A Blockchain Model in Tamarin and Formal Analysis of Hash Time Lock Contract

Colin Boyd
NTNU - Norwegian University of Science and Technology, Trondheim, Norway
colin.boyd@ntnu.no

Kristian Gjøsteen
NTNU - Norwegian University of Science and Technology, Trondheim, Norway
kristian.gjosteen@ntnu.no

Shuang Wu
NTNU - Norwegian University of Science and Technology, Trondheim, Norway
shuang.wu@ntnu.no

Abstract

Formal analysis and verification methods can aid the design and validation of security properties in blockchain based protocols. However, to generate a reasonable and correct verification, a proper model for the blockchain is needed. In this paper, we give a blockchain model in Tamarin. Based on our model we analyze and give a formal verification for the hash time lock contract, an atomic cross chain trading protocol. The result shows that our model is able to identify an underlying assumption for the hash time lock contract and that the model is useful for analyzing blockchain based protocols.

2012 ACM Subject Classification Security and privacy → Formal security models

Keywords and phrases Blockchain model, Tamarin, Hash time lock contract, Formal verification

Digital Object Identifier 10.4230/OASIcs.FMBC.2020.5

Supplementary Material The source code can be found at
`https://github.com/ShuangWu121/Tamarin-code-for-HTLC-verification`

1 Introduction

In a blockchain based protocol, the blockchain serves as a reliable public ledger to deliver ordered outcomes to all its agents. Protocols can be executed by using smart contracts and the execution states are recorded on the blockchain. The blockchain essentially performs as a distributed trusted party to reduce the direct trust between the entities in the system.

In order to formally verify the security properties of protocols built on top of blockchains, a proper model for blockchains is needed. The model must capture the interesting properties of blockchains, without becoming too complicated. A blockchain is more than a public ledger. The dynamics of the growing chain provide a time reference: the relatively stable growth of the blockchain height offers a "global time". With respect to this global time, a blockchain enables a time lock function used as a restriction specifying that a transaction cannot be added to blockchain before a set time (actually a given chain height). Thus in order to capture properties of time lock contracts a blockchain model should include the following features.

Model time. The blockchain model should contain a global time reference in the system. Different blockchains contain different global time references. The model should be able to capture time-relevant risks, such as race conditions.

Model the time lock restriction. The time out event of a time lock should be triggered by the time reference. It should be possible to model the risk introduced by a time lock that times out earlier or later than is expected.

Clarify the underlying assumptions. If a certain property of the blockchain fails, a protocol built on top of it will not be safe either.

Related work

In 2014, Andrychowicz et al. [2] modeled a multiparty computation contract in Bitcoin by using timed automata. Back then the time lock functionality in Bitcoin was limited and consequently the structure of the contract is different today. Bursuc and Kremer [4] used Tamarin to model the blockchain as a public ledger, and analysed the ZKCP [7] protocol built on top of it. But in their model, the executions are not time-relevant. Turuani et al. [10] give a formal model in ASLan++ of the two-factor authentication protocol used by the Electrum Bitcoin wallet. Bentov et al. [3] propose a real-time cryptocurrency exchange service, and they give an informal cryptographic proof for the security of a hash time lock protocol, with a probabilistic modeling of forking. Sun and Yu [9] give a formal verification model for five kinds of security issues in the Ethereum blockchain using Coq.

Our contributions

To address the above challenges, we build a blockchain model in Tamarin [8]. The model defines a public ledger and a global time reference for the system, with time lock functionality built on top. We also define the security properties of an atomic cross chain trading protocol and give a formal proof for the security of the hash time lock contract (HTLC). To our knowledge, this is the first HTLC analysis by formal verification tools. The proof clarifies a “hidden” security assumption: the growth speed of the two blockchains need to be stable, otherwise security will fail. We further use our model to analyze an older version of the hash time lock contract, and Tamarin is able to find a flaw. Even if the assumption looks trivial and the flaw is somewhat obscure, this demonstrates that our model is able to address the above challenges and can be used for formal verification of blockchain based protocols.

2 Background

2.1 Hash time lock contract

The goal of the hash time lock contract (HTLC) is to exchange different cryptocurrencies between two players in a decentralized way. Consider Alice who wants to exchange Bitcoin for Altcoin, and Bob who wants to exchange Altcoin for Bitcoin. They could do the following:

1. Alice creates a transaction that is locked by a hash value $\mathsf{h} := \mathsf{H}(\mathsf{sk})$ to send Bob 1 Bitcoin. Bob can take the funds only if he can provide the hash pre-image. This is Alice’s *commitment transaction.*
2. After Alice’s commitment transaction has been confirmed on the Bitcoin blockchain, Bob creates a transaction (contract) to send 1 Altcoin to Alice, locked using the same hash value $\mathsf{h} := \mathsf{H}(\mathsf{sk})$. This is Bob’s commitment transaction.
3. Alice takes Bob’s Altcoin by providing her signature and the pre-image of the hash lock. Bob learns the hash pre-image and unlocks the Bitcoin that Alice sent to him.

