Making Bitcoin Exchanges Transparent

Christian Decker(✉), James Guthrie, Jochen Seidel, and Roger Wattenhofer

Distributed Computing Group, ETH Zurich, Zürich, Switzerland
{cdecker,guthriej,seidelj,wattenhofer}@ethz.ch

Abstract. Bitcoin exchanges are a vital component of the Bitcoin ecosystem. They are a gateway from the classical economy to the cryptocurrency economy, facilitating the exchange between fiat currency and bitcoins. However, exchanges are also single points of failure, operating outside the Bitcoin blockchain, requiring users to entrust them with their funds in order to operate. In this work we present a solution, and a proof-of-concept implementation, that allows exchanges to prove their solvency, without publishing any information of strategic importance.

1 Introduction

Since the conceptual introduction of Bitcoin in 2008 by Satoshi Nakamoto [15] and the appearance of the first Bitcoin client in 2009, Bitcoin has seen massive growth on a multitude of fronts. Bitcoin currently has a market capitalisation of 3 billion US dollars and an average daily transaction volume of approximately 50 million US dollars.

One factor which has driven widespread adoption of Bitcoin is the emergence of Bitcoin exchanges: companies which allow trading bitcoins with fiat currency, such as Euros and US dollars. Bitcoin exchanges have helped the adoption of Bitcoin in two ways. Firstly, before the advent of Bitcoin exchanges, the only way to come by bitcoins was to *mine* them oneself or to informally trade bitcoins with other participants. Exchanges have opened the Bitcoin market to parties who might otherwise not have been able to participate. Secondly, exchanges publish their trade books which establishes an accepted exchange rate between fiat currencies and bitcoins. This in turn allowed vendors to value their goods and services in bitcoins in accordance to the market rates in fiat currency.

Although Bitcoin exchanges have had a positive contribution to the Bitcoin economy, they are not without risks. In Moore and Christin's analysis of the risks involved with Bitcoin exchanges [14] they analyse 40 Bitcoin exchanges, at the time of publication 18 of the 40 exchanges had ceased operation. Of those 18, 5 exchanges did not reimburse customers on closure, 6 exchanges claim that they did and for the remaining 7 there is no data available. Most of the collapsed Bitcoin exchanges were not long-lived, with their closure either being immediate or over a relatively short period of time.

Since the publication of that analysis the most high-profile exchange closure took place: the bankruptcy and closure of the Mt. Gox Bitcoin exchange, in which 650,000 bitcoins belonging to customers of the exchange were lost or stolen.

G. Pernul et al. (Eds.): ESORICS 2015, Part II, LNCS 9327, pp. 561–576, 2015.
DOI: 10.1007/978-3-319-24177-7_28

At the time, Mt. Gox claimed that a flaw in the Bitcoin protocol was to blame for the loss of its client's bitcoins, a claim which has since been refuted [7]. At the time of the event, Mt. Gox was one of the longest-running exchanges in the Bitcoin market with its cumulative number of transactions accounting for approximately 70 % of all Bitcoin transactions.

Bitcoin transactions are irreversible by design. Once a user has transferred her bitcoins to another user there is no way that she will get them back without the cooperation of the recipient. There is little recourse for the customer of an exchange: Bitcoin is new ground for insurers, regulators, and law enforcement who do not yet have any established methods for dealing with Bitcoin related legal issues.

In an effort to calm customers fears, some exchanges have taken to periodically publishing data proving their solvency: an anonymised list of their customers account balances and a list of Bitcoin addresses owned by the exchange along with a signature that proves the ownership. If the balance of the bitcoins available on the addresses is at least as large as the sum of the amounts owed by the exchange, the exchange is solvent. Although customers may be appreciative of this type of transparency, it may put the exchange at a disadvantage as it reveals information of strategic importance, such as the number of customers, the amounts the exchange's customers keep on hand and the total balance of bitcoins held by the exchange.

In conventional financial markets trust is placed in the financial statements made by institutions such as exchanges or investment funds through the process of auditing. An independent third party, which is perceived to be trustworthy by the customers of the institution, or a state mandated auditor inspects the financial records of the institution and publishes an audit result. Such an audit is an expensive and time-consuming process and is typically only performed in well-spaced intervals.

In this paper, we propose to perform a software-based audit of Bitcoin exchanges without revealing any information about the bitcoins that are possessed by either the exchange, or its customers to the public. This is achieved by replacing the human financial auditor by a piece of software. To ensure that the software is executed correctly we rely on Trusted Computing (TC) technology. In our scenario, the traditional limitations of financial auditing no longer apply. Software executes orders of magnitudes faster than humans, and the execution of a piece of software is generally not costly at all and it becomes feasible to provide daily audits of a Bitcoin exchange. Our contribution is twofold: we propose a system that uses Trusted Computing to prove the exchange's solvibility to its customers and we implement the proposed solution on consumer hardware minimizing obstacles to its deployment.

