```
// PMPCS-CT

// Authors: Catalin V. Birjoveanu
// Date:    August 2025

theory PMPCS_CT
begin

builtins: hashing, symmetric-encryption, signing

// Registering a public key
rule Register_pk:
    [ Fr(~ltk) ]
  -->
    [ !Ltk($A, ~ltk), !Pk($A, pk(~ltk)), Out(pk(~ltk)) ]

// Generating a shared symmetric key
rule Init_Shared_Keys:
  [ Fr(~k) ]
  -->
  [ !SharedKey($A,$B,~k) ]

// Modeling dishonest behavior of an agent
rule Init_Finalize:
  [ !Pk(A, pkA) ]
  --[Dish(A)]->
  [ Dishonest(A) ]

// Establishing a contract
rule Contracts_Setting:
  [ Fr(~c) ]
  --[ Neq($A,$B) ]->
  [ !Contract($A,$B,~c) ]

// Merkle tree generation
rule Customer_Constructs_MerkleTree:

    let
        H1 = h(<C, P1, 'Id1', 'D', h(c1)>)
        H2 = h(<C, P2, 'Id2', 'D', h(c2)>)
        Root = h(<H1, H2>)
    in
  [ !Contract(C,P1,c1)
  , !Contract(C,P2,c2)
  ]
  --[ Neq(P1,P2) ]->
  [
    !Merkle(C,P1,P2,Root,H1,H2,c1,c2)
    , SCSendFinalize(C,'D',Root,'0')
    , Deadlinenotexpired('D')
  ]

// Signing.1.1  C sends commitment to P1
rule Customer_11:
    let
        m = <'t11',Root,H2,'Id1'>
        c = senc(<m,sign(m,ltkC)>, k1)
    in
    [ !Contract(C,P1,c1)
    , !Merkle(C,P1,P2,Root,H1,H2,c1,c2),!SharedKey(C,P1,k1)
    , !Ltk(C, ltkC)
    ]
  --[ SendCommit1(C,P1,Root,H2,'Id1',k1) ]->
    [ CustomerSendCommit_1(C,P1,Root,H2,'Id1',k1 )
    , Out(c) ]

// Signing.1.2  After C sends commitment to P1, C sends commitment to P2
rule Customer_12:
    let
```

```
        m = <'t12',Root,H1,'Id2'>
        c = senc(<m,sign(m,ltkC)>, k2)
    in
    [ CustomerSendCommit_1(C,P1,Root,H2,'Id1',k1 )
    , !Contract(C,P1,c1)
    , !Merkle(C,P1,P2,Root,H1,H2,c1,c2),!SharedKey(C,P2,k2)
    , !Ltk(C, ltkC)
    ]
  --[ SendCommit2(C,P2,Root,H1,'Id2',k2) ]->
    [ CustomerSendCommit_2(C,P2,Root,H1,'Id2',k1,k2 )
    , Out(c) ]

// Signing.2.1  P1 receives commitment (message 1) from C and sends the confirmation (message 2)
// to C
rule Provider_12:
    let
        m = <'t11',Root,H2,'Id1'>
        m1 = <'t21',Root,H2,'Id1'>
        H1 = h(<C, P1, 'Id1', 'D', h(c1)>)
        Root1 = h(<H1, H2>)
        c = senc(<m1,sign(m1,ltkP1)>,k1)
    in
    [ !Contract(C,P1,c1)
    , !Merkle(C,P1,P2,Root,H1,H2,c1,c2)
    , !SharedKey(C,P1,k1), !Pk(C, pkC), !Ltk(P1, ltkP1)
    , In(senc(<m,sig>, k1))
    ]
    --[ Eq(verify(sig,m,pkC),true), Eq(Root1,Root)
    , AuthenticCommit1(P1,C,Root,H2,'Id1'), SendConf1(P1,C,Root,H2,'Id1',k1) ]->
    [ ProviderSendConf_1(P1,C,Root,H2,'Id1',c1,k1)
    , Out(c) ]

// Signing.2.2  P2 receives commitment (message 1) from C and sends the confirmation (message 2)
// to C
rule Provider_22:
    let
        m = <'t12',Root,H1,'Id2'>
        m1 = <'t22',Root,H1,'Id2'>
        H2 = h(<C, P2, 'Id2', 'D', h(c2)>)
        Root1 = h(<H1, H2>)
        c = senc(<m1,sign(m1,ltkP2)>,k2)
    in
    [ !Merkle(C,P1,P2,Root,H1,H2,c1,c2)
    , !Contract(C,P2,c2)
    , !SharedKey(C,P2,k2)
    , !Pk(C, pkC)
    , !Ltk(P2, ltkP2)
    , In(senc(<m,sig>, k2))
    ]
    --[ Eq(verify(sig,m,pkC),true), Eq(Root1,Root)
