```
// PMPCS-CT

// Authors:     Catalin V. Birjoveanu
// Date:        July 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) ]
  -->
  [ 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 = <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 = <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 = <Root,H2,'Id1'>
        m1 = <'Id1',Root,H2>
        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 = <Root,H1,'Id2'>
        m1 = <'Id2',Root,H1>
        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 = <'Id1',Root,H2>
        //c = senc(<m,sign(m,ltkP1)>,k1)
        // m = <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 = <'Id2',Root,H1>
    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 = <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 = <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 (message3) from C and sends his signature (message4) to
C
rule Provider_14:
    let
        m = <Root,c1>
        m1 = <c1,Root>
        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),SendSig41(P1,C,c1,Root,k1) ]->
    [ Out(c) ]


// Signing.4.1  P2 receive C's signature (message3) from C and sends his signature (message4) to
C
rule Provider_24:
```

```
    let
        m = <Root,c2>
        m2 = <c2,Root>
        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),SendSig42(P2,C,c2,k2,Root) ]->
    [ Out(c) ]

// Signing.4.1  C receives P1's signature (message4) from P1
rule Customer_41:
    let
        m = <c1,Root>
    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) ]->
    [ CustomerRecSig41(C,P1,c1,Root)]

// Signing.4.1  C receives P2's signature (message4) from P2
rule Customer_42:
    let
        m = <c2,Root>
    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) ]->
    [ CustomerRecSig42(C,P2,c2,Root) ]

// Finalization: P2 receives the signature from C and
// behaves dishonestly by sending a cancel signature to C
rule Dishonest_P2:
    let
        m2 = <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)
    ]
    --[ 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 = <c2,'cancel',Root>
```

```
        m1= <'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,SC,k)
    , !Ltk(C, ltkC)
    , !Pk(P2, pkP2)
    , In(senc(<m,sig>,k))
    ]
  --[ Eq(verify(sig,m,pkP2),true),
NotAuthentic32(P2,C,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 senc(<'Finalized',m,sig>,k) to
C
rule SmartContract_finalizeMPCS:
    let
        m = <'D',Root>
        m1 =<'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(C,SC,P1,P2,'D',Root,c1,c2)]->
    [ SCSendFinalize(C,'D',Root,'1')
    , Out(c)
    ]

//Finalization.2 C receives 'Finalized' from SC
rule Customer_Finalization_2:
     let
        m = <'D',Root>
        m1 =<'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)
    ]

// Status Retrieval.1  P2 calls the retrieveStatusMPCS() function
rule Provider2_Status_Retrieval_1:
    let
        m = <Root>
        m1 =<'RetriveStatus',m>
        c = senc(<m1,sign(m1,ltkP2)>,k)
    in
    [ ProviderSendCancelSig_2(P2,C,c2,k2,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
senc(<'Finalized',m,sig>,k) to C
rule SmartContract_retrieveStatusMPCS:
    let
        m = <'D',Root>
        m1 =<'Finalized',m>
        c = senc(<m1,sign(m1,ltkSC)>,k)
    in
    [ SCSendFinalize(C,'D',Root,'1')
    , !Merkle(C,P1,P2,Root,H1,H2,c1,c2)
    , !SharedKey(C,SC,k)
    , !Pk(P2, pkP2)
    , !Ltk(SC, ltkSC)
    , In(senc(<m,sig>,k))
    ]
  --[ Eq(verify(sig,m,pkP2),true), SendStatusMPCS(P2,SC,Root,c2)]->
    [ SCSendStatus(SC,P2,C,Root)
    , Out(c)
    ]

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

        m =<'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), RetrieveStatusFinalizedMPCS(P2,SC,Root)]->
    [ Provider2RecStatus(SC,P2,C,Root)
    ]

// 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"

// Model executability
lemma Customer_Providers_honest_session:
  exists-trace
  " Ex C P1 Root P2 c1 c2 #i #j.
        RecSig41(C,P1,c1,Root) @ #i & RecSig42(C,P2,c2,Root) @ #j
  "

// Confidentiality of the content of contracts
lemma Contracts_Confidentility:
  " not(
    Ex C P1 c1 P2 c2 Root #i #j #k #l.
    RecSig41(C,P1,c1,Root) @ #i
    & RecSig42(C,P2,c2,Root) @ #j
    & K(c1) @ #k
    & K(c2) @ #l
    )
  "

// C authenticates the commitment to P1
lemma C_Auth_commit_to_P1:
  " ( 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 )
    )
  "

// C authenticates the commitment to P2
lemma C_Auth_commit_to_P2:
  " ( 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 )
    )
  "

// P1 authenticates the confirmation to C
lemma P1_Auth_conf_to_C:
  " ( 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 )
    )
  "

// P2 authenticates the confirmation to C
lemma P2_Auth_conf_to_C:
  " ( 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 )
    )
  "

// C authenticates the signature (message 3) to P1
lemma C_Auth_sig_to_P1:
  " ( All P1 C Root c1 #i. Authentic31(P1,C,Root,c1) @ #i
    ==>
     (Ex k1 #j. SendSig31(C,P1,Root,c1,k1) @ j & j < i )
    )
  "

// C authenticates the signature (message 3) to P2
lemma C_Auth_sig_to_P2:
  " ( All C P2 Root c2 #i. Authentic32(P2,C,Root,c2) @ #i
    ==>
     (Ex k2 #j. SendSig32(C,P2,Root,c2,k2) @ j & j < i )
    )
  "

// P1 authenticates the signature (message 4) to C
lemma P1_Auth_sig_to_C:
  " ( All P1 C Root c1 #i. Authentic41(C,P1,c1,Root) @ #i
    ==>
     (Ex k1 #j. SendSig41(P1,C,c1,Root,k1) @ j & j < i )
    )
  "

// P2 authenticates the signature (message 4) to C
lemma P2_Auth_sig_to_C:
  " ( All P2 C Root c2 #i. Authentic42(C,P2,c2,Root) @ #i
    ==>
     (Ex k2 #j. SendSig42(P2,C,c2,k2,Root) @ j & j < i )
    )
  "

// Fairness in the Signing sub-protocol under the conditions of honest behavior of the entities
lemma FairnessSigning:
  " ( All C P1 P2 Root c1 c2 #i #j. Authentic41(C,P1,c1,Root) @ #i & Authentic42(C,P2,c2,Root) @
#j
    ==>
     (Ex #k #l. Authentic31(P1,C,Root,c1) @ #k & k < i  & Authentic32(P2,C,Root,c2) @ #l & l < j
)
    )
```

```
  "

// Fairness in dispute scenarios due to P2's dishonest behavior
/*lemma FairnessSigningFinalization:
  "( All C P1 P2 Root c1 c2 #i #j. Authentic41(C,P1,c1,Root) @ #i & NotAuthentic32(P2,C,Root,c2)
@ #j
     ==>
     (Ex SC #k #l. Authentic31(P1,C,Root,c1) @ #k & k < i &
AuthenticFinalizeMPCS(C,SC,P1,P2,'D',Root,c1,c2) @ l & l <j)
   )
   "*/

end
```