1.1 Related Work

Auditing Bitcoin exchanges has been previously discussed by Todd and Maxwell [10], and later by Maxwell and Wilcox [11]. Both approaches rely on modifying the merkle tree computation to defend against insertion of negative

subtrees. Our use of TC for the merkle tree computation obviates any such modifications as the secure code would error out on negative sums.

Trusted Computing, and more specifically TPMs, have been proposed previously as a method to secure Bitcoin wallets by Hal Finney [8], storing sensitive keying material in the tamper proof storage. Since then several additional methods of securing funds have been proposed, including multisignature accounts [3], the creation of deterministic public keys that do not require private keys during the generation [4] and locking funds for a predetermined period of time [17]. The latter may also be used to extend the audit to guarantee the solvency for a certain period, by making the funds inaccessible until they are unlocked.

While regular audits may help detect fraud at an early stage, a regulatory framework is needed to prosecute perpetrators. Some initial work has been done in the field of regulation, examining the impact of Bitcoin on current anti-money laundering (AML) policies and on know-your-customer (KYC) policies [1,2,5,6,16].

2 Preliminaries

The software-based audit of a Bitcoin exchange relies on an understanding of both the Bitcoin project as well as Trusted Computing. This section introduces the fundamentals of Bitcoin and Trusted Computing, as needed in this paper.

2.1 Bitcoin

Bitcoin is a decentralized digital currency built on cryptographic protocols and a system of *proof of work*. Instead of relying on traditional, centralized financial institutions to administer transactions as well as other aspects concerning the economic valuation of the currency, peers within the Bitcoin network process transactions and control the creation of bitcoins. The major problems to be solved by a distributed currency are related to how consensus can be reached in a group of anonymous participants, some of whom may be behaving with malicious intent.

Transactions within the Bitcoin network are based on public key cryptography, users of Bitcoin generate an *address* which is used to receive funds. The Bitcoin address is derived, through cryptographic hash functions, from the public-key of an ECDSA key pair. A Bitcoin transaction records the transfer of bitcoins from some input address to output addresses. A transaction consists of one or more inputs and one or more outputs, each input to a transaction is the output of a previous transaction. The output of a transaction may only be used as the input to a single transaction. The outputs are associated with an address, whose private key is then used to sign transactions spending these outputs.

Transactions are generated by the sender and distributed amongst the peers in the Bitcoin network. Transactions are only valid once they have been accepted into the public history of transactions, the *blockchain*. As the blockchain contains Bitcoin's entire transaction history and is publicly distributed, any user can determine the bitcoin balance of every address at any time, by summing the value of unspent transaction outputs (UTXOs) associated with the address.

Bitcoin Exchanges. Bitcoin exchanges facilitate trade between fiat currency and bitcoins. In order to trade on the exchange, users create an account with the exchange and transfer fiat currency and/or bitcoins to the exchange. Should the user wish to retrieve their bitcoins, they must make a request that the exchange transfers the bitcoins to an address which the user controls. The exchange manages a balance of the bitcoins that the user has deposited with the exchange or traded for against fiat currency.

The user may place buy and sell orders for bitcoins or fiat currency which are executed for the user by the exchange, adjusting the balances of the user's Bitcoin or fiat currency accounts. The orders are executed internally within the exchange, that is they are not recorded in the blockchain. Given this model of operation, a Bitcoin exchange is not merely a marketplace but also acts as a fiduciary, administrating both fiat currency and bitcoin accounts for its clients.

2.2 Trusted Computing

When a third party, such as a Bitcoin exchange, is tasked with performing a computation, there is no method for the verification of the integrity of the result, short of performing the computation locally, which in some circumstances may not be feasible. Trusted Computing allows the creation of a *trusted platform* which provides the following features [18]:

Protected Capabilities are commands which may access *shielded locations*, areas in memory or registers which are only accessible to the trusted platform. These memory areas may contain sensitive data such as private keys or a digest of some aspect of the current system state.

Integrity Measurement is the process of *measuring* the software which is executing on the current platform. A measurement is the cryptographic hash of the software which is executing throughout each stage of execution.

Integrity Reporting is the process of delivering a platform measurement to a third party such that it can be verified to have originated from a trusted platform.

These features of the trusted platform are deployed on consumer hardware in a unit called the Trusted Platform Module (TPM), a secure cryptographic co-processor, which is usually incorporated on the mainboard of the hardware.

An important component in proving trust are the Platform Configuration Registers (PCRs), 20-byte registers which are only modifiable through the *extend* operation based on cryptographic hash digests. The properties of a cryptographic hash ensure that the value held in a PCR cannot be deliberately set.

Initially the TPM is equipped with a Storage Root Key (SRK) which may be used to sign and thus authenticate further keys which may be generated or loaded into the TPM. A number of different types of cryptographic keys may be present on the TPM, however we limit our description to Attestation Identity Keys (AIK). AIKs are signing keys that reside solely on the TPM, which are used to sign data, which originates from the TPM, in order to attest to the

values originating from the TPM. In order to verify a TPM attestation, the verifying party requires the signed attestation, the AIK public key, and a valid SRK signature authenticating the AIK.