      , AuthenticCommit2(P2,C,Root,H1,'Id2'), SendConf2(P2,C,Root,H1,'Id2',k2) ]->
    [  ProviderSendConf_2(P2,C,Root,H1,'Id2',c2,k2), Out(c) ]

// Signing.2.1  C receives confirmation from P1
rule Customer_21:
    let
        m = <'t21',Root,H2,'Id1'>
    in
    [ CustomerSendCommit_2(C,P2,Root,H1,'Id2',k1,k2 )
    , !Merkle(C,P1,P2,Root,H1,H2,c1,c2)
    , !Contract(C,P1,c1)
    , !SharedKey(C,P1,k1)
    , !Pk(P1, pkP1)
    , In(senc(<m,sig>,k1))
    ]
  --[ Eq(verify(sig,m,pkP1),true), ReceiveConf1(C,P1,Root,H2,'Id1',k1) ]->
    [ CustomerReceiveConf_1(C,P1,Root,H2,'Id1',c1,k1) ]

// Signing.2.2  After C receives confirmation from P1, C receives confirmation from P2
```

```
rule Customer_22:
    let
        m = <'t22',Root,H1,'Id2'>
    in
    [ CustomerReceiveConf_1(C,P1,Root,H2,'Id1',c1,k1 )
    , !Merkle(C,P1,P2,Root,H1,H2,c1,c2)
    , !Contract(C,P2,c2)
    , !SharedKey(C,P2,k2)
    , !Pk(P2, pkP2)
    , In(senc(<m,sig>, k2))
    ]
  --[ Eq(verify(sig,m,pkP2),true), ReceiveConf2(C,P2,Root,H1,'Id2',k2) ]->
    [ CustomerReceiveConf_2( C,P2,Root,H1,'Id2',c2,k1,k2) ]

// Signing.3.1  C receives confirmations from P1 and P2 and sends its signature (message 3) to P1
rule Customer_31:
    let
        m = <'t31',Root,c1>
        c = senc(<m,sign(m,ltkC)>,k1)
    in
    [ CustomerReceiveConf_2(C,P2,Root,H1,'Id2',c2,k1, k2)
    , !Merkle(C,P1,P2,Root,H1,H2,c1,c2)
    , !Contract(C,P1,c1)
    , !Ltk(C, ltkC)
    ]
  --[ SendSig31(C,P1,Root,c1,k1) ]->
    [ CustomerSendSig_31(C,P1,Root,c1,k1,k2)
    , Out(c) ]

// Signing.3.2  C sends his signature to P2
rule Customer_32:
    let
        m2 = <'t32',Root,c2>
        c = senc(<m2,sign(m2,ltkC)>,k2)
    in
    [ CustomerSendSig_31(C,P1,Root,c1,k1,k2)
    , !Merkle(C,P1,P2,Root,H1,H2,c1,c2)
    , !Contract(C,P2,c2)
    , !Ltk(C, ltkC)
    ]
  --[ SendSig32(C,P2,Root,c2,k2) ]->
    [ CustomerSendSig_32(C,P2,Root,c2)
    , Out(c) ]

// Signing.4.1  P1 receive C's signature (message 3) from C and sends his signature (message 4)
// to C
rule Provider_14:
    let
        m = <'t31',Root,c1>
        m1 = <'t41',Root,c1>
        c = senc(<m1,sign(m1,ltkP1)>,k1)
    in
    [ ProviderSendConf_1(P1,C,Root,H2,'Id1',c1,k1)
    , !Merkle(C,P1,P2,Root,H1,H2,c1,c2)
    , !Contract(C,P1,c1)
    , !Pk(C, pkC)
    , !Ltk(P1, ltkP1)
    , In( senc(<m,sig>, k1) )
    ]
    --[ Eq(verify(sig,m,pkC),true), Authentic31(P1,C,Root,c1,k1), SendSig41(P1,C,c1,Root,k1) ]->
    [ Out(c) ]


// Signing.4.1  P2 receive C's signature (message 3) from C and sends his signature (message 4)
// to C
rule Provider_24:
    let
        m = <'t32',Root,c2>
        m2 = <'t42',Root,c2>
```

```
        c = senc(<m2,sign(m2,ltkP2)>,k2)
    in
    [ ProviderSendConf_2(P2,C,Root,H1,'Id2',c2,k2)
    , !Merkle(C,P1,P2,Root,H1,H2,c1,c2)
    , !Contract(C,P2,c2)
    , !Pk(C, pkC)
    , !Ltk(P2, ltkP2)
    , In( senc(<m,sig>, k2) )
    ]
    --[ Eq(verify(sig,m,pkC),true), Authentic32(P2,C,Root,c2,k2), SendSig42(P2,C,c2,k2,Root) ]->
