randomness.

Casino dice are created specifically to remove any potential dice bias (square corners, filled in pips, low manufacturing tolerance, etc.) That's why casinos use them!

TerraSlate paper is extremely rugged, but you might also consider laminating the paper for additional protection. You'll need a thermal laminator (http://a.co/cZBN1YU) and laminating pouches (http://a.co/ifISzje).

Jameson Lopp has also done extreme stress testing (https://medium.com/@lopp/metal-bitcoin-seed-storage-stress-test-part-ii-d309e04aefeb) for metal seed storage devices.

2.3. Protocol structure

The overall CryptoGlacier protocol consists of several distinct subprotocols:

- **Setup**: Prepares hardware, and downloads and verifies needed software & documentation.
- **Key Generation**: Private keys are generated..
- **Multisig Account Creation**: Multisig wallets, contracts, and accounts are setup.
- **Deposit**: For securely storing crypto.
- **Withdrawal**: For transferring some or all of your stored funds to another crypto address.
- **Viewing**: For viewing the balance of your funds in secure storage.
- **Maintenance**: For ensuring funds in cold storage remain accessible and secure.

Sensitive Data

Critically-sensitive data (e.g. private keys) will be highlighted in red, like this: `critically-sensitive-data-here`.

Critically sensitive data can be used by thieves to to steal your assets. If you follow the protocol precisely, your critically sensitive data will remain secure.

Do *not* do anything with critically sensitive data that the protocol does not specifically instruct you to. In particular:

- Never send it over email or instant messenger
- Never save it to disk (hard drive, USB drive, etc.)
- Never paste or type it into any non-eternally-quarantined device
- Never take a picture of it
- Never let any untrusted person see it

Moderately-sensitive data (e.g. a cold storage addresses or redemption script) will be highlighted in yellow, like this: `moderately-sensitive-data-here`.

Moderately sensitive data impacts privacy, but does not directly impact security. It cannot be used to steal your funds, but it *can* be used to see how much crypto you own (if someone knows that the moderately sensitive data in question belongs to you).

It does indirectly impact security, in that if someone knows you own a lot of difficult-to-trace money, they have some incentive to rob, extort, or attack you to get it.

The protocol recommends storing copies of moderately-sensitive data electronically, in a "conventionally secure" manner (for example, in a password manager such as 1Password (https://1password.com/)).

This means that knowledge of your cold storage balances will be as secure as access to any accounts which have their credentials stored in your password manager. For most people, this is sufficient.

If you use only hardcopies, you'll need to manually type in a large amount of gibberish data, by hand, with no errors, every time you withdraw funds from cold storage.

Terminal Usage

Many protocol steps involve typing commands into a *terminal window*. Working in a terminal window is analogous to working under the hood of a car. It allows you to give the computer more precise commands than you can through the regular interface.

Commands to be entered into a terminal window will be displayed in a fixed-width font like this:

```
$ echo "everything after the $ could be copy-pasted into a terminal window"
```

The `$` at the beginning of the line represents a *terminal prompt*, indicating readiness for user input. The actual prompt varies depending on your operating system and its configuration; it may be `$` , `>` , or something else. Usually the terminal will show additional information (such as a computer name, user ID and/or folder name) preceding every prompt.

In the above example, the text splits across two lines because of the margins of this document. Each line is *not* a separate command; it is all one command, meant to be entered all at once. This is clear because there is no terminal prompt at the beginning of the second line. Proceed Carefully

If you encounter **anything that is different** from what the protocol says you should expect, **the recommendation is that you stop and seek help** unless your expert opinion gives you high confidence that you understand all possible causes and implications of the discrepancy.

In general, follow the protocol carefully, keep track of what step you are on, and double-check your work. Any errors or deviations can undermine your security.

3. Setup

3.1. Verify and print protocol document

The Setup Protocol is used to prepare hardware, and download and verify needed software & documentation.

The first thing we need to do is verify the integrity of the CryptoGlacier protocol document (the one you are reading) to ensure that it has not been tampered with. After verifying the document, we'll print a hardcopy.

Printing is important, because a verified electronic copy will not be accessible at all times during protocol execution due to reboots and other changes to the computing environment. Printing a hardcopy ensures there is always a verified copy of the document available.

Each signatory will need to do the following:

1. Find a computer which has Internet access, printer access, and which you have permission to install new software on. We'll refer to this computer as the "SETUP 1" computer.
2. Review the errata for the version of Glacier you are using at https://github.com/vogelito/ CryptoGlacierProtocol/releases (https://github.com/vogelito/CryptoGlacierProtocol/releases).
3. Download the latest full release of CryptoGlacier (*not* just the protocol document) at https:// github.com/vogelito/CryptoGlacierProtocol/releases (https://github.com/vogelito/ CryptoGlacierProtocol/releases).
4. If your browser does not automatically extract the ZIP file contents into a folder within your downloads directory, do so.
5. Rename the folder to "cryptoglacier".
6. If you have used CryptoGlacier before, *and* you know you have the CryptoGlacier public key imported into a local GPG keyring, skip the next step. (If you don't know, that's fine; proceed as normal.)
7. Obtain the CryptoGlacier "public key," used to cryptographically verify the protocol document.

   If you are ever using CryptoGlacier in the future and notice that this step has changed (or that this warning has been removed), there is a security risk. Stop and seek assistance.

   a. Access CryptoGlacier's Keybase profile at https://keybase.io/vogelito (https://keybase.io/ vogelito).
   b. Click the string of letters and numbers next to the key icon.
   c. In the pop-up that appears, locate the link reading "this key".
   d. Right-click the link and select "Save Link As…" or "Download Linked File As…"
   e. Name the file "cryptoglacier.asc".

8. Download and install GnuPG (https://gnupg.org/), the software we'll use for doing the cryptographic verification. See tech details.

    a. **Windows**: Download and install the latest available version of Gpg4win (https://www.gpg4win.org/). Use the default options.

    b. **macOS**: Download and install the latest available version of GPG Suite (https://gpgtools.org/).

    c. **Linux**: GnuPG comes pre-installed with Linux distributions.

9. Open a terminal window:

    a. **Windows**: Press Windows-R, type "powershell" and click OK.

    b. **macOS**: Click the Searchlight (magnifying glass) icon in the menu bar, and type a terminal window. "terminal". Select the Terminal application from the search results.

    c. **Linux**: Varies; on Ubuntu, press Ctrl-Alt-T.

10. Change the terminal window's active folder to your downloads folder. The commands below are based on common default settings; if you put your downloads is in a different place, you will need to customize this command.

    a. **Windows**: `> cd $HOME/Downloads/cryptoglacier`

    b. **macOS**: `$ cd $HOME/Downloads/cryptoglacier`

    c. **Linux**: `$ cd $HOME/Downloads/cryptoglacier`

11. Verify the integrity of the downloaded document.

    a. Import the CryptoGlacier public key into your local GPG installation:

    ```
    $ gpg --import $HOME/Downloads/cryptoglacier.asc
    ```

    b. Use the public key to verify that the Glacier "fingerprint file" is legitimate:

    ```
    $ gpg --verify SHA256SUMS.sig SHA256SUMS
    ```

    Expected output (timestamp will vary, but e-mail and fingerprint should match):

    ```
    gpg: Signature made Thu Jun 20 18:01:31 2019 CDT
    gpg:                using RSA key 3378240146B53C307FBA4B0D97F10485CCBACA30
    gpg: Good signature from "Daniel Vogel <vogel@bitso.com>" [unknown]
    gpg:                 aka "Daniel Vogel <dvogel@cs.stanford.edu>" [unknown]
    gpg: WARNING: This key is not certified with a trusted signature!
    gpg:          There is no indication that the signature belongs to the owner.
    Primary key fingerprint: 3378 2401 46B5 3C30 7FBA  4B0D 97F1 0485 CCBA CA30
    ```

    The warning message is expected, and is not cause for alarm.

c. Verify the fingerprints in the fingerprint file match the fingerprints of the downloaded Glacier files.

    i. On Linux or Mac:

        Linux: `$ sha256sum -c SHA256SUMS 2>&1`

        Mac: `$ shasum -a 256 -c SHA256SUMS 2>&1`

        Expected output:

```
CryptoGlacier.pdf: OK
README.md: OK
mnemonic_entropy.py: OK
package.json: OK
package-lock.json: OK
setup.js: OK
```

    ii. On Windows 10:

```
> Get-FileHash -a sha256 CryptoGlacier.pdf
> cat SHA256SUMS | select-string -pattern "CryptoGlacier.pdf"
```

        Ensure that the hash output from the first command matches the output by the second command. Upper/lower case doesn't matter.

d. If you do not see the expected output, your copy of the document has not been verified. Stop and seek assistance.

12. Switch to use the new document.

    a. Open the version of the document that you just verified.

    b. Close this window (of the unverified version of the document you had been using).

    c. Delete the old, unverified copy of the document.

13. Print the verified document.

You are strongly encouraged to use the printed copy as a checklist, physically marking off each step as you complete it. This reduces the risk of execution error by ensuring you don't lose your place.

3.2. Prepare non-quarantined hardware

Each signatory will need to do the following:

1. Select two (2) computers which will be used as "Setup Computers" to set up USB drives.

    a. Both Setup Computers must have Internet access.
    b. You should have administrator access to both Setup Computers.
    c. Importantly, at least one computer should be a computer that you *do not* own, or that doesn't spend much time on your home or office network.

       It's not technically ownership that's important. But computers you own are more likely to run the same software, have visited the same websites, or have been exposed to the same USB drives or networks – and therefore to have the same malware.

2. Using sticky notes, label the two Setup Computers "SETUP 1" and "SETUP 2".
3. With a permanent marker, label two USB drives "SETUP 1 BOOT" and "SETUP 2 BOOT".

    a. Remember that, per the equipment list, each signatory should have 4 remaining USB drives – two from one manufacturer, and two from a *different* manufacturer.

4. Run a virus scan on the Setup Computers. If you don't have virus scanning software installed, here are some options:

    ◦ Windows: Kaspersky (https://usa.kaspersky.com/) ($39.99/yr), Avira (https://www.avira.com) (Free)
    ◦ macOS: BitDefender (https://www.bitdefender.com/) ($59.95/yr), Sophos (https://home.sophos.com/) (Free)
    ◦ Linux: Unnecessary

5. If the virus scan comes up with any viruses, take steps to remove them.
6. Once you have a clean virus scan, your Setup Computers are ready.

3.3. Prepare quarantined hardware

Each signatory should do the following:

1. Separate your quarantined hardware into two parallel sets. Each set should contain:
    ◦ One laptop
    ◦ Two USB drives from the same manufacturer

Each component should be supplied by *different* manufacturers from the other set. I.e. your two laptops should be from two different manufacturers, and the USB drives in one set should be from a different manufacturer than the USB drives in the other set.

2. In each set, label all hardware with a permanent marker. Write directly on the hardware.

   a. Label the laptops ("Quarantined Computers") "Q1" and "Q2".
   b. Label one USB drive from each set with "Q1 BOOT" or "Q2 BOOT". These USBs will have the operating system you'll boot the computer with.
   c. Label the other USB drive from each set with "Q1 APP" or "Q2 APP". These USBs will have the software applications you'll use.

3. **Labeled hardware should only be used with hardware that shares the same label ("Q1", "Q2", or "SETUP 1", or "SETUP 2").** For example:

   a. **Don't** plug a **"Q1"** USB drive into a **"Q2"** laptop.
   b. **Don't** plug a **"SETUP 2"** USB drive into a **"Q1"** or **"Q2"** laptop.
   c. **Don't** plug an **unlabeled** USB drive into a **"Q1"** or **"Q2"** laptop.

4. Quarantine the network and wireless interfaces for both laptops:

   a. Unbox laptop. Do **not** power it on.
   b. Put a tamper-resistant seal (https://www.amazon.com/Security-Warranty-Hologram-Sequential-Numbering/dp/B0051JNB6A/ref=sr_1_1?ie=UTF8&qid=1471760406&sr=8-1&keywords=tamper+resistant+stickers) over the Ethernet port, if it has one.
   c. Physically remove the wireless card.

      i. For the recommended Dell laptop, Dell's official instructions for doing so are here (http://topics-cdn.dell.com/pdf/inspiron-11-3162-laptop_Service%20Manual_en-us.pdf). A YouTube video showing an abbreviated procedure is here (https://www.youtube.com/watch?v=nFYXQQPoh90).
      ii. For the recommended Acer laptop, the process is similar to the Dell. Note there are two cover screws hidden underneath rubber feet on the bottom of the laptop.
      iii. For any other models, use YouTube to learn how to remove the wireless card.

   d. After removing the wireless card, cover the ends of the internal WiFi antennae with electrical tape.
   e. If the computer has separate cards for WiFi and Bluetooth, be sure to remove both. (Most modern laptops, including the recommended Acer and Dell, have a single wireless card which handles both.)

5. Fully charge both laptops.

3.4. Create boot USBs

Because the eternally quarantined computers cannot connect to a network, they cannot download software. We'll be using USB drives to transfer the necessary software to them.

Each signatory should do the following:

We will prepare four bootable Ubuntu (https://en.wikipedia.org/wiki/Ubuntu_(operating_system)) USB drives. ("Bootable" means that the Ubuntu operating system will be booted directly from the USB drive, without using the computer's hard drive in any way.)

The *first two* USB drives ("Setup Boot USBs") are the USB drives you labeled "SETUP 1 BOOT" and "SETUP 2 BOOT" in Section II. They will be prepared using your Setup Computers, which may be running Windows, macOS, or something else.

The *last two* USB drives ("Quarantined Boot USBs") are the USB drives you labeled "Q1 BOOT" and "Q2 BOOT" in Section II. They will be prepared using your Setup Computers *while booted off* a *Setup Boot USB*.

Technical details: The Non-Quarantined OS USBs serve two purposes:

- First, they are used for creating the Quarantined App USBs in the next section, which greatly simplifies the process of doing so because we know it'll always be done from an Ubuntu environment. (We can't use the Quarantined OS USBs for this – they're eternally quarantined, so they need to be permanently unplugged from their Setup Computer the moment they are created.)
- Second, it will be harder for any malware infections on a Setup Computer's default OS to undermine a Quarantined USB setup process (the malware would first have to propagate itself to the Non-Quarantined OS USB).

1. Perform the following steps on your SETUP 1 computer.
2. If you are not already reading this document on the SETUP 1 computer, open a copy there.
3. Download Ubuntu by going to this link:

   http://old-releases.ubuntu.com/releases/xenial/ubuntu-16.04.1-desktop-amd64.iso (http://old-releases.ubuntu.com/releases/xenial/ubuntu-16.04.1-desktop-amd64.iso) Wait until the download is complete.

4. Open a terminal window.

   a. **Windows**: Press Windows-R, type "powershell" and click OK.
   b. **macOS**: Click the Searchlight (magnifying glass) icon in the menu bar, and type "terminal". Select the Terminal application from the search results.
   c. **Linux**: Varies; on Ubuntu, press Ctrl-Alt-T. (On Ubuntu, press Ctrl-Alt-T.)

5. Verify the integrity of the Ubuntu download.

    a. Change the terminal window's active folder to the folder where you downloaded Ubuntu, customizing the folder name if necessary:

        i. **Windows**: `> cd $HOME/Downloads`

        ii. **macOs**: `$ cd $HOME/Downloads`

        iii. **Linux**: `$ cd $HOME/Downloads`

    b. View the fingerprint of the file:

        i. **Windows**: `> Get-FileHash -a sha256 ubuntu-16.04.1-desktop-amd64.iso`

        ii. **macOs**: `$ shasum -a 256 ubuntu-16.04.1-desktop-amd64.iso`

        iii. **Linux**: `$ sha256sum ubuntu-16.04.1-desktop-amd64.iso`

    c. The following fingerprint should be displayed:

```
dc7dee086faabc9553d5ff8ff1b490a7f85c379f49de20c076f11fb6ac7c0f34
```

It's not important to check every single character when visually verifying a fingerprint. It's sufficient to check the **first 8 characters, last 8 characters, and a few somewhere in the middle.**

Technical details: Because you verified the checksum & checksum signature for this document in Section I, we are omitting the GPG verification of some other fingerprints in the protocol. For a detailed security analysis, see the design document.