The TPM can be used to *seal* data which encrypts the data with a key which is loaded in the TPM and binds the data to the state of some of the PCRs. The encrypted data may only be decrypted or *unsealed* if PCRs are in the same state as when the data was sealed, thus binding the ability to decrypt to the measured state. TPMs provide two distinct paradigms:

SRTM (Static Root of Trust for Measurement): the system begins to boot in a piece of firmware which is trusted (the static root) and each component of the boot process is measured and verified against a known-good configuration before it is executed in order to assert that no component has been tampered with.

DRTM (Dynamic Root of Trust for Measurement): allows for a trusted platform to be established dynamically without requiring a system reboot. It even allows for a trusted platform to be established within a platform which is known to be compromised with malicious software.

DRTM is implemented in consumer general purpose processors from Intel and AMD under the names Intel Trusted eXecution Technology (TXT) and AMD Secure Virtual Machine (SVM). Intel TXT and AMD SVM provide additional security features when executing in the secure mode on top of the capabilities of the TPM. These include turning off system interrupts to prevent other execution paths, as well as memory protection for specific memory ranges which also prevents DMA access [9].

3 Auditing

The audit should determine the solvency of the exchange. In principle this is a binary result, either *solvent* in the case that the exchange's assets in bitcoins cover its liabilities in bitcoins, or *insolvent* otherwise. It is plausible that there are situations in which this binary result does not suffice, for instance an exchange which wishes to prove fractional reserves. In these cases a multiplicative factor can be applied to the liabilities of the exchange to show that the exchange can cover some percentage of its liabilities with its assets.

The auditing process can be broken into three individual steps: summing the user account balances (proof of liabilities), summing the assets, i.e., address balances, the exchange controls (proof of reserves), and proof that the reserves cover the liabilities (proof of solvency). Figure 1 illustrates the components of the audit, the inputs to each of the components of the calculation and the outputs of the audit.

The publicly available inputs are the address balance and the fraction factor, which determines the percentage of coverage that the exchange wishes to prove. The address balances can be computed by a third party by replaying transactions in the blockchain until the time of the audit. The non-public inputs consist of

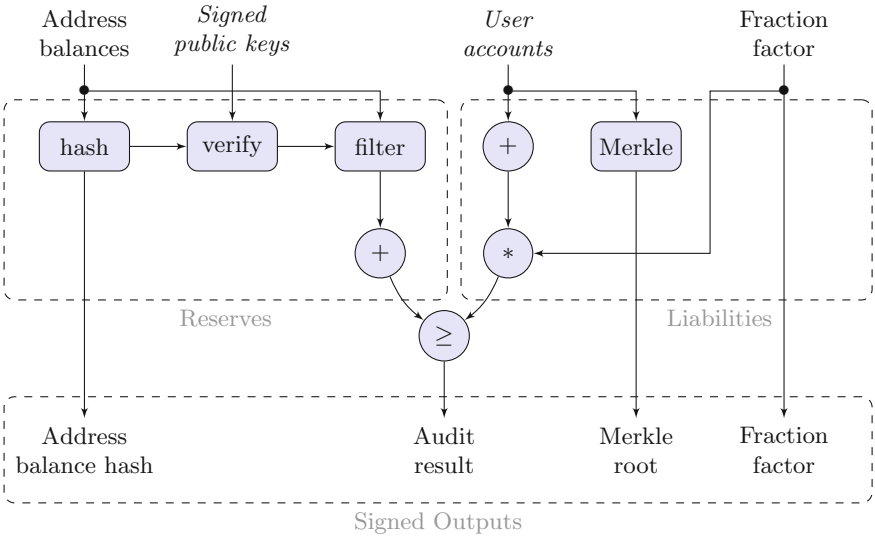

Fig. 1. An overview of the audit process. Italicised values are not published.

a list of signed public keys owned by the exchange and the list of user account information, including account balances and customer identifiers.

Unlike the inputs, the outputs of the auditing process should all be disclosed publicly. The address balance hash proves that the latest snapshot of the address balances was used in the audit and should match an independently computed hash. The audit result is the boolean result, either `true` if the exchanges assets are greater than the fraction of the liabilities or `false` otherwise. To prove that all liabilities have been considered a merkle tree is computed and its root is included in the outputs as well as the fraction which determines the coverage percentage. The output values are signed by the TPM, which also signs a hash digest of the binary which was executing at the time.

3.1 Proof of Reserves

The assets that the exchange possesses are in the form of bitcoins in the blockchain. The sum of assets is therefore calculated by determining which balances in the blockchain the exchange has access to and calculating the sum of those balances. In order for the exchange to access the bitcoins it needs to be in possession of the private keys belonging to the addresses.

To simplify the calculation, the audit program does not need to parse the entire blockchain to determine which balances should be summed. Instead, a preprocessor can be used to compute the address balances from the blockchain. This is secure as it is a deterministic aggregation over publicly available data.

