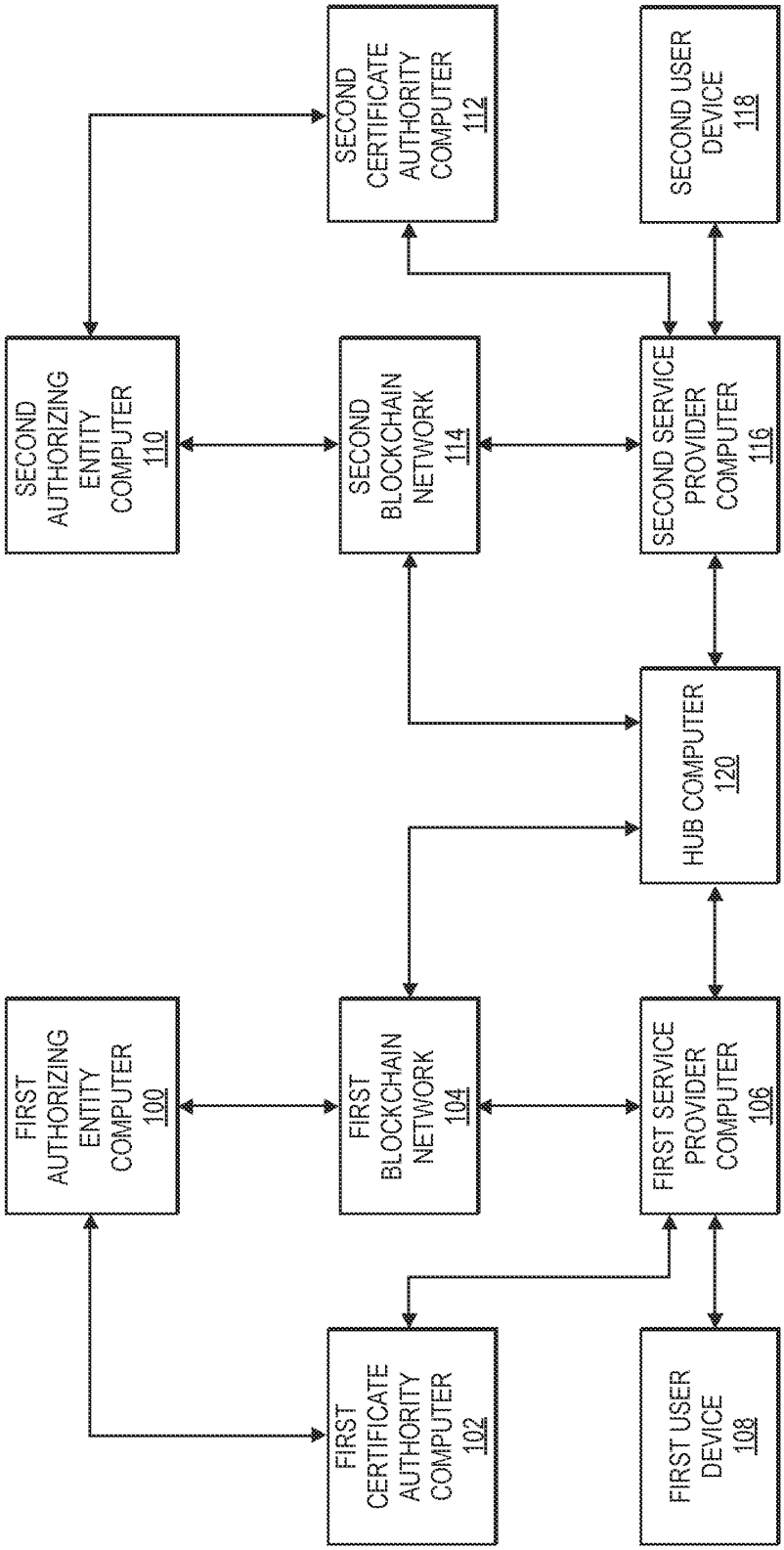
SECOND CERTIFICATE AUTHORITY COMPUTER 112
SECOND USER DEVICE 118
SECOND AUTHORIZING ENTITY COMPUTER 110
SECOND BLOCKCHAIN NETWORK 114
SECOND SERVICE PROVIDER COMPUTER 116
HUB COMPUTER 120
FIRST AUTHORIZING ENTITY COMPUTER 100
FIRST BLOCKCHAIN NETWORK 104
FIRST SERVICE PROVIDER COMPUTER 106
FIRST CERTIFICATE AUTHORITY COMPUTER 102
FIRST USER DEVICE 108

FIG. 1

SMART CONTRACT
200

Init(cid, vk1, vk2, T):

1. Set channelID ← cid, vk_S ← vk_1, vk_C ← vk_2, disputeTime ← T
2. Set status ← "Active", finalidx ← 0, finalCredit ← 0, serverDeposit ← 0, clientDeposit ← 0, channelExpiry ← 0, closeRequestor ← ⊥

GetParams():

Output[channelID, vk_S, vk_C, disputeTime]

GetState():

