Security Audit of RSKJ Ginger 0.2.0

Patrick McCorry[1][2] and Andrew Miller[2]

[1] University College London, UK
[2] University of Illinois at Urbana-Champaign, US

Abstract. RSK commissioned us to audit both the Remasc and Bridge contract in the RSKJ implementation. Remasc is responsible for computing distributing the block reward amongst the miner's while Bridge is responsible for transferring coins from Bitcoin to RSKJ and vice versa. We highlight that RSKJ does not consistently implement the deterministic block selection rule used by miners to decide which block to extend. On the other hand, we highlight future planned improvements for the design of Bridge. Overall, we propose changes to both Remasc and Bridge before discussing potential attacks that can be explored in future research.

1 Introduction

RSK (Rootstock) released their smart contract platform RSKJ as a side-chain for Bitcoin in May 2017 and its implementation relies on both EthereumJ and BitcoinJ. The former permits contracts in RSKJ to be deterministically executed by all peers on the network using their copy of the RVM (Rootstock Virtual Machine). On the other hand, the former permits RSKJ to verify that bitcoins from the Bitcoin blockchain are locked into this sidechain for use in smart contracts.

In this security audit, we focus on two pre-defined contracts Remasc and Bridge:

– **Remasc.** Distributes the block reward amongst miners,
– **Bridge.** Locks and unlocks bitcoins for use RSKJ.

There is no block subsidy (i.e. minting of new coins) in RSKJ and instead the block reward distributed by Remasc relies solely on transaction fees collected by miners in their blocks. It is also worth mentioning that RSKJ implements a variation of the GHOST protocol thus miners are rewarded for creating stale blocks that are included as *uncle blocks* in the blockchain. We highlight that while Ethereum supports paying for uncle blocks via a block subsidy that creates new coins, RSKJ implements a reward scheme that evenly splits transaction fees amongst all miners that create blocks for a given block height. Furthermore Remasc is the first reward scheme that introduces the concept of burning coins[3] if miners deviate from the protocol or if the block reward cannot be evenly split.

[3] It is worth mentioning that coins are not in fact burnt or destoryed. Instead the coins are deallocated for use in RSKJ and the Federation members are trusted not to allow the pegged coins to be returned into Bitcoin.

On the other hand, Bridge is responsible for verifying if bitcoins are locked into the RSKJ sidechain. A group of semi-trusted Federation members (i.e. functionaries) interact with Bridge for locking and releasing coins. All transfers require signatures from a threshold M of N Federation members before it is released in the respective blockchain. Of course to avoid blockchain re-organisation attacks coins must be locked in the respective blockchain for a sufficient period of time before Bridge releases coins for use in RSKJ or Federation members release coins back to Bitcoin.

Overall, the implementation of RSKJ is of excellent quality. The RSK developers have followed best practice in terms of Java Programming and have included extensive unit testing for both Remasc and Bridge. Although, we did find implementation and design issues for both which includes the following:

– We have identified a bug where miners and Remasc will execute different block selection algorithms in RSKJ.
– We have implemented a simulator to evaluate Remasc's reward scheme,
– We identify and explore future planned improvements and the associated security implications for Bridge.

This report outlined as follows. We begin by describing the reward scheme implemented in Remasc before highlighting the selection rule bug and the impact of burning coins if unconfirmed blocks held by miners is not synchronised. Then, we introduce the Bridge protocol and highlight implementation issues that may impact its future performance. Finally, we explore some future planned improvements for Bridge.

2 Remasc Contract

In this section, we provide background information on uncle blocks, the maturity period before the block reward is computed, and RSKJ's selection rule algorithm that dictates whether block b_i should extend sibling block s_1 or s_2. Next, we present how RSKJ computes the block reward such that it is split amongst two or more miners. Finally, we highlight the selection rule bug we found during our investigation and the impact of burnt coins if miners have partially disjoint sets of unconfirmed blocks.

2.1 Background

We briefly discuss the concept of uncle blocks, the maturity period before the block reward is distributed and the selection rule to dictate which block to extend.

Uncle Block. If there are two or more blocks competing for entry into the blockchain, then one block is accepted into the blockchain as a *main block*. The other competing blocks are later included in the blockchain by future main blocks and become known as *uncle blocks*. The purpose of uncle blocks is to allow their proof of work to contribute towards the blockchain's overall computational weight and as such the transactions included in an uncle block are not processed.