The exchange can prove control of a Bitcoin address by providing the public key belonging to that Bitcoin address and signing it with the private key. For

additional safety, the exchange should also sign a value which can be used to prove the freshness of the signature, a nonce. The hash of the last block added to the blockchain is an ideal candidate for the nonce, as it uniquely identifies the state of the blockchain and thus the address balances, it is not predictable and changes frequently. Thus, the second input to the audit process consists of a list of tuples of a public keys, and a signatures of the public key and the nonce:

$$\langle \text{PubKey}, \{\text{PubKey}, \text{Nonce}\}_\sigma \rangle$$

where $\{\text{data}\}_\sigma$ indicates that *data* is signed with the corresponding private key.

The overview of the steps of the calculation of reserves is shown in Fig. 1, internally it consists of four different stages. The first stage computes the hash of the address balances, which is required in the *verify* stage. The verify stage asserts that the signatures for the public keys are valid and that the provided nonce matches the hash of the provided address balances. It then passes the public keys to the *filter* stage, determines the Bitcoin address and filters for entries in the address balances which match the exchange's addresses. Finally, the balances of these entries are summed. The sum, as well as the hash of the address balances are produced as outputs of the proof of reserves.

3.2 Proof of Liabilities

The liabilities of the exchange are the balances in bitcoins owed to its customers. The audit process requires a list of tuples consisting of a customer identifier and a positive balance owed to the customer:

$$\langle \text{CustID}, \text{Balance} \rangle$$

An additional input to the proof of liabilities is the *fraction factor*, which is multiplied with the sum of client account balances to prove fractional reserves.

Using the above definition of liabilities, the total liabilities of the exchange are calculated as the sum of all customer account balances. The calculated sum is later compared against the sum of reserves to determine solvency. Additionally to the sum, the proof of liabilities component calculates the root of a merkle tree [13], as well as a hash of the fraction factor.

The basic schema is to construct a merkle tree with the user account information. That is, in order to compute a leaf in the tree one would take the cryptographic hash of the customer identifier and the balance owed to the customer. The leaves are then combined in a pairwise fashion and hashed, forming the nodes in the next layer of the tree. Nodes are combined and hashed until the root of the tree is constructed.

As the root of the merkle tree is dependent on all of the individual values within the tree, it serves as public record of the account balances which were counted in the summation of all account balances, without revealing individual customers account balances.

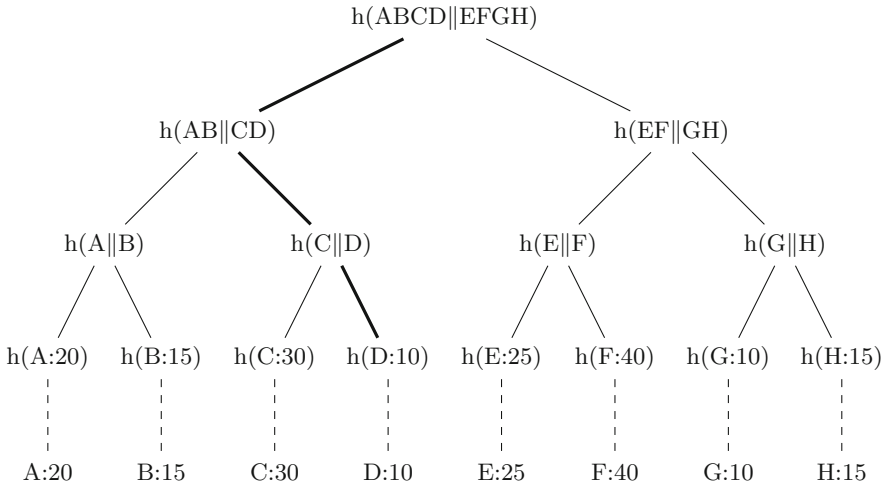

Fig. 2. An example merkle tree with the path from h(D:10) to the root highlighted

3.3 Proof of Solvency and Verification

The proof of the solvency of a bitcoin exchange consists of two components, one is the outputs of the audit, the other is an attestation which can be used to verify that the auditing software was executed in the trusted environment, and that it computed the outputs which are attested. The final output is the *Audit result*, which is a binary value, *true* if the reserves are greater than or equal to the liabilities, and *false* otherwise. The attestation is a signature for the outputs as well as the platform measurements, i.e., the hashes of the executed program.

Given the audit program, its public inputs and outputs and the attestation, a customer can independently verify the validity of the audit. By hashing the program and validating it against the attested measurements she can ensure that the TPM has executed the program. The validity of the program could be proven by publishing its source code. The customer can then proceed to validating the outputs, by checking the signatures, that the address balance hash matches the blockchain and that she is included in the merkle tree.

The customer can use the root of the merkle tree to verify that its account balance was included in the calculation. The merkle tree in Fig. 2 shows a potential scenario in which customer D wishes to determine whether it was accounted for in the hash h(ABCD‖EFGH). The nodes which D requires are the children of the nodes along the path from D's leaf node to the root excluding the nodes along that path. These are the nodes h(C:30), $h(A \parallel B)$, h(EF‖GH). With these node values, D can reconstruct the path from its leaf node to the root, calculating the same value of h(ABCD‖EFGH) that was provided by the exchange.