    [ Out(c) ]

// Signing.4.1  C receives P1's signature (message 4) from P1
rule Customer_41:
    let
        m = <'t41',Root,c1>
    in
    [ CustomerSendSig_32(C,P2,Root,c2)
    , !Merkle(C,P1,P2,Root,H1,H2,c1,c2)
    , !Contract(C,P1,c1)
    , !SharedKey(C,P1,k1)
    , !Pk(P1, pkP1)
    , In(senc(<m,sig>, k1))
    ]
  --[ Eq(verify(sig,m,pkP1),true), RecSig41(C,P1,c1,Root), Authentic41(C,P1,c1,Root,k1) ]->
    [ CustomerRecSig41(C,P1,c1,Root)]

// Signing.4.1  C receives P2's signature (message 4) from P2
rule Customer_42:
    let
        m = <'t42',Root,c2>
    in
    [ CustomerRecSig41(C,P1,c1,Root)
    , !Merkle(C,P1,P2,Root,H1,H2,c1,c2)
    , !Contract(C,P2,c2)
    , !SharedKey(C,P2,k2)
    , !Pk(P2, pkP2)
    , In(senc(<m,sig>, k2))
    ]
  --[ Eq(verify(sig,m,pkP2),true), RecSig42(C,P2,c2,Root), Authentic42(C,P2,c2,Root,k2) ]->
    [ CustomerRecSig42(C,P2,c2,Root) ]

// ---------- begin of Finalization sub-protocol modeling ----------------------
// Finalization: C sends its contract signatures to P1 and P2 in message 3, but does not receive
// the corresponding contract signatures from P2; P2 behaves dishonestly by sending a cancel
// signature to C
rule Dishonest_P2:
    let
        m = <'t32',Root,c2>
        m2 = <'t42c',c2,'cancel',Root>
        c = senc(<m2,sign(m2,ltkP2)>,k2)
    in
    [ ProviderSendConf_2(P2,C,Root,H1,'Id2',c2,k2)
    , Dishonest(P2)
    , !Merkle(C,P1,P2,Root,H1,H2,c1,c2)
    , !Contract(C,P2,c2)
    , !Pk(C, pkC)
    , !Ltk(P2, ltkP2)
    , In(senc(<m,sig>, k2))
    ]
    --[ Eq(verify(sig,m,pkC),true), Authentic32(P2,C,Root,c2,k2), SendCancelSig(P2,C,c2,k2,Root)
]->
    [ ProviderSendCancelSig_2(P2,C,c2,k2,Root),Out(c) ]

// Finalization.1: C sends the signature (message 3) to P2, but instead of receiving the
// signature (message 4) from P2, it receives a 'cancel' message, which triggers the
// Finalization sub-protocol;
// C calls the finalizeMPCS function of SC
rule Customer_Finalization_1:
```

```
    let
        m = <'t42c',c2,'cancel',Root>
        m1= <'t1f','D',Root>
        c = senc(<m1,sign(m1,ltkC)>,k)
    in
    [ CustomerRecSig41(C,P1,c1,Root)
    , !Merkle(C,P1,P2,Root,H1,H2,c1,c2)
    , Deadlinenotexpired('D')
    , !Contract(C,P2,c2)
    , !SharedKey(C,P2,k2)
    , !SharedKey(C,SC,k)
    , !Ltk(C, ltkC)
    , !Pk(P2, pkP2)
    , In(senc(<m,sig>,k2))
    ]
  --[ Eq(verify(sig,m,pkP2),true), CustomerReceiveCancel(C,P2,Root,c2)
    , SendFinalizeMPCS(C,SC,P1,P2,'D',Root,c1,c2) ]->
    [ CustomerSendFinalize(C,SC,P1,P2,'D',Root,c1,c2)
    , Out(c)
    ]

// Finalization.2  SC applies the finalizeMPCS function and sends 'Finalized'
// ( message senc(<'t2f','Finalized','t1f','D',Root,sig>,k) ) to C
rule SmartContract_finalizeMPCS:
    let
        m = <'t1f','D',Root>
        m1 =<'t2f','Finalized',m>
        c = senc(<m1,sign(m1,ltkSC)>,k)
    in
    [ SCSendFinalize(C,'D',Root,'0'),
      !Merkle(C,P1,P2,Root,H1,H2,c1,c2)
    , !SharedKey(C,SC,k)
    , !Pk(C, pkC)
    , !Ltk(SC, ltkSC)
    , In(senc(<m,sig>,k))