Algorithm 1 Remasc's fee distribution

Input: Blockchain $bc := (b_0, ..., b_k)$, Siblings $s := (s_1, ..., s_n)$, block b, reward pot λ, Selection rule was previously broken δ_k

Output: Boolean δ_k if selection rule was broken

```
1: $\lambda := \lambda + b.\textsc{fee}$    ▷ Contribute collected transaction fees to reward pot
2: $\Lambda_{br} := \lambda/5$ and $\lambda := \lambda - \Lambda_{br}$.    ▷ 20% Block Reward.
3: $\Lambda_{RSK} := \Lambda_{br}/5$ and $\Lambda_{br} := \Lambda_{br} - \Lambda_{RSK}$    ▷ 20% RSK Founder Reward
4: sendReward(RSK,$\Lambda_{RSK}$)
5: $\delta_{k+1}$ := selectionRule($b, s$)    ▷ Check if $b_{i+1}$ broke selection rule.
6: if length($s$) == 0 then
7:     if $\delta_k$ == TRUE then
8:         $\beta sr := \Lambda_{br}/10$ and $\Lambda_{br} := \Lambda_{br} - \beta_{sr}$    ▷ Selection rule penalty
9:     end if
10:    sendReward($b.\textsc{miner}$, $\Lambda_{br}$) and return $\delta_{k+1}$    ▷ Pay miner of $b$ and exit.
11: end if
12: $\Lambda_{pr} := \Lambda_{br}/10$ and $\Lambda_{br} := \Lambda_{br} - \Lambda_{pr}$.    ▷ 10% Publisher Reward.
13: $\beta_{pr} := \Lambda_{pr}$ mod length($s$)    ▷ Remainder after splitting the publisher reward.
14: $\Lambda_{pr} := \Lambda_{pr} - \beta_{pir}$
15: for $s_i$ in $s$ do    ▷ Pay miner that included a sibling in their block
16:     sendReward($s_i.\textsc{inclusion\_miner}$, $\Lambda_{pr}$/length($s$))
17: end for
18: sendReward($m_i$, $\Lambda_{pr}$/length($s$))    ▷ 1 share per sibling included in a miner's block.
19: if $\delta_k$ == TRUE then
20:     $\beta sr := \Lambda_{br}/10$ and $\Lambda_{br} := \Lambda_{br} - \beta_{sr}$    ▷ Selection rule penalty
21: end if
22: $\beta_{br} := \Lambda_{br}$ mod length($s$) + 1 and $\Lambda_{br} := \Lambda_{br} - \beta_{br}$.
23: $\Lambda_{sib} := \Lambda_{br}$/(length($s$) + 1)    ▷ Individual reward for siblings and miner of $b$
24: for each $s_i$ in $s$ do
25:     $d$ :=delay($s_i, bs$)    ▷ Compute inclusion delay for $s_i$ as an uncle in $bc$.
26:     if $d \geq 0$ then    ▷ 5% penalty for each delayed block
27:         $\beta_{sib,i} := (\Lambda_{sib} * d)/5$ and $\Lambda_{sib,i} := \Lambda_{sib,i} - \beta_{sib}$
28:     end if
29:     sendReward($s_i.\textsc{miner}$, $\Lambda_{sib}$)
30: end for
31: sendReward($b.\textsc{miner}$, $\Lambda_{sib}$) and return $\delta_k$
```

Maturity Period. There is a maturity period Δ from when b_i is accepted into the blockchain and when the reward for b_i is distributed. This grace period is necessary to provide time for miners to include any siblings (i.e. competing blocks) $s_1, ..., s_n$ in the blockchain as uncle blocks. The distribution of the block reward for b_i is computed once block $b_{i+\Delta}$ is included in the blockchain.

Selection Rule. Remasc is responsible for checking if the selection rule presented in Algorithm 2 was correctly followed and will burn 10% of the block reward if the rule is broken. Notably, this rule dictates that if there were two or more potential siblings $s_1, ..., s_n$ competing for block height $i - 1$, then block b_i should extend the sibling with either the most computational weight or at least double the fees than any other sibling.

Algorithm 2 Selection Rule (Current implementation in Remasc)

Input: Block BLOCK-1 and siblings $s = (s_1, ..., s_n)$
Output: Boolean

```
1: For each sibling $s_i$, if $s_i$.INDIVIDUAL-WEIGHT > BLOCK-1.INDIVIDUAL-WEIGHT OR
   $s_i$.FEES*2 > BLOCK-1.FEES then return FALSE
2: Return TRUE
```

2.2 Reward Distribution

We outline in Algorithm 1 how the block reward is distributed amongst the miners. RSKJ does not incorporate a subsidy in the block reward and instead relies solely on transaction fees collected in each block. All fees are sent to the reward pot λ and 20% of this pot's coins are allocated as the block reward Λ_{br} for b_i. Unlike other cryptocurrencies, this block reward Λ_{br} is further split into a RSK Founder Reward Λ_{RSK}, Publisher Reward Λ_{pr}, and Sibling Reward Λ_{sib}. Remarkably, a small portion of the block reward can also be burnt (i.e. destroyed) to penalise miners if deviation from the consensus protocol is detected. In the following we present how to compute each reward and penalty.