4 Implementation

The proof-of-concept presented in this work is built on the Flicker platform [12]. Flicker is a software platform which leverages DRTM to allow security sensitive components of software applications to execute in a secure, isolated environment. The developers of Flicker call such a component a Piece of Application Logic (PAL). The PAL comprises only the routines required to perform some security critical computation component of the application. Flicker consists of two components, the kernel module which prepares and launches the DRTM process, and the Secure Loader Block (SLB) core which performs bootstrapping of the secure execution environment for the PAL.

The execution scenario in which the PAL runs is made up of four distinct components: the user application, the Flicker kernel module, an Authenticated Code Module (ACM), and one or more PAL binaries, each consisting of the SLB core and PAL. The ACM is the root of dynamic trust for the DRTM in Intel TXT and is digitally signed by the chipset vendor. It functions as a secure bootloader for a lightweight piece of code which is to be executed on the processor in complete isolation from any other software or hardware access.

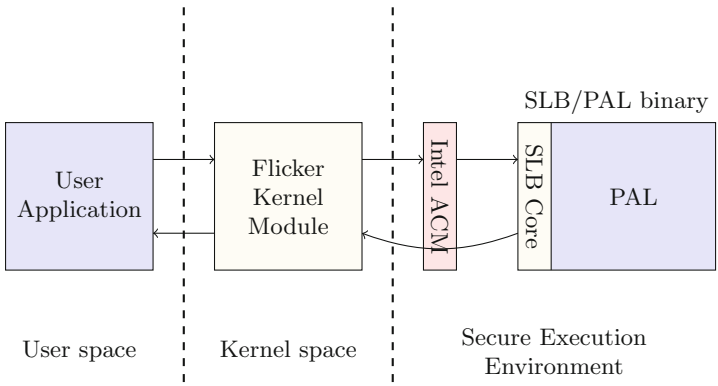

Fig. 3. Flicker PAL execution scenario.

The user application is a conventional application executing in userspace. The Flicker kernel module provides `data` and `control` file system entries with which the user application may interact in order to provide the Flicker kernel module with the SLB, PAL, and the inputs, as well as to read the outputs when execution of the PAL terminates.

Figure 3 illustrates the control flow when the user application needs to perform a security-critical task. First the application passes the PAL binary and inputs to the Flicker kernel module and instructs the kernel module to execute the PAL. The Flicker kernel module prepares the necessary data structures and memory protection to launch the DRTM and start the PAL, it then invokes

the GETSEC[SENTER] CPU instruction which disables interrupts and triggers the start of the DRTM. These data structures are measured by the Intel ACM, which forms the root of the DRTM. The ACM hands control over to the Flicker SLB core which invokes the PAL and contains the necessary data structures to return the control flow directly to the Flicker kernel module when the PAL has finished executing.

During the execution of the SENTER operation, the dynamic TPM PCRs (17–23) are initialised to zero. PCR 17 is then extended with the hashes of the ACM and a number of configuration parameters. During the execution, PCR 18 is extended with the hash of the PAL. These PCR values are provided in the TPM's attestation, which can be used to prove to a third party that the PAL binary was executed and calculated the output values.

The Flicker platform was designed with lightweight, short-lived computations in mind, as such it imposes a number of restrictions which make a direct implementation of the audit as outlined in Sect. 3 unfeasible. The major restriction which poses problems for the automated software audit is memory. The Flicker environment has a stack size of 4KB, a heap size of 128KB, and a maximum input size of approximately 116KB. In addition each Flicker session has a significant overhead, between 0.2 and 1 second, depending on which TPM functionality is used during the invocation [12].

4.1 Architecture

Three of the four inputs to the audit process may be of considerable size: the address balances, the public keys and signatures, and the user accounts. At the time of writing there are a total of 3.7 million addresses with a non-zero balance. Each of the entries in the address balance input consists of an address of up to 35 byte and a 64 bit integer for the balance. The size of all address balances therefore is just under 160 MB. The size of the user accounts depends on the number of user accounts of the exchange. Generating a unique identifier from the account information by hashing results in a 32 byte identifier. Each account therefore has a 32 byte identifier with a 64 bit integer for the balance. Estimating the user base of the exchange at 1 million users this results in a total size for the user account input of 40 MB. While the number of addresses owned by the exchange is under control of the exchange, the prototype should support any number of addresses.