Output[status, finalidx, finalCredit, serverDeposit, clientDeposit, channelExpiry, closeRequestor]

Deposit(amount):

1. Abort is status "Active"
2. Abort if caller.vk ≠ vk_S and caller.vk ≠ vk_C
3. If caller.vk = vk_S, then set serverDeposit ← serverDeposit + amount
4. If caller.vk = vk_C, then set clientDeposit ← clientDeposit + amount

InitClose():

1. Abort if status ∈ {CooperativeClosing, Closed}
2. If status = "Active", then channelExpiry ← currentTime + disputeTime, status ← "UnilateralClosing" and close Requester ← caller.vk
3. If status = "UnilateralClosing" and caller.vk ← closeRequester, then set status ← "CooperativeClosing"

Withdraw():

1. Abort if status ∈ {Active, Closed}
2. Abort if status = "Active" and currentTime > channelExpiry
3. Invoke vk_C.transfer(clientDeposit + finalCredit) and vk_S.transfer(serverDeposit – finalCredit)
4. Set status ← "Closed"

Claim(cid, idx, clientCredit, txAmount, hash, expiry, secretReceived, σ , secret):

1. Abort if idx ≤ finalidx
2. If secretReceived = 0, abort if currentTime > Expiry or if Hash(secret) ≠ hash
3. If caller.vk = vk_S:
   (a) Abort if SigVerify([channelID, idx, clientCredit, txAmount, hash, expiry, secretReceived, σ, vk_C) ≠ 1], , vk_C) ≠ 1
   (b) Abort if txAmount > clientDeposit + clientCredit
   (c) Update finalCredit ← clientCredit – txAmount
4. Update finalidx ← idx
5. Invoke InitClose()

FIG. 2

HELPER FUNCTIONS
300

CreatePromise(cid, state, txAmount, hash, expiry, sk):

1. Set secretReceived ← 0
2. Output [cid, idx, clientCredit, txAmount, hash, expiry, secretReceived, σ], where σ = Sign[cid, idx, clientCredit, txAmount, hash, expiry, secretReceived]

VerifyPromise(promise, cid, state, txAmount, hash, expiry, vk):

1. Set secretReceived ← 0
2. Output SigVerify(promise.σ, [cid, idx, clientCredit, txAmount, hash, expiry, secretReceived], vk)

CreateReceipt(cid, state, sk):

1. Set txAmount ← ⊥, hash ← ⊥, expiry ← ⊥, secretReceived ← 1
2. Output [cid, idx, clientCredit, txAmount, hash, expiry, secretReceived, σ], where σ = Sign[cid, idx, clientCredit, txAmount, hash, expiry, secretReceived]

VerifyReceipt(receipt, cid, state, vk):

1. Set txAmount ← ⊥, hash ← ⊥, expiry ← ⊥, secretReceived ← 1
2. Output SigVerify(receipt.σ, [cid, idx, clientCredit, txAmount, hash, expiry, secretReceived], vk)

UpdateLocalState(localState, promise, increaseCredit):

1. localState.idx ← localState.idx + 1
2. If increaseCredit = 1, then localState.clientCredit ← localState.clientCredit + promise.txAmount
3. Otherwise, localState.clientCredit ← localState.clientCredit – promise.txAmount
4. Output localState

FIG. 3

CONTRACT AND TIME EVENT HANDLER
400

While TRUE:
For each channel C in party's list of channels:
1. C.contractState ← C.contract.GetState()
2. If C.contractState.status = "Closed"
    a) C ← ⊥ and return [ChannelClosed]
3. If C.contractState.status = "CooperativeClosing"
    a) Invoke C.contract.Withdraw()
4. If C.contractState.status = "UnilateralClosing"
    a) If C.promise.secret ≠ ⊥, then invoke C.contract.Claim(C.promise)
    b) Otherwise, if C.receipt ≠ ⊥, then invoke C.contract.Claim(C.receipt)
    c) If C.receipt = ⊥ and C.promise.secret = ⊥, then invoke C.contract.InitClose()
    d) If currentTime > C.contractState.channelExpiry, then invoke C.contract.Withdraw()
5. If C.contractState.status = "Active" and currentTime > C.contractState.promise.expiry
    a) If C.promise.secret ≠ ⊥, then invoke C.contract.Claim(C.promise)
    b) Set C.promise ← ⊥