In order to avoid an interrupted protocol leaving players’ funds locked forever, the commitment transactions are also locked by time locks. After the time lock times out, the transaction can be redeemed by the sender. The time lock of Alice’s commitment transaction

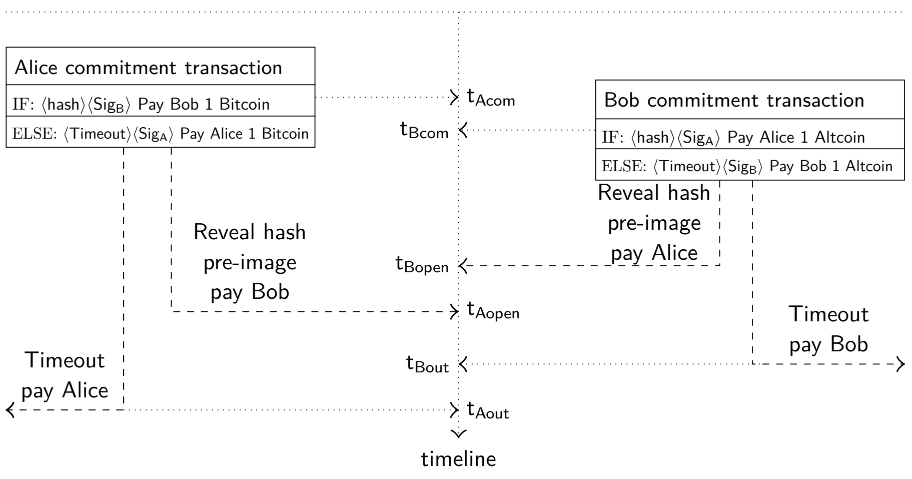
Alice commitment transaction
IF: ⟨hash⟩⟨Sig_B⟩ Pay Bob 1 Bitcoin
ELSE: ⟨Timeout⟩⟨Sig_A⟩ Pay Alice 1 Bitcoin
Bob commitment transaction
IF: ⟨hash⟩⟨Sig_A⟩ Pay Alice 1 Altcoin
ELSE: ⟨Timeout⟩⟨Sig_B⟩ Pay Bob 1 Altcoin
t_Acom
t_Bcom
Reveal hash pre-image pay Alice
t_Bopen
Reveal hash pre-image pay Bob
t_Aopen
Timeout pay Bob
t_Bout
Timeout pay Alice
t_Aout
timeline

Figure 1 Hash time lock contract execution.

should be longer than Bob's commitment transaction, since in the case that Alice takes Bob's Altcoin at the last moment before Bob's commitment transaction timed out, her commitment is still locked by the time lock and Bob still has time to take Alice's Bitcoin. A successful execution of a hash time lock contract can be seen in figure 1. Notice that in the figure we use the same structure (script) to describe Altcoin and Bitcoin, but in fact we just consider two Bitcoin-like blockchains. As long as the blockchain supports both timelock and hash lock functionalities, the hash time lock contract protocol can be used.

The above description is the latest version of the hash time lock contract [6], where a time lock restricts when a transaction can be spent by its following transaction. Thus the two potential outputs of a commitment transaction are specified inside the commitment transaction. The previous version [5, 1] utilizes a time lock that only restricts when a transaction can be added to blockchain. The time lock is then not specified in the commitment transaction, but in the redeem transaction. In this case the redeem transaction must be signed by multiple signatures, thus the procedure involves two players exchanging signatures on transactions.

2.2 About Tamarin

Tamarin [8] is an automatic symbolic protocol verification tool. Given a protocol, the user specifies the roles running the protocol and their behaviors, the adversary model and the security properties by using the Tamarin programming language. Tamarin applies malicious adversarial behavior to the roles and uses a backward search method to generate counter-examples to the security claims. Tamarin ends up with either a proof that demonstrates that the given protocol satisfies the security properties, or Tamarin would give an attack for a failed security claim.

In Tamarin, the communication messages, fresh randomness and the states of the protocols are represented by symbolic terms called facts. There are two special facts to model the interaction with the untrusted environment: $\mathsf{In}(*)$, $\mathsf{Out}(*)$,representing the protocol's input and output from and to the environment. All the messages forwarded by $\mathsf{In}(*)$ and $\mathsf{Out}(*)$ can be learned by the adversary. The fact $\mathsf{K}(\mathsf{x})$ denotes the adversary learning x. Some facts are linear, which means that they can be used only once. The protocols and the specifications of the adversaries are modeled by using multiset rewriting rules. These rules and facts define a labeled transition system. Security properties are either defined in terms of traces of the transition system or the observational equivalence of two transition systems.

A role in the protocol is specified by Tamarin multiset rewriting rules. A rule consists of three elements: $(\mathsf{L}, \mathsf{A}, \mathsf{R})$:$[\mathsf{L}] - [\mathsf{A}] \to [\mathsf{R}]$, the left side facts L (states, messages of the protocol) are the premises of the rule, the right side facts R are rule conclusions, and the actions in the middle square brackets A are to label the traces. A rule can be executed as long as its premises exist in the current system states. Then the facts in the premises will be removed from the current system states, while the facts in conclusion will be added. Users can also add restrictions to enforce that only traces satisfying the restrictions are considered by Tamarin's backward search.