It is clear from the memory requirements posed by the input data and the available input sizes of the Flicker platform that the monolithic architecture of the audit as proposed in Fig. 1 must be broken into smaller components. The input data is split into input-sized chunks and processed in an incremental fashion. This does not change the result of the audit, however the calculation of the outputs which are required to verify the input data must change as a result of the components only having a view of a small subset of the input data in each iteration. The individual invocations of a component of the audit require a secure method of storing intermediate values, for instance a sum which is calculated over multiple iterations. The PAL can use the TPM to seal intermediate values

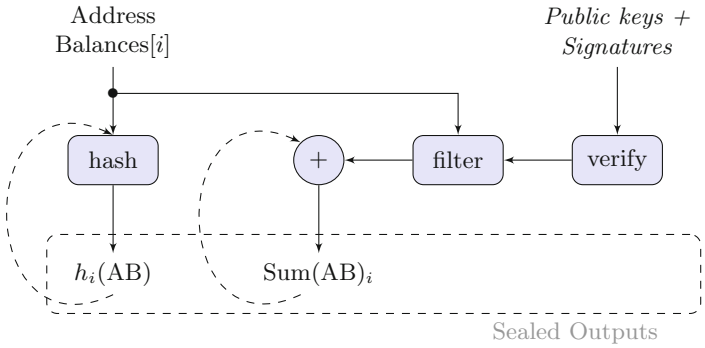

Fig. 4. An overview of the proof of reserves component. Italicised values are not published publicly. Dashed arrows indicate values which are passed from invocation to invocation

to the current PCR state, encrypting them such that they can only be decrypted by the TPM when it is executing the same PAL. The encrypted data is passed back to the user application which should provide it as an input to the next iteration of the component.

The process is driven by a user application, external to the trusted platform, which repeatedly invokes the computation in the trusted platform. As the encrypted data is passed back to the user application, which is executing in an untrusted and potentially malicious environment, there is the potential for a replay attack to be performed. However, the process of hashing the input ensures that replay attacks can be detected by the client when verifying the result of the audit.

We consider each component of the system individually and describe how it is implemented in order to support incremental invocations.

Proof of Reserves. The Proof of Reserves can be split into iterative invocations by splitting the address balance list, and the list of signatures and public keys into equal sized batches. Initially the address balance list is sorted lexicographically, in order to allow a verifier to compute the same hash. Each batch contains a list of address balances and a possibly empty set of signatures and public keys matching the address balances of the batch. This allows the system to verify the signatures and sum up the respective values. The hash of the address balances is computed by concatenating the hash from the previous round with the current hash and hashing the result: $h_0 = h(AB_0)$ and $h_i = h(h_{i-1} \parallel AB_i)$. The output includes the last considered address from the current batch. Due to the lexicographic ordering of addresses it is trivial for the proof of reserves to detect a replay attack, since it would require a lexicographically lower first address than the last address from the previous batch. Figure 4 shows an overview of the new POR component.

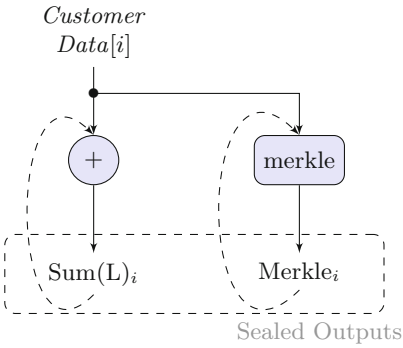

Fig. 5. An overview of the proof of liabilities component. Italicised values are not published publicly. Dashed arrows indicate values which are passed from invocation to invocation

Proof of Liabilities. The Proof of Liabilities (POL) is invoked iteratively, similarly to the Proof of Reserves. Figure 5 shows an overview of the POL component. The merkle tree computation accepts a list of tuples consisting of merkle subtree root hashes, the root's height and the associated sum of the tree's value. It then iteratively computes the roots of the trees by combining the subtrees, summing the values and increasing the height. The resulting merkle root, height and value sum is then sealed for the next iteration or to be passed to the proof of solvency. In order to initiate the process, the proof of liabilities also accepts subtrees that are not sealed for height 0, i.e., the hashes of the account identifier and the account's value. Missing branches in the merkle tree are replaced with a single leaf with value 0.

Given that the merkle tree computation does not allow negative value sums for subtrees guarantees that, if an account was included in the computation, its value is included in the sum. A replay attack in this case does not benefit the exchange as it may only increase the sum that is to be covered.

Proof of Solvency. The Proof of Solvency (POS) component takes as inputs the sealed outputs from the Proof of Reserves (POR) and Proof of Liabilities (POS) components as well as the Fraction factor. As it handles only constant size inputs it is sufficient to call the proof of solvability once. Its main purpose is to compute the fraction that is to be covered and whether or not the assets are sufficient to cover the liabilities. A secondary purpose is to unseal the results from the other components and sign in order to publish them (Fig. 6).