RSK Founders Reward. First, all blocks send 20% of the block reward to the RSK Founders.

Full Reward (no siblings) In most cases it is likely that b_i is the only block competing for entry into the blockchain for block height i and as such the Publisher and Sibling rewards are not computed. Instead, the Remasc contract will check if the Selection Rule presented in Algorithm 2 was correctly followed. If the rule is broken then Remasc will burn 10% of the block reward. The miner is sent the remaining block reward (either 80% or 70%).

Publisher Reward. The contract allocates 10% of the block reward as a publisher fee Λ_{pr} and the remainder $\beta_{pr} := \Lambda_{pr} \mod \text{length}(s)$ is burnt. Each miner $m_1, ..., m_k$ is sent d_i publisher rewards, where d_i is the number of siblings that miner m_i included in their blocks such that:

$$\sum_{i=1}^{k} \text{sendReward}(m_i, d_i(\Lambda_{pr}/\text{length}(s))),$$

Sibling Reward. Two penalities can be inflicted on the Sibling Reward. First, Remasc burns 10% of the block reward if the Selection Rule presented in Algorithm 2 is not correctly followed. The remaining block reward Λ_{br} is evenly split amongst the siblings and the miner m_i of b_i such that the individual block reward is $\Lambda_{sib} := \Lambda_{br}/(\text{length}(s)+1)$, and the remainder $\beta_{br} := \Lambda_{br} \mod \text{length}(s)+1$ is burnt.

Second, the reward for each sibling can be penalised an additional 5% for each block in which the sibling block was delayed entry into the blockchain as an uncle block. For example, if the sibling was destined for block 100, and it was included as an uncle in block 102, then it was delayed by 1 block. This incurs a

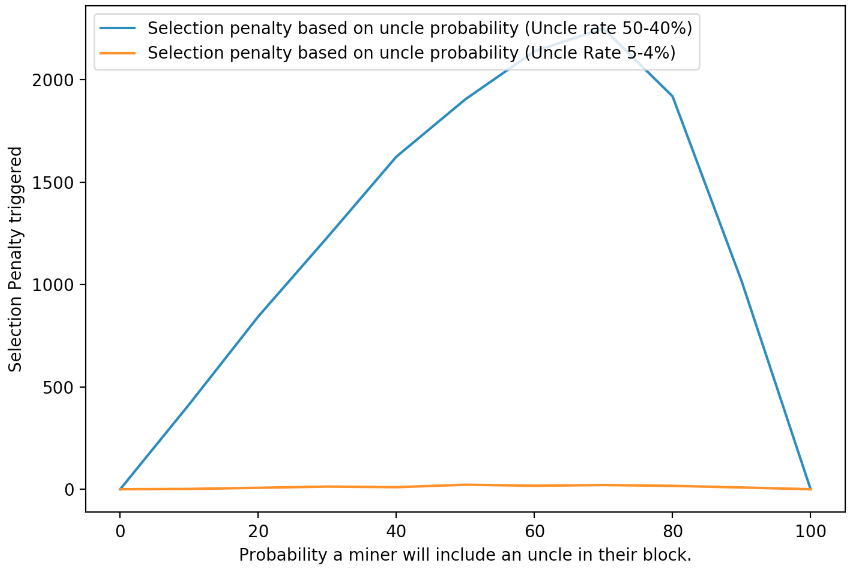

Fig. 1: The number of times the selection rule penalty is triggered when the probability that a sibling is included as an uncle block increases by 10% increments. This assumes an uncle rate of 50-40% and 10,000 blocks for each simulation run.

single 5% penalty which is burnt β_{sib}. For the rest of this paper the penalty is known as the *late inclusion of an uncle block penalty*. Finally, each sibling and the miner of b is sent their individual reward Λ_{sib}.

2.3 Selection Rule Bug

The Selection Rule algorithm is run twice in RSKJ. First by miners to choose which block to extend and second by Remasc to determine if the miner's deviated from RSKJ's consensus protocol. We briefly reiterate how the selection rule is applied before highlighting how the current implementation does not execute the same algorithm in both cases.