FIG. 4

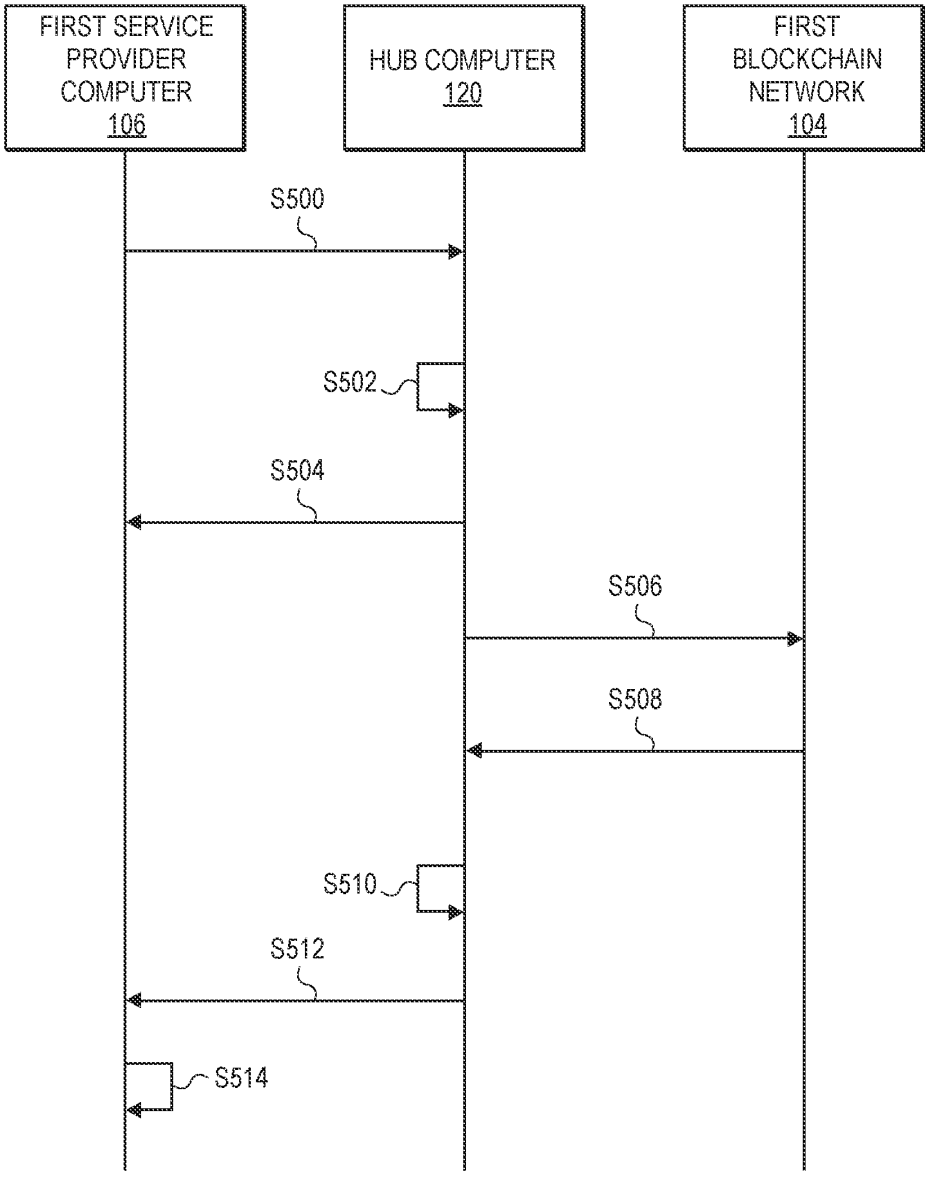
FIRST SERVICE PROVIDER COMPUTER 106
HUB COMPUTER 120
FIRST BLOCKCHAIN NETWORK 104
S500
S502
S504
S506
S508
S510
S512
S514