You can verify this is the official Ubuntu fingerprint here (http://releases.ubuntu.com/16.04.1/SHA256SUMS), or follow Ubuntu's full verification process using this guide.

6. Create the SETUP 1 BOOT USB.

    a. **Windows**

        i. Download the Rufus disk utility (https://rufus.akeo.ie/) and run it.

        ii. Insert the SETUP 1 BOOT USB in an empty USB slot.

        iii. In the "Device" dropdown at the top of the Rufus window, ensure the empty USB drive is selected.

        iv. Next to the text "Create a bootable disk using", select "ISO Image" in the dropdown.

        v. Click the CD icon next to the "ISO Image" dropdown.

        vi. A file explorer will pop up. Select `ubuntu-16.04.1-desktop-amd64.iso` from your downloads folder and click Open.

        vii. Click Start.

        viii. If prompted to download Syslinux software, click "Yes".

        ix. When asked to write in "ISO Image Mode (Recommended)" or "DD Image Mode", select "ISO Image Mode" and press OK.

x. The program will take a few minutes to write the USB.

b. **macOS**

    i. Prepare the Ubuntu download for copying to the USB.

```
$ cd $HOME/Downloads
$ hdiutil convert ubuntu-16.04.1-desktop-amd64.iso -format
UDRW -o ubuntu-16.04.1-desktop-amd64.img
```

    ii. Determine the macOS "device identifier" for the Boot USB.

        1. `$ diskutil list`

        2. Insert the SETUP 1 BOOT USB in an empty USB slot.

        3. Wait 10 seconds for the operating system to recognize the USB.

        4. Once more: `$ diskutil list`

        5. The output of the second command should include an additional section that was not present in the first command's output.

            i. This section will have (external, physical) in the header.

            ii. The first line of the section's SIZE column should reflect the capacity of the USB drive.

        6. Make a note of the device identifier.

            i. The device identifier is the part of the new section header that comes before (external, physical) (for example /dev/disk2).

    iii. Put Ubuntu on the SETUP 1 BOOT USB.

        1. First, unmount the USB

```
$ diskutil unmountDisk USB-device-identifier-here
```

        2. Enter the following command, **making sure to use the correct device identifier; using the wrong one could overwrite your hard drive!**

```
$ sudo dd if=ubuntu-16.04.1-desktop-amd64.img.dmg \
of=USB-device-identifier-here bs=1m
```

        Example:

```
$ sudo dd if=ubuntu-16.04.1-desktop-amd64.img.dmg \
of=/dev/disk2 bs=1m
```

        3. Enter your administrator password when requested.

        4. Wait several minutes for the copying process to complete. When it does, you may see an error box pop up. This is expected; it's because the USB is written in a format readable by Ubuntu, but not readable by macOS.

        5. Click Ignore.

iv. Verify the integrity of the SETUP 1 BOOT USB (i.e. no errors or malware infection).

1. Remove the USB drive from the USB slot and immediately reinsert it.

2. Wait 10 seconds for the operating system to recognize the USB.

3. You may see the same error box pop up again. Select Ignore.

4. The USB's device identifier may have changed. Find it again:

```
$ diskutil list
```

5.
```
$ cd $HOME/Downloads
```

6.
```
$ sudo cmp -n `stat -f '%z' \
ubuntu-16.04.1-desktop-amd64.img.dmg` \
ubuntu-16.04.1-desktop-amd64.img.dmg \
USB-device-identifier-here
```

7. Wait a few minutes for the verification process to complete.

8. If all goes well, the command will output no data, returning to your usual terminal prompt.

9. If there is a discrepancy, you'll see a message like:

```
ubuntu-16.04.1-desktop-amd64.img.dmg /dev/disk2
differ: byte 1, line 1
```

If you see a message like this, STOP – this may be a security risk. Restart this section from the beginning. If the issue persists, try using a different USB drive or a different Setup Computer.

c. **Ubuntu**

i. If this is your first time using Ubuntu, note:

1. You can copy-paste text in most applications (e.g. Firefox) by pressing **Ctrl-C** or **Ctrl-V**.

2. You can copy-paste text in a *terminal window* by pressing **Ctrl-Shift-C** or **Ctrl-Shift-V**.

ii. Put Ubuntu on the SETUP BOOT 1 USB.

1. Open the Ubuntu search console by clicking the purple circle/swirl icon in the upper-left corner of the screen.

2. Type "startup disk creator" in the text box that appears

3. Click on the "Startup Disk Creator" icon that appears.

4. The "Source disc image" panel should show the `.iso` file you downloaded. If it does not, click the "Other" button and find it in the folder you downloaded it to.

5. In the "Disk to use" panel, you should see two lines. They may vary from system to system, but each line will have a device identifier in it, highlighted in the example below.

```
Generic Flash Disk (/dev/sda)
Kanguru Flash Trust (/dev/sdb)
```

6. Select the line containing SETUP 1 BOOT USB and make note of the disk identifier (e.g. /dev/sdb).

7. Click "Make Startup Disk" and then click "Yes".

8. Wait a few minutes for the copying process to complete.

iii. Verify the integrity of the SETUP 1 BOOT USB (i.e. no errors or malware

1. On your desktop, right-click the corresponding USB drive icon in your dock and select Eject from the pop-up menu.

2. Remove the USB drive from the USB slot and immediately re-insert it.

3. Wait 10 seconds for the operating system to recognize the USB.

4.
```
$ cd $HOME/Downloads
```

5.
```
$ sudo cmp -n `stat -c '%s' \
ubuntu-16.04.1-desktop-amd64.iso` \
ubuntu-16.04.1-desktop-amd64.iso USB-device-identifier-here
```

6. If prompted for a password, enter the computer's root password.

7. Wait a few minutes for the verification process to complete.

8. If all goes well, the command will output no data, returning to your usual terminal prompt.

9. If there is an issue, you'll see a message like:

```
ubuntu-16.04.1-desktop-amd64.iso /dev/sda differ:
byte 1, line 1
```

If you see a message like this, STOP – this may be a security risk. Restart this section from the beginning. If the issue persists, try using a different USB drive or a different Setup Computer.

7. Create the Q1 BOOT USB **We do not recommend you boot Ubuntu on Mac devices released after 2015. Ubuntu support for newer generation Macs and vice-versa is still very poor at the time of writing and you might find your mouse, keyboard, or network to be unusable.**

    a. Boot the SETUP 1 computer from the SETUP 1 BOOT USB.

        i. Reboot the computer.

        ii. Press your laptop's key sequence to bring up the boot device selection menu. (Some PCs may offer a boot device selection menu; see below.)

            1. **PC**: Varies by manufacturer, but is often **F12** or **Del**. The timing may vary as well; try pressing it when the boot logo appears.

                i. On the recommended Dell laptop, press F12. You should see a horizontal blue bar appear underneath the Dell logo.

                ii. The recommended Acer laptop does not have a boot menu. See below for instructions.

            2. **Mac**: When you hear the startup chime, **press and hold Option (⌥).**

        iii. Select the proper device to boot from.

            1. **PC**: Varies by manufacturer; option will often say "USB" and/or "UEFI".

                i. On the recommended Dell laptop, select "USB1" under "UEFI OPTIONS".

                ii. The recommended Acer laptop does not have a boot menu. See below for instructions.

            2. **Mac**: Click the "EFI Boot" option and then click the up arrow underneath it.

              You do not need to select a network at this time. If more than one identical "EFI boot" option is shown, you may need to guess and reboot if you pick the wrong one.

        iv. Some laptops don't have a boot device selection menu, and you need to go into the BIOS configuration and change the boot order so that the USB drive is first.

            1. On the recommended Acer laptop:

                i. Press F2 while booting to enter BIOS configuration.

                ii. Navigate to the Boot menu.

                iii. Select USB HDD, and press F6 until it is at the top of the list.

                iv. Press F10 to save and automatically reboot from the USB.

        v. If the computer boots into its regular OS rather than presenting you with a boot device or BIOS configuration screen, you probably pressed the wrong button, or waited too long.

            1. Hold down your laptop's power button for 10 seconds. (The screen may turn black sooner than that; keep holding it down.)

            2. Turn the laptop back on and try again. Spam the appropriate button(s) repeatedly as it boots.

3. If the computer boots *immediately* to where it left off, you probably didn't hold down the power button long enough.

vi. You'll see a menu that says "GNU GRUB" at the top of the screen. Select the option "Try Ubuntu without installing" and press Enter.

vii. The computer should boot into the USB's Ubuntu desktop.

b. Enable WiFi connectivity.

i. Click the cone-shaped WiFi icon near the right side of the menu bar.

ii. If the dropdown says "No network devices available" at the top, you need to enable your networking drivers:

1. Click on "System Settings". It's the gear-and-wrench icon along the left side of the screen.

2. A System Settings window will appear. Click the "Software & Updates" icon.

3. A Software & Updates window will appear. Click the "Additional Drivers" tab.

4. In the Additional Drivers tab, you'll see a section for a Wireless Network Adapter. In that section, "Do not use the device" will be selected. Select any other option besides "Do not use the device.""

5. Click "Apply Changes".

6. Click the cone-shaped WiFi icon near the right side of the menu bar again. There should be a list of WiFi networks this time.

iii. Select your WiFi network from the list and enter the password.

c. Repeat steps 1-6 using the SETUP 1 computer to create the Q1 BOOT USB rather than the SETUP 1 BOOT USB.

i. **The instruction to plug a Quarantined Boot USB into your Setup computer should raise a red flag for you, because you should never plug a quarantined USB into anything other than the quarantined computer it is designated for!**

This setup process is the ONE exception.

ii. Because you have booted the SETUP 1 computer off the SETUP 1 BOOT USB, you will follow the instructions for Ubuntu, even if your computer normally runs Windows or macOS.

iii. Immediately after you are finished executing steps 1-6 with the Q1 BOOT USB, remove the Q1 BOOT USB from the SETUP 1 computer.

1. On your desktop, right-click the corresponding USB drive icon in your dock and select Eject from the pop-up menu.

2. Remove the USB drive from the USB slot.

iv. **The Q1 BOOT USB is now eternally quarantined. It should never again be plugged into anything besides the Q1 computer.**

8. Create the SETUP 2 BOOT USB and Q2 BOOT USB

a. Repeat steps 1-7 using the SETUP 2 computer, SETUP 2 BOOT USB, and Q2 BOOT USB.

3.5. Create App USBs

Each signatory should do the following:

We will prepare two (2) "Quarantined App USB" drives with the software needed to execute the remainder of the protocol. These are the USB drives you labeled "Q1 APP" and "Q2 APP" in Section III.

1. Boot the SETUP 1 computer off the SETUP 1 BOOT USB if it is not already. (See the instructions in Section III for details.)
2. Insert the Q1 APP USB into the the the SETUP 1 computer.

    a. **The instruction to plug a Quarantined App USB into your Setup computer** *should* **raise a red flag for you, because you should never plug a quarantine USB into anything other than the quarantined computer it is designated for!**

        This setup process is the ONE exception.

3. Press Ctrl-Alt-T to open a terminal window.
4. Download the CryptoGlacier protocol document and accompanying scripts

    a. Download the latest full release of CryptoGlacier (*not* just the protocol document) at https://github.com/vogelito/CryptoGlacierProtocol/releases (https://github.com/vogelito/CryptoGlacierProtocol/releases).

    b. Unpack the CryptoGlacier ZIP file into a staging area.

        i. When the download starts, Firefox will ask you if you want to open the ZIP file with Archive Manager. Click OK.

            When the ZIP file download completes, it will be opened with Archive Manager.

        ii. There will be a single entry in a list named "CryptoGlacierProtocol-`version-here`", where `version-here` is replaced with the current version number (like "v0.02"). Click on that and then click the "Extract" button.

        iii. The Archive Manager will ask you where you want to extract the ZIP file to. Select "Home" on the left panel and then press the extract button.

        iv. When the Archive Manager is finished extracting the ZIP archive it will ask you what to do next. Click "Show the Files".

        v. Rename the unzipped folder from "CryptoGlacierProtocol-`version-here`" to "cryptoglacier".

    c. Obtain the CryptoGlacier "public key," used to cryptographically verify the CryptoGlacier document and CryptoGlacierScript.

If you are ever using CryptoGlacier in the future and notice that this step has changed (or that this warning has been removed), there is a security risk. Stop and seek assistance.

    i. Access CryptoGlacier's Keybase profile at https://keybase.io/vogelito (https://keybase.io/vogelito).

    ii. Click the string of letters and numbers next to the key icon.

    iii. In the pop-up that appears, locate the link reading "this key".

    iv. Right-click the link and select "Save Link As…"

    v. Name the file "cryptoglacier.asc".

d. Verify the integrity of the CryptoGlacier download.

    i. Import the CryptoGlacier public key into your local GPG installation:

```
$ gpg --import ~/Downloads/cryptoglacier.asc
```

    ii. Switch to the cryptoglacier folder:

```
$ cd ~/cryptoglacier
```

    iii. Use the public key to verify that the CryptoGlacier "fingerprint file" is legitimate:

```
$ gpg --verify SHA256SUMS.sig SHA256SUMS
```

Expected output (timestamp will vary, but e-mail and fingerprint should match):

```
gpg: Signature made Thu Jun 20 18:01:31 2019 CDT
gpg:                using RSA key 3378240146B53C307FBA4B0D97F10485CCBACA30
gpg: Good signature from "Daniel Vogel <vogel@bitso.com>" [unknown]
gpg:                 aka "Daniel Vogel <dvogel@cs.stanford.edu>" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: 3378 2401 46B5 3C30 7FBA  4B0D 97F1 0485 CCBA CA30
```

The warning message is expected, and is not cause for alarm.

    iv. Verify the fingerprints in the fingerprint file match the fingerprints of the downloaded CryptoGlacier files:

```
$ sha256sum -c SHA256SUMS 2>&1
```

Expected output:

```
CryptoGlacier.pdf: OK
README.md: OK
mnemonic_entropy.py: OK
package.json: OK
package-lock.json: OK
setup.js: OK
```

5. Open the CryptoGlacier protocol document so that it is available for copy-pasting terminal commands.

6. Install the remaining application software on the Q1 APP USB.

    a. Configure our system to enable access to the software we need in Ubuntu's "package repository".On Ubuntu 16.04.01 there is a bug (https://bugs.launchpad.net/ubuntu/ +source/appstream/+bug/1601971) in Ubuntu's package manager that affects systems running off a bootable Ubuntu USB. The commands in steps a and b are a workaround.

        i.
        ```
        $ sudo mv /var/cache/app-info/xapian/default \
        /var/cache/app-info/xapian/default_old
        ```

        ii.
        ```
        $ sudo mv /var/cache/app-info/xapian/default_old \
        /var/cache/app-info/xapian/default
        ```

        iii.
        ```
        $ sudo apt-add-repository universe
        ```

        iv.
        ```
        $ curl -sSL \
        https://deb.nodesource.com/gpgkey/nodesource.gpg.key \
        | sudo apt-key add -
        $ echo "deb https://deb.nodesource.com/node_10.x xenial main" \
        | sudo tee /etc/apt/sources.list.d/nodesource.list
        $ echo "deb-src https://deb.nodesource.com/node_10.x xenial \
        main" | sudo tee -a /etc/apt/sources.list.d/nodesource.list
        ```

        v.
        ```
        $ sudo apt-get update
        ```

    b. Download and perform integrity verification of software available from Ubuntu's package repository:

        ■ **libappindicator1** and **libindicator7**: Dependencies for multisigweb (see below)

        ■ **nodejs** and **npm**: Required to run the cryptoglacierscript

        ■ **qrencode**: Used for creating QR codes to move data off quarantined computers

        ■ **zbar-tools**: Used for reading QR codes to import data into quarantined computers

```
$ sudo apt-get -y install \
libindicator7=12.10.2+16.04.20151208-0ubuntu1 \
libappindicator1=12.10.1+16.04.20170215-0ubuntu1 \
nodejs=10.18.1-1nodesource1 \
qrencode=3.4.4-1 \
zbar-tools=0.10+doc-10ubuntu1
```

- **multisigweb**: Used to manage multi-location multisig cold wallets for ETH and ERC20 token

```
$ mkdir ~/dls
$ cd ~/dls
$ wget https://github.com/gnosis/MultiSigWallet/releases/
download/v1.6.0/multisigweb-1.6.0-amd64.deb.zip
```

Make sure the output of `$ sha256sum multisigweb-1.6.0-amd64.deb.zip` is:

```
607e1e94cb5d4d9deb2b05eb0d9f6aaa6a41eaba531b3333dea5da90e2f29350
```

Unzip:

```
$ unzip multisigweb-1.6.0-amd64.deb.zip
```

- **Electrum**: Used to manage multi-location multisig cold wallets

```
$ wget https://download.electrum.org/3.3.6/electrum-3.3.6-
x86_64.AppImage
$ wget https://download.electrum.org/3.3.6/electrum-3.3.6-
x86_64.AppImage.asc
```

Import the signing keys

```
$ wget https://raw.githubusercontent.com/spesmilo/electrum/
3.3.6/pubkeys/ThomasV.asc
$ gpg --import ThomasV.asc
```

Verify the file

```
$ gpg --verify electrum-3.3.6-x86_64.AppImage.asc
electrum-3.3.6-x86_64.AppImage
```

You should see something similar to (verify fingerprint matches):

```
      gpg: Signature made Thu 16 May 2019 06:14:30 PM UTC using
RSA key ID 7F9470E6
      gpg: Good signature from "Thomas Voegtlin (https://
electrum.org) <thomasv@electrum.org>"
      gpg:                   aka "ThomasV <thomasv1@gmx.de>"
      gpg:                   aka "Thomas Voegtlin
<thomasv1@gmx.de>"
      gpg: WARNING: This key is not certified with a trusted
signature!
      gpg:           There is no indication that the signature
belongs to the owner.
      Primary key fingerprint: 6694 D8DE 7BE8 EE56 31BE  D950
2BD5 824B 7F94 70E6
```

- **ElectronCash**: Used to manage multi-location multisig cold wallets

```
  $ wget https://github.com/Electron-Cash/Electron-Cash/
releases/download/4.0.6/Electron-Cash-4.0.6-x86_64.AppImage
  $ wget https://github.com/Electron-Cash/keys-n-hashes/raw/
master/sigs-and-sums/4.0.6/win-linux/Electron-Cash-4.0.6-
x86_64.AppImage.asc
  $ wget https://github.com/Electron-Cash/keys-n-hashes/raw/
master/sigs-and-sums/4.0.6/win-linux/SHA256.Electron-Cash-4.0.6-
x64_64.AppImage.txt
```

Import the signing keys

```
  $ gpg --import <(curl -L https://raw.githubusercontent.com/
fyookball/keys-n-hashes/master/pubkeys/jonaldkey2.txt)
```

Make sure the output of `$ sha256sum -c SHA256.Electron-Cash-4.0.6-x64_64.AppImage.txt` is:

```
  Electron-Cash-4.0.6-x86_64.AppImage: OK
```

Verify the signatures

```
  $ gpg --verify Electron-Cash-4.0.6-x86_64.AppImage.asc
Electron-Cash-4.0.6-x86_64.AppImage
```

You should see something similar to (verify fingerprint matches):

```
      gpg: Signature made Thu 06 Jun 2019 02:46:52 PM UTC using
DSA key ID EFF1DDE1
      gpg: Good signature from "Jonald Fyookball
<jonf@electroncash.org>"
      gpg: WARNING: This key is not certified with a trusted
signature!
      gpg:              There is no indication that the signature
belongs to the owner.
      Primary key fingerprint: D56C 110F 4555 F371 AEEF  CB25
4FD0 6489 EFF1 DDE1
```

- **Electrum-LTC**: Used to manage multi-location multisig cold wallets

```
  $ wget https://electrum-ltc.org/download/electrum-ltc-3.3.6.1-
x86_64.AppImage
  $ wget https://electrum-ltc.org/download/electrum-ltc-3.3.6.1-
x86_64.AppImage.asc
```

Import signing keys from keyserver

```
  $ gpg --keyserver pool.sks-keyservers.net --recv-keys
0x6fc4c9f7f1be8fea 0xfe3348877809386c
```

You should see something similar to

```
      gpg: requesting key F1BE8FEA from hkp server pool.sks-
keyservers.net
      gpg: requesting key 7809386C from hkp server pool.sks-
keyservers.net
      gpg: key F1BE8FEA: public key "pooler
<pooler@litecoinpool.org>" imported
      gpg: key 7809386C: public key "Adrian Gallagher
<thrasher@addictionsoftware.com>" imported
      gpg: no ultimately trusted keys found
      gpg: Total number processed: 2
      gpg:               imported: 2  (RSA: 2)
```

Verify that the fingerprints are correct

```
  $ gpg --fingerprint 0x6fc4c9f7f1be8fea 0xfe3348877809386c
```

You should see:

```
    pub   2048R/F1BE8FEA 2013-07-21
          Key fingerprint = CAE1 092A D355 3FFD 21C0  5DE3
6FC4 C9F7 F1BE 8FEA
    uid                    pooler <pooler@litecoinpool.org>
    sub   2048R/A31415A6 2013-07-21


    pub   2048R/7809386C 2013-06-19
          Key fingerprint = 59CA F0E9 6F23 F537 4794  5FD4
FE33 4887 7809 386C
    uid                    Adrian Gallagher
<thrasher@addictionsoftware.com>
    sub   2048R/6FB978EE 2013-06-19
```

Verify signature of downloaded files

```
  $ gpg --verify electrum-ltc-3.3.6.1-x86_64.AppImage.asc
electrum-ltc-3.3.6.1-x86_64.AppImage
```

You should see something like:

```
    gpg: Signature made Wed 22 May 2019 07:07:53 AM UTC using
RSA key ID F1BE8FEA
    gpg: Good signature from "pooler
<pooler@litecoinpool.org>"
    gpg: WARNING: This key is not certified with a trusted
signature!
    gpg:         There is no indication that the signature
belongs to the owner.
    Primary key fingerprint: CAE1 092A D355 3FFD 21C0  5DE3
6FC4 C9F7 F1BE 8FEA
```

- **BIP39**: Used to manage multi-location multisig cold wallets

```
  $ wget https://github.com/iancoleman/bip39/releases/download/
0.3.11/bip39-standalone.html
```

Make sure the output of `$ sha256sum bip39-standalone.html` is:

954691257c7ab59175de365688e3bc7c1112e1392d308c1b97336b23854fe397

c. Copy that software to the Q1 APP USB.

    i. Create a folder for the application files that will be moved to the USB:

```
$ mkdir ~/apps
```

    ii. Copy the software into the apps folder:

```
$ cp /var/cache/apt/archives/*.deb ~/apps
$ cp ~/dls/electrum-3.3.6-x86_64.AppImage ~/apps
$ cp ~/dls/Electron-Cash-4.0.6-x86_64.AppImage ~/apps
$ cp ~/dls/electrum-ltc-3.3.6.1-x86_64.AppImage ~/apps
$ cp ~/dls/bip39-standalone.html ~/apps
$ cp ~/dls/multisigweb-1.6.0-amd64.deb ~/apps
```

    iii. Copy the contents of the apps folder to the Q1 APP USB:

      1. Click on the File Manager icon in the launching dock:

      2. Navigate to the "Home" folder.

      3. Click and drag "apps" folder to the icon representing the USB drive on the left panel.

7. Install the required packages for the CryptoGlacier scripts and install CryptoGlacier on the Q1 APP USB.

a. Install the required packages for the node script:

```
$ cd ~/cryptoglacier
$ npm install
```

b. Copy the cryptoglacier folder to the Q1 APP USB.

    i. Click on the File Manager icon in the launching dock along the left side of the screen.

    ii. Find the "cryptoglacier" folder under "Home".

    iii. Click and drag the cryptoglacier folder to the icon representing the USB drive on the left.

    iv. You may get errors related to copying `rlp`, `sha.js`, and `uuid`. You can safely skip those

    v. If you see a different "Error while copying" pop-up, you may be suffering from this Ubuntu bug (https://bugs.launchpad.net/ubuntu/+source/nautilus/+bug/1021375). To fix it, do the following and then retry copying the files:

1. `$ mv ~/.config/nautilus ~/.config/nautilus-bak`

     1. Log out of Ubuntu: Click the power icon in the top right of the screen and select "logout" from the drop-down menu.
     2. Login again with user "ubuntu" and leave the password blank.

8. Click on the USB drive icon to verify that it has the correct files. The contents should look like this

```
apps
cryptoglacier
```

Click the apps folder. It will have the following content:

```
bip39-standalone.html
Electron-Cash-4.0.6-x86_64.AppImage
electrum-3.3.6-x86_64.AppImage
electrum-ltc-3.3.6.1-x86_64.AppImage
libappindicator1_12.10.1+16.04.20170215-0ubuntu1_amd64.deb
libdbusmenu-gtk4_16.04.1+16.04.20160927-0ubuntu1_amd64.deb
libindicator7_12.10.2+16.04.20151208-0ubuntu1_amd64.deb
libqrencode3_3.4.4-1_amd64.deb
libzbar0_0.10+doc-10ubuntu1_amd64.deb
multisigweb-1.6.0-amd64.deb
nodejs_10.18.1-1nodesource1_amd64.deb
qrencode_3.4.4-1_amd64.deb
zbar-tools_0.10+doc-10ubuntu1_amd64.deb
```

Click the cryptoglacier folder. It will have the following content:

```
node_modiules
CryptoGlacier.pdf
LICENSE
mnemonic_entropy.py
package.json
package-lock.json
README.md
setup.js
SHA256SUMS
SHA256SUMS.sig
```

9. Eject and physically remove the Q1 APP USB from the SETUP 1 computer.

   The Q1 APP USB is now eternally quarantined. It should never again be plugged into anything besides the Q1 computer.

10. Repeat all above steps using the SETUP 2 computer, SETUP 2 BOOT USB, and Q2 APP USB.

11. Find a container in which to store all of your labeled hardware, along with the CryptoGlacier document hardcopy, when you are finished.

3.6. Prepare quarantined workspaces

Each signatory must always prepare a quarantined workspace before executing the Key Generation, Deposit or Withdrawal protocols. If you are executing the Setup Protocol for the first time and do **not** plan on executing the Key Generation, Deposit or Withdrawal protocol now, you can stop here.

1. Block side channels

   Side-channel attacks (https://en.wikipedia.org/wiki/Side-channel_attack) are a form of electronic threat based on the physical nature of computing hardware (as opposed to algorithms or their software implementations). Side channel attacks are rare, but it's relatively straightforward to defend against most of them.

   a. Visual side channel
      i. Ensure that no humans or cameras (e.g. home security cameras, which can be hacked) have visual line-of-sight to the Quarantined Computers.
      ii. Close doors and window shades.
   b. Acoustic side channel (https://en.wikipedia.org/wiki/Acoustic_cryptanalysis)
      i. Choose a room where sound will not travel easily outside.
      ii. Shut down nearby devices with microphones (e.g. smartphones and other laptops).
      iii. Plug in and turn on a table fan to generate white noise.
   c. Power side channel (http://sharps.org/wp-content/uploads/CLARK-ESORICS13.pdf)
      i. Unplug both Quarantined Computers from the wall.
      ii. Run them **only on battery power** throughout this protocol.
      iii. Make sure they are fully charged first! If you run out of battery, you'll need to start over.
   d. Radio (https://cyber.bgu.ac.il/how-leak-sensitive-data-isolated-computer-air-gap-near-mobile-phone-airhopper/) and other side channels. Including seismic (https://www.cc.gatech.edu/fac/traynor/papers/traynor-ccs11.pdf), thermal (https://cyber.bgu.ac.il/bitwhisper-heat-air-gap/), and magnetic (http://fc15.ifca.ai/preproceedings/paper_14.pdf).
      i. Turn off all other computers and smartphones in the room.
      ii. Put portable computing devices in the Faraday bag and seal the bag.
      iii. Unplug desktop computers.
2. Put your Q1 BOOT USB into an open slot in your Q1 computer.

3. Boot off the USB drive. If you've forgotten how, refer to the procedure in Section IV.
4. Plug the Q1 APP USB into the Q1 computer
5. Copy the software from the Q1 computer's RAM disk.
    a. Click the File Manager icon from the launchpad on the left side of the screen.
    b. Click on the App USB on the left of the file manager.
    c. Drag the contents of the USB to the "Home" directory on the left side of file manager.
6. Open a copy of this document on the Q1 computer.
    a. In the File Manager find the cryptoglacier folder. The PDF file for this document should be visible with the name "CryptoGlacier.pdf". Open it.

    You won't be able to click any external links in the document, since you don't have a network connection. If you need to look something up on the internet, do so in a distant room. Do not remove devices from the Faraday bag before doing going to the other room.

7. Open a Terminal window by pressing Ctrl-Alt-T.
8. Install the application software on the Q1 computer's RAM disk.

```
$ cd ~/apps
$ sudo dpkg -i *.deb
```

9. Prepare the AppImage files for execution

```
$ chmod +x ~/apps/electrum-3.3.6-x86_64.AppImage
$ chmod +x ~/apps/Electron-Cash-4.0.6-x86_64.AppImage
$ chmod +x ~/apps/electrum-ltc-3.3.6.1-x86_64.AppImage
```

10. Change into the cryptoglacier directory. You'll be using this directory to execute software for the protocol

```
$ cd ~/cryptoglacier
```

11. Prepare the "Quarantined Scratchpad" – an empty file you'll use as a place to jot notes.
    a. Click the "Search your computer" icon at the top of the launcher along the left side of the screen.
    b. Type "text editor".
    c. Click the Text Editor icon.
    d. A blank window should appear.
12. Repeat the above steps using the Q2 computer, Q2 SETUP USB and Q2 APP USB.

4. Key Generation

4.1. Generate BIP39 Mnemonic

The Key Generation Protocol will securely generate a `BIP39 Mnemonic` that will be used to store all your assets.

Through the BIP39 (https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki) standard we will create 24 words (a mnemonic) which will then be used to derive private keys across protocols. Each signatory will only need to secure their 24-word phrase in order to be able to access funds .

By the end of this section, each signatory will generate the following information:

- A `BIP39 Mnemonic`: Also called a seed phrase, this is a 24-word combination that will later be used to unlock your funds across BTC, BCH, LTC, XRP & ETH. Each seed phrase should be created in a different device by a different signatory. In your M-of-N policy there should be N signatories each with 1 seed phrase. **These phrases should always be separated and signatories should never see each other's seed phrases.**
- A `Master public key for Bitcoin`: An alphanumeric string to allow Electrum to generate the public keys for the BTC cold HD wallet
- A `Master public key for Litecoin`: An alphanumeric string to allow Electrum-LTC to generate the public keys for the LTC cold HD wallet
- A `Master public key for Bitcoin Cash`: An alphanumeric string to allow Electron-Cash to generate the public keys for the BCH cold HD wallet
- An `XRP address`: Address to give ownership of an XRP multisign account
- An `ETH address`: Address to give ownership of an ETH multisig contract

Only quarantined hardware should be used during the execution of the Key Generation Protocol.