Miner Selection. Miners select the block with the heaviest weight (including its uncles). If competing blocks have an equal combined weight, then miners will choose the block with either the heaviest individual weight or the block that collects at least double the transaction fee's than any other sibling.

Remasc. The code checks if the chosen block has the heaviest individual weight, or if it has collected at least double the fees than any other competing siblings.

Unlike the miners selection algorithm, Remasc does not check the combined weight of blocks (i.e. the uncle weights are not included in computing the block's weight). This can result in Remasc falsely penalising the miner for deviating from the consensus protocol. For example, there are two blocks b_1, s_1 and the combined weight for the main block is b_1.COMBINED-WEIGHT = 230 and the sibling block is s_1.COMBINED-WEIGHT = 200. On the other hand, the individual weights for the main block is b_1.INDIVIDUAL-WEIGHT = 80 and the sibling block is s_1.INDIVIDUAL-WEIGHT = 90. Remarkably, the miners will select b_1 when mining blocks, whereas the Remasc contract will select s_1 when deciding whether the miner should be penalised. To conclude, this inconsistency will result in Remasc penalising miners for breaking the selection rule.

3 Remasc Simulator

We implemented a simulator to evaluate Remasc and its reward scheme. Our simulation creates a blockchain with 10,000 blocks and each block collects 1,000 coins (i.e. transaction fees). It assumes there are five miners with $40, 10, 10, 20, 20$ percent hashing power and the simualtor can be configured in the following way:

– Uncle rate[4] can be set to 50-40%[5] or 5-4%[6],
– Late inclusion of uncle block probability[7] can be set from 0% to 100%,
– Selection rule can be consistently or inconsistently applied by Remasc.

The rest of this section considers two penalities for studying the impact of burning coins. First, we investigate the selection rule bug and highlight that in the worst-case over 3.5% coins are burnt. Second, we investigate the late inclusion of uncle block's penalty impact and highlight that in the worst-case over 35% coins are burnt. Next, we present how to trigger both penalties before discussing their impact.

3.1 Triggering the Selection Rule Penalty

Figure 1 demonstrates the frequency of triggering the selection rule. We observe that this trigger only occurs if two or more competing blocks include different sets of uncle blocks. In fact, there are three cases to consider:

1. Two or more competing blocks always include the same set of uncles.
2. Two or more competing blocks do not always include the same set of uncles.
3. Two or more competing blocks do not include any uncles.

The first and third case are conceptually identicial as only a block's individual weight needs to be considered.[8] The second case triggers the selection rule penalty

[4] Frequency that two or more competing blocks for a block height are created.
[5] Uncle rate of 50-40% is computed by allowing miners to create blocks with a probability α, where α represents their hash power.
[6] Uncle rate of 5-4% is computed by allowing miners to create blocks with probability $\alpha * (1/10)$, where 10 represents the desired 10 second interval.
[7] Likelihood that a sibling is included as an uncle in a new block.
[8] Either no uncles are included, or all competing blocks include the same uncle set and as such have the same weight.

Algorithm 3 Selection Rule (Proposed fix for Remasc contract)

```
Input: Block BLOCK-1 and siblings $s = (s_1, ..., s_n)$
Output: Boolean
1: For each sibling $s_i$, if BLOCK-1.TOTALWEIGHT $>$ $s_i$.TOTALWEIGHT then return
   TRUE.
2: For each sibling $s_i$, if BLOCK-1.TOTALWEIGHT $\neq$ $s_i$.TOTALWEIGHT then return
   FALSE
3: For each sibling $s_i$, if $s_i$.INDIVIDUAL-WEIGHT $>$ BLOCK-1.INDIVIDUAL-WEIGHT OR
   $s_i$.FEES*2 $>$ BLOCK-1.FEES then return FALSE
4: Return TRUE
```

as two or more competing blocks can include disjoint sets of uncle blocks and thus have different weights.

Figure 1 highlights that the selection rule and the late inclusion of uncle blocks penalty is most likely to be triggered if miners have a 70% probability that each sibling is included as an uncle block in their new block. Therefore all simulated scenarios in this report assumes the 70% probability in order to evaluate the worst-case scenario. Next, we explore the selection rule's impact in terms of burnt coins before highlighting that it is less severe than the late inclusion of uncle block penalty.

3.2 Selection Rule Penalty Impact

We simulated four scenario that varied whether the selection rule is inconsistently/consistently applied and if the uncle rate is set to 50-40% or 5-4%. As well, each scenario is simulated forty times. Our results show that the selection rule penalty is neglibile in 3/4 simulated scenarios as it was unlikely to be triggered.