FIG. 5

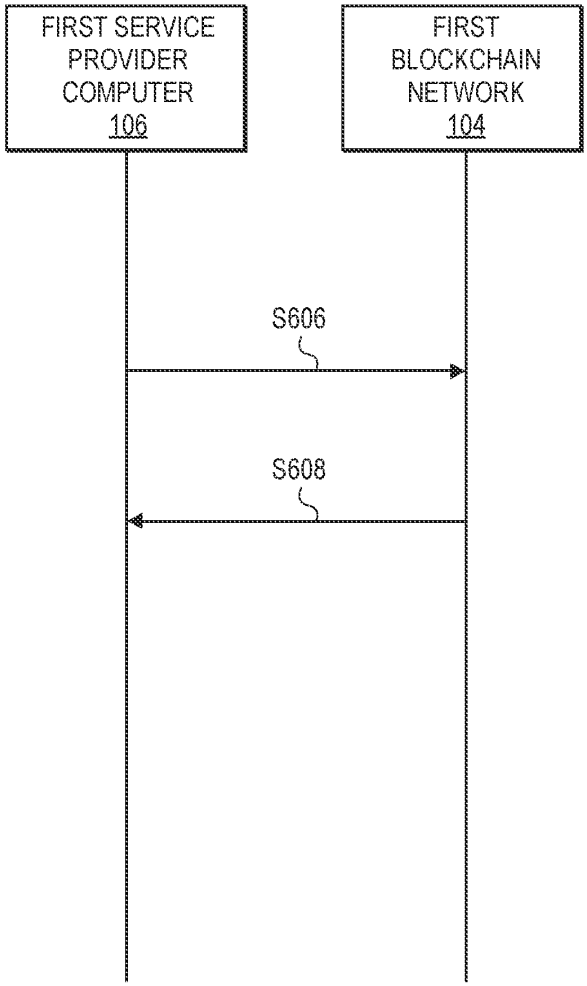
FIRST SERVICE PROVIDER COMPUTER 106
FIRST BLOCKCHAIN NETWORK 104
S606
S608

FIG. 6

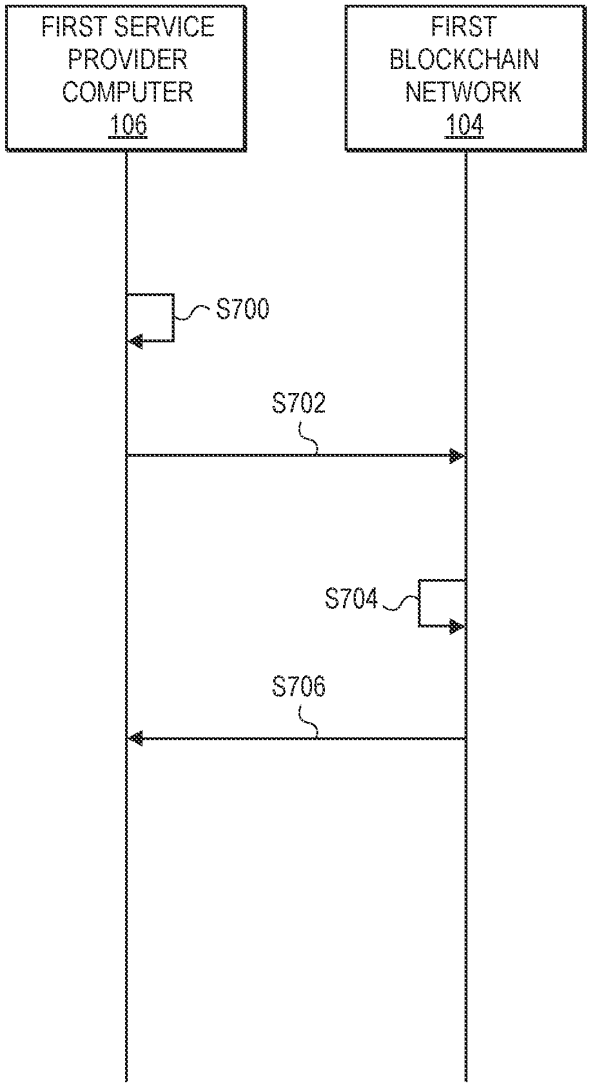
FIRST SERVICE PROVIDER COMPUTER 106
FIRST BLOCKCHAIN NETWORK 104
S700
S702
S704
S706

FIG. 7

AUTHORIZE PROTOCOL
800

Party A initiates transfer to Party B. Party A and B agree on: (hash, txAmount, txExpiry), where Party B knows secret such that hash = Hash(secret)

1. Party A:

a) $P_A \leftarrow$ CreatePromise(cid_A, C_A.localstate, txAmount, hash, txExpiry + $ledger_A.\Delta$, sk_A)

b) Set C_A.promise $\leftarrow P_A$

c) Transmit SenderPromise[P_A, B] to Server S

2. Server S:

a) Set $C_{S,A}$.promise $\leftarrow P_A$

b) $P_S \leftarrow$ CreatePromise(cid_B, $C_{S,B}$.localState, P_A.txAmount, P_A.hash, P_A.expiry – $ledger_A.\Delta$, sk_S)