    ]
  --[ Eq(verify(sig,m,pkC),true), AuthenticFinalizeMPCS(SC,C,P1,P2,'D',Root,c1,c2)
    , SCSendFinalizedMPCS(SC,C,'D',Root) ]->
    [ SCSendFinalize(C,'D',Root,'1')
    , Out(c)
    ]

//Finalization.2 C receives 'Finalized' from SC
rule Customer_Finalization_2:
     let
        m = <'t1f','D',Root>
        m1 =<'t2f','Finalized',m>
    in
    [ CustomerSendFinalize(C,SC,P1,P2,'D',Root,c1,c2)
    , !Merkle(C,P1,P2,Root,H1,H2,c1,c2)
    , !SharedKey(C,SC,k)
    , !Pk(SC, pkSC)
    , In(senc(<m1,sig>, k))
    ]
  --[ Eq(verify(sig,m1,pkSC),true), CustomerRecFinalizedMPCS(C,SC,P1,P2,'D',Root,c1,c2) ]->
    [ CustomerRecFinalized(C,SC,'D',Root)
    ]
// ---------- end of Finalization sub-protocol modeling -----------------------

// ---------- begin of Status Retrieval sub-protocol modeling -----------------
// Status Retrieval: C obtains the contract signing confirmations from P1 and P2, but C behaves
// dishonestly executing the Finalization sub-protocol without having sent its signature on
// contract c2 to P2
rule Dishonest_C:
    let
        m= <'t1fsr','D',Root>
        c = senc(<m,sign(m,ltkC)>,k)
    in
    [ CustomerSendSig_31(C,P1,Root,c1,k1,k2)
```

```
    , Dishonest(C)
    , !Merkle(C,P1,P2,Root,H1,H2,c1,c2)
    , Deadlinenotexpired('D')
    , !SharedKey(C,SC,k)
    , !Ltk(C, ltkC)
    ]
  --[ CSendFinalizeMPCS(C,SC,P1,P2,'D',Root,c1,c2) ]->
    [ CSendFinalize(C,SC,P1,P2,'D',Root,c1,c2)
    , Out(c) ]

// Finalization.22  SC applies the finalizeMPCS function and sends 'Finalized' to C
rule SmartContract_finalizeMPCS2:
    let
        m = <'t1fsr','D',Root>
        m1 =<'t2fsr','Finalized',m>
        c = senc(<m1,sign(m1,ltkSC)>,k)
    in
    [ SCSendFinalize(C,'D',Root,'0')
    , !Merkle(C,P1,P2,Root,H1,H2,c1,c2)
    , !SharedKey(C,SC,k)
    , !Pk(C, pkC)
    , !Ltk(SC, ltkSC)
    , In(senc(<m,sig>,k))
    ]
  --[ Eq(verify(sig,m,pkC),true), AuthFinalizeMPCS(SC,C,P1,P2,'D',Root,c1,c2)
    , SCSendFinMPCS(SC,C,'D',Root) ]->
    [ SCSendFinalize(C,'D',Root,'2')
    , Out(c)
    ]

//Finalization.22 C receives 'Finalized' from SC
rule Customer_Finalization_22:
     let
        m = <'t1fsr','D',Root>
        m1 =<'t2fsr','Finalized',m>
    in
    [ CSendFinalize(C,SC,P1,P2,'D',Root,c1,c2)
    , !Merkle(C,P1,P2,Root,H1,H2,c1,c2)
    , !SharedKey(C,SC,k)
    , !Pk(SC, pkSC)
    , In(senc(<m1,sig>, k))
    ]
  --[ Eq(verify(sig,m1,pkSC),true), CRecFinalizedMPCS(C,SC,P1,P2,'D',Root,c1,c2) ]->
    [ CRecFinalized(C,SC,'D',Root)
    ]

// Status Retrieval.1  P2 calls the retrieveStatusMPCS() function requesting
// the status and evidence of the corresponding multi-party contract
rule Provider2_Status_Retrieval_1:
    let
        m1 = <'t1s','RetrieveStatus',Root>
        c = senc(<m1,sign(m1,ltkP2)>,k)
    in
    [ ProviderSendConf_2(P2,C,Root,H1,'Id2',c2,k2)
    , CRecFinalized(C,SC,'D',Root)
    , !Merkle(C,P1,P2,Root,H1,H2,c1,c2)
    , !Contract(C,P1,c1)
    , !Contract(C,P2,c2)
    , !SharedKey(P2,SC,k)
    , !Ltk(P2, ltkP2)
    ]
