

1/17

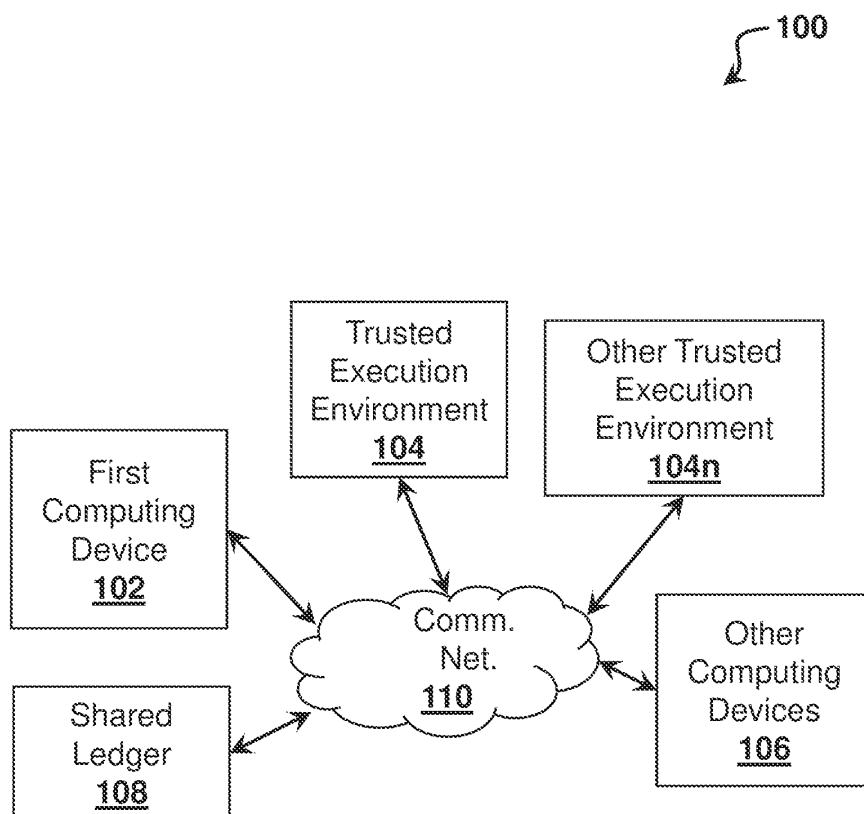


FIG. 1

2/17

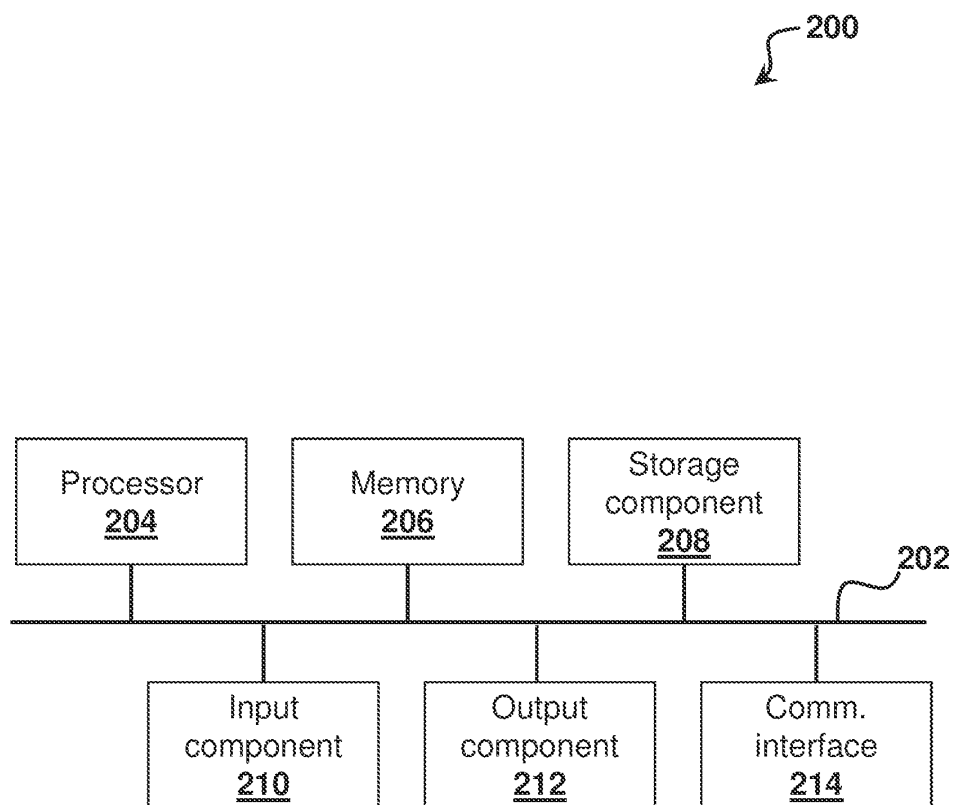


FIG. 2

3/17

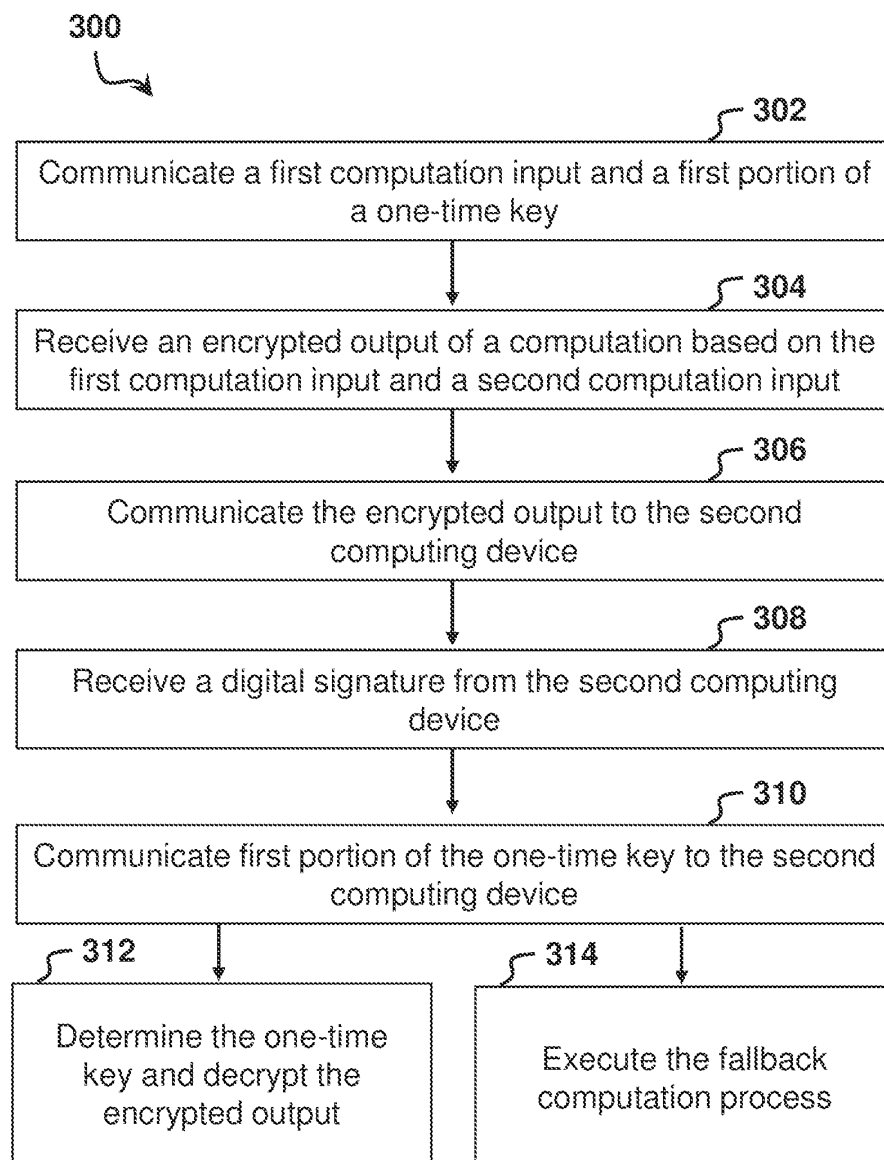


FIG. 3

4/17

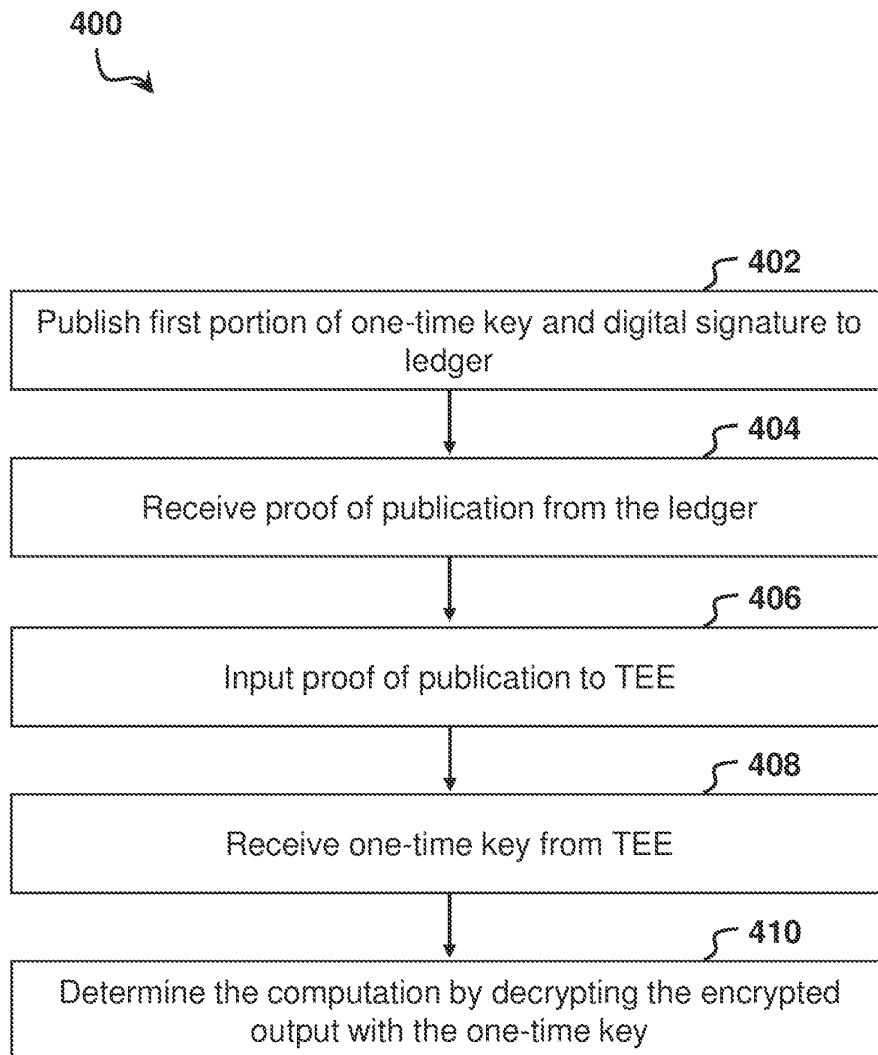


FIG. 4

5/17

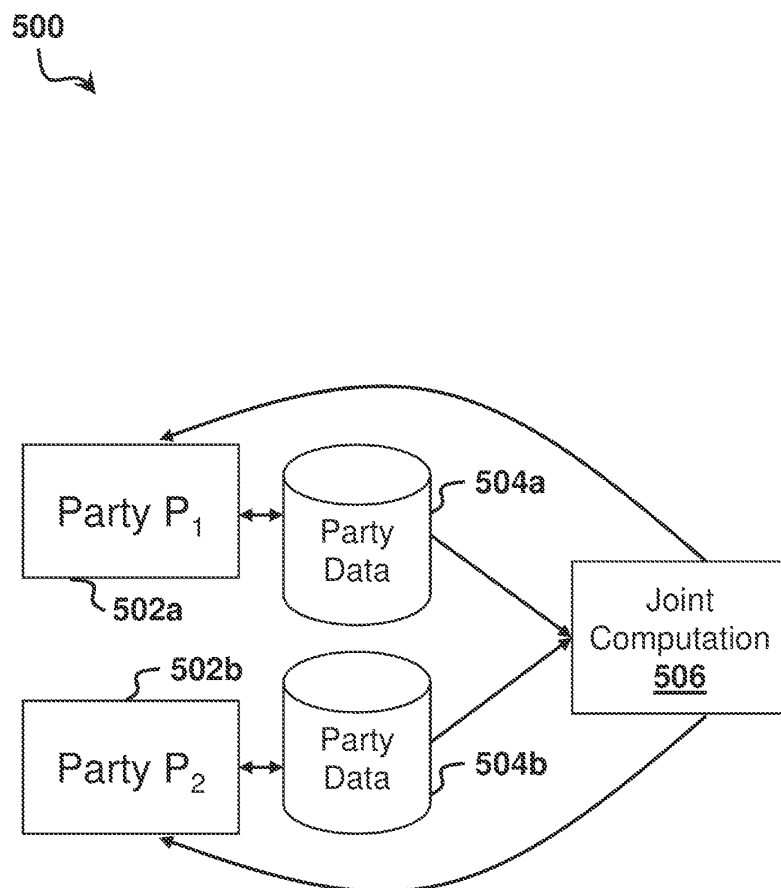


FIG. 5

6/17

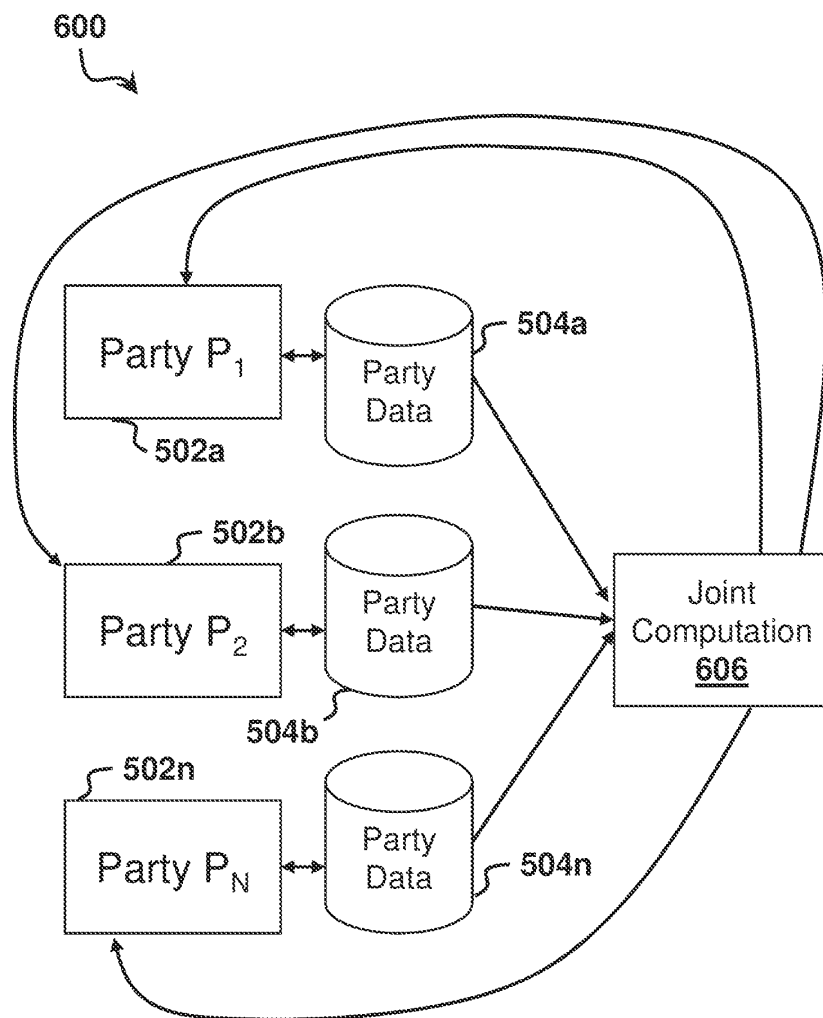


FIG. 6

7/17

- On input ("genKey", i)