We illustrate Tamarin syntax by introducing a toy Diffie-Hellman key exchange protocol:

```
rule Server_1:
[ Fr(~a) ]-->[ S_1( ~a, 'g'^~a), Out( 'g'^~a) ]

rule Client:
[ Fr(~b), In( X ) ]--[ Key(X ^~b) ]->[ Out('g' ^~b ) ]

rule Server_2:
[ S_1( a, 'g'^a ), In( Y ) ]--[ Key( Y^a ) ]->[ ]
```

In the first step, the server generates fresh randomness ~a (the symbol ~ denotes a fresh nonce, the function Fr(∗) means generating a fresh nonce), sends g^a to client by the fact Out('g' ^~a), and it records the inner state by the fact S_1(~ a, 'g' ^~a) . This state will be used in next step of the server with the name Server_2.

The client receives the message from server by fact In(X), it then generates the session key according to the Diffie-Hellman key exchange protocol. This trace and its parameters are recorded by the action Key(X^~b), this will later be used to claim the security property of the protocol. The server's next step generates a similar action.

The security properties to be evaluated are defined by lemmas. In the above example we want to claim there is no adversary that can learn the secret key.

```
lemma Key_secrecy:
" All key #i . Key( key )@i ==> not Ex #j . K( key )@j "
```

The lemma Key_secrecy specifies that in all the traces that have an action Key(key), no adversary could learn the input of the action, namely, the value key, expressed by statement that there is no fact K(key) in the trace.

3 Tamarin Blockchain model

3.1 Simplification

A complete blockchain model would be too complex for Tamarin to work with, if it could even be expressed. We have simplified the structures of the transactions and blocks to make our blockchain model simpler, while still expressive enough to capture the essential elements for describing attacks on the protocols, and thus making verification possible. We let the blocks only include zero or one transactions, and forks are not allowed, thus we only consider the blocks that are already stable. The consensus protocol and cost are not modeled in our work. A transaction contains six elements: the id of the transaction that is being spent, the sender's address (we simply denote the addresses as public keys), the input signature (or script), output address (or script), the block sequence and the id of this transaction.

We set the relative growing speeds of the two blockchains to be the same. This simplification will not change the primary mechanism of the protocol because if the speed of Alice's blockchain is two time faster than Bob's blockchain, the time lock of Alice will be twice as long to ensure that it is longer than Bob's time lock in real time.

3.2 Tamarin blockchain model rules

We describe the rules of our blockchain model in two parts: the ledger rules and the global time rules. The ledger rules add a transaction to a block. The global time rules generate the time state called 'Tick' to specify the time point of a block being added to the blockchain.

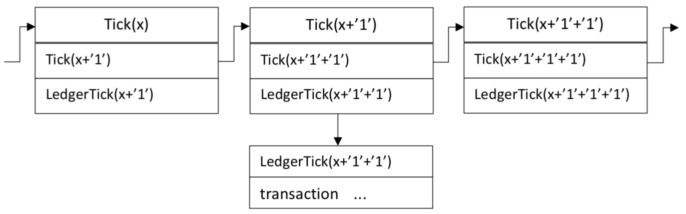

Figure 2 Tick chain.

In the global time rules, each time Tick has a unique parameter time. (It also has another parameter to tie a Tick to a specific blockchain, so that block chains can grow at different speeds. For simplicity we leave it out of Figure 2.) When generating a new Tick, an older Tick that has the largest time will be consumed and the time will be increased by one. Thus the Ticks form a time state transition chain that we call a Tickchain. Given the uniqueness of each Tick and since "time" is always increasing, each Tick can be considered as an empty block and the Tickchain can serve as base for a blockchain. We refer the blockchain in our Tamarin model as Tickchain and its blocks are called Tickblocks. In order to model adding a transaction to a certain Tickblock, a LedgerTick with a parameter Height equal to time is generated along with a new Tick. The ledger rules consume a LedgerTick to create a new transaction. In this way we bind a transaction to a Tickblock. The parameter Height also implies a sequence of transactions. After the executions of a protocol, there may be some LedgerTicks left without being consumed, which means that no transaction was added to the corresponding Tickblock.

Global time rules. There are two rules: Tick_start and Tick to create a blockchain. (We also use Tick to name the rule that generates a Tick state.) The rule Tick_start initiates the clock and the rule Tick updates the clock, i.e. increase the clock by adding '1'. There are three facts involved in the global time rules: $\mathsf{Chain}(\mathsf{BC})$, $\mathsf{Tick}(\mathsf{BC}, \mathsf{time})$ and $\mathsf{LedgerTick}(\mathsf{BC}, \mathsf{height})$. $\mathsf{Chain}(\mathsf{BC})$ specifies which blockchain. $\mathsf{Tick}(\mathsf{BC}, \mathsf{x})$ and $\mathsf{LedgerTick}(\mathsf{BC}, \mathsf{x})$ denote a certain block with the block height x. $\mathsf{Tick}(\mathsf{BC}, \mathsf{x})$ will be consumed by the Tick rules to updates the clock by iteration. $\mathsf{LedgerTick}(\mathsf{BC}, \mathsf{x})$ will be consumed by the ledger rules to link a transaction to a block. All these facts are linear facts that can be only consumed one time.