Signatories should not be in close proximity of each other when executing the Key Generation Protocol. Signatory should always use distinct quarantined computers and should never ever share seed phrases.

1. If this is **not** your first time working with CryptoGlacier:
   a. Use a networked computer to access the latest full release of CryptoGlacier (not just the protocol document) at https://github.com/vogelito/CryptoGlacierProtocol/releases (https://github.com/vogelito/CryptoGlacierProtocol/releases).
   b. Open the protocol document (CryptoGlacier.pdf) within the ZIP file.
   c. Check the Release Notes (Appendix E) of the protocol document to see if there are any new versions of CryptoGlacier recommended.

d. Whether or not you decide to upgrade, review the errata for the version of CryptoGlacier you are using at https://github.com/vogelito/GlacierProtocol/releases (https://github.com/vogelito/GlacierProtocol/releases).

2. Execute Section VI of the Setup Protocol to prepare your quarantined workspace.

3. Create entropy for private keys

Creating an unguessable private key requires *entropy* – random data. We'll combine two sources of entropy to generate our keys. This ensures securely random keys even if *one* entropy source is somehow flawed or compromised to be less-than-perfectly random.

a. Generate dice entropy

    i. Type "DICE ENTROPY" into both Quarantined Scratchpads.

    ii. Roll 62 six-sided dice, shaking the dice thoroughly each roll. 62 dice rolls corresponds to 160 bits of entropy. See the design document for details.

    iii. If you are rolling multiple dice at the same time, read the dice left-to-right. **This is important.** Humans are horrible at generating random data (http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0041531) and great at noticing patterns. Without a consistent heuristic like "read the dice left to right", you may subconsciously read them in a non-random order (like tending to record lower numbers first). This can drastically undermine the randomness of the data, and could be exploited to guess your private keys.

    iv. Manually enter the `numbers` into the Quarantined Scratchpad of **ONLY ONE** of the quarantined computers. Put all rolls on the same line to create `one line of 62 numbers`. (It's fine to add spaces for readability.)

b. Generate BIP39 seed phrase

    i. Make sure you are in the `~/cryptoglacier folder`:

```
$ cd ~/cryptoglacier
```

    ii. **On the Q1 computer** enter the following command to generate your BIP39 seed phrase

```
$ node setup.js --init
```

    iii. The script will prompt you to enter the 62-number line of dice entropy

    iv. The script will output your cold storage data:
       - Dice entropy
       - Generated Computer entropy
       - Final entropy
       - BIP39 Mnemonic
       - Bitcoin Master Public Key
       - Litecoin Master Public Key

- Bitcoin Cash Master Public Key
- Ethereum Address
- Ethereum Private Key
- Ripple Address
- Ripple Private Key

Example output:

```
Dice entropy:                          1111 1111 1111 1111 1111 1111 111
Generated Computer entropy:            aaaa aaaa aaaa aaaa aaaa aaaa aaaa
Final entropy:                         ae29155ab1b3f5a1fc0c7cee883cd39457
BIP39 Mnemonic:                        purchase emerge find gloom dismiss
Bitcoin Master Public Key (Zpub):      Zpub75EpZYVWcoTQ1WChsnabzLUAm91t31
Litecoin Master Public Key (Zpub):     Zpub75CcoXN1LVH6qE3MNYAAbd8f6c3pg1
BitcoinCash Master Public Key (xpub):  xpub6BjfaUaRAzkDfaYWKGNDxewXDUpgk1
Ethereum Address:                      0x6bff50edf67a2eae30e9eef7007b3129
Ethereum Private Key:                  0xB31B0A016562839D6D6D137489924C50
Ripple Address:                        rHzjZTF4nD1ta2oPz7EYvYKXx26n8ZKFU
Ripple private key:                    F477BA0925608BCBBF870078E17030DB11
```

v. Type "COMPUTER ENTROPY" into both computers' Quarantined Scratchpads. (This is a descriptive heading to keep your notes organized and minimize risk of error.)

vi. Copy-paste the Generated Computer entropy into the Quarantined Scratchpad.

vii. Manually enter the Generated Computer entropy into the Quarantined Scratchpad on the other quarantined computer.

c. Verify the integrity of the cold storage data

If there are discrepancies in any of the verification steps, please restart the protocol. If discrepancies continue, please DO NOT PROCEED and seek assistance.

i. **On the Q2 computer** enter the following command:

```
$ node setup.js --check
```

1. The script will prompt you to enter the 62-number dice entropy and the Generated computer entropy.

2. Verify that the BIP39 Mnemonic shown in the terminal window is identical on both computers.

   Make sure you carefully verify every word.

   There are attack vectors which could replace just a portion of BIP39 seed phrase, making the private keys easier to brute force, so it's important to check them thoroughly.

ii. **On the Q2 computer** enter the following command:

```
$ python mnemonic_entropy.py entropy --integrity
```

1. The script will prompt you to enter the `62-number dice entropy` and the `Generated computer entropy`.

   Example output of the python script:

   ```
   Generated Entropy (copy this string into bip39-standalone.ht
   ```

2. Verify that the `Generated Entropy` string shown in the terminal window is identical to the `Final entropy` string shown in the node script terminal output of the **Q1 Computer**.

iii. **On the Q2 computer** open bip39-standalone.html

```
$ firefox ~/apps/bip39-standalone.html
```

1. Check the `Show entropy details` checkbox
2. Copy the `Generated Entropy` output of the python script into the `Entropy` box in Firefox and verify that the BIP39 seed matches that of the node script terminal output of the **Q1 Computer**.

   Example output in the `BIP39 Mnemonic` section in Firefox:

   ```
   purchase emerge find gloom dismiss special usual moon update
   ```

   Verify every word of the BIP39 Mnemonic so it matches the output of the scripts on both computers.

3. In the `Coin` dropdown menu in Firefox, select `ETH - Ethereum`
   i. Verify that the `derived Ethereum address` for the `m/44'/60'/0'/0/0` Path matches the `Ethereum address` output of the script **on the Q1 computer**. (Case insensitive)
   ii. Verify that the `derived Ethereum Private Key` for the `m/44'/60'/0'/0/0` Path matches the `Ethereum Private Key` output of the script **on the Q1 computer**. (Case insensitive)

   For the Ethereum private key, verify each character. Again, there are attack vectors which could replace just a portion of private keys, making the private keys easier to brute force so it's important to check them thoroughly. For **Ethereum**, both the private key and the address **ARE NOT** case sensitive.

4. In the `Coin` dropdown menu in Firefox, select `XRP - Ripple`
     i. Verify that the <mark>derived Ripple address</mark> for the `m/44'/144'/0'/0/0` Path matches the <mark>Ripple address</mark> output of the script **on the Q1 computer**.
     ii. Verify that the <mark>derived Ripple Private Key</mark> for the `m/44'/144'/0'/0/0` Path matches the <mark>Ripple Private Key</mark> output of the script **on the Q1 computer**.

    For the Ripple private key, verify each character. Again, there are attack vectors which could replace just a portion of private keys, making the private keys easier to brute force so it's important to check them thoroughly.

iv. **On the Q2 computer** open Electrum

```
$ ~/apps/electrum-3.3.6-x86_64.AppImage
```

1. Leave `default_wallet` and click `Next` .
2. Select `Multi-signature wallet` and click `Next` .
3. Select your `M-of-N` policy. The top bar, `cosigners` is the `N` and the lower bar, `Required signatures` is the `M` . Click `Next` .
4. Select `I already have a seed` and click `Next` .
5. Click on `Options` , select `BIP39 seed` and click `OK` .
6. Enter your <mark>BIP39 seed phrase</mark> and click `Next` .
7. Leave `native segwit multisig (p2wsh)` selected and click `Next` .
8. Verify that the <mark>Master Public Key</mark> is the same as the output of the script **on the Q1 computer**.

   Again, please make sure you verify each character.

9. Close the window (click on the top left `x`)

v. **On the Q2 computer** open Electrum-LTC

```
$ ~/apps/electrum-ltc-3.3.6.1-x86_64.AppImage
```

1. Leave `default_wallet` and click `Next` .
2. Select `Multi-signature wallet` and click `Next` .
3. Select your `M-of-N` policy. The top bar, `cosigners` is the `N` and the lower bar, `Required signatures` is the `M` . Click `Next` .
4. Select `I already have a seed` and click `Next` .
5. Click on `Options` , select `BIP39 seed` and click `OK` .
6. Enter your <mark>BIP39 seed phrase</mark> and click `Next` .
7. Leave `native segwit multisig (p2wsh)` selected and click `Next` .

8. Verify that the <mark>Master Public Key</mark> is the same as the output of the script **on the Q1 computer**.

   Again, please make sure you verify each character.

9. Close the window (click on the top left `x`)

vi. **On the Q2 computer** open Electron-Cash

```
$ ~/apps/Electron-Cash-4.0.6-x86_64.AppImage
```

1. Leave `default_wallet` and click `Next` .
2. Select `Multi-signature wallet` and click `Next` .
3. Select your `M-of-N` policy. The top bar, `cosigners` is the `N` and the lower bar, `Required signatures` is the `M` . Click `Next` .
4. Select `I already have a seed` and click `Next` .
5. Click on `Options` , select `BIP39 seed` and click `OK` .
6. Enter your <mark>BIP39 seed phrase</mark> and click `Next` .
7. Leave the default `m/44'/145'/0'` derivation path selected and click `Next` .
8. Verify that the <mark>Master Public Key</mark> is the same as the output of the script **on the Q1 computer**.

   Again, please make sure you verify each character.

9. Close the window (click on the top left `x`)

vii. **On the Q2 computer** open multisigweb

```
$ multisigweb
```

1. Agree to the Terms of Use and Privacy Policy
2. Select `Light Wallet`
3. Go to the `Accounts` tab and click `Import`
4. Click `Browse...` and go to Home -> cryptoglacier
5. Select the `ethereum.json` file and click `Open`
6. In the password field enter `cryptoglacier`
7. In the account name field enter anything you'd like, such as `coolest signatory`
8. Click `Import Account`
9. Verify that the <mark>Ethereum Address</mark> is the same as the output of the script **on the Q1 computer**.

   Again, please make sure you verify each character.

10. Close the window (click on the top left `x`)

4.2. Transfer cold storage data to paper

In this section, you'll move the cold storage data you generated in Section I from the quarantined computing environments onto physical paper. This will be done using a combination hand transcription and QR codes (https://en.wikipedia.org/wiki/QR_code).

Each signatory will need to do the following:

1. Transfer the `BIP39 Mnemonic` to paper.
    a. Write the `24-word BIP39 Mnemonic` on a piece of TerraSlate paper.
        i. Do **not** write anything else on the paper unless specifically instructed (such as "Bitcoin", "CryptoGlacier", "private key", etc.) In the event the key is seen by someone untrustworthy or stolen by a random thief, such clues help them understand the significance of the key and give them an incentive to plot further thefts or attacks.
        ii. Transcribe **by hand**. Do not use QR codes or any other method to transfer.
        iii. Seed phrases **are not** case-sensitive.
        iv. **Write clearly**.
            1. Use care to distinguish between "l" (lower-case "L") and "I" (upper case "i")
            2. Use care to distinguish between "u" and "v"
            3. Use care to distinguish between "U" and "v"
    b. **Double-check that you transcribed all 24 words in the** `BIP39 Mnemonic` **correctly.** If you make a mistake, you'll have to redo a lot of work.
    c. Manually count that you have transcribed 24 words.
    d. Label each page with:
        i. Today's date
        ii. Your initials
        iii. The **version** of CryptoGlacier used (listed on the front page of this document) Do not write "CryptoGlacier", simply the version of CryptoGlacier used (e.g. `0.94.1`)
        iv. Write down the chosen M-of-N configuration
2. Visually hide all critically sensitive data.

    We'll be using a smartphone with a live Internet connection to read QR codes from the quarantined computer screens. Any malware (or a malicious QR reader app) could steal sensitive data if it is not visually hidden.

    This step is important. Failing to execute it properly creates a substantial security risk.

    a. Put your `handwritten BIP39 Mnemonic` out of sight (don't just turn them face down; paper is not completely opaque). This prevents a smartphone camera from accidentally seeing them.

b. Delete all text from the Quarantined Scratchpad on the **Q1 and Q2** computers.

c. **On the Q1 computer Quarantined Scratchpad:**

    i. Copy-paste the following items from the node script's terminal window output:

- Bitcoin Master Public Key
- Litecoin Master Public Key
- Bitcoin Cash Master Public Key
- Ethereum Address
- Ripple Address

    ii. Double check that only the items above are included in the Scratchpad.

    iii. No really, triple check that.

    iv. Enable line wrapping so the entire contents can be seen.

        1. With the Quarantined Scratchpad window active, go to the menu bar at the top of the screen.

        2. Select Edit.

        3. Select Preferences.

        4. Select the View tab.

        5. Uncheck "Do not split words over two lines".

d. **On the Q2 computer** close Firefox.

e. Clear the terminal windows on the **Q1 and Q2** computers.

```
$ clear
```

3. QR reader setup

    a. Remove a smartphone from the Faraday bag and turn it on.

    b. If the smartphone doesn't already have a QR code reader on it, install one.

Any reader is fine as long as it can read all types of QR codes, but here are recommendations we've tested with Glacier: iOS (https://itunes.apple.com/us/app/qr-reader-for-iphone/id368494609?mt=8), Android (https://play.google.com/store/apps/details?id=com.application_4u.qrcode.barcode.scanner.reader.flashlight&hl=en).

4. Transfer the Bitcoin Master Public Key to a non-quarantined computer.

    a. **On the Q1 computer**, display the Bitcoin Master Public Key as a QR code on the screen:

```
$ cd ~/cryptoglacier
$ eog bitcoin_master_public_key.png
```

    b. Use the smartphone's QR code reader to read the QR code. When the QR code is successfully read, the smartphone should display the text version of the Bitcoin Master Public Key.

    c. Verify the Bitcoin Master Public Key address on the smartphone matches the Bitcoin Master Public Key in the Quarantined Scratchpad.

If it does not match, do not proceed. Try using a different QR reader application or restarting the Deposit Protocol. Seek assistance if discrepancies persist.

    d. Use the smartphone to send the <mark>Bitcoin Master Public Key</mark> to yourself using a messaging app which you'll be able to access from a laptop (e.g. Whatsapp, Slack, etc..). When sending the message, add **"B"** as an **identifier** to the message. This is an indication that the <mark>Public Key</mark> belongs to Bitcoin. (E-mail is not recommended for security reasons.)

    e. Take a picture of the QR Code

    f. Use the smartphone to send the <mark>QR Code</mark> picture to yourself using a messaging app which you'll be able to access from a laptop. Also use the **identifier "B"**. (E-mail is not recommended for security reasons.)

    g. Close the program displaying the QR code (click on the top left <mark>x</mark>)

5. Repeat the previous step for the following:
   - <mark>Litecoin Master Public Key</mark>, stored in "litecoin_master_public_key.png", with **identifier "L"**.
   - <mark>Bitcoin Cash Master Public Key</mark>, stored in "bitcoincash_master_public_key.png", with **identifier "BC"**.
   - <mark>Ethereum Address</mark>, stored in "ethereum_address.png", with **identifier "E"**.
   - <mark>Ripple Address</mark>, stored in "ripple_address.png", with **identifier "X"**

6. Power down the smartphone and return it to the Faraday bag.

7. Shut down **both** quarantined computers entirely. As a precaution against side channel attacks, the quarantined computers should not be active except when they absolutely need to be.

```
$ sudo shutdown now
```

The recommended Acer laptop may require you to hold down the power button for several seconds to complete the shutdown.