 let (epk, esk) $\leftarrow$ PKE.Keygen(1^λ)
 ret HW.Quote(epk)
- On input ("loadMskShares", msk_1 , $encmsk_2$)
 let msk_2 , sig $\leftarrow$ PKE.Dec(esk, $encmsk_2$)
 assert S.Verify(epk, sig, msk_2)
 let $msk \leftarrow msk_1 \oplus msk_2$
- On input ("ledgerPost", sig, nonce)
 assert S.Verify(pk₂, sig, nonce)
 let entry $\leftarrow$ HW.Quote(sig || PKE.Enc(pk₂, PRF(msk, nonce)))
 ret entry
- On input ("verifyProofOfPublication", entryFromL, σ)
 assert Verify_L(σ , entryFromL) $\wedge$ entry = entryFromL
 ret PRF(msk, nonce)

FIG. 7

8/17

(0) Pre-processing Phase

```

if  $i = 1$  {
    let  $e \leftarrow \text{HW.Load}(\text{prog}_{2\text{pfx}})$ 
    let  $(\text{epk}, \text{quote}) \leftarrow \text{HW.Run}(e, (\text{"genKey"}))$ 
    send  $(\text{epk}, \text{quote})$  to  $p_2$ 
    recv  $\text{encmsk}_2$  from  $p_2$ 
     $\text{HW.Run}(e, (\text{"loadMskShares"}, \text{msk}_1, \text{encmsk}_2))$ 
} else if  $i = 2$  {
    recv  $(\text{epk}, \text{quote})$  from  $p_1$ , and assert  $\text{HW.QuoteVerify}(\text{quote})$ 
    send  $\text{PKE.Enc}(\text{epk}, \text{msk}_2)$  to  $p_1$ 
}