c) Set $C_{S,B}$.promise $\leftarrow P_S$

d) Transmit ServerPromise[P_S] to Party B

3. Party B

a) Set C_B.promise $\leftarrow P_S$

b) Set C_B.promise.secret $\leftarrow$ secret

FIG. 8

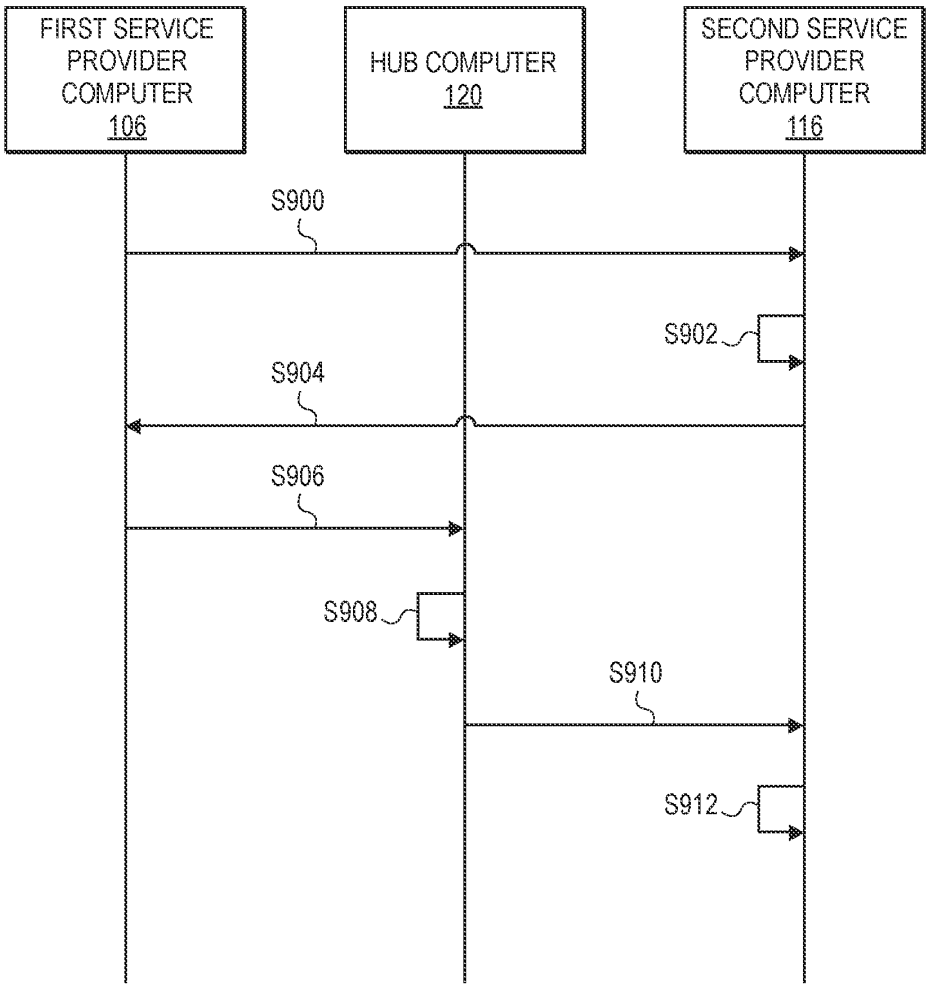
FIRST SERVICE PROVIDER COMPUTER 106
HUB COMPUTER 120
SECOND SERVICE PROVIDER COMPUTER 116
S900
S902
S904
S906
S908
S910
S912

FIG. 9

TRANSFER PROTOCOL
1000

1. Party B:

a) C_B.localState ← UpdateLocalState(C_B.localState, C_B.promise, 1)

b) C_B.promise.secret ← secret

c) Transmit [secret, A] to Server S

2. Server S:

a) $C_{S,B}$.localState ← UpdateLocalState($C_{S,B}$.localState, $C_{S,B}$.promise, 1)

b) $receipt_S$ ← CreateReceipt(cid_B, $C_{S,B}$.localState, sk_S)

c) $C_{S,B}$.promise ← ⊥

d) Transmit ReceiverReceipt[receiptS] to Party B

e) $C_{S,A}$.localState ← UpdateLocalState($C_{S,A}$.localState, $C_{S,A}$.promise, 0)

f) $C_{S,A}$.promise.secret ← secret

g) Transmit [secret] to Party A

3. Party A

a) localState ← UpdateLocalState(C_A.localState, C_A.promise, 0)

b) $receipt_A$ ← CreateReceipt(cid_A, C_A.localState, sk_A)