8. Create your <mark>Signatory Information Packet</mark>.

Using your setup (Internet-connected) computer:

    a. Access the material you sent yourself from your smartphone previously. In total you should have 10 pieces of information:
   - <mark>Bitcoin Master Public Key</mark> (message "B")
   - <mark>Bitcoin Master Public Key QR Code</mark> (message "B")
   - <mark>Litecoin Master Public Key</mark> (message "L")
   - <mark>Litecoin Master Public Key QR Code</mark> (message "L")
   - <mark>Bitcoin Cash Master Public Key</mark> (message "BC")
   - <mark>Bitcoin Cash Master Public Key QR Code</mark> (message "BC")
   - <mark>Ethereum Address</mark> (message "E")
   - <mark>Ethereum Address QR Code</mark> (message "E")
   - <mark>Ripple Address</mark> (message "X")

- ■ Ripple Address QR Code (message "X")

b. Open an empty document in any word processing application (Word, Pages, etc…) This will be used to create the Signatory Information Packet.

c. Put the following information into the document:

  i. Paste each message code followed by its corresponding QR-code and clear-text version

  ii. On the header:

    - ■ Type today's date

    - ■ Your initials

    - ■ Type the version of CryptoGlacier used (listed on the first page of this document)

    - ■ Write down the chosen M-of-N configuration

d. Do **not** put anything else in the document (such as "Bitcoin", "CryptoGlacier", "private key", etc.)

e. Print a copy of the Signatory Information Packet.

f. Share the contents of the Signatory Information Packet:

  i. Save the document as a PDF.

  ii. GPG encrypt the PDF file using your keys and the keys of every other signatory (TODO: expand this).

  iii. Electronically send the gpg-encrypted file to the other signatories

  iv. You can also save an electronic copy of the PDF in a "conventionally secure" location of your choosing. Most password managers include a way to securely store files. Because the Signatory Information Packet contains moderately-sensitive data, there are some privacy considerations with keeping an electronic copy of it. See the Sensitive Data subsection for details.

  v. Delete the unencrypted PDF from your computer

g. Shut down the computer. (It has a camera, and you will be working with critically sensitive data in a moment.)

9. Prepare the *temporary* version of your Cold Storage Information Packet:

a. Put the handwritten BIP39 Mnemonic page along with one Signatory Information Packet in its own opaque envelope. While this obviously won't deter a determined thief, it makes it quite difficult for a thief to steal a key without leaving evidence they have done so – and noticing theft of a single key gives you a chance to move your funds away before the thief can steal a second key.

b. Your Cold Storage Information Packet is incomplete because you still need to go through the Multisig Account Creation Protocol

c. Once complete, each set of pages will be referred to as a Cold Storage Information Packet.

10. Immediate storage of Cold Storage Information Packet:

a. While the other signatories finish the Key Generation Protocol, we need to temporarily store the Cold Storage Information Packet

b. Double-check to make sure the envelope contains a handwritten BIP39 Mnemonic page and your Signatory Information Packet .

c. Seal the envelope

d. Use tamper-resistant seals in addition to the envelope's normal adhesive to seal it.

e. **Immediately** put the Cold Storage Information Packet in the safest possible location in your home or office that is immediately accessible.

f. No, really. Like right now. That's basically one of the keys to a huge pile of cash you have just sitting there in envelopes on your desk.

11. Hardware storage

a. Put tamper-resistant seals on the ends of all USB drives.

b. Close the quarantined laptops, and seal the screen shut with a tamper-resistant seal.

c. Store the hardware somewhere where it is unlikely to be used by accident.

12. It's crucial that only a small amount of time goes by (e.g. a few days) between the execution of the Key Generation Protocol and the Multisig Account Creation Protocol. Each of the signatory's Cold Storage Information Packet will go into long-term storage after the Multisig Account Creation Protocol is executed.

5. Multisig Account Creation

5.1. Create your Multisig Accounts

In the next few sections we will create Multisig wallets for Bitcoin, Litecoin, and Bitcoin Cash. We will also deploy a Multisig Contract for Ethereum and setup a Multisign Account for XRP.

Only one signatory needs to deploy the Ethereum Contract and setup the XRP Account. You should agree with the other signatories who will be responsible for deploying the contract and setting up the XRP account.

Also of note: setting up Ethereum and XRP will require a small amount of funds. The other coins can be setup without any funds.

Before proceeding, every other signatory (N-1 of them) must have sent you their gpg-encrypted Signatory Information Packets so you will have a total of **N** packets (your own included). Make sure you print every Signatory Information Packet.

5.2. Create your Bitcoin, Litecoin, and Bitcoin Cash Multisig Wallets

In this section you'll setup the Multisig Wallets for Bitcoin, Litecoin, and Bitcoin Cash. You will also verify your deposit addresses.

Your Bitcoin, Litecoin, and Bitcoin Cash wallets are derived from your chosen M-of-N policy and the Master Public Keys from the N signatories. The derivation is deterministic, so as long as you keep a copy of the N Signatory Information Packets and you know the chosen M-of-N policy that was chosen, you should be able to derive your wallets. In order to access your funds, you will need to know the M-of-N policy that was chosen, N Master Public Keys, and at least M signatories should have access to their BIP39 Mnemonic.

The following process can be done at an internet connected computer and should be verified across signatories.

1. Derive Wallet Address Using Electrum
2. Open Electrum

```
$ ~/apps/electrum-3.3.6-x86_64.AppImage
```

3. Select `Auto connect` and click `Next` .

4. Leave the default wallet name and click `Next` .

5. Select `Multi-signature wallet` and click `Next` .

6. Select your chosen M-of-N policy. The top bar, `cosigners` is the `N` and the lower bar, `Required signatures` is the `M` . Click `Next` .

7. Select `Use a master key` and click `Next` .

8. You will be asked to enter your Bitcoin Master Public Key . For Bitcoin, this is marked as **Message B** in your Signatory Information Packets . You can scan the QR Code or enter it manually. Click `Next` twice.

9. Select `Enter cosigner key` and click `Next` .

10. You will now need to scan or enter the other N-1 Bitcoin Master Public Keys . These are all marked with **Message B** on your Signatory Information Packets .

11. You don't need to enter a password, just click `Next` .

12. Click on the `Receive` tab. This will show your receiving address which is basically your bitcoin cold storage address .

13. Verify the Bitcoin cold storage address

14. Click on the QR Code

15. Use the smartphone's QR code reader to read the QR Code . When the QR Code is successfully read, the smartphone should display the text version of the bitcoin cold storage address .

16. Send this address to all your signatories and make sure they have derived the same address across their devices.

17. If the verification fails, seek assistance

18. Repeat steps 1 and 2 for Litecoin and Bitcoin Cash

19. For Litecoin use the following program and the **Message L** keys:

    ```
    $ ~/apps/electrum-ltc-3.3.6.1-x86_64.AppImage
    ```

20. For Bitcoin Cash use the following program and the **Message BC** keys:

    ```
    $ ~/apps/Electron-Cash-4.0.6-x86_64.AppImage
    ```

    Electron Cash calls `master keys` simply `public or private keys`

5.3. Create your Ethereum Multisig Contract

In this section you will deploy an Ethereum Multisig Smart Contract based on the Gnosis Multisignature Wallet Contract (https://github.com/gnosis/MultiSigWallet).

As a reminder, this section should only be executed by one agreed upon signatory. If you are not that signatory, please skip this section.

5.4. Deploying the Contract

1. Download Multisigweb
2. On an internet enabled computer, download and install Gnosis Multisignature Wallet (https://github.com/gnosis/MultiSigWallet/releases).
3. Please make sure you verify the SHA256 checksums. At the time of writing, v1.6.0 was the latest version
4. Create a new Ethereum Account that will be used to deploy the contract
5. Open multisigweb
6. Agree to the `Terms of Use` and `Privacy Policy`
7. Select `Light Wallet`
8. Click on `Accounts`
9. Click on `Add`
10. Choose and confirm a password
11. Choose a name for the account (can be anything)
12. Click `Create Account`
13. Send funds to the account that was just created
14. Check the current gas prices at ethgasstation (https://ethgasstation.info/)
15. Multiply the gas price by `2,057,168` and divide by `10^18` to see how much ETH you will need to deploy the contract. We recommend you send twice the amount from the calculation.
16. Click the `Copy` button next to the address created in the step above
17. Send the ETH (beyond the scope of this protocol) to the address above
18. Wait for the transaction to be confirmed, you should see the new balance on the top right section of the screen, next to the account address.
19. Deploy new Multisig Wallet
20. Go the the `Wallets` tab
21. Click on `Add`
22. Select `Create new wallet`
23. Select a wallet name (can be anything)
24. Enter **M** (for your chosen M-of-N policy)
25. Leave daily limit at 0
26. Click on `Remove` so the address that deployed the contract does not have signatory rights over the contract
27. Enter the Ethereum Addresses of the **N** signatories. **You should have N Owners**.
28. Click on `Deploy with factory`
29. You can check ethgasstation (https://ethgasstation.info/) to validate the gas price. Leave all the other fields to their default.

30. Monitor Contract Deployment
31. Go to the `Transactions` tab
32. Wait for the transaction to be mined
33. If you go to the `Wallets` tab, you should now see your Multisig contract
34. Share your `Ethereum Cold Wallet Address` with the other signatories
35. Go to the `Wallets` tab
36. Click on `Copy` on the `Address` Column
37. Send to the `Ethereum Cold Wallet Address` to the other signatories.

5.5. Create your XRP Multisign Account

In this section you will create a Ripple Multisign Account (https://xrpl.org/multi-signing.html).

CryptoGlacierScript has automated the process to set up Multi-Signing (https://xrpl.org/set-up-multi-signing.html).

As a reminder, this section should only be executed by one agreed upon signatory. If you are not that signatory, please skip this section.

1. On an internet enabled computer with CryptoGlacierScript already downloaded, run the multisign account setup and follow the steps in the screen:

   ```
   $ node setup.js -m
   ```

2. The script will output the `Ripple Cold Storage Address`
3. It will also output the `Ripple Cold Storage Secret`. Even though the secret should be disabled, it is good practice to store it somewhere in case of an emergency.
4. Send to the `Ripple Cold Storage Address` to the other signatories.

5.6. Create Cold Storage Information Packet

Only one signatory needs to execute this section. You should agree with the other signatories who will be responsible for creating the `Cold Storage Information Packet`.

In this section you will create a `Cold Storage Information Packet` which will replace the **N** `Signatory Information Packets`.

1. Open an empty document in any word processing application

2. Put the following information into the document:

    a. On the header:

        ■ Type today's date

        ■ Type the version of CryptoGlacier used (listed on the first page of this document)

        ■ Write down the chosen M-of-N configuration

    b. In the body of the document, write the following information. Pay attention to the order:

        i. **MESSAGE B**

            1. Insert the Bitcoin Master Public Key QR Code

            2. Insert the Bitcoin Cold Storage Address for all **N** signatories

        ii. **MESSAGE L**

            1. Insert the Litecoin Master Public Key QR Code for all **N** signatories

            2. Insert the Litecoin Cold Storage Address

        iii. **MESSAGE BC**

            1. Insert the Bitcoin Cash Master Public Key QR Code for all **N** signatories

            2. Insert the Bitcoin Cash Cold Storage Address

        iv. **MESSAGE E**

            1. Insert the Ethereum Cold Storage Address

            2. Insert the MultisigWeb Wallet Configuration

        v. **MESSAGE X**

            1. Insert the Ripple Cold Storage Address

3. gpg encrypt this document (TODO)

4. Send the gpg-encrypted document to all signatories

6. Deposit

6.1. Test deposit and withdrawal

It's important to make sure everything is working properly before proceeding. You'll verify this by making a token deposit to, and withdrawal from, your cold storage address on each of the protocols.

Depositing funds requires the Internet, and does not require handling any critically sensitive cold storage data, so you can use any Internet-connected computer for this section.

1. Open your electronic copy of the <mark>Cold Storage Information Packet</mark>.
2. Perform a test deposit.
   a. Use the wallet software or service of your choice to send a small transaction to your <mark>bitcoin cold storage address</mark>.
      i. Scan your <mark>bitcoin cold storage address QR Code</mark> from the <mark>Cold Storage Information Packet</mark> into the wallet software.
   b. Wait for the Bitcoin network to confirm the transaction at least once. The way you tell whether a transaction has been confirmed varies depending on the software or service you are using to send funds, but it should be displayed prominently.
3. Perform a test withdrawal.
   a. Execute the Withdrawal Protocol to withdraw the remaining balance from cold storage to a regular Bitcoin address of your choice.
   b. Wait for the Bitcoin network to confirm the transaction at least once. (Instructions for doing this are in the Withdrawal protocol.)
4. Repeat the above for Litecoin, Bitcoin Cash, Ether and XRP

6.2. Deposit execution

Depositing funds requires the Internet, and does not require handling any critically sensitive cold storage data, so you can use any Internet-connected computer for this section.

You will need access to an electronic *and* paper copy of your <mark>Cold Storage Information Packet</mark>.

1. If you are depositing a large amount, consider making a small deposit as a test followed by a second deposit for the remainder.
2. Verify cold storage address.
   a. Get one of the paper <mark>Cold Storage Information Packets</mark> containing your cold storage address.

b. Open your electronic copy of the Cold Storage Information Packet. If you've lost access to it, you'll need to recreate a new electronic copy by transcribing one of the hardcopies (attached to each public key) by hand.

c. **Visually verify that the cold storage addresses are identical in the electronic copy and paper copy.** This is to insure that the electronic copy was not damaged, hacked, accidentally changed due to a typo, etc.

d. Return the paper Cold Storage Information Packet to its normal secure storage.

e. Confirm this deposit address with your signatories.

3. Perform the deposit.

a. Use the wallet software or service of your choice to **prepare** to send the desired amount of funds to your cold storage address. Crypto networks require a fee to process transactions. We recommend you use a wallet service that either covers the fees for you or recommends a fee amount automatically, which most do.

Enter all necessary transaction information, but do not actually execute the transaction.

i. Copy-paste your cold storage address from the Cold Storage Information Packet into the wallet software. You may also choose to scan the Cold storage address QR Code

b. **Double-check that the address you pasted or scanned matches the address in the Cold Storage Information Packet and that confirmed by the other signatories. If you use the wrong address, you will lose all of your funds with no recourse.**

c. Execute the transaction.

4. Verify the deposit on the public blockchain.

a. Go to a block explorer, paste the address into the search bar, and press Enter. You'll be taken to a page that shows your cold storage address listed.

b. Within a couple of minutes (and often much faster), you should be able to refresh this page and see your funds listed.

c. Periodically refresh the page until you see the funds moved from "Unconfirmed" to be reflected in "Balance". This generally happens within 15 minutes; if the network is unusually congested, it may take longer.

Your funds are now secured in cold storage.

If this was your first deposit to this cold storage address, proceed to the next section. Otherwise, you have completed the Deposit Protocol.