  --[ SendRetrieveStatusMPCS(P2,SC,Root,c2) ]->
    [ ProviderSendRetrieve(P2,SC,Root,c2)
    , Out(c)
    ]

// Status Retrieval.2 SC applies the retrieveStatusMPCS() function and sends the evidence
// senc(<'t2s','Finalized','D',Root,sig>,k) to P2
rule SmartContract_retrieveStatusMPCS:
```

```
    let
        m1 = <'t1s','RetrieveStatus',Root>
        m2 = <'t2s','Finalized','D',Root>
        c = senc(<m2,sign(m2,ltkSC)>,k)
    in
    [ SCSendFinalize(C,'D',Root,'2')
    , !Merkle(C,P1,P2,Root,H1,H2,c1,c2)
    , !SharedKey(C,SC,k)
    , !Pk(P2, pkP2)
    , !Ltk(SC, ltkSC)
    , In(senc(<m1,sig>,k))
    ]
  --[ Eq(verify(sig,m1,pkP2),true), AuthenticStatusretrievalMPCS(SC,P2,'D',Root)
    , SCSendStatusMPCS(SC,C,P1,P2,'D',Root,c1,c2) ]->
    [ SCSendStatus(SC,P2,C,Root,c2)
    , Out(c)
    ]

//Status Retrieval.2 P2 receives the 'Finalized' status of the contract
 rule Provider2_RetrieveStatusMPCS:
    let

        m =<'t2s','Finalized','D',Root>
    in
    [ ProviderSendRetrieve(P2,SC,Root,c2)
    , !Merkle(C,P1,P2,Root,H1,H2,c1,c2)
    , !SharedKey(C,SC,k)
    , !Pk(SC, pkSC)
    , In(senc(<m,sig>,k))
    ]
  --[ Eq(verify(sig,m,pkSC),true), ProviderRetrieveStatusFinalizedMPCS(P2,SC,Root)]->
    [ ProviderRecStatus(P2,SC,C,Root)
    ]
// ---------- end of Status Retrieval sub-protocol modeling -----------------

// Inequality restriction
restriction Inequality:
 " All x #i. Neq(x,x) @ #i ==> F "

// Equality restriction
restriction Equality:
  " All x y #i. Eq(x,y) @i ==> x = y "

// ---------- Security properties specification -----------------
// Model executability for Signing sub-protocol:
// There is a complete session of Signing sub-protocol in which C, P1, P2 behave honestly;
// C receives signatures from P1 and P2 in Signing sub-protocol
lemma Customer_Providers_honest_session:
  exists-trace
  " Ex C P1 P2 Root c1 c2 #i #j.
        RecSig41(C,P1,c1,Root) @ #i & RecSig42(C,P2,c2,Root) @ #j
  "

// Confidentiality of the content of contracts in Signing in which C, P1, P2 behave honestly:
// If C receives signatures on both contracts c1 and c2 from P1 and P2 (from the protocol
// execution mode this means that P1 and P2 also received the corresponding signatures on
// the contracts from C), then the attacker does not know the contents of either c1 or c2
lemma Contracts_Confidentiality_Signing:
  " ( All C P1 c1 P2 c2 Root #i #j.
      RecSig41(C,P1,c1,Root) @ #i
      & RecSig42(C,P2,c2,Root) @ #j
    ==> ( not (Ex #k. K(c1) @ #k) & not (Ex #l. K(c2) @ #l) )
    )
  "

// P1 authenticates C on commit: If P1 receives and verifies the commitment (message 1) as
// coming from C, then C previously sent the commitment.
lemma P1_authenticates_C_on_commit:
  " ( All P1 C Root H2 #i. AuthenticCommit1(P1,C,Root,H2,'Id1') @ #i
```

```
    ==>
     ( Ex k1 #j. SendCommit1(C,P1,Root,H2,'Id1',k1) @ #j & j < i )
    )
  "

// P2 authenticates C on commit: If P2 receives and verifies the commitment (message 1) as
// coming from C, then C previously sent the commitment.