c) C_A.promise ← ⊥

d) Transmit ServerReceipt[$receipt_A$] to Server S

4. Server S

$C_{S,A}$.promise ← ⊥, $C_{S,A}$.receipt ← $receipt_A$

5. Party B

C_B.promise ← ⊥, C_B.receipt ← $receipt_S$

FIG. 10

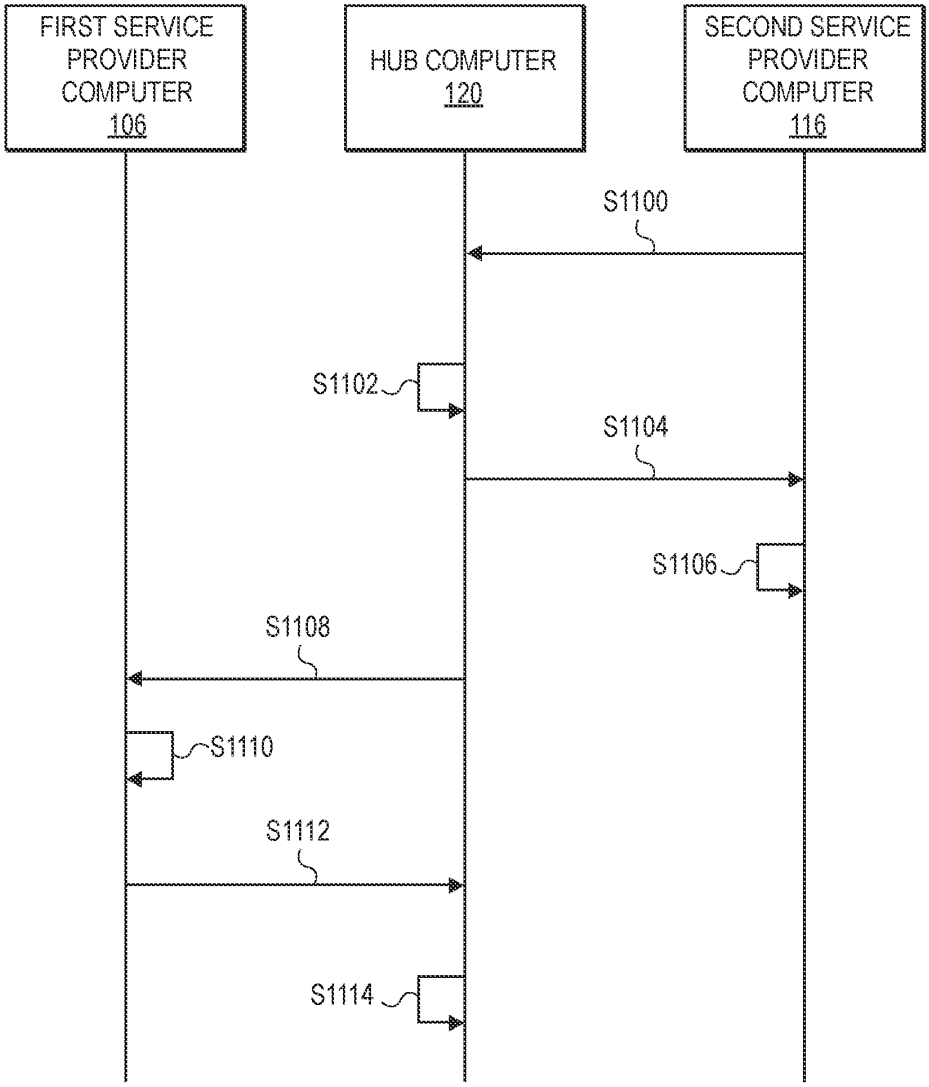
FIRST SERVICE PROVIDER COMPUTER 106
HUB COMPUTER 120
SECOND SERVICE PROVIDER COMPUTER 116
S1100
S1102
S1104
S1106
S1108
S1110
S1112
S1114

FIG. 11

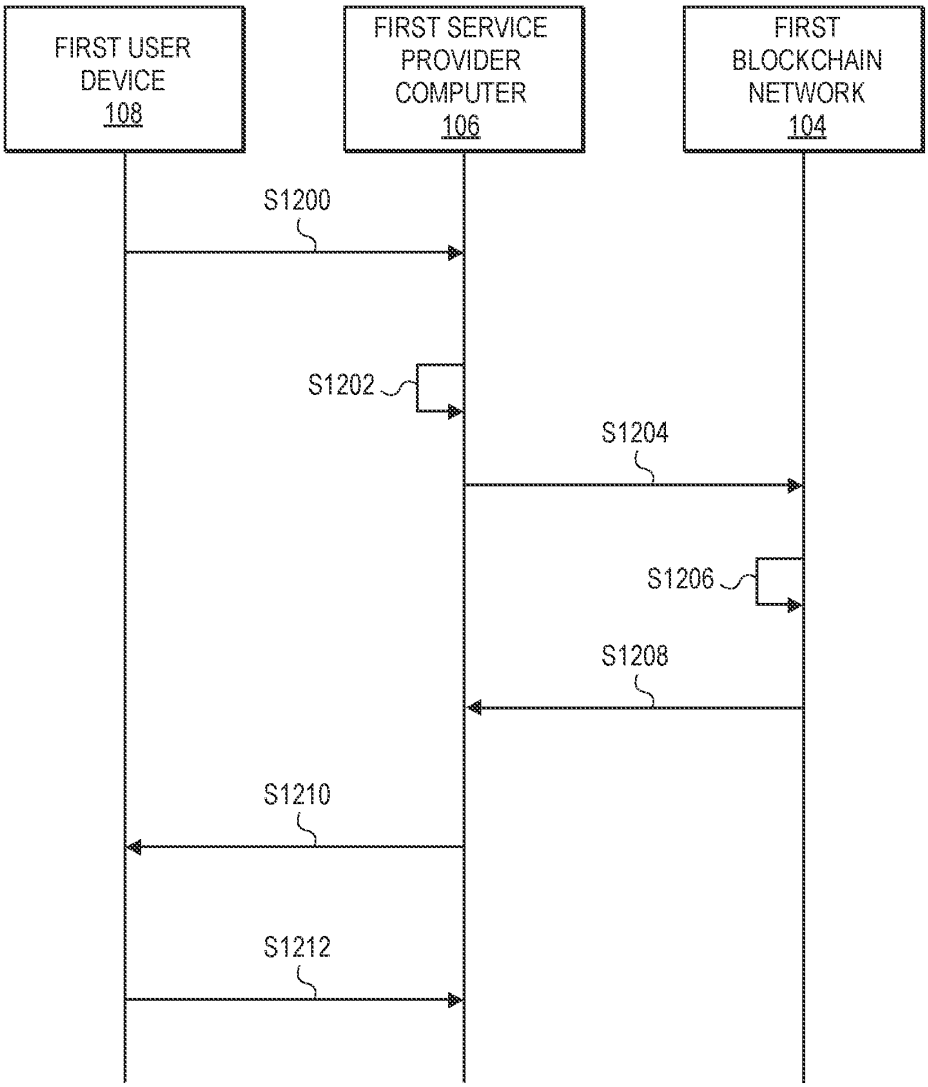
FIRST USER DEVICE 108
FIRST SERVICE PROVIDER COMPUTER 106
FIRST BLOCKCHAIN NETWORK 104
S1200
S1202
S1204
S1206
S1208
S1210
S1212