The final scenario demonstrated that around 35k coins are burnt if the selection rule is inconsistently applied and the uncle rate is set to 50-40%[9]. This represents around 3.5% of coins in circulation (35,000/1,000,000) if we assume that transaction fees collected in blocks are never re-spent and thus represent fresh coins. In practice, we suspect that around 3.5% or more coins will be burnt as it is unlikely that all collected transaction fees represent new coins.

Next we explain two approaches for fixing the selection bug before presenting our simulations for the late uncle block inclusion penalty.

3.3 Fixing Selection Rule Bug

Fixing the selection rule bug requires updating Remasc or changing how miners select blocks. We explore both solutions in the following.

Update Remasc. The first approach can be seen in Algorithm 3. This involves updating Remasc to include checking the weight of uncle blocks, and

[9] Private conversation identified that this is in the realm of possibility and thus worth investigating.

only checking the individual weight if two blocks have the same combined weight. As well, RSKJ must be updated to maintain each sibling block's list of uncle hashes. If the uncle blocks are not verified, then this can result in a miner creating fake uncle blocks. Although, there does not appear to be a clear benefit to creating fake uncle blocks as this still requires performing the proof of work.

Update Miner Selection. The second approach requires no changes to Remasc as the miner's simply stop checking a block's combined weight when it is deciding which fork to extend. It is worth mentioning that this is how Ethereum works today as it relies on the individual weight of a block, and not the combined weight of uncles. This appears to be the easiest approach to implement in RSKJ, but provides less computational weight towards its blockchain immutability.

3.4 Delayed Uncle Block Inclusion Impact

We tested two scenarios that varied whether the uncle rate was 50-40% or 5-4% and each scenario is simulated forty times. As well, we assume the uncle block inclusion probability is set to 70%. This simulation represents the scenario where miners do not have a consistent view of uncle blocks on the network.

Our results highlight that 35-40k coins are burnt if the uncle rate is 5-4% and 350-400k coins are burnt if the uncle rate is 50-40%. The severity of burning these coins can be seen if we consider that in the ideal (and non-realistic) case 1 million coins are collected as transaction fees by Remasc. In the former approximately 3.5-4% of coins are burnt, whereas in the latter 35-40% of coins are burnt.

Figure 2 highlights that approximately 4.1k uncle blocks are included in the blockchain. Around 37% of sibling are included as uncle blocks immediately and incur no penalty. On the other hand, the remaining 63% of uncle blocks incurred a 5% to 25% penalty.[10] We leave it as future work to further explore the impact of this penalty and whether there is a feasible selfish-mining strategy that can leverage these results.[11]

4 Bridge

Bridge records the number of bitcoins locked into RSKJ and supports co-ordination amongst Federation members for locking/releasing coins. In this section, we outline the protocol that is currently implemented in RSKJ, highlight performance issues before exploring future planned improvements.

[10] Every delayed block will incur an additional 5% penalty and our simulation permits blocks to be delayed for up to 5 blocks.

[11] One notable strategy we noticed was that miners can include only their own siblings as uncle blocks and this provided a slight advantage. We omitted these results as it requires further investigation.

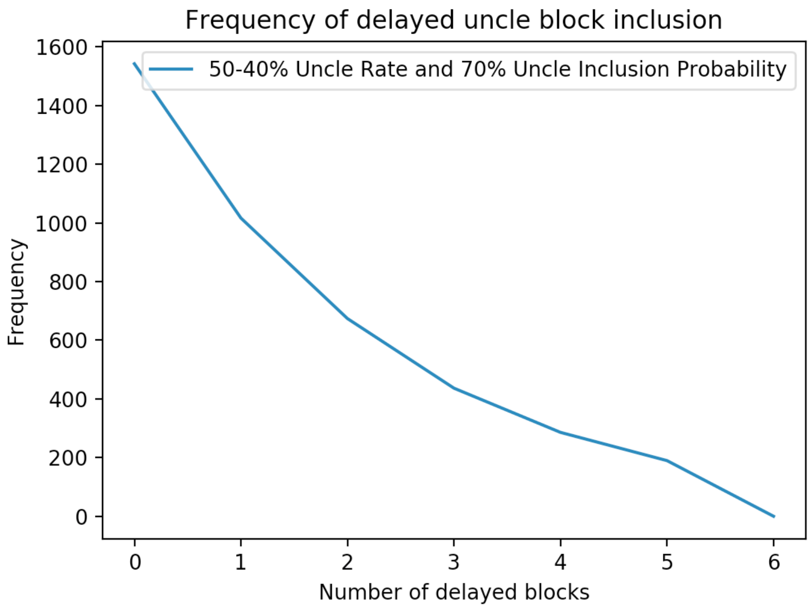

Fig. 2: The selection rule penalty is triggered if miners create blocks that include different siblings, and if the same selection rule is not applied by both the miners and Remasc contract. All four scenarios assume that a miner will include each sibling with a 70% probability.