6.3. Store cold storage data

1. Shut down any nearby computers or smartphones, or other devices with cameras.

2. Immediate storage of `Cold Storage Information Packets`
   a. Double-check to make sure each envelope contains a handwritten `24-word BIP39 Mnemonic` and a `Cold Storage Information Page`.
   b. Seal each envelope.
   c. Use tamper-resistant seals in addition to the envelope's normal adhesive to seal it.
   d. **Immediately** put all `Cold Storage Information Packets` in the safest possible location in your home or office that is immediately accessible.
   e. No, really. Like right now. That's basically a huge pile of cash you have just sitting there in envelopes on your desk.
3. Hardware storage
   a. Put tamper-resistant seals on the ends of all USB drives.
   b. Close the quarantined laptops, and seal the screen shut with a tamper-resistant seal.
   c. Store the hardware somewhere where it is unlikely to be used by accident.
4. Maintenance planning
5. Create a reminder for yourself in six months to execute the Maintenance Protocol . (If you don't have a reminder system you trust, find one on the web.)
6. Long-term storage of `Cold Storage Information Packets`
   a. As soon as possible, transfer each `Cold Storage Information Packet` to its secure storage location (e.g. safe deposit box).
   b. Don't put more than one `packet` in long-term storage in the same building! Signatories should make sure that their respective keys are never stored in the same building, in the same city, or in the same institution. Doing so increases the risk of losing access to your funds in a disaster (e.g. fire) or thorough a thief/social attack.
   c. Remember the following
      i. Do **not** send your `packets` electronically – no e-mail, no photograph, no "secure instant message".
      ii. Do **not** keep any related notes *on* or *with* the `packets`. In the event the key is seen by someone untrustworthy or stolen by a random thief, such clues help them understand the significance of the key and give them an incentive to plot further thefts or attacks.

You have finished securing your cold storage funds.

7. Withdrawal

7.1. Preparation

The Withdrawal Protocol is used to transfer funds out of high-security cold storage.

In this first section, we'll gather physical hardcopies of all information needed to do the withdrawal. This is done with the help of a regular networked computer to find some of this information online and translate it into printed QR codes.

On any Internet-connected computer:

1. If this is **not** your first time working with CryptoGlacier:
   a. Use a networked computer to access the latest full release of CryptoGlacier (not just the protocol document) at https://github.com/vogelito/CryptoGlacierProtocol/releases (https://github.com/vogelito/CryptoGlacierProtocol/releases).
   b. Open the protocol document (CryptoGlacier.pdf) within the ZIP file.
   c. Check the Release Notes (Appendix E) of the protocol document to see if there are any new versions of CryptoGlacier recommended.
   d. Whether or not you decide to upgrade, review the errata for the version of Glacier you are using at https://github.com/vogelito/CryptoGlacierProtocol/releases (https://github.com/vogelito/CryptoGlacierProtocol/releases).
2. Open your electronic copy of the `Cold Storage Information Packet`.

7.2. Withdrawing Bitcoin, Litecoin or BitcoinCash

In this section, we first construct a transaction, which we then pass to a quarantined environment for signature, verify it, and then use QR codes to extract it from the quarantined environment to pass on to additional quarantined environments (of the other signatories) for additional signatures and eventually extract it for execution.

For brevity purposes, we will use Bitcoin and Electrum as an example, but this should be easily replicable on Litecoin and Electrum-LTC or Bitcoin Cash and ElectronCash.

Building the Transaction

On any Internet-connected computer, set-up the **Watch-Only Electrum Wallet** (this setup can be re-used from previous transactions):

1. Download and install Electrum (see prior steps for information on where to do it)

2. Open Electrum

3. Select `Auto connect` and click `Next`

4. If you wish name this wallet and click `Next`

5. Select `Multi-signature wallet` and click `Next`

6. Select your `M-of-N` policy. The top bar, `cosigners` is the `N` and the lower bar, `Required signatures` is the `M`. Click `Next`.

7. Select `Use a master key` and click `Next`

8. Scan your Bitcoin Master Public Key QR Code

9. Select `Enter cosigner key` and click `Next`

10. Scan the rest of the Bitcoin Master Public Key QR Codes

11. Do not enter a password, just click `Next` You should be able to see the balance in your bitcoin cold storage address

12. Click on `Send`, enter the destination address and the amount

13. Click on `Preview`

14. Click on the `QR Code` icon

15. Print the QR Code. This is the transaction that will need to be signed by the Quarantined computers of M signatories.

Signing the transaction

The following steps will need to be done by M signatories:

1. Gather required information
   a. Make sure you have your Cold Storage Information Packets on hand (you'll need the 24-word BIP39 Mnemonic).
      i. You will also need to coordinate with M-1 signatories who will in turn need their Cold Storage Information Packets.
   b. You should print the QR Code of the transaction to be signed.

2. Execute Section VI of the Setup Protocol to prepare your quarantined workspace.

3. Sign the transaction

   On any of your Quarantined computers:

   a. Import QR code data
      i. Start Electrum

      ```
      $ ~/apps/electrum-3.3.6-x86_64.AppImage
      ```

      ii. Load the wallet and your key
         1. Choose `auto connect` and click `Next`.
         2. Choose whatever name for the wallet and click `Next`
         3. Select `Multi-signature wallet`

4. Select your `M-of-N` policy. The top bar, `cosigners` is the `N` and the lower bar, `Required signatures` is the `M` . Click `Next` .

5. Select `I already have a seed` and click `Next`

6. Click on `Options` , select `BIP39 seed` and click `OK` .

7. Enter your BIP39 seed phrase and click `Next` .

8. Leave the default settings selected and click `Next` .

9. Click `Next`

10. Select `Enter cosigner key`

11. Use the QR Code reader to read the next signatories' Bitcoin Master Public Key QR Code

12. Repeat steps 9 to 11 for the other N-2 signatories

13. Do not enter a password and click `Next`

iii. Sign the Transaction

1. From the top bar menu, select `Tools` -> `Load Transaction` -> `From QR Code`

2. Scan the printed QR Code of the transaction that needs signing

3. Verify that the outputs of the transaction make sense

   i. You should see one output for the desired amount and destination.

   ii. You will likely see a second output going to a change address belonging to the same wallet. You can verify that the change addresses belong to the wallet in the `Addresses` tab

4. Click on `Sign`

5. You should see the `Status` of the transaction, indicating how many signatories have signed

4. Visually hide all critically sensitive data.

We'll be using a smartphone with a live Internet connection to read QR codes from the quarantined computer screens. Any malware (or a malicious QR reader app) could steal sensitive data if it is not visually hidden.

This step is important. Failing to execute it properly creates a substantial security risk.

a. Put your Cold Storage Information Packets out of sight – this prevents a smartphone camera from accidentally seeing them.

b. Make sure your BIP39 Mnemonic is nowhere to be seen in your Computer screen or in the physical world.

5. Extract the signed transaction from the quarantined environment.

a. Remove a smartphone from the Faraday bag and turn it on.

b. Open your QR Reader App

c. **On the Quarantined computer**, display the signed transaction QR code by clicking the `QR Icon` on the bottom left of the screen.

d. Use the smartphone's QR code reader to read the QR code.

e. Take a picture of the QR code using the smartphone.

f. Send both the text version and the QR Code's picture to the next signatory using a messaging app which the signatory will be able to access from a laptop. If you are the last signatory, send the contents to yourself

6. Shut down **all** quarantined computers entirely. As a precaution against side channel attacks, the quarantined computers should not be active except when they absolutely need to be.

```
$ sudo shutdown now
```

The recommended Acer laptop may require you to hold down the power button for several seconds to complete the shutdown.

7. Repeat the steps above until M signatories have signed the transaction

a. The status of Electrum should change to `Signed` once all required signatories have signed the transaction

Verifying and broadcasting the transaction

On any Internet-connected computer, the last signatory should:

1. Access the QR Code of the fully signed transaction she previously sent herself.

2. Open the watch-only Bitcoin Electrum Wallet

3. Load the transaction

a. Click on `Tools` -> `Load Transaction` -> `From QR Code`

b. Scan the QR Code of the fully signed transaction

4. Under "Outputs"

a. **Verify the destination address is correct.**

b. Verify the amount going to the destination address is correct.

c. If you did *not* withdraw all funds from these unspent transactions, you'll also see a second output which "sends" the remainder of the funds "back" to your cold storage address. This is necessary; it's how Bitcoin is designed to operate.

5. Execute the transaction.

a. Click on `Broadcast`

b. You should see a `Payment sent` pop up with the Transaction ID

c. Copy-paste the Transaction ID

6. Verify the withdrawal on the public blockchain.

a. Go to blockchair (https://www.blockchair.com/), paste the Transaction ID into the search bar, and press Enter.

b. Within a couple of minutes (and often much faster), you should be able to refresh this page and see your transaction listed as "Unconfirmed".

c. Periodically refresh the page until you see the funds moved from "Unconfirmed" to be reflected in "Balance". This generally happens within 15 minutes; if the Bitcoin network is unusually congested, it may take longer.

d. You should be able to verify this in the Watch-Only Electrum wallet as well

7.3. Withdrawing Ethereum & ERC20 Tokens

In this section, we construct a "signed transaction" in our quarantined environments, verify it, and then use QR codes to extract it from the quarantined environment to pass on to additional quarantined environments for additional signatures and eventually extract it for execution.

This protocol is divided into two sub-protocols: Proposing Transfers and Confirming Transfers. The flow is one signatory will propose and the rest will confirm.

Gather the required information

Every signatory needs to execute this section

1. Make sure you have your Cold Storage Information Packets on hand (you'll need the 24-word BIP39 Mnemonic).

   a. You will also need to coordinate with M-1 signatories who will in turn need their Cold Storage Information Packets.

On any Internet-connected computer:

1. Find your current account's Nonce

   a. Navigate to etherscan.io (https://etherscan.io) and enter your Ethereum Public Address, also known as **MESSAGE E** on your Cold Storage Information Packet.

   b. Find your last outgoing transaction and click on it, find the `Nonce` value and write it down on a piece of paper. If there are no outgoing transactions, then record the number `0`.

2. Navigate to ethgasstation.info (https://ethgasstation.info) and record the recommended gas price in Gwei on the same piece of paper

3. If you are **Proposing A Transfer**:

   a. Install the required software (on a Mac, only required the first time):

   ```
   $ /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/
   Homebrew/install/master/install)"
   $ brew install qrencode
   ```

b. Create and print QR code with the destination address

    i. On terminal

```
$ qrencode -s 5 -o destination.png <ENTER_DESTINATION_ADDRESS>
```

    ii. Open the QR Code:

```
$ open destination.png
```

    iii. Print the QR Code

c. If you are withdrawing from an ERC20 Token, make sure you repeat the step above for the ERC20 Token Contract Address. You can find the ERC20 Contract Address on etherscan.io **Make sure you also note the Contract Decimals (usually 18)** in the piece of paper.

d. On the same piece of paper carefully write down the amount of ETH or ERC20 Tokens that you are withdrawing.

4. If you are **Confirming a Transfer**:

a. Obtain the Transfer ID:

    i. Open multisigweb on an internet connected device

    ii. Import the `Ethereum Cold Wallet Address`.

    iii. Open the Wallet and navigate to the `Multisig Transactions` section

    iv. Verify the details (amount + destination) of the Transfer you are looking to confirm and write down on the piece of paper the ID (left-most column)

5. Finally, remember that each signatory's Ethereum account will be making transactions on the blockchain, so make sure each account has some ETH balance.

Proposing Transfers

Only one signatory needs to propose transfers. If a signatory has already proposed a transfer and you need to confirm it, see section below on Confirming Transfers

Again, the following steps will need to be done by 1 signatories:

1. Execute Section VI of the Setup Protocol to prepare your quarantined workspace.
2. Create and sign the transaction

On the Q1 computer:

a. Create your ETH account keystore file

    i. Execute the cryptoglacier script

```
$ node ~/cryptoglacier/setup.js --ether
```

    ii. You will be prompted to enter your `24-word BIP39 Mnemonic`

    iii. If this is your first transaction, please see the "First transaction" section below

iv. The script will write an `ethereum.json` file to your `~/cryptoglacier` directory

b. Set the gas price on multisigweb

   i. Start multisigweb

```
$ multisigweb
```

   ii. Accept the TOS and Privacy Policy

   iii. Select `Light Wallet`

   iv. Click on `Settings` tab

   v. Enter the gas price in Wei (multiply by 10^9)

   vi. Under `Wallet factory contract` select `Custom contract`

   vii. Click `Update settings`

   viii. Exit the program (Upper left Menu, Application -> Quit)

c. Import your keystore file into multisigweb

   i. Start multisigweb

```
$ multisigweb
```

   ii. Click on `Accounts` tab

   iii. Click on `Import`

   iv. Select `Browse` and select the `ethereum.json` file in the `cryptoglacier` directory

   v. Enter `cryptoglacier` as your password

   vi. Name the account as you wish

   vii. Import Account

d. Import required data

   i. Start the QR code reader

      1. On Terminal, open a new tab with `Ctrl+Shift+T`

      2. Start zbarcam

```
$ zbarcam
```

      A window will appear with your laptop's video feed.

   ii. Scan the Ethereum Cold Wallet Address from your Cold Storage Information Packet.

      1. Hold the QR code up to the webcam

      2. When a green square appears around the QR code on the video feed, it has been successfully read.

      3. Verify the decoded QR code is shown in the terminal window. Example:

```
QR-Code:0xe46295248fab5f8749af13eeea7021aec098c4ba
```

    4. Copy-paste the data into the Quarantined Scratchpad under a "CONTRACT ADDRESS" header

  iii. Repeat the step above for the destination address QR code with a "DESTINATION ADDRESS" header in the Quarantined Scratchpad

  iv. If you are withdrawing an ERC20 Token, repeat the step above for the ERC20 Token Contract Address QR code with a "TOKEN ADDRESS" header in the Quarantined Scratchpad

e. Import your Wallet Contract into multisigweb

  i. Click on `Wallets` tab

  ii. Click on `Add`

  iii. Select `Restore deployed wallet` and click `Next`

  iv. Enter any name you wish and then copy the Ethereum Cold Wallet Address from your Quarantined Scratchpad and click `Ok`

f. Create the transaction

  i. Click on the Name of the Wallet

  ii. For ETH Transactions

    1. Click on `Add` next to `Execute offline`

    2. Enter the `Destination` (from the Scratchpad) and the `Amount` (from the piece of paper) and click `Sign offline`

  iii. For ERC20 Transactions

    1. Click the `Add` button in the `Tokens` section

    2. Enter the "TOKEN ADDRESS" in the `Address` field

    3. Enter any `Symbol`

    4. Enter the `Decimals` from the piece of paper and click `Ok`

    5. Scroll through the `Tokens` until you find the `Symbol` you just added. Click `Withdraw`

    6. Enter the `Amount` (from the piece of paper) and the `Destination` (from the Scratchpad) and click `Sign offline`

  iv. Enter the `Nonce` you recorded on the piece of paper and click `Ok`

  v. Enter `cryptoglacier` as the password and click `Ok`

  vi. You will receive the hex code. Select it and click `Copy`

g. Build the QR Code for the transaction

```
$ qrencode -o tx.png [PASTE USING CTRL+SHIFT+V]
```

h. Display the QR Code

```
$ eog tx.png
```

      i. If this is your first transaction, please also display the QR Code for the first transaction

           i. On Terminal, open a new tab with `Ctrl+Shift+T`

           ii. Display the QR Code for the first transaction

```
$ eog tx0.png
```

3. Visually hide all critically sensitive data.

   We'll be using a smartphone with a live Internet connection to read QR codes from the quarantined computer screens. Any malware (or a malicious QR reader app) could steal sensitive data if it is not visually hidden.

   This step is important. Failing to execute it properly creates a substantial security risk.

   a. Put your `Cold Storage Information Packets` out of sight – this prevents a smartphone camera from accidentally seeing them.

4. Extract the signed transaction from the quarantined environment.
   a. QR reader setup
      i. Remove a smartphone from the Faraday bag and turn it on.
      ii. If the smartphone doesn't already have a QR code reader on it, install one.

         Any reader is fine as long as it can read all types of QR codes, but here are recommendations we've tested with this protocol: iOS (https://itunes.apple.com/us/app/qr-reader-for-iphone/id368494609?mt=8), Android (https://play.google.com/store/apps/details?id=com.application_4u.qrcode.barcode.scanner.reader.flashlight&hl=en).

   b. Transfer the signed transaction data to a non-quarantined computer.
      i. Use the smartphone's QR code reader to read the QR code.
      ii. Visually inspect that the hex code is the same and send it to yourself using a messaging app which you can access from a laptop.