FIG. 12

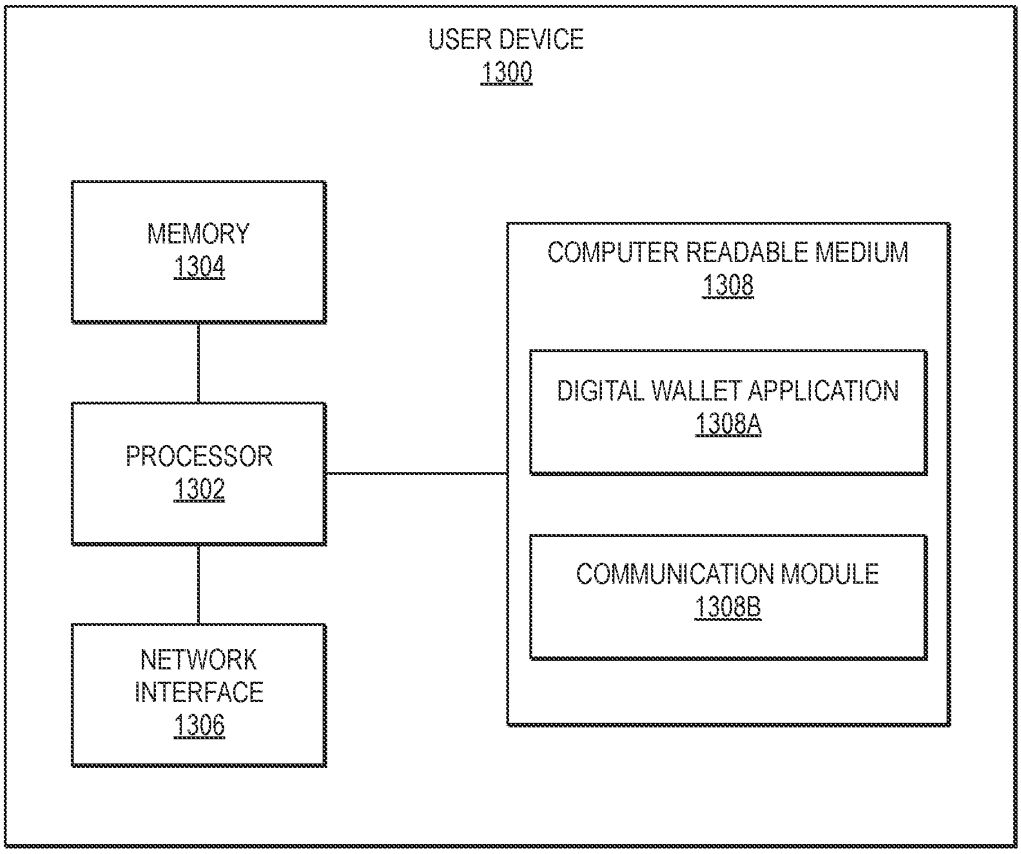
USER DEVICE
1300
MEMORY
1304
PROCESSOR
1302
NETWORK INTERFACE
1306
COMPUTER READABLE MEDIUM
1308
DIGITAL WALLET APPLICATION
1308A
COMMUNICATION MODULE
1308B