4.1 Lock and Release

RSK is responsible for appointing members of the Federations whom are trusted to lock and release coins in RSKJ. We briefly describe the lock and release procedures as currently implemented in RSKJ below:

Lock. Users send bitcoins to the Federation Address (i.e. a multisig address) in Bitcoin. After a contest period has passed, one Federation member is responsible for sending RSKJ an SPV (Simplified Payment Verification) proof[12] that these bitcoins are under the Federation's control. Once the SPV proof is verified then RSKJ releases an equal portion of coins to the sender's address (i.e. the user's public key that authorised sending these coins in Bitcoin).

Release. The current coin owner in RSKJ is responsible for initating the process of releasing these coins for use in Bitcoin[13]. Bridge creates the bitcoin transaction that will release the coins and requires signatures from M of N Federation members to authorise the release.[14] These coins are considered released

[12] A merkle tree branch and a Bitcoin block header to prove this transaction was accepted into the blockchain at a specified block height.

[13] User calls `BridgeSupport.releaseBtc()`.

[14] BTC Transaction is stored in the array `rskTxsWaitingForConfirmations` and `BridgeSupport.addSignature()` collects the Federation's signatures for a btc release transaction.

and removed from RSKJ once a sufficient number of signatures are collected as anyone can broadcast the Bitcoin transaction.

It is also worth mentioning how RSKJ distinguishes if a transaction will lock or release bitcoins:

– A lock transaction must send bitcoins to the Federation's address and cannot contain an input that also spends coins from the Federation's address.
– A release transaction must contain at least one input that spends coins under the Federation's control.

Issue with Releasing Coins. The lock inspection rule only permits the Federation to release bitcoins from RSKJ if there is one or more corresponding spendable outputs that match its value. This is restrictive as a transaction cannot both release and lock bitcoins into RSKJ. As a result the Federation must release all bitcoins spent in a transaction. To overcome this issue requires removing the rule '*cannot contain an input that also spends coins from the Federation's address*' when inspecting the lock transaction. Instead RSKJ can simply track bitcoins that are sent and received using the Federation's address.

Recording Transactions. RSKJ currently records the identification hash of bitcoin transactions that either lock or release coins.[15] The purpose of this set is to prevent Bridge processing the same transaction two or more times. However a transaction hash removal policy does not exist and thus the set will grow indefinitely. We feel that while this is not necessarily a security issue it is important to highlight for RSKJ's long-term performance.

We propose that timers can be used for including and removing transaction hashes. One possible approach involves recording both the transaction hash and the inclusion block height (i.e. which block this transaction was included in) if it appears within the most recent Δ_{ACCEPT} blocks (i.e. 5k blocks). Once this timer expires then Bridge can verify that the hash is confirmed in the blockchain expires before releasing the coins and removing the transaction hash. A concrete protocol for this approach is left as future work.

4.2 Access to Bridge.

Rizzo suggested to investigate the implications of permitting any user to call functions in Bridge and whether access should be restricted to only Federation members. All functions can be called by any user in the current implementation and we will discuss each function seperately to evaluate if there are any potential attack vectors. A brief summary of the functions are provided before discussing each one in-depth:

– **Receive Headers.** Update RSKJ with the latest set of block headers from Bitcoin.

[15] This is recorded in `BridgeStorageProvider.getBtcTxHashesAlreadyProcessed()` which is a SortedSet.

- **Register BTC Transaction.** Initiate the process of transferring coins from Bitcoin to RSKJ.
- **Release BTC.** Initiate the process of transfering coins from RSKJ to Bitcoin.
- **Add Signature.** Submit a Federation Memember's signature for a BTC Transaction that releases coins from RSKJ to Bitcoin.
- **Update Collections.** Responsible for transiting a Release Transaction from initiation to finalisation.

Receive Headers. This function accepts a list of serialised block headers. Bridge will deserialise each block header and append it to the best known blockchain. Currently, the implementation processes every block header even if one fails and is not appended to the blockchain. This function can be provided a list of erroneous block headers and every block header will be processed by Bridge. We propose that this function should roll-back any changes if a single block header cannot be appended to the blockchain. Furthermore, there are currently no unit tests for this function and we propose that some are written.