lemma P2_authenticates_C_on_commit:
  " ( All P2 C Root H1 #i. AuthenticCommit2(P2,C,Root,H1,'Id2') @ #i
    ==>
     ( Ex k2 #j. SendCommit2(C,P2,Root,H1,'Id2',k2) @ #j & j < i )
    )
  "

// C authenticates P1 on confirmation
lemma C_authenticates_P1_on_conf:
  " ( All P1 C Root H2 k1 #i. ReceiveConf1(C,P1,Root,H2,'Id1',k1) @ #i
    ==>
     ( Ex #j. SendConf1(P1,C,Root,H2,'Id1',k1) @ #j & j < i )
    )
  "

// C authenticates P2 on confirmation
lemma C_authenticates_P2_on_conf:
  " ( All P2 C Root H1 k2 #i. ReceiveConf2(C,P2,Root,H1,'Id2',k2) @ #i
    ==>
     ( Ex #j. SendConf2(P2,C,Root,H1,'Id2',k2) @ #j & j < i )
    )
  "

// P1 authenticates C on signature (message 3)
lemma P1_authenticates_C_on_sig:
  " ( All P1 C Root c1 k1 #i. Authentic31(P1,C,Root,c1,k1) @ #i
    ==>
     ( Ex #j. SendSig31(C,P1,Root,c1,k1) @ #j & j < i )
    )
  "

// P2 authenticates C on signature (message 3)
lemma P2_authenticates_C_on_sig:
  " ( All C P2 Root c2 k2 #i. Authentic32(P2,C,Root,c2,k2) @ #i
    ==>
     ( Ex #j. SendSig32(C,P2,Root,c2,k2) @ #j & j < i )
    )
  "

// C authenticates P1 on signature (message 4)
lemma C_authenticates_P1_on_sig:
  " ( All P1 C Root c1 k1 #i. Authentic41(C,P1,c1,Root,k1) @ #i
    ==>
     ( Ex #j. SendSig41(P1,C,c1,Root,k1) @ #j & j < i )
    )
  "

// C authenticates P2 on the signature (message 4)
lemma C_authenticates_P2_on_sig:
  " ( All P2 C Root c2 k2 #i. Authentic42(C,P2,c2,Root,k2) @ #i
    ==>
     ( Ex #j. SendSig42(P2,C,c2,k2,Root) @ #j & j < i )
    )
  "

// C authenticates P1: whenever C (acting as customer) completes a run of the Signing
// sub-protocol, apparently with provider P1, then the provider P1 has previously been
// running the protocol, apparently with C, and the two entities agreed on
// contract c1 and Root
lemma C_authenticates_P1:
  " ( All C P1 Root c1 k1 #i.
      Authentic41(C,P1,c1,Root,k1) @ #i
```

```
    ==>
     ( Ex #j. Authentic31(P1,C,Root,c1,k1) @ #j & j < i )
    )
  "

// P1 authenticates C: whenever P1 (acting as provider) completes a run of the Signing
// sub-protocol, apparently with customer C, then the customer C has previously been
// running the protocol, apparently with P1, and the two entities agreed on
// contract c1 and Root
lemma P1_authenticates_C:
  " ( All C P1 Root c1 k1 #i.
      Authentic31(P1,C,Root,c1,k1) @ #i
    ==>
     ( Ex #j. SendSig31(C,P1,Root,c1,k1) @ #j & j < i )
    )
  "

// C authenticates P2: whenever C completes a run of the Signing sub-protocol, apparently with
// P2, then P2 has previously been running the protocol, apparently with C, and the two entities
// agreed on contract c2 and Root
lemma C_authenticates_P2:
  " ( All C P2 Root c2 k2 #i.
      Authentic42(C,P2,c2,Root,k2) @ #i
    ==>
     ( Ex #j. Authentic32(P2,C,Root,c2,k2) @ #j & j < i )
    )
  "
// P2 authenticates C: whenever P2 completes a run of the Signing sub-protocol, apparently with
// C, then C has previously been running the protocol, apparently with P2, and the two entities
// agreed on contract c2 and Root
lemma P2_authenticates_C:
  " ( All C P2 Root c2 k2 #i.
      Authentic32(P2,C,Root,c2,k2) @ #i
    ==>
     ( Ex #j. SendSig32(C,P2,Root,c2,k2) @ #j & j < i )
    )
  "

// Fairness in Signing under the conditions of honest behavior of entities C, P1 and P2:
// In any session of the protocol in which C receives both authentic signatures on individual
// contracts from P1 and P2 (represented by action facts Authentic41 and Authentic42),
// then both P1 and P2 have previously received authentic signatures from C on their
// corresponding contracts (represented by action facts Authentic31 and Authentic32)
lemma FairnessSigning:
  " ( All C P1 P2 Root c1 c2 k1 k2 #i #j.
      Authentic41(C,P1,c1,Root,k1) @ #i
      & Authentic42(C,P2,c2,Root,k2) @ #j
    ==>
     ( Ex #k #l. Authentic31(P1,C,Root,c1,k1) @ #k & k < i & Authentic32(P2,C,Root,c2,k2) @ #l &
l < j )
    )
  "

// Model executability for Signing and Finalization sub-protocols:
// There is a session of Signing sub-protocol in which C and P1 behave honestly, and
// P2 behaves dishonestly by sending a cancel signature to C;
// Then, a complete session of Finalization sub-protocol is executed in which C receives
// the signature from SC
lemma Customer_Finalize:
  exists-trace
  " Ex C SC P1 c1 P2 c2 Root #i #j.
      RecSig41(C,P1,c1,Root) @ #i
      & CustomerRecFinalizedMPCS(C,SC,P1,P2,'D',Root,c1,c2) @ #j
  "

// Confidentiality of the content of contracts in Signing and Finalization sub-protocols
// when P2 behaves dishonestly
lemma Contracts_Confidentiality_SignFin:
  " ( All C SC P1 c1 P2 c2 Root #i #j.