Global time rules

Tick_Start	Tick
Input: $\mathsf{Chain}(\mathsf{BC})$	Input: $\mathsf{Tick}(\mathsf{BC}, \mathsf{time})$
Output: $\mathsf{Tick}('1'), \mathsf{LedgerTick}('1')$	Output: $\mathsf{Tick}(\mathsf{BC}, \mathsf{time} +' 1')$, $\mathsf{LedgerTick}(\mathsf{BC}, \mathsf{time} +' 1')$

Ledger rules. The ledger rules model the nodes in blockchain network: the nodes get transaction information from the network, check its validity and then record the transaction to the blockchain.

There are two types of transaction in our model: SimpleTx(BC, InTx, InSig, OutPk, tx, height) to model the transactions without the hash and time lock, and CommitTx(BC, InTx, InSig, OutScript, tx, height) to model the transactions locked by a hash and a time lock. In these two transactions, BC denotes which blockchain the transaction belongs to; InTx is a nonce that identifies a previous unspent transaction owned by the sender; InSig is the sender's signature. tx is a nonce that identifies this transaction; height specifies in which block this transaction has been recorded. In the simple transaction, the OutPk is the receiver's address, while the OutScript in a commitment transaction is a hash time lock contract script, specifying the hash value, time lock value and receiver's address.

There are five rules to model the blockchain behaviors, Mine_Coin, Simple_Tx, Commit_Tx, Commit_open and Commit_timeout. The purpose of these rules are to generate blocks that contain different types of transactions and append the block to the blockchain. Mine_Coin creates the original coins of the blockchain. Simple_Tx spends a simple transaction and creates a new unspent simple transaction. Commit_Tx creates a transaction that is locked by a hash and a time lock. Commit_open models the transaction that is spent by revealing the hash pre-image. The Commit_timeout model the commit transaction that is spent by sender redeeming the transaction in the case of timeout.

Ledger rules

Mine_Coin

Input: Fr(∼ n), PK(A, pkA), ledgerTick(BC, t)
Output: SimpleTx(BC,′0′,′0′, pkA, n, t)

Simple_Tx

Input: SimpleTx(BC, InTx, InSig, Pk, n, height), In(⟨tx, Sig, pkB⟩), ledgerTik(BC, t)
Output: SimpleTx(BC, n, Sig, pkB, tx, t)

Commit_Tx

Input: SimpleTx(BC, InTx, InSig, Pk, n, height), In(⟨Sig, ⟨pkA, timelock, hash, pkB⟩⟩), LedgerTick(BC, t)
Output: Commit_Tx(BC, n, Sig, ⟨pkA, timelock, hash, pkB⟩, tx, t)

Commit_open

Input: Commit_Tx(BC, InTx, InSig, ⟨pkA, timelock, hash, pkB⟩, n, height), In(⟨⟨Script1, Script2⟩, PKaddress⟩), LedgerTick(BC, t),
Output: SimpleTx(BC, n, Sig, pkB, tx, t)

Commit_timeout

Input: Commit_Tx(BC, InTx, InSig, ⟨pkA, timelock, hash, pkB⟩, n, height) In(⟨Script1, PKaddress⟩) LedgerTik(BC, t)
Output: SimpleTx(BC, n, Script1, Pk, tx, t)

Restrictions. The no double spending property of blockchain is guaranteed by restriction rule. When a transaction has been spent, an action Spend(BC, tx, M, t) will be recorded in the Tamarin system to identify the event. On a single blockchain, for a transaction x, there can only exist one Spend(BC, x, M, t).

```
restriction DoubleSpending:
"All BC x n m t1 t2 #i #j .Spend(BC,x,n,t1)@i &Spend(BC,x,m,t2)@j==>#i=#j"
```

To help Tamarin reason more efficiently, we add one more restriction HappenBefore. This restriction simply tells Tamarin that a transaction that has a larger block number should happen later than a transaction that has a smaller block number.

```
restriction HappenBefore:
"All BC t1 t2 #i .HappenBefore(BC,t1,t2)@i==>Ex x .t2=t1+x"
```

4 Model HTLC in Tamarin

We model the two roles Alice and Bob in the hash time lock contract. Alice is the contract initiator, Bob is the responder. Alice is not allowed to set up a hash time lock contract with herself. The roles send data to blockchain network by using fact Out(∗), i.e. they send data to the environment directly. It models the blockchain network as public, where the adversary learns anything sent to and received from the network.

4.1 HTLC rules

Alices' rules. Alice is defined by two rules: Alice_send and Alice_receive. The rule Alice_send broadcasts Alice's commitment transaction and redeem transaction to the blockchain network. The rule Alice_receive broadcasts the transaction to open Bob's commitment transaction. Note that even though Alice broadcasts her commitment transaction and its redeem transaction at the same time, the redeem transaction cannot be added to the blockchain until the time lock of Alice's commitment transaction expires.