FIG. 13

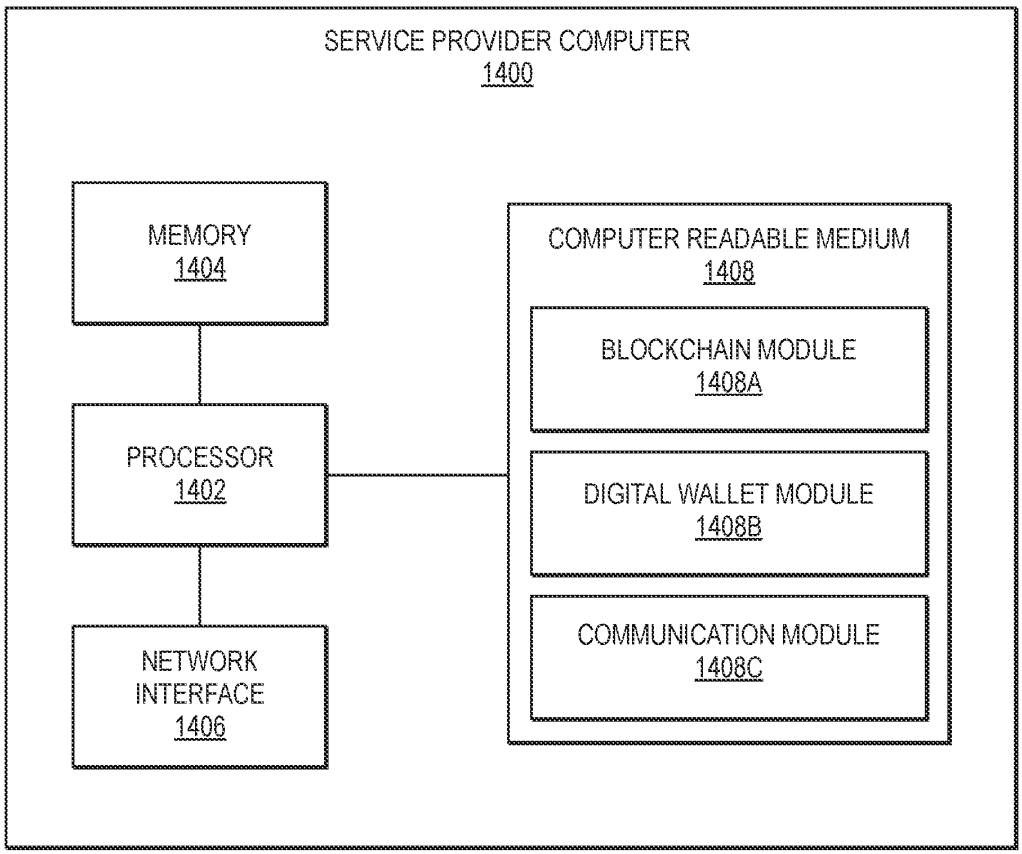
SERVICE PROVIDER COMPUTER
1400
MEMORY
1404
PROCESSOR
1402
NETWORK
INTERFACE
1406
COMPUTER READABLE MEDIUM
1408
BLOCKCHAIN MODULE
1408A
DIGITAL WALLET MODULE
1408B
COMMUNICATION MODULE
1408C

FIG. 14

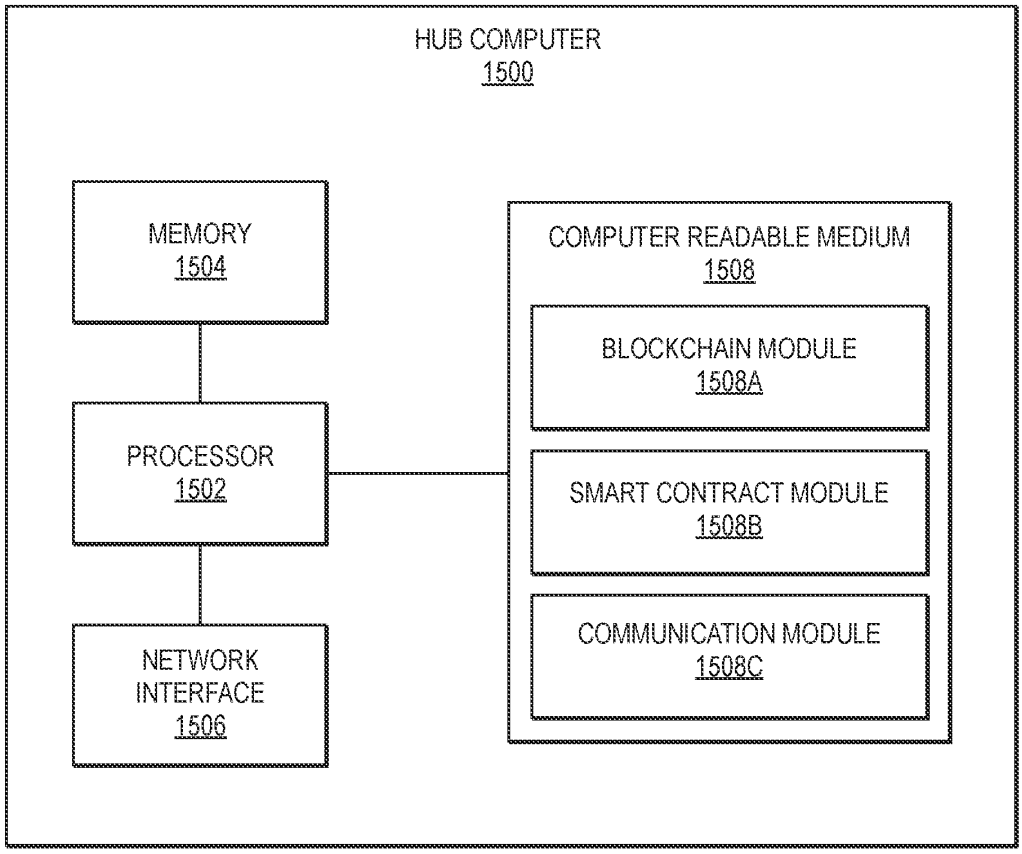
HUB COMPUTER
1500
MEMORY
1504
PROCESSOR
1502
NETWORK
INTERFACE
1506
COMPUTER READABLE MEDIUM
1508
BLOCKCHAIN MODULE
1508A
SMART CONTRACT MODULE
1508B
COMMUNICATION MODULE
1508C

FIG. 15