Register BTC Transaction. This function requires a bitcoin transaction, a merkle tree branch and the block height of its inclusion. The SPV (Simplified Payment Verification) proof is checked to confirm that the bitcoin transaction is indeed accepted into the blockchain and that the transaction sends bitcoins to the Federation's address. Bridge confirms that this transaction has not been previously processed and that it has achieved sufficient depth in the Bitcoin blockchain. Next, Bridge checks if the transaction is either locking or releasing coins. In the former the transaction's creator is sent an equal number of coins in RSKJ and in the latter Bridge does not perform any action. Finally, a copy of the transaction hash is stored to prevent it being re-processed in the future.

We investigated the possibility of performing a replay attack where the transaction is submitted twice to the function. First, the function can detect if the same transaction is submitted twice using its hash and will not re-process it. Second, Bridge waits for the transaction to achieve sufficient depth in the blockchain and thus it is not feasible to perform a transaction malleability attack.

Next, we considered the possibility of modifying the Bitcoin Script such that it contained the Federation address, but sent money back to the sender.[16] It is worth mentioning that `Bridge.isLocktx()` relies on BITCOINJ to determine if the coins are sent to the Federation's address. This eventually calls `TransactionOutput.IsMine()` which inspects the Bitcoin Script for the following three types of addresses:

1. A single public key (i.e. a 'raw' public key).
2. A pay to public key hash (i.e. a single bitcoin address).
3. A pay to script hash (i.e. multi-signature address).

Bridge only stores the pay to script hash of the Federation's multi-signature address in its wallet and thus any malicious transaction must satisfy the third

[16] For example, it is feasible to craft a Bitcoin script such that the sender can claim the coins immediately, whereas the Federation member can claim the output after $\Delta_{FEDERATION}$.

inspection rule. Technically, this requires any new coins sent to the Federation to follow the Bitcoin Script template `OP_HASH160 <HASH> OP_EQUAL` and for `<HASH>` to represent a pay to script hash that contains the Federation's multi-signature address. It does not appear feasible to craft a malicious Bitcoin script that appeared to send coins to the Federation due to the deterministic nature of this type of Bitcoin Script.

Release BTC. The RSKJ transaction that calls Bridge is used as input to the function. First, sender's public key is extracted and a new transaction output for Bitcoin is created that sends equal number of coins to the sender.[17] Second, the new transaction output is stored in `rskTxsWaitingForConfirmations` for future use in `Bridge.UpdateCollections()`. It is worth mentioning that there does not appear to be any obvious exploitable feature. There is no direct user-defined input and it requires a real RSKJ transaction to successfully call the Bridge contract.

Add Signature. This function requires a public key, the corresponding signature and the respective RSKJ transaction hash. Bridge checks that the pubic key belongs to a Federation member before fetching the corresponding transaction that matches the hash provided and verifying that this signature is for this transaction. This signature is stored in Bridge and if a threshold of signatures for this transaction is reached then it is moved from the list `rskTxsWaitingForSignatures` to the `rskTxsWaitingForBroadcasting` list. Again, we could not find any exploitable flaws as Bridge verifies all user-defined input correctly.

Update Collection requires no user-defined input. This function is responsible for transiting a BTC transaction through three stages such as `CONFIRM, SIGNATURES, BROADCAST` once a Release RSKJ transactio is registered using the `Release BTC` function. A transaction can transition from `CONFIRM` to `SIGNATURES` once it has achieved depth in the blockchain. Next, it requires a threshold of signatures from Federation members to be sent using the `Add Signature` function before it can transition from `SIGNATURES` to `BROADCAST`. Finally a grace period is provided to broadcast the BTC transaction before it is removed altogether from Bridge.

We highlight that the current implementation only permits one RSK transaction to be stored in `rskTxsWaitingForSignatures`. This can be seen as there is a while loop that checks if any release transaction can transition from `CONFIRM` to `SIGNATURES`. The inner code of this while loop will only be executed if `rskTxsWaitingForSignatures` is empty.[18] Of course, if a single release transaction transitions from `CONFIRM` to `SIGNATURES` then this inner code will not execute. We propose fixing this while loop to permit the list to contain more than one item.

4.3 Future Planned Improvements

We briefly identify and explore future improvements for Bridge. This includes how to replace compromised federation keys, incentives for Federation members

[17] The public key used in the RSKJ transaction is converted to a Bitcoin address.