Alice_send

The rule takes a simple transaction tx, Alice's secret keys, Bob's address and a fresh nonce as input. It outputs Alice's commitment transaction, Alice's redeem transaction, and a state Alice_1_record to record the hash pre-image. It spends the simple transaction tx with signature SigA and outputs a commitment transaction that has ⟨pk(ltkA1), timelock_A, hash, pkB3⟩ as output. The two potential ways to spend this commitment transaction are: 1) Redeem by Alice: when the time lock timelock_A timed out, Alice could redeem the commitment transaction by providing the signature of the public key pk(ltkA1). 2) Opened by Bob: Bob can take the funding by providing the pre-image of the hash lock and the signature of pkB3. The rule outputs the redeem transaction at the same time, since Alice desires to broadcast the redeem transaction earlier so that it can be added on the blockchain as soon as the time lock expires. The Tamarin code is listed below.

```
rule Alice_send:
   let
     timelock_A='1'+'1'
     hash=HTLChash(~hsk)
     SigA=sign(<'BC1',tx,pk(ltkA),<pk(ltkA1),timelock_A,hash,pkB3>>,ltkA)
     CommitTxAlice=TXhash(<tx,SigA,<pk(ltkA1),timelock_A,hash,pkB3>>)
     SigA1=sign(<'BC1',CommitTxAlice,<pk(ltkA1),timelock_A,hash,pkB3>,pkA2>,ltkA1)
   in
     [ !SimpleTx('BC1','0','0',pk(ltkA),tx,t) ,!PK(A,pk(ltkA1)),!PK(A,pkA2),!PK(B,pkB3),
     Fr(~hsk)]
  --[ InEq(A,B) ]->
     [ Out(<tx,SigA,<pk(ltkA1),timelock_A,hash,pkB3>>),Out(<CommitTxAlice,SigA1,pkA2>)
       ,Alice_1_record(hash,~hsk)]
```

Alice_receive

The rule takes a commitment transaction that has Alice as receiver, a state record Alice_1_record and Alice's address as inputs. It outputs a transaction spending the commitment transaction and a fact Reveal(hsk). The rule opens the pre-image of hash and provide the signature SigA3 for pkA3. This spending transaction will be added to the blockchain by the ledger rule Commit_open, it transfers the funding to the target address.

```
rule Alice_receive:
  let
    SigA3=sign(<'BC2',CommitTxBob,<pkB1,timelock_B,hash,pk(ltkA3)>,pkA4>,ltkA3)
  in
    [!CommitTx('BC2',tx0,SigB0,<pkB1,timelock_B,hash,pk(ltkA3)>,CommitTxBob,t)
    ,Alice_1_record(hash,hsk),!PK(A,pkA4)]
  --[ Alice_receive(CommitTxBob) ]->
    [ Out(<CommitTxBob,<hsk,SigA3>,pkA4,hsk>)]
```

Bob's rules. Bob is specified by the rules Bob_send, Bob_receive and a restriction Not_Spend. The rule Bob_send generates the commitment transaction and its redeem transaction, and broadcasts them to the blockchain network. The rule Bob_receive is to open Alice's commitment transaction and the restriction Not_Spend checks that Alice's commitment transaction has not been spent.

Bob_send

The rule takes Alice's commitment transaction, a simple transaction, Alice's receiving address, Bob's redeem address and Bob's secret key as input. The restriction Not_Spend checks if Alice's commitment transaction is just added to the blockchain. If it is, the rule will output Bob's commitment transaction, its redeem transaction and Bob_1_record to record the hash lock and the transaction id of Alice's commitment transaction. The output script in Bob's commitment transaction is $\langle \mathsf{pk(ltkB)}, \mathsf{timelock\_B}, \mathsf{hash}, \mathsf{pkA3} \rangle$. The hash is the same with the hash lock in Alice's commitment transaction.

```
rule Bob_send:
  let
    timelock_B='1'
    SigB=sign(<'BC2',tx,pk(ltkB),<pk(ltkB1),timelock_B,hash,pkA3>>,ltkB)
    CommitTxBob=TXhash(<tx,SigB,<pk(ltkB1),timelock_B,hash,pkA3>>)
    SigB1=sign(<'BC2',CommitTxBob,<pk(ltkB1),timelock_B,hash,pkA3>,pkB2>,ltkB1)
  in
    [!SimpleTx('BC2','0','0',pk(ltkB),tx,t1),!PK(B,pk(ltkB1)) ,!PK(B,pkB2),!PK(A,pkA3)
    ,!CommitTx('BC1',tx_0,SigA_0,<pkA,timelock_A,hash,pkB>,CommitTxAlice,t)]
  --[Not_Spend(CommitTxAlice)]->
    [ Bob_1_record(hash,CommitTxAlice)
     ,Out(<tx,SigB,<pk(ltkB1),timelock_B,hash,pkA3>>)
     ,Out(<CommitTxBob,SigB1,pkB2>)]
```

Bob_receive

The rule takes a state record Bob_1_record, the hash pre-image In(hsk), Bob's address and Alice's commitment transaction as inputs. It outputs the transaction to open Alice's commitment transaction. By providing the signature of the public key pk(ltkB3) and the pre-image of the hash lock, Bob transfers the funding to his address pkB4.

```
rule Bob_receive:
  let
    SigB3=sign(<'BC1',CommitTxAlice,<pkA1,timelock_A,hash,pk(ltkB3)>,pkB4>,ltkB3)
  in
    [ Bob_1_record(hash,CommitTxAlice),In(hsk),!PK(B,pkB4)
    ,!CommitTx('BC1',tx0,SigA0,<pkA1,timelock_A,hash,pk(ltkB3)>,CommitTxAlice,t)]
  --[ Bob_receive(CommitTxAlice) ]->
    [ Out(<CommitTxAlice,<hsk,SigB3>,pkB4,hsk>) ]
```