The final step of the POS component is the attestation of the PAL binary. The audit no longer consists of an individual invocation of a PAL, instead the POR and POL components are invoked hundreds or thousands of times each of these invocations requires attestation. The solution to this problem is to put the separate logic for the POR, POL and POS components into a single binary. The initial invocations of both the POR and POL logic of the PAL produce a

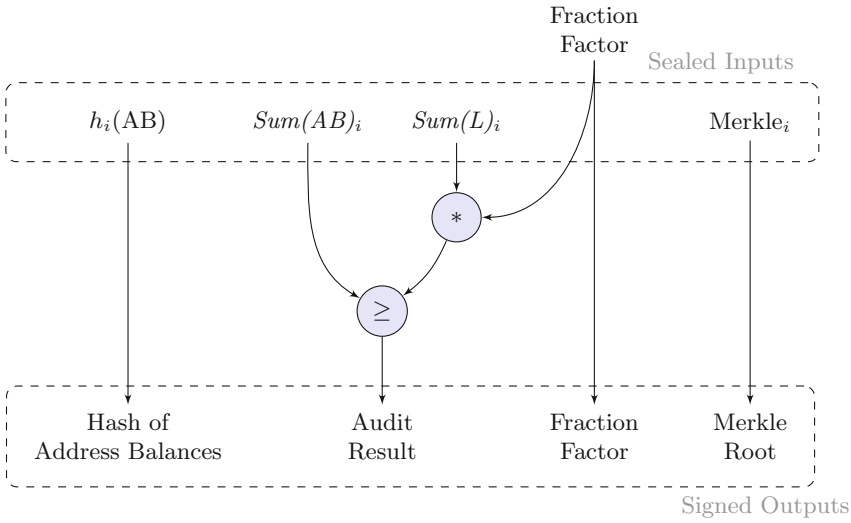

Fig. 6. An overview of the proof of solvency component.

sealed intermediate values which are tied to that PAL. The sealed blob is then unsealed by the next invocation, the intermediate values are modified and then resealed. When the POS is invoked, it unseals the intermediate results produced by the POR and POL.

The fact that the sealed blobs are unsealed and modified in each invocation of the POR and POL and that they can only be unsealed by the same PAL that initially created them means that the values in the sealed blob form a chain of trust from their respective first invocations until the invocation of the POS. An attestation of the POS is transitive to all previous PAL executions which were able to unseal the blobs that the POS unseals.

4.2 Execution Time

As previously mentioned, the Flicker invocation and some TPM operations pose a significant overhead of up to 1 s, when repeatedly entering and leaving the PAL. During the execution time of the PAL, the operating system on which the Flicker session is invoked does not process any interrupts. When the Flicker session ends, the operating system requires a small amount of time to process any interrupts and respond to system events. Tests showed that the operating system needs pauses of 500 ms to 1 s in order to continue processing without locking up or crashing. As the processing time for such a small number of inputs is quite low in comparison to the TPM overhead, we can safely assume that each Flicker invocation costs approximately two seconds.

For input sizes in the range previously discussed, 3.7 million address balances and 1 million customer accounts, the POR must be invoked approximately 1300 times if each invocation of the POR can process 3000 address balances, the

POL must be invoked approximately 500 times. This comes to a total of 1800 invocations, each of which requires 2 s to execute and wait for the operating system to recover. The overall execution time for an audit with inputs of this size is approximately one hour and scales linearly in the number of address balances and user accounts.

4.3 Additional Interfaces

Although the audit is the core component of proving solvency of a Bitcoin exchange, the signed audit output is not all that a customer requires in order to verify the audit. Customers must retrieve additional values from the exchange and perform some local computations in order to be able to verify the audit, and to have some form of recourse should the verification fail. The implementation of these interfaces is not in the scope of this work, what follows is an outline of the requirements of the peripheral software and interfaces.

Audit Verification. Most important for customers is the ability to verify the audit's result. This consists of the verification that the customer's balance was included in the calculation, verification of the address balances, and verification of the attestation. The customer of an exchange must be able to retrieve the nodes in the merkle tree which can be used to calculate the path from the customer's leaf node to the root of the merkle tree. If the customer is able to reproduce the merkle root using the nodes provided by the exchange and their own customer identifier and account balance at the time of the audit, then they can be assured that they were accounted for in the calculation. The interface for this purpose must take the hash of the tuple

$$\langle \text{Customer Identifier}, \text{Balance} \rangle$$

as an argument and deliver the set of nodes required to calculate the path from the customer's leaf node to the merkle root. Each node would consist of a tuple

$$\langle \text{Height}, \text{Hash} \rangle$$

where *Height* is the height of the node in the merkle tree and *Hash* is the hash digest stored at that node in the tree. The customer must also be able to verify that the hash of the address balances provided by the exchange represents the true account balances for a set blockchain height. For this purpose, the customer must be able to determine the blockchain height that was used to determine the address balances. The customer would require a software client which can determine the address balances for the blockchain at a given height. This consists of: extraction and aggregation of UTXOs, and sorting of the address balances. With the address balances calculated, the customer can calculate the hash and compare it with the hash provided by the audit. Finally, the customer must be able to verify the attestation. This consists of two components: verification that the attestation originates from a TPM, and verification of the binary which was executed in the trusted platform.

Attestation Verification. The customer needs to be able to verify that the attestation was indeed issued by a TPM, in other words, what the customer needs to know is that the Attestation Identity Key (AIK) used to sign the attestion was provided by a TPM. The method proposed by the TCG is Direct Anonymous Attestation (DAA) which allows for a customer to verify directly that an AIK belongs to a TPM. For this the exchange must provide an interface which performs DAA and the customer requires client software which can verify the DAA provided by the exchange.