[18] `iter.hasNext() && provider.getRskTxsWaitingForSignatures().size()==0`

to participate and whether access to some functions in the Bridge contract should be restricted to Federation members only.

Federation Key Replacement. Locking coins into RSKJ involves sending bitcoins to a multi-signature address that is under the Federation's control. Each member of the Federation has a single public-private key pair that is included as part of this multi-signature address. Of course, releasing these coins requires M out of N signatures, where N is the membership size of the Federation. Unfortunately there is no contingency plan to replace a member's public-private key pair in the event that it is potentially compromised. There are two issues that must be considered:

1. **New Consensus Rule.** A new public-private key pair for the Federation member must be computed and included as a new consensus rule in RSKJ.
2. **Bitcoin Transactions.** All locked coins in Bitcoin must be sent to the Federation's new multi-signature address.

One solution for updating a Federation member's public key is to permit a threshold of Federation members to sign an key replacement message. This message invalidates the compromised public key and replaces it with a public key that was freshly generated by the respective member. Once replaced Bridge can enforce that all newly locked coins are under controlled by the new Federation's multi-signature address before an equal portion of coins can be released in RSKJ. Of course, there are other problems such as authenticating the ownership of this new public key and whether the miners should be required to also approve a Federation's new public key.

On the other hand, the Federation may need to transfer all locked coins in Bitcoin to the new multi-signature address in the event that two or more Federation member's public keys are compromised. RSK have not allocated coins to pay for this transfer which is likely to be expensive as transaction fees in Bitcoin were 420 satoshis/byte in June 2017. Furthermore, Bridge cannot yet support transferring coins due to the transaction inspection rules outlined earlier.

We recommend that Bridge is updated to support transferring coins to a new Federation's multi-signature address. Furthermore, we recommend that either all Federation Members must deposit coins into Bridge (i.e. a stake) to cover the transaction fees. Otherwise, the block reward's burnt coins or a portion of the reward can be allocated for this constingency plan.

Incentive for Federation Participation. Federation members are trusted to participate in Bridge in order to provide transparency to the community. However there is currently no financial incentive for them to use Bridge as opposed to running a classical consensus protocol for signing release transactions. It is possible to encourage usage of this contract by creating a Federation Reward in Remasc. This reward can be proportionally split amongst members of the Federation based on the number of signatures (i.e. the signatures for releasing coins from RSKJ to Bitcoin) collected in Bridge. Otherwise, Bridge can be simplified by removing the functionality to collect signatures altogether and simply deallocate the coins from usage in RSKJ once a user initiates this process.

Privacy of a Release Transaction. Bridge does not provide privacy to the sender when coins are released from RSKJ to Bitcoin. We suspect that a publicly verifiable on-chain mixing protocol is required such that Bridge can deterministically construct a CoinJoin transaction upon releasing the coins. Although we leave this as future work to construct a concrete protocol. It is also worth mentioning RSK may not incorporate an on-chain and trustless mixer within the forseeable future due to implementation complexity. Thus, we recommend that senders mix their coins prior to initiating a release transaction and if this proves successful then RSK can later decide to incorporiate the mixing protocol into Bridge.

5 Conclusion

Our security audit of RSKJ Ginger 0.2.0 was comissioned by RSK. We focused on Remasc that controls how the block reward is distributed amongst miners and Bridge that allows coins to be transferred from Bitcoin to RSKJ and vice versa. Before we highlight our findings it is worth mentioning that the RSK developers have implemented both Remasc and Bridge to an excellent standard.

In Remasc we identified that the block selection rule is not consistently applied for both miners and Remasc. To evaluate the impact of this inconsistency we built a simulator that can vary the uncle rate, probability that an uncle block is included in the blockchain and whether the selection rule is consistently applied. Out of simulated four scenarios we identified one scenario that represented the worst-case and enabled over 3.5% coins in circulation to be destroyed. This led us to also identify that if this bug was fixed then it remains feasible for 35-40% of coins in circulation to be burnt in the worst case if miners do not have a consistent view of unconfirmed blocks.

In Bridge, we identified performance issues such as the contract only permitting one release transaction to collect signatures from the Federation members at a time. As well, RSKJ cannot release coins to Bitcoin unless there is a combination of locked coins of equal value. Next, we highlight several future improvements to resolve security implications in RSKJ. This includes a contingency plan in the event that one or more Federation member's public keys are potentially compromised, providing an incentive (or reward) for Federation member's to participate in Bridge and whether releasing coins from RSKJ to Bitcoin can also provide financial privacy.

6 Acknowledgements

We thank the RSK developers for interacting and answering questions throughout the process of this audit.