5 Tamarin Security analysis

5.1 Preliminaries

We describe a transaction as a tuple of six elements: $\mathsf{TX}\{\mathsf{BC}, \mathsf{InTx}, \mathsf{InSig}, \mathsf{Output}, \mathsf{n}, \mathsf{height}\}$, where BC is the blockchain which this transaction belongs to, InTx is the ID of the input transaction, InSig is the input signature, n is the id of this transaction, and height specifies which block contains this transaction. For a simple transaction, the parameter Output is simply a public key, while for a commitment transaction, the Output will be a tuple $\langle \mathsf{pk}_1, \mathsf{timelock}, \mathsf{hash}, \mathsf{pk}_2\rangle$ that contains two public keys pk_1 and pk_2, a time lock and a hash lock. The commitment transaction can be spent by revealing the hash pre-image and the signature of pk_2 or providing the signature of pk_1 if the time lock timed out. The parameter height is ignored if a transaction is not recorded on the blockchain yet.

For a specific time lock, we denotes its value as Δ. It restricts a commitment transaction can be spent only if there is at least Δ blocks appended after the block that contains this commitment transaction. We specify the corresponding real time duration of generating these Δ blocks as δ. The relationship is typically simple, for instance, in the Bitcoin blockchain the approximate time to generate 20 blocks is 200 minutes, so with timelock $\Delta = 20$ we get real time $\delta = 200$. The reason why the real time also involved in the formula is that we are dealing with two blockchains. Each of the two blockchains can be seen as a time reference, but these two time references might get out of sync, thus we need a single global clock.

When a commitment transaction is added on the blockchain, we denote the event as $\{\Gamma^{\mathsf{h_{sk}},\Delta_\mathsf{A}}_{\mathsf{Acom}}, \mathsf{t}, \mathsf{Tick}\}$, which means Alice's commitment transaction is recorded on blockchain at time point t in block sequence Tick, locked by Δ_A and a hash with pre-image $\mathsf{h_{sk}}$. The open and timeout of the commitment transaction are specified as $\{\Gamma^{\mathsf{h_{sk}},\Delta}_{\mathsf{Aopen}}, \mathsf{t}, \mathsf{Tick}\}$ and $\{\Gamma^{\mathsf{h_{sk}},\Delta}_{\mathsf{Ared}}, \mathsf{t}, \mathsf{Tick}\}$, respectively.

5.2 Security claim

For Alice, the hash time lock contract should satisfy the first two properties. For Bob, the protocol should guarantee the last two security properties:

Property 1. Bob cannot open Alice's commitment transaction and take her funding unless Bob has created a commitment transaction to Alice.

$$\forall\{\Gamma^{\mathsf{h_{sk}},\Delta}_{\mathsf{Aopen}}, \mathsf{t_{Aopen}}, \mathsf{Tick_{Aopen}}\} ==>\exists\{\Gamma^{\mathsf{h_{sk}},\Delta}_{\mathsf{Bcom}}, \mathsf{t_{Bcom}}, \mathsf{Tick_{Bcom}}\}$$

The equation claims that for all the events that Alice's commitment transactions have been opened, there must exist an event that Bob made a commitment transaction before. The commitment transaction made by Bob should use the same hash lock generated from $\mathsf{h_{sk}}$ and it sends funding to Alice's address.

```
lemma Security_1_Alice:
" All A tx1 SigA pkA1 timelock_A hash pkB3 CommitTxAlice
        TickAcom TickAopen #tAcom #tAopen #Apk1 .


     !PK(A,pkA1)@Apk1
     &!CommitTx('BC1',tx1,SigA,<pkA1,timelock_A,hash,pkB3>,CommitTxAlice,TickAcom)@tAcom
     &Spend('BC1',CommitTxAlice,'CommitOpen',TickAopen)@tAopen



  ==>Ex tx2 SigB pkB1 timelock_B pkA3 CommitTxBob
        TickBcom #tBcom #Apk2 .
        !PK(A,pkA3)@Apk2
        &!CommitTx('BC2',tx2,SigB,<pkB1,timelock_B,hash,pkA3>,CommitTxBob,TickBcom)@tBcom
        &#tBcom<#tAopen
"
```

Tamarin verifies the first security claim is true.