Binary Verication. In order for the customer to verify that the PAL executed in the trusted environment actually calculates the audit, as opposed to always returning true, the customer must have access to the source code of the PAL and be able to reproduce the value of PCR 17 which is signed in the TPM's attestation. The exchange needs to provide a platform from which the PAL source code can be retrieved, as well as a method for compiling a reproducible binary, and instructions on how to transform the hash of the binary to the value of PCR 17 in the attestation.

Signed Account Balance. If a customer should determine that their account balance was not included in an audit, they require some form of proof that their account balances ought to have been taken into account in the audit. For this purpose the exchange should provide an interface which allows a customer to retrieve a signature of the hash of their $\langle \text{CustomrID}, \text{balance} \rangle$ tuple. With this signature, other customers or the community at large could verify that the exchange signed a value which is not included in the latest audit.

5 Conclusion

A string of Bitcoin exchange closures as well as various thefts from Bitcoin exchanges have left customers of exchange services somewhat hesitant as there has often been little transparency when such events took place. Exchanges have published customer account balances as well as proof of ownership of Bitcoin addressed which allow for customers and the public to determine the Bitcoin assets of the exchange.

In this work we propose using an automated software-based audit to determine the solvency of Bitcoin exchanges without revealing any private data. Methods are proposed, based on the Flicker Trusted Computing platform, with which the audit result can be verified and trusted to be correct. An architecture is proposed which allows for the computation to be split into individual pieces which iteratively compute a subset of the complete input to overcome the memory limitations posed by the Flicker platform. The verification methodology is expanded to cover the iterative execution scenario, allowing for customers of an exchange to verify the inputs to the audit. An analysis of the execution time showed that it is entirely feasible to conduct audits on a daily basis at the current estimate size of the Bitcoin ecosystem.

References

1. Bitcoin virtual currency: Unique features present distinct challenges for deterring illicit activity. Technical report, Federal Bureau of Investigation (2012)
2. Application of fincen's regulations to persons administering: exchanging, or using virtual currencies. Technical report, Financial Crimes Enforcement Network, US Department of the Treasury (2013)
3. Andresen, G.: Bitcoin improvement proposal 11: M-of-N standard transactions (2011). https://github.com/bitcoin/bips/blob/master/bip-0011.mediawiki. Accessed February 2015
4. Araoz, M., Charles, R.X., Garcia, M.A.: Bip 45: Structure for deterministic P2SH multisignature wallets (2014). https://github.com/bitcoin/bips/blob/master/bip-0045.mediawiki. Accessed February 2015
5. Brito, J., Castillo, A.: Bitcoin: A Primer for Policymakers. Mercatus Center at George Mason University, Arlington (2013)
6. Bryans, D.: Bitcoin and money laundering: mining for an effective solution. Indiana Law J. **89**, 441–472 (2014)
7. Decker, C., Wattenhofer, R.: Bitcoin transaction malleability and MtGox. In: 19th European Symposium on Research in Computer Security (ESORICS), Wroclaw, Poland, September 2014
8. Finney, H.: bcflick - using TPM's and trusted computing to strengthen bitcoin wallets (2013). https://bitcointalk.org/index.php?topic=154290.msg1635481. Accessed February 2015
9. Intel Corporation: Intel Trusted Execution Technology Software Developers Guide, May 2014
10. Maxwell, G., Todd, P.: Fraud proof (2013). https://people.xiph.org/greg/bitcoin-wizards-fraud-proof.log.txt. Accessed March 2015
11. Maxwell, G., Wilcox, Z.: Proving your bitcoin reserves, February 2014. https://iwilcox.me.uk/2014/proving-bitcoin-reserves. Accessed 5th January 2015
12. McCune, J.M., Parno, B.J., Perrig, A., Reiter, M.K., Isozaki, H.: Flicker: An execution infrastructure for TCB minimization. In: ACM SIGOPS Operating Systems Review
13. Merkle, R.C.: A digital signature based on a conventional encryption function. In: Pomerance, C. (ed.) CRYPTO 1987. LNCS, vol. 293, pp. 369–378. Springer, Heidelberg (1988)
14. Moore, T., Christin, N.: Beware the middleman: empirical analysis of bitcoin-exchange risk. In: Sadeghi, A.-R. (ed.) FC 2013. LNCS, vol. 7859, pp. 25–33. Springer, Heidelberg (2013)
15. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008). https://bitcoin.org/bitcoin.pdf. Accessed February 2015
16. New York State Department of Financial Services: Virtual Currencies (2015). http://www.dfs.ny.gov/legal/regulations/revised_vc_regulation.pdf. Accessed February 2015
17. Todd, P.: BIP 65: OP_CHECKLOCKTIMEVERIFY (2014). https://github.com/bitcoin/bips. Accessed March 2014
18. Trusted Computing Group: TCG Specification Architecture Overview, rev. 1.4, August 2007