```

```
      RecSig41(C,P1,c1,Root) @ #i
      & CustomerRecFinalizedMPCS(C,SC,P1,P2,'D',Root,c1,c2) @ #j
    ==> ( not (Ex #k. K(c1) @ #k) & not (Ex #l. K(c2) @ #l) )
    )
  "

// Dishonest behaviour of P2: C authenticates P2 on cancel signature
lemma C_authenticates_P2_on_cancel:
  " ( All P2 C Root c2 #i. CustomerReceiveCancel(C,P2,Root,c2) @ #i
    ==>
     ( Ex k2 #j. SendCancelSig(P2,C,c2,k2,Root) @ #j & j < i )
    )
  "

// If C is not dishonest (is honest) then SC authenticates C on message 1 from Finalization
// sub-protocol
lemma SC_authenticates_C_on_finalization:
  " not (Ex C #j. Dish(C) @ #j )
  ==>
  ( All C SC P1 P2 Root c1 c2 #i. AuthenticFinalizeMPCS(SC,C,P1,P2,'D',Root,c1,c2) @ #i
    ==>
     ( Ex #k. SendFinalizeMPCS(C,SC,P1,P2,'D',Root,c1,c2) @ #k & k < i )
    )
  "

// C authenticates SC on message 2 from Finalization sub-protocol
lemma C_authenticates_SC_on_rec:
  " ( All C SC P1 P2 Root c1 c2 #i. CustomerRecFinalizedMPCS(C,SC,P1,P2,'D',Root,c1,c2) @ #i
    ==>
     ( Ex #j. SCSendFinalizedMPCS(SC,C,'D',Root)  @ #j & j < i )
    )
  "

// C authenticates P2 in Finalization sub-protocol case (P2 behaves dishoneslty):
// whenever C completes a run of the Signing and Finalization sub-protocols, apparently with P2,
// then P2 has previously been running the protocol, apparently with C, and the two entities
// agreed on Root and Id2
lemma C_authenticates_P2_Finalization:
  " ( All C P2 SC P1 Root H1 k2 c1 c2 #i #j.
      ReceiveConf2(C,P2,Root,H1,'Id2',k2) @ #i
      & CustomerRecFinalizedMPCS(C,SC,P1,P2,'D',Root,c1,c2) @ #j
    ==>
     ( Ex #k. SendConf2(P2,C,Root,H1,'Id2',k2) @ #k & k < i )
    )
  "

// P2 authenticates C in Finalization sub-protocol case (P2 behaves dishoneslty):
// whenever P2 completes a run of the Signing and Finalization sub-protocols, apparently with C,
// then C has previously been running the protocol, apparently with P2, and the two entities
// agreed on Root and Id2
lemma P2_authenticates_C_Finalization:
  " ( All C P2 Root c2 k2 #i.
      Authentic32(P2,C,Root,c2,k2) @ #i
    ==>
     ( Ex H1 #j. SendCommit2(C,P2,Root,H1,'Id2',k2) @ #j & j < i )
    )
  "

// Fairness in Signing and Finalization sub-protocols when P2 behaves dishonestly
// by sending a cancel signature to C:
// In any session of the protocol in which C receives
// - authentic signature on individual contract c1 from P1 (represented by action fact
//    Authentic41) and
// - authentic signature from SC after Finalization (represented by action fact
//    CustomerRecFinalizedMPCS),
// then P1 has previously received the signature from C and C has previously received a
// cancel signature from P2
lemma FairnessSigningFinalization:
  "( All C SC P1 P2 Root c1 c2 k1 #i #j.
```

```
     Authentic41(C,P1,c1,Root,k1) @ #i
     & CustomerRecFinalizedMPCS(C,SC,P1,P2,'D',Root,c1,c2) @ #j
     ==>
     (( Ex #k #l. Authentic31(P1,C,Root,c1,k1) @ #k & k < i
        & CustomerReceiveCancel(C,P2,Root,c2) @ #l & l < j)
     )
    )
  "

// Model executability for Signing, Finalization and Status Retrieval sub-protocols:
// There is a session of Signing sub-protocol in which C behaves dishonestly:
//  - C obtains the confirmations from P1 and P2, but then executes Finalization
//    without having sent its signature to P2
lemma C_Dishonest_Customer_Finalize2:
  exists-trace
  " Ex C SC P1 c1 P2 c2 k1 Root #i #j.
    Authentic31(P1,C,Root,c1,k1) @ #i
    & CRecFinalizedMPCS(C,SC,P1,P2,'D',Root,c1,c2) @ #j
  "

// Confidentiality of the content of contracts in Signing, Finalization and Status Retrieval
lemma Contracts_Confidentiality_SignFinStRet:
  " ( All C SC P1 c1 P2 c2 Root k1 #i #j #k.