Property 2. Bob can redeem his funding only if the time lock of his commitment transaction timed out.

$$\forall(\{\Gamma^{h,\Delta_B}_{Bcom}, t_{Bcom}, Tick_{Bcom}\} \wedge \{\Gamma^{h,\Delta_A}_{Bred}, t_{Bred}, Tick_{Bred}\}) ==> t_{Bred} > t_{Bcom} + \delta_B$$

Since in a single blockchain, a transaction recorded early has a smaller height than those recorded later. We reduce this security property to:

$$\forall(\{\Gamma^{h,\Delta_B}_{Bcom}, t_{Bcom}, Tick_{Bcom}\} \wedge \{\Gamma^{h,\Delta_A}_{Bred}, t_{Bred}, Tick_{Bred}\}) ==> Tick_{Bred} > Tick_{Bcom} + \Delta_B$$

The equation claims that the duration between the time point Bob's commitment transaction is added to the blockchain and the time point Bob's redeem transaction is added to the blockchain is always larger than the duration of its time lock. Bob cannot redeem his commitment transaction before it timed out. This property guarantees that there is no race condition between Bob's redeem transaction and the transaction of Alice to open Bob's commitment transaction.

```
lemma Security_2_Alice:
" All tx2 SigB pkB1 timelock_B hash pkA3 CommitTxBob TickBcom #tBcom
        TickBTout #tBTout .

       !CommitTx('BC2',tx2,SigB,<pkB1,timelock_B,hash,pkA3>,CommitTxBob,TickBcom)@tBcom
       &Spend('BC2',CommitTxBob,'CommitTout',TickBTout)@tBTout
 ==>Ex x. TickBTout=TickBcom+timelock_B+x
"
```

Tamarin verifies the above security claim is true.

Property 3. After Alice takes Bob's funding, Bob has time to take Alice's funding before Alice's commitment transaction time out.

$$\forall(\{\Gamma^{h,\Delta_A}_{Acom}, t_{Acom}, Tick_{Acom}\} \wedge \{\Gamma^{h,\Delta_B}_{Bopen}, t_{Bopen}, Tick_{Bopen}\} ==> t_{Acom} + \delta_A > t_{Bopen}$$

The equation claims that if Alice takes Bob's funding at the last moment before it timed out, Bob should always have some time left before Alice's commitment transaction is timed out. This property avoids the risk of the race condition between Bob opening Alice's commitment transaction and Alice redeems her commitment transaction.

```
lemma Security_3_Bob:
"
All CommitTxAlice hash timelock_A pkA1 tx1 SigA pkB3 TickAcom #tAcom
      CommitTxBob timelock_B pkB1 tx2 SigB pkA3 TickBcom #tBcom
        #tBopen1 #tATout1 #tBopen #tATout .

    !CommitTx('BC1',tx1,SigA,<pkA1,timelock_A,hash,pkB3>,CommitTxAlice,TickAcom)@tAcom
    &!CommitTx('BC2',tx2,SigB,<pkB1,timelock_B,hash,pkA3>,CommitTxBob,TickBcom)@tBcom


    &Spend('BC2',CommitTxBob,'CommitOpen',TickBcom+timelock_B)@tBopen1
    &LedgerTick('BC2',TickBcom+timelock_B)@tBopen

    &LedgerTick('BC1',TickAcom+timelock_A+'1')@tATout
    &Spend('BC1',CommitTxAlice,'CommitTout',TickAcom+timelock_A+'1')@tATout1
==> #tBopen<#tATout
"
```

Tamarin gives a counterexample to this security claim, because the growth speed of the blockchains may differ. We explain in detail in the next subsection.

Property 4. Alice could redeem her funding only if her commitment transaction timed out.

$$\forall(\{\Gamma^{h,\Delta_A}_{Acom}, t_{Acom}, Tick_{Acom}\} \wedge \{\Gamma^{h,\Delta_A}_{Ared}, t_{Ared}, Tick_{Ared}\}) ==> t_{Ared} > t_{Acom} + \delta_A$$

The equation removes the same race condition risk that Alice has as described in security property 2.

```
lemma Security_4_Bob:
"
All tx1 SigA pkA1 timelock_A hash pkB3 CommitTxAlice TickAcom #tAcom #tATout
      TickATout .

    !CommitTx('BC1',tx1,SigA,<pkA1,timelock_A,hash,pkB3>,CommitTxAlice,TickAcom)@tAcom
    &Spend('BC1',CommitTxAlice,'CommitTout',TickATout)@tATout
==>Ex x. TickATout=TickAcom+timelock_A+x
"
```

Tamarin verifies this security claim is true.

5.3 Discussion on property 3

The failure of property 3 claims that after Alice taking Bob's funding, there exists a case that Bob has no time to open Alice commitment transaction before it expires. Tamarin shows that this attack happens in the case that the blockchain on which Alice made a commit transaction grows faster than is expected. Thus Alice's commitment transaction expires earlier even before Bob's commitment transaction expires. Therefore Alice has the chance to redeem her funding and also take Bob's funding.

Therefore, we need to have a blockchain that not only has liveness and consistency but also keeps a stable growth speed for the block height. Based on the Tamarin result, we add an extra restriction to restrict the growing speed of the blockchain "BC2" to be at least as fast as "BC1", and then evaluate the security property again.

```
restriction stable_growing_blockchain:
"All height #i .Tick('BC1',height)@i==>Ex #j.Tick('BC2',height)@j"
```

Tamarin now proves that property 3 holds. Notice that in the real scenario we expected both two blockchains should have stable growing speed, but this condition is not necessary for HTLC. The result shows that as long as "BC" grows relatively no slower than "BC1", HTLC is secure. The reason is Alice holds the pre-image of the hash, she only needs to observe the height of "BC2" to take Bob's funding before its timelock expires, she doesn't need to worry Bob will take her funding since he doesn't know the hash pre-image. While for Bob, if he publishes his commitment transaction, he needs to make sure Alice cannot withdraw her funding earlier than Alice taking his funding.