5. Shut down the quarantined computer entirely. As a precaution against side channel attacks, the quarantined computers should not be active except when they absolutely need to be.

```
$ sudo shutdown now
```

   The recommended Acer laptop may require you to hold down the power button for several seconds to complete the shutdown.

6. Skip to the section "Broadcasting and verifying transactions" below

Confirming Transfers

M-1 signatories need to confirm transfers. Only Transfers that have been proposed can be confirmed.

If you are a signatory and are looking to confirm a transfer:

1. Execute Section VI of the Setup Protocol to prepare your quarantined workspace.
2. Sign the confirmation transaction

   On the Q1 computer:

   a. Create your ETH account keystore file
      i. Execute the cryptoglacier script

      ```
      $ node ~/cryptoglacier/setup.js --ether
      ```

      ii. You will be prompted to enter your 24-word BIP39 Mnemonic
      iii. If this is your first transaction, please see the "First transaction" section below
      iv. The script will write an `ethereum.json` file to your `~/cryptoglacier` directory
   b. Set the gas price on multisigweb
      i. Start multisigweb

      ```
      $ multisigweb
      ```

      ii. Accept the TOS and Privacy Policy
      iii. Select `Light Wallet`
      iv. Click on `Settings` tab
      v. Enter the gas price in Wei (multiply by 10^9)
      vi. Under `Wallet factory contract` select `Custom contract`
      vii. Click `Update settings`
      viii. Exit the program (Upper left Menu, Application -> Quit)
   c. Import your keystore file into multisigweb
      i. Start multisigweb

      ```
      $ multisigweb
      ```

      ii. Click on `Accounts` tab
      iii. Click on `Import`
      iv. Select `Browse` and select the `ethereum.json` file in the `cryptoglacier` directory

v. Enter `cryptoglacier` as your password

vi. Name the account as you wish

vii. Import Account

d. Import required data

    i. Start the QR code reader

        1. On Terminal, open a new tab with `Ctrl+Shift+T`

        2. Start zbarcam

```
$ zbarcam
```

        A window will appear with your laptop's video feed.

    ii. Scan the `Ethereum Cold Wallet Address` from your `Cold Storage Information Packet`.

        1. Hold the QR code up to the webcam

        2. When a green square appears around the QR code on the video feed, it has been successfully read.

        3. Verify the decoded QR code is shown in the terminal window. Example:

```
QR-Code:0xe46295248fab5f8749af13eeea7021aec098c4ba
```

        4. Copy-paste the data into the Quarantined Scratchpad under a "CONTRACT ADDRESS" header

e. Import your Wallet Contract into multisigweb

    i. Click on `Wallets` tab

    ii. Click on `Add`

    iii. Select `Restore deployed wallet` and click `Next`

    iv. Enter any name you wish and then copy the `Ethereum Cold Wallet Address` from your Quarantined Scratchpad and click `Ok`

f. Confirm the transaction

    i. Click on the Name of the Wallet

    ii. Click `Confirm offline`

    iii. Enter the `Transaction ID` as written on the piece of paper and click `Confirm offline`

    iv. Enter the `Nonce` you recorded on the piece of paper and click `Ok`

    v. Enter `cryptoglacier` as the password and click `Ok`

    vi. You will receive the hex code. Select it and click `Copy`

g. Build the QR Code for the transaction

```
$ qrencode -o tx.png [PASTE USING CTRL+SHIFT+V]
```

h. Display the QR Code

```
$ eog tx.png
```

  i. If this is your first transaction, please also display the QR Code for the first transaction

    i. On Terminal, open a new tab with `Ctrl+Shift+T`

    ii. Display the QR Code for the first transaction

```
$ eog tx0.png
```

3. Visually hide all critically sensitive data.

We'll be using a smartphone with a live Internet connection to read QR codes from the quarantined computer screens. Any malware (or a malicious QR reader app) could steal sensitive data if it is not visually hidden.

This step is important. Failing to execute it properly creates a substantial security risk.

  a. Put your Cold Storage Information Packets out of sight – this prevents a smartphone camera from accidentally seeing them.

4. Extract the signed transaction from the quarantined environment.

  a. QR reader setup

    i. Remove a smartphone from the Faraday bag and turn it on.

    ii. If the smartphone doesn't already have a QR code reader on it, install one.

Any reader is fine as long as it can read all types of QR codes, but here are recommendations we've tested with this protocol: iOS (https://itunes.apple.com/us/app/qr-reader-for-iphone/id368494609?mt=8), Android (https://play.google.com/store/apps/details?id=com.application_4u.qrcode.barcode.scanner.reader.flashlight&hl=en).

  b. Transfer the signed transaction data to a non-quarantined computer.

    i. Use the smartphone's QR code reader to read the QR code(s).

    ii. Visually inspect that the hex code is the same and send it to yourself using a messaging app which you can access from a laptop.

5. Shut down the quarantined computer entirely. As a precaution against side channel attacks, the quarantined computers should not be active except when they absolutely need to be.

```
$ sudo shutdown now
```

The recommended Acer laptop may require you to hold down the power button for several seconds to complete the shutdown.

6. Follow the section "Broadcasting and verifying transactions" below

Broadcasting and verifying transactions

On any Internet-connected computer:

1. Send the Transaction
   a. Access the <mark>hex Code of the fully signed transaction</mark> you sent yourself from your smartphone previously.
   b. Open alpha.myetherwallet.com/pushTx (https://alpha.myetherwallet.com/pushTx) or etherscan.io/pushtx (https://etherscan.io/pushtx) and paste the hex code in the `Signed Transaction` box
   c. Click on `Send Transaction`
   d. Confirm the contents of the transaction and click `Yes, I am sure! Make transaction.`
   e. Wait until the transaction gets into an ethereum block by checking etherscan.io (https://etherscan.io)
2. Verify the transaction status by opening multisigweb on an internet connected device and importing the <mark>Ethereum Cold Wallet Address</mark>

First Transaction

There is a known bug (https://github.com/gnosis/MultiSigWallet/issues/298) that prevents Multisigweb from signing the very first ETH transaction which would have a nonce of zero. While Multisigweb is fixed, `setup.js` script will prompt you to see if this is your first transaction. If this is your first transaction, the script will create a `tx0.png` file that you will also need to extract from your quarantined environment and broadcast to the network.

7.4. Withdrawing XRP

In this section, we construct a "signed transaction" in our quarantined environments, verify it, and then use QR codes to extract it from the quarantined environment to pass on to additional quarantined environments for additional signatures and eventually extract it for execution.

This protocol requires one signatory to **Create a transaction** and then M-1 signatories to **Sign a transaction**.

Gather the required information

1. Make sure you have your <mark>Cold Storage Information Packets</mark> on hand (you'll need the <mark>24-word BIP39 Mnemonic</mark>).

   a. You will also need to coordinate with M-1 signatories who will in turn need their <mark>Cold Storage Information Packets</mark>.

2. If you are the first signatory and will **Create a transaction**, then on any Internet-connected computer:

   a. Find your address' sequence number

      i. Navigate to https://xrpl.org/xrp-ledger-rpc-tool.html (https://xrpl.org/xrp-ledger-rpc-tool.html), enter your <mark>Ripple Cold Storage Address</mark> and click `Get info`.

      ii. On the `Result` section, expand the `account_data` information and record the `Sequence` number on a piece of paper

   b. On the same piece of paper carefully write down the amount of XRP that you are withdrawing.

   c. Create and print the QR codes with the <mark>Ripple Cold Storage Address</mark> and the destination address

      i. Install the required software (on a Mac, only required the first time). On terminal:

      ```
      $ /usr/bin/ruby -e "$(curl -fsSL https://
      raw.githubusercontent.com/Homebrew/install/master/install)"
      $ brew install qrencode
      ```

      ii. Create the QR Code for the <mark>Ripple Cold Storage Address</mark>

      ```
      $ qrencode -s 5 -o source.png
      <PASTE_RIPPLE_COLD_STORAGE_ADDRESS>
      ```

      iii. Open the QR Code and paste it in a word-editing doc under the header "SOURCE"

      ```
      $ open source.png
      ```

      iv. Create the QR Code for the destination address

      ```
      $ qrencode -s 5 -o destination.png <PASTE_DESTINATION_ADDRESS>
      ```

      v. Open the QR Code and paste it in a word-editing doc under the header "DESTINATION"

      ```
      $ open destination.png
      ```

vi. Print the word editing doc

d. If the transfer requires a Destination Tag, please write it down carefully on the piece of paper

3. If you are a signatory that will **Sign a transaction**, please make sure you write down on a piece of paper

a. The Destination Address

b. The Destination TAG

c. The Transaction amount

Create a new Transaction

Only one signatory needs to create a new transaction. If another signatory has already created a transaction and you need to sign over it, see section below on Sign a Transaction

Again, the following steps will need to be done by 1 signatories:

1. Execute Section VI of the Setup Protocol to prepare your quarantined workspace.
2. Create and sign the transaction

On the Q1 computer:

a. Import required data

i. Start zbarcam

```
$ zbarcam
```

A window will appear with your laptop's video feed.

ii. Scan the destination address QR code

1. Hold the QR code up to the webcam
2. When a green square appears around the QR code on the video feed, it has been successfully read.
3. Verify the decoded QR code is shown in the terminal window. Example:

```
QR-Code: r8HgVGenRTAiNSM5iqt9PX2D2EczFZhZr
```

4. Copy-paste the data into the Quarantined Scratchpad under a "DESTINATION ADDRESS" header

b. Create Transaction

i. Execute the cryptoglacier script

```
$ cd ~/cryptoglacier/
$ node setup.js --xrp
```

      ii. You will be prompted to enter your `24-word BIP39 Mnemonic`

      iii. The script will ask you a few questions and write a `ripple_tx.png` file to your `~/cryptoglacier` directory

  c. Display the QR Code

```
$ eog ripple_tx.png
```

3. Visually hide all critically sensitive data.

   We'll be using a smartphone with a live Internet connection to read QR codes from the quarantined computer screens. Any malware (or a malicious QR reader app) could steal sensitive data if it is not visually hidden.

   This step is important. Failing to execute it properly creates a substantial security risk.

   a. Put your `Cold Storage Information Packets` out of sight – this prevents a smartphone camera from accidentally seeing them.

4. Extract the signed transaction from the quarantined environment.

   a. QR reader setup

       i. Remove a smartphone from the Faraday bag and turn it on.

       ii. If the smartphone doesn't already have a QR code reader on it, install one.

       Any reader is fine as long as it can read all types of QR codes, but here are recommendations we've tested with this protocol: iOS (https://itunes.apple.com/us/app/qr-reader-for-iphone/id368494609?mt=8), Android (https://play.google.com/store/apps/details?id=com.application_4u.qrcode.barcode.scanner.reader.flashlight&hl=en).

   b. Transfer the signed transaction data to a non-quarantined computer.

       i. Use the smartphone's QR code reader to read the QR code.

       ii. Visually inspect that the json is the same

       iii. Take a picture of the QR code and send it to the next signatory using a messaging app which they can access from a laptop.

5. Shut down the quarantined computer entirely. As a precaution against side channel attacks, the quarantined computers should not be active except when they absolutely need to be.

```
$ sudo shutdown now
```

The recommended Acer laptop may require you to hold down the power button for several seconds to complete the shutdown.

Sign a Transaction

M-1 signatories need to sign the transaction.

If you are a signatory and are looking to sign a transfer:

1. Execute Section VI of the Setup Protocol to prepare your quarantined workspace.
2. Sign the confirmation transaction

    On the Q1 computer:

    a. Import required data

        i. Start zbarcam

        ```
        $ zbarcam
        ```

        A window will appear with your laptop's video feed.

        ii. Scan the QR code you received from the prior signatory

            1. Hold the QR code up to the webcam
            2. When a green square appears around the QR code on the video feed, it has been successfully read.
            3. Verify the decoded QR code is shown in the terminal window. Example:

            ```
            QR-Code:{"Account":"rp3rEms99VB7uMyU8GnGyPmo6uejJ4XbEV","Destination":"rp3rEms99VB7uMyU8GnGyPmo6uejJ4XbEV","DestinationTag":5,"Amount":"600000000","Sequence":40,"TransactionType":"Payment","Fee":"100","SigningPubKey":"","Signers":[{"Signer":{"Account":"rp3rEms99VB7uMyU8GnGyPmo6uejJ4XbEV","SigningPubKey":"0368C9DEE202196D3FFEA2A81F7BBAE8673775F54B286379F8E7C3AB31B53B4666","TxnSignature":"304502FB6C45A46912E522100E346752EF9E816D55F63F3F7FC010D80CFD1B0CEFD2672ACD7D562B575125094E602200B6243B4575D044984A10020A9B26BDE2347E7AB8B7E076"}}]}
            ```

            4. Copy-paste the data into the Quarantined Scratchpad under a "TX TO SIGN" header
            5. Inspect the transaction and make sure the following details are correct:
                - **Destination Tag**
                - **Destination Address**
                - **Amount**

    b. Execute the cryptoglacier script

    ```
    $ cd ~/cryptoglacier/
    $ node setup.js --xrp
    ```

        i. You will be prompted to enter your 24-word BIP39 Mnemonic

   ii. The script will ask you a few questions, including to paste the "TX TO SIGN" and the script will write a new `ripple_tx.png` file to your `~/cryptoglacier` directory

  c. Display the QR Code

```
$ eog ripple_tx.png
```

3. Visually hide all critically sensitive data.

  We'll be using a smartphone with a live Internet connection to read QR codes from the quarantined computer screens. Any malware (or a malicious QR reader app) could steal sensitive data if it is not visually hidden.

  This step is important. Failing to execute it properly creates a substantial security risk.

  a. Put your Cold Storage Information Packets out of sight – this prevents a smartphone camera from accidentally seeing them.

4. Extract the signed transaction from the quarantined environment.

  a. QR reader setup

   i. Remove a smartphone from the Faraday bag and turn it on.

   ii. If the smartphone doesn't already have a QR code reader on it, install one.

   Any reader is fine as long as it can read all types of QR codes, but here are recommendations we've tested with this protocol: iOS (https://itunes.apple.com/us/app/qr-reader-for-iphone/id368494609?mt=8), Android (https://play.google.com/store/apps/details?id=com.application_4u.qrcode.barcode.scanner.reader.flashlight&hl=en).

  b. Transfer the signed transaction data to a non-quarantined computer.

   i. Use the smartphone's QR code reader to read the QR code.

   ii. Visually inspect that the json is the same

   iii. Take a picture of the QR code and send it to the next signatory using a messaging app which they can access from a laptop. If you are the last signatory, send the json **contents** to yourself using a messaging app that you can access from a laptop.

5. Shut down the quarantined computer entirely. As a precaution against side channel attacks, the quarantined computers should not be active except when they absolutely need to be.

```
$ sudo shutdown now
```

The recommended Acer laptop may require you to hold down the power button for several seconds to complete the shutdown.

Broadcasting the transactions

On any Internet-connected computer:

1. Send the Transaction
   a. Access the final JSON of the fully signed transaction you sent yourself from your smartphone previously.
   b. Open xrpl.org/websocket-api-tool.html (https://xrpl.org/websocket-api-tool.html) and paste the json string in the `Request` box
   c. Click on `Send request`
2. Verify the transaction
   a. You can check the result of your transaction by visiting bithomp (https://bithomp.com/explorer/)

8. Balance and maintenance

8.1. Check your balance

The Viewing Protocol is a simple procedure for viewing your balance of funds currently in one cold storage address.

1. Open your electronic copy of the Cold Storage Information Page (see Section II for details). If you've lost access to it, you'll need to recreate a new electronic copy by transcribing one of the hardcopies (stored with each private key) by hand.
2. Go to Blockr (https://www.coinbase.com/), paste your cold storage address into the search bar, and press Enter.
3. You'll be taken to a page that says "Bitcoin Address" at the top, with your cold storage address listed underneath.
4. Your balance will be listed on the page.