      Authentic31(P1,C,Root,c1,k1) @ #i
      & CRecFinalizedMPCS(C,SC,P1,P2,'D',Root,c1,c2) @ #j
      & ProviderRetrieveStatusFinalizedMPCS(P2,SC,Root) @ #k
    ==> ( not (Ex #l. K(c1) @ #l) & not (Ex #m. K(c2) @ #m) )
    )
  "

// Case C is dishonest: SC authenticates C on message 1 from Finalization sub-protocol
lemma SC_authenticates_C_on_finalization2:
  " ( All C SC P1 P2 Root c1 c2 #i. AuthFinalizeMPCS(SC,C,P1,P2,'D',Root,c1,c2) @ #i
    ==>
     ( Ex #k. CSendFinalizeMPCS(C,SC,P1,P2,'D',Root,c1,c2) @ #k & k < i )
    )
  "
// Case C is dishonest: C authenticates SC on message 2 from Finalization sub-protocol
lemma C_authenticates_SC_on_rec2:
  " ( All C SC P1 P2 Root c1 c2 #i. CRecFinalizedMPCS(C,SC,P1,P2,'D',Root,c1,c2) @ #i
    ==>
     ( Ex #j. SCSendFinMPCS(SC,C,'D',Root)  @ #j & j < i )
    )
  "

// SC authenticates P2 on message 1 from Status Retrieval sub-protocol
lemma SC_authenticates_P2_on_statusret:
  " ( All SC P2 Root #i. AuthenticStatusretrievalMPCS(SC,P2,'D',Root) @ #i
    ==>
     (Ex c2 #j. SendRetrieveStatusMPCS(P2,SC,Root,c2)  @ #j & j < i )
    )
  "

// P2 authenticates SC on message 2 from Status Retrieval sub-protocol
lemma P2_authenticates_SC_on_statusret:
  " ( All SC P2 Root #i. ProviderRetrieveStatusFinalizedMPCS(P2,SC,Root) @ #i
    ==>
     ( Ex C P1 c1 c2 #j. SCSendStatusMPCS(C,SC,P1,P2,'D',Root,c1,c2)  @ #j & j < i )
    )
  "

// C authenticates P2 in Status Retrieval sub-protocol case (C behaves dishoneslty):
// whenever C completes a run of the Signing and Finalization sub-protocols, apparently with P2,
// then P2 has previously been running the protocol, apparently with C, and the two entities
// agreed on Root and Id2
lemma C_authenticates_P2_StatusRetrieval:
  " ( All C P2 SC P1 Root H1 k2 c1 c2 #i #j.
      ReceiveConf2(C,P2,Root,H1,'Id2',k2) @ #i
      & CRecFinalizedMPCS(C,SC,P1,P2,'D',Root,c1,c2) @ #j
```

```
    ==>
     ( Ex #k. SendConf2(P2,C,Root,H1,'Id2',k2) @ #k & k < i )
    )
  "

// P2 authenticates C in Status Retrieval sub-protocol case (C behaves dishoneslty):
// whenever P2 completes a run of the Signing and Status Retrieval sub-protocols, apparently
// with C, then C has previously been running the protocol, apparently with P2, and the two
// entities agreed on Root and Id2
lemma P2_authenticates_C_StatusRetrieval:
  " ( All C P2 SC Root H1  #i #j.
      AuthenticCommit2(P2,C,Root,H1,'Id2') @ #i
      & ProviderRetrieveStatusFinalizedMPCS(P2,SC,Root) @ #j
    ==>
     ( Ex k2 #k. SendCommit2(C,P2,Root,H1,'Id2',k2) @ #k & k < i )
    )
  "

// Fairness in Signing, Finalization and Status Retrieval sub-protocols when C behaves
// dishonestly by executing Finalization, without having sent its signature to P2:
// In any session of the protocol in which C receives authentic signature on individual
// contract c1 from P1 (represented by action fact Authentic41) and P2 receives authentic
// signature from SC (represented by action fact ProviderRetrieveStatusFinalizedMPCS)
// after execution of Status Retrieval,
// then P1 has previously received the signature from C and C has previously received
// the signature from SC after Finalization
lemma FairnessSigningFinalizationRetrieval:
  "( All C SC P1 P2 Root c1 k1 #i #j.
     Authentic41(C,P1,c1,Root,k1) @ #i
     & ProviderRetrieveStatusFinalizedMPCS(P2,SC,Root) @ #j
     ==>
     (( Ex c2 k1 #k #l. Authentic31(P1,C,Root,c1,k1) @ #k & k < i
        & CRecFinalizedMPCS(C,SC,P1,P2,'D',Root,c1,c2) @ #l & l < j )
     )
    )
  "

end
```