6 Analysis of the old version of HTLC

The old version of the hash time lock contract was used when the time lock functionality could only constrain the time point that a certain transaction is allowed to be added to blockchain. In this case, the time lock is specified in the redeem transaction rather than the commitment transaction. To make an agreement for the time lock duration of the redeem transaction, the two players need to exchange their signatures on the redeem transaction. The multi-signatures are checked by the nodes before they add the transaction into a block. The signature exchanging procedure is done before the players publish their commitment transaction, otherwise, they might be unable to redeem their commitment transactions.

We claim the same four security properties from section 5 for the old version hash time lock contract. Tamarin verifies that the protocol satisfies the security claims given that Alice is allowed to only use a fixed duration of timelock in the contract. However, in reality, Alice might use the timelock with different durations. In this case, there is an attack that allows Alice to redeem her funding earlier than the time period that Bob has signed.

The attack is as follows: Alice will initiate two hash time lock contracts with Bob, these two hash time lock contracts are the same except the second one has longer time lock than the first one. She aborts the first one when she gets Bob's signature on her redeem transaction. Bob will also abort the contract since Alice doesn't publish her commitment transaction. Alice initiates the second contract with Bob, using the same hash lock, but longer time lock. (Bob could in principle notice that he has signed the same hash before, but this requires Bob to keep track of earlier contracts, which is impractical.) In this scenario, after both players publish their commitment transactions to blockchains, Alice can use the redeem transaction of the fist hash time lock contract to unlock her commitment transaction in the second hash time lock contract. Because the second redeem transaction has a shorter time lock, she can redeem the commitment transaction earlier than Bob's expectation.

When we enable different timelocks in our Tamarin model, Tamarin finds the attack and shows that security property 3 fails even with the synchronization between the two blockchain growth speeds.

7 Conclusion

In this paper, we give a formal model for blockchain in Tamarin. Using this model we give a formal verification for security of the hash time lock contract. The verification result from Tamarin shows that the security of HTLC is based on the security assumptions of the underlying blockchain, but also requires that the responder blockchain (the blockchain that Bob operates on) needs to grow at least as fast as the initiator's blockchain. This result demonstrates that our Tamarin blockchain model can be used to find security issues in blockchain-based protocols. We note that the verification process of Tamarin needs human guidance to some extent, which could be improved in future work. Also, the model can be improved to allow forks to be more comprehensive.

References

1 Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Łukasz Mazurek. Fair two-party computations via bitcoin deposits. In Rainer Böhme, Michael Brenner, Tyler Moore, and Matthew Smith, editors, *Financial Cryptography and Data Security*, pages 105–121, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

2 Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Łukasz Mazurek. Modeling bitcoin contracts by timed automata. In Axel Legay and Marius Bozga, editors, *Formal Modeling and Analysis of Timed Systems*, pages 7–22, Cham, 2014. Springer International Publishing.

3 Iddo Bentov, Yan Ji, Fan Zhang, Lorenz Breidenbach, Philip Daian, and Ari Juels. Tesseract: Real-time cryptocurrency exchange using trusted hardware. In Lorenzo Cavallaro et al., editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019*, pages 1521–1538. ACM, 2019. doi:10.1145/3319535.3363221.

4 Sergiu Bursuc and Steve Kremer. Contingent payments on a public ledger: models and reductions for automated verification. In *ESORICS 2019 - The 24th European Symposium on Research in Computer Security*, Luxembourg, Luxembourg, September 2019. URL: https://hal.archives-ouvertes.fr/hal-02269063.

5 Christian Decker and Roger Wattenhofer. A fast and scalable payment network with bitcoin duplex micropayment channels. In Andrzej Pelc and Alexander A. Schwarzmann, editors, *Stabilization, Safety, and Security of Distributed Systems*, pages 3–18, Cham, 2015. Springer International Publishing.

6 Thaddeus Dryja Joseph Poon. The Bitcoin Lightning Network: Scalable off-chain instant payments, 2016. URL: `https://lightning.network/lightning-network-paper.pdf`.

7 Gregory Maxwell. Zero knowledge contingent payments, 2011. URL: `https://en.bitcoin.it/wiki/ZeroKnowledgeContingentPayment`.

8 Simon Meier, Benedikt Schmidt, Cas Cremers, and David Basin. The tamarin prover for the symbolic analysis of security protocols. In *Proceedings of the 25th International Conference on Computer Aided Verification - Volume 8044*, CAV 2013, page 696–701, Berlin, Heidelberg, 2013. Springer-Verlag.

9 Tianyu Sun and Wensheng Yu. A formal verification framework for security issues of blockchain smart contracts. *Electronics*, 9:255, February 2020. `doi:10.3390/electronics9020255`.

10 Mathieu Turuani, Thomas Voegtlin, and Michael Rusinowitch. Automated verification of electrum wallet. In Jeremy Clark, Sarah Meiklejohn, Peter Y.A. Ryan, Dan Wallach, Michael Brenner, and Kurt Rohloff, editors, *Financial Cryptography and Data Security*, pages 27–42, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.