8.2. Maintenance

The Maintenance Protocol is designed to minimize the risk of losing funds due to:

- **Loss of private keys**: Obviously if too many keys are compromised or lost (due to theft, damage, or misplacement), your funds are lost. It's therefore important to know ASAP if even a single key is lost, so you can generate a replacement before more keys are lost.
- **New security threats**: Glacier may contain weaknesses which are currently undiscovered – perhaps related to attacks which are not part of the current security landscape.
- Bit rot (https://en.wikipedia.org/wiki/Software_rot): The Withdrawal Protocol depends on the availability of certain software (including not just the applications themselves, but also software libraries those applications depend on), plus hardware and networks that are compatible with that software. If your funds are in storage for a long time, the withdrawal tools may become obsolete and no longer function.

We recommend the Maintenance Protocol be executed **six months** after the initial deposit into cold storage, and **annually** thereafter.

1. Execute the Viewing Protocol to view the balance of the cold storage address and ensure that it is as expected.
2. Check for Glacier security upgrades
   a. Access the latest full release of Glacier (*not* just the protocol document) at https://github.com/GlacierProtocol/GlacierProtocol/releases (https://github.com/GlacierProtocol/GlacierProtocol/releases).
   b. Open the protocol document (Glacier.pdf) within the ZIP file.

c. In Appendix E, locate release notes for all versions since the last time you executed the Maintenance Protocol (or if it's the first time, since the Glacier version specified on your Cold Storage Information Page).

d. See whether any of those releases recommend any security upgrades. (Any recommendations are prominently mentioned at the top of the notes for each version.)

e. Whether or not you decide to upgrade, review the errata for the version of Glacier you are using at https://github.com/GlacierProtocol/GlacierProtocol/releases (https://github.com/GlacierProtocol/GlacierProtocol/releases).

3. Have each Cold Storage Information Packet visually inspected (either by you, or the signatory that has it in custody):

a. Verify the packet is in its expected location.

b. Verify the packet's location is secured as expected (any locks in working order, etc.)

c. Verify the packet is in good physical condition.

d. Verify the tamper-resistant seals appear to be intact.

4. Execute the Withdrawal Protocol for a small test amount.

5. Create a reminder for yourself in one year to execute the Maintenance Protocol again. (If you don't have a reminder system you trust, find one on the web.)

9. Extend Glacier

9.1. Extend Glacier security

Glacier is designed to provide strong protection for almost everyone – even those storing many millions of dollars.

However, it is not designed to provide adequate protection for truly exceptional circumstances, such as a targeted attack/surveillance effort (electronic or physical) by a well-resourced criminal organization. This appendix briefly outlines additional measures one might consider if further security were needed above and beyond those in the formal Glacier protocol.

We do not recommend considering these measures unless you feel you have a strong need. This list is neither complete nor are the practices cost-effective for almost any circumstances. In addition, implementing these measures incorrectly may decrease security rather than increase it.

Digital software security

- **Verify GnuPG installation**: When downloading a new copy of GnuPG on the setup computer, one would ideally also verify the integrity of the download using the signed checksum. This requires having a pre-existing trusted installation of GnuPG available for verifying the checksum signature.
- **Cross-network checksum sourcing**: Using two different computers on two different networks, obtain all the software checksums from the Internet and verify they are identical, to reduce the risk that the checksums are being compromised by a man-in-the-middle attack.
- **Quarantined checksum verification**: Verify all USB checksums on the quarantined computers to eliminate any risk that software was altered between checksum verification on the Setup Computers and when the USB is used in the quarantined environment. The only reason Glacier doesn't currently do this is because the process of verifying the App USB checksums happens as part of Ubuntu's apt-get application, which requires network connectivity. It can be done by hand without apt-get, but it's significantly more involved and so was not included in the protocol.
- **Greater differentiation of quarantined environments**: Instead of simply using different hardware in each quarantined environment, use different software (including a non-Linux-derived OS and a different Bitcoin wallet), different smartphones (and different smartphone software, i.e. QR code readers). Different software stacks eliminate the risk that a software bug or vulnerability may generate a flawed key. See the design document for details on why this risk is small enough to justify leaving it unaddressed in the formal protocol.
- **Dedicated pair of environments for each private key**: Use extra environments such that each environment only touches one key both when generating keys and signing transactions. Expand the definition of "environment" to include the physical location in which Glacier is executed. This way, compromising one environment will only compromise one key.

- **Deposit transaction verification**: If depositing bitcoins out of self-managed storage, don't immediately send a transaction directly from one's own wallet software. Instead, first export a raw signed transaction, and use a service like Blockr to verify the transaction is actually sending the funds to the correct address.
- **Avoid software random number generators**: Use a hardware random number generator (https://en.wikipedia.org/wiki/Comparison_of_hardware_random_number_generators) instead.

Side channel security

- **Faraday cage**: Use a Faraday cage (https://en.wikipedia.org/wiki/Faraday_cage) to protect against electromagnetic side channels (example (https://www.usenix.org/legacy/event/sec09/tech/full_papers/vuagnoux.pdf)). Faraday cages can be self-built (https://www.thesurvivalistblog.net/build-your-own-faraday-cage-heres-how/) or professionally built (https://www.faradaycages.com/server-rooms).

- **No QR codes**: Reading and relaying QR codes to a printer requires a networked device, such as a smartphone, which could potentially receive data from side channels. Instead of using QR codes, copy all redemption scripts and transactions by hand, and keep all nearby smartphones powered off and in Faraday bags through protocol execution. Note that transcription of redeem scripts and transactions is not only a painstakingly long process, but dangerously vulnerable to human error: any mistakes in the initial transcription & storage of the redemption script will cause all funds to be lost.

Hardware security

- **Purchase factory-new Setup Computers**: Don't use existing computers for your Setup Computers. Purchase them factory-new, and never use them on the same network (to reduce the risk of infection by identical malware).
- **Use firmware-protected USBs**: Some USBs have extra features to protect against malware targeting their firmware (e.g. BadUSB (https://arstechnica.com/information-technology/2014/07/this-thumbdrive-hacks-computers-badusb-exploit-makes-devices-turn-evil/) or Psychson (https://github.com/brandonlw/Psychson)). Examples include Kanguru drives (https://www.kanguru.com/secure-storage/defender-secure-flash-drives.shtml) and IronKey drives (http://www.ironkey.com/en-US/encrypted-storage-drives/250-basic.html).
- **Purchase a factory-new printer**: Printers can have malware, which could conceivably interfere with printing the hardcopy of the Glacier document. Use a new printer for printing the Glacier document. Choose one without wireless capabilities.
- **Purchase non-recommended equipment**: Don't purchase any of the suggested equipment linked in this document – if Glacier achieves widespread adoption, that particular equipment may be

targeted for sabotage to undermine the protocol (e.g. loaded dice, malware pre-installed on computers, etc.) Select your own comparable equipment from different manufacturers.

- **Purchase from stores**: Buy all equipment from stores, to reduce the risk it will be tampered with before it is delivered to you (https://arstechnica.com/tech-policy/2014/05/photos-of-an-nsa-upgrade-factory-show-cisco-router-getting-implant/). Don't choose the stores nearest your home or office. Don't leave the equipment unattended until you are done using it.
- **Improved tamper-evident seals on laptops**: After you are done using the laptop, paint over the hinge joints and cover screws with glitter nail polish and take a picture. The randomness of the glitter is difficult to recreate, so if the laptop is tampered with, you can see it, and know not to use it for future protocol operations.
- **Secure or destroy quarantined hardware after use**: If sensitive data was somehow stored on quarantined hardware's permanent media due to a protocol error or malware, then physical theft of the hardware becomes a concern. Store the hardware somewhere secure such as a vault, or physically destroy it first. Glacier is designed to only use a RAM disk, but it's possible some data is saved to permanent media (hard drive or USB) without us realizing it.

Paper key security

- **Paper key encryption**: Encrypt the contents of your paper keys using BIP38 (https://github.com/bitcoin/bips/blob/master/bip-0038.mediawiki) to further protect against physical theft. Note that the question of how to securely store the passphrase is non-trivial. It should be unique and hard to guess, which means it is non-trivial to remember. If you are confident you can remember it, storing it only in your own memory will not address estate planning needs. If you record it on paper, you need to make sure those papers are stored securely – they should not be stored with the keys, and there should be a process for checking on them periodically to make sure they are not lost or damaged.
- **Durable storage medium**: TerraSlate paper is extremely rugged, but you might also consider laminating the paper for additional protection. You'll need a thermal laminator (http://a.co/cZBN1YU) and laminating pouches (http://a.co/ifISzje). An even more durable approach would be to engrave the private keys in metal.
- **High-security vaults**: Store keys in high-security vaults that are more resistant to theft and disaster. See example (http://mountainvault.net/).
- **Geographically distributed storage**: Store keys in distant cities for resilience against a major disaster that wipes out all keys at once.
- **Multiple fund stores**: Mitigate risk by splitting funds across more than one Bitcoin address, each secured using Glacier, and don't keep printed keys from different store in the same place.

Personal security

- **Unique protocol execution site**: Rather than executing Glacier at your home, office, or anywhere else you frequent, choose a new location (e.g. a hotel) that is unlikely to have compromised or surveillance devices present.
- **Avoid location tracking**: To avoid surveillance (including from adjacent rooms, via side channels like radio waves), take steps to avoid location tracking when executing Glacier. Don't carry a GPS-enabled smartphone with you, don't use credit cards for purchases, etc.
- **Deliver keys by hand**: Don't use couriers or [phones (https://www.cbsnews.com/news/60-minutes-hacking-your-phone/)](https://www.cbsnews.com/news/60-minutes-hacking-your-phone/) to send keys to trusted associates. Hand-deliver them personally or using a trusted party.
- **Conventional personal security**: Home surveillance systems, bodyguards, etc.

9.2. Possible improvements to CryptoGlacier

Don't store electronic copy of Cold Storage Information Page

Glacier recommends stores an electronic copy of the Cold Storage Information Page for easy copy-pasting for subsequent deposits or withdrawals. However, this is slightly less secure & complicated – and it's still a good idea to check a physical copy of the Cold Storage Information Page to verify the electronic copy hasn't been tampered with.

Printing QR codes on the Cold Storage Information Page would be another way to avoid the need to manually transcribe the deposits and withdrawals

No Address Reuse

Currently, Glacier reuses addresses for both depositing and withdrawing funds. As discussed in the protocol design document, this has both privacy and security implications.

This could be implemented with HD wallets, which would allow one to generate one master key and then use new derived addresses for each deposit or change transaction. Bitcoin Core does not yet support importing user-generated HD wallets in a straightforward way.

Avoiding address re-use would also prevent the use of a test withdrawal. Careful consideration would need to be given as to whether there is another way to safely test funds access, perhaps using something like the signrawtransaction Bitcoin Core RPC.

BIP39 Mnemonic Support

BIP39 (https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki) supports the creation of private keys encoded as an English mnemonic for ease and reliability of transcription. It's not yet supported by Glacier because it's not supported by Bitcoin Core.

Sign Withdrawal Transactions With Individual Signatures

Bringing multiple private keys together in the same physical location for the Withdrawal Protocol entails risk (they could be physically stolen). It would be good to have an option to sign the withdrawal one transaction at a time, probably by bringing a QR-encoded physical hardcopy of the partially-signed transaction to the storage location of each private key.

Consider Shamir's Secret Sharing or Vanilla Multisig vs. P2SH Transactions

Glacier currently uses P2SH transactions. This allows all signatories storing private keys to view the user's balance, because a copy of the redeem script must be kept with each private key.

Vanilla multisig transactions would address this, but it's not clear if it's possible to do vanilla multisig configurations with over 3 keys (https://bitcoin.stackexchange.com/questions/23893/what-are-the-limits-of-m-and-n-in-m-of-n-multisig-addresses). Another option is to use a single Bitcoin private key, split into n pieces using Shamir's Secret Sharing (https://en.wikipedia.org/wiki/Shamir%27s_Secret_Sharing), which would not have any limitations on the number of keyholders, but would require additional cryptographic software be integrated into Glacier.

Automate Quarantined USB creation

Many of the steps for creating the Quarantined USBs could be automated in a simple script.

Security With Biased Dice

Assess integration of this paper and/or this algorithm (http://pit-claudel.fr/clement/blog/generating-uniformly-random-data-from-skewed-input-biased-coins-loaded-dice-skew-correction-and-the-von-neumann-extractor/) so that the quality of our randomness is not vulnerable to dice bias.

Entropy Quality Testing

Use an entropy test suite such as ent (http://www.fourmilab.ch/random/) to verify the quality of generated entropy before it's used.

Bitcoin Core Version

Pinning Currently, we download Bitcoin Core on to the Quarantined App USBs via the Ubuntu Package archive. However, because Bitcoin is a privately-managed archive, it only hosts the latest release, rather than all previous versions. This prevents us from pinning the protocol to use a specific release (desirable for ongoing compatibility).

9.3. Ecosystem improvements

The Glacier protocol is lengthy and complex because the tools for high-security cold storage do not exist. This appendix briefly outlines some of the tool functionality that would address this gap. For additional technical details, see the Glacier design document.

Ideally, the Bitcoin community (and other cryptocurrency communities) will create these tools as soon as possible and render Glacier obsolete. We invite inquiry and consultation by others interested in developing these tools.

Cold Storage Hardware Wallets

- Function like conventional hardware wallets, but eternally quarantined (no wireless or wired connections)
- I/O
    - Keyboard for entering data (key recovery, user entropy for key generation)
    - Camera for reading QR codes (for unsigned transactions)
    - Screen for displaying data, including QR codes (for complex data such as signed transactions)

- Generate keys from user-provided entropy (ideally two combined sources)
- Support for BIP39 and HD keys
- Multisig support
    - Each wallet storing one key is probably the way to go
    - Ability to for each device to add one single signature to a transaction, so only one key needs to be stored on a given device
    - Compatibility with HD keys
- Verifiability
    - All deterministic algorithms (for key generation, transaction generation, etc.)
    - Multiple wallet products on the market which use as many different hardware components as possible (to minimize the possibility of a common flaw / vulnerability)
- Simple to use
- Display steps user through security steps – how to safely generate their entropy, double-checking that addresses are correct, verifying duplicate algorithm results on an alternate device, etc.
- Optional side channel protection
    - Partner with a company that manufactures some sort of Faraday glove box, and market it to customers who have extra-high security concerns

Bitcoin Core improvements

Until robust cold storage hardware wallets are created, improvements in Bitcoin Core could go a long way towards improving and simplifying Glacier, including reducing the necessary complexity of GlacierScript.

- Add support for importing and using BIP32 HD keys.
- Generate keys based on raw user entropy so that key generation can be deterministically checked by a second quarantine computer.
- BIP39 key generation support
    - Promotes security through ease of use, and reduces risk of transcription errors

10. Contribute

10.1. License

All the documents are distributed under the Creative Commons Attribution-ShareAlike 4.0 International License (https://creativecommons.org/licenses/by-sa/4.0/).

The CryptoGlacierScript software is distributed under the MIT license (https://opensource.org/licenses/MIT).

10.2. Acknowledgments

CryptoGlacier is based on the popular Glacier Protocol (https://glacierprotocol.org)

The following individuals have offered feedback or other contributions that were incorporated into CryptoGlacier:

- Fulvia Morales
- Your name here!
- Your name here!
- Your name here!

11. Design documents

11.1. Design document

If you want to learn more about the security considerations for CryptoGlacier, we recommend you to check the Glacier design document:

- v0.9 Beta (latest version)
- v0.1 Alpha