```

(1) Signature confirming receipt of encrypted output

```

if encrypted output was not given, then abort
if  $i = 1$  { recv sig from  $p_2$ , and assert  $\text{S.Verify}(\text{pk}_2, \text{sig}, \text{nonce})$  }
if  $i = 2$  { send  $\text{S.Sig}(\text{sk}_2, \text{nonce})$  to  $p_1$  }

```

(2) TEE-possessing party p_1 sends share to non-TEE party p_2

```

if  $i = 1$  { send  $\text{PKE.Enc}(\text{pk}_2, k_1)$  to  $p_2$  }
if  $i = 2$  { recv msg from  $p_1$ , and let  $k_1 \leftarrow \text{PKE.Dec}(\text{sk}_2, \text{msg})$  }

```

FIG. 8

9/17

- (3) **Non-TEE Party p_2 sends share to TEE-possessing party p_1**
 if $i = 1$ {
 recv msg from p_2
 let $k_2 \leftarrow \text{PKE.Dec}(sk_1, \text{msg})$
 } else if $i = 2$ {
 if p_1 aborted in step 2 { **sleep** in this round, and **goto** step (4) }
 send $\text{PKE.Enc}(pk_1, k_2)$ to p_1
 }
output $k_1 \oplus k_2$
- (4) **Initiating Fallback Mechanism**
 if $i = 1$ {
 assert p_2 aborted in step 3 (else we should have terminated)
 keyenc $\leftarrow \text{HW.Run}(e, (\text{"ledgerPost"}, \text{sig}))$
 post sig || keyenc to L
 } else if $i = 2$ {
 assert p_1 aborted in step 2 (else we should have terminated)
 sleep in this round, and **goto** step (5)
 }
 }
- (5) **Completing Fallback Mechanism**
 get (sig || keyenc, σ) from L
 if $i = 1$ {
 $k \leftarrow \text{HW.Run}(e, (\text{"verifyProofOfPublication"}, \text{sig} \parallel \text{keyenc}, \sigma))$
 } else if $i = 2$ {
 $k \leftarrow \text{PKE.Dec}(sk_2, \text{keyenc})$
 }
output k

FIG. 9

10/17

- On input (“genKey”, i)
 - let (epk, esk) $\leftarrow$ PKE.Keygen(1^λ)
 - ret HW.Quote($i \parallel$ epk)
- On input (“loadMskShares”, emsk)
 - let $msk_1 \parallel \dots \parallel msk_n \leftarrow$ PKE.Dec(esk, emsk)
 - assert** verify commitments
 - let $msk \leftarrow msk_1 \oplus \dots \oplus msk_n$
- On input (“ProofOfPublication1”, signatures, σ , nonce)
 - let $k \leftarrow$ PRF(msk, nonce); let $sig_1, \dots, sig_n \leftarrow$ signatures
 - assert** $verify_L(\sigma) \wedge \forall j \in \{1, \dots, n\}. S.Verify(pk_j, sig_j, nonce)$
 - let entry $\leftarrow$ PKE.Enc(pk_{t+1}, k) $\parallel \dots \parallel$ PKE.Enc(pk_n, k)
 - ret HW.Quote(entry)
- On input (“ProofOfPublication2”, entryFromL, σ)
 - assert** $Verify_L(\sigma, entryFromL) \wedge entry = entryFromL$
 - ret PRF(msk, nonce)

FIG. 10

11/17

(0) Pre-processing Phase

```

if  $i \leq t$  {
    let  $e \leftarrow \text{HW.Load}(\text{prog}_{\text{npfx}})$ 
     $(\text{epk}_i, \text{quote}_i) \leftarrow \text{HW.Run}(e, (\text{"genKey"}))$ 
    send  $(\text{epk}_i, \text{quote}_i)$  to  $\{p_1, \dots, p_n\} \setminus \{p_i\}$ 
    recv  $\text{msg}_1, \dots, \text{msg}_t$  from  $\{p_1, \dots, p_t\} \setminus \{p_i\}$ 
    let  $(\text{epk}_j, \text{quote}_j) \leftarrow \text{msg}_j$ , and assert  $\forall j. \text{HW.QuoteVerify}(\text{quote}_j)$ 
    let  $\text{emsk}_i \leftarrow \text{PKE.Enc}(\text{epk}_1, \text{msk}_i), \dots, \text{PKE.Enc}(\text{epk}_t, \text{msk}_i)$ 
    send  $\text{emsk}_i$  to  $\{p_1, \dots, p_t\} \setminus \{p_i\}$ 
    recv  $\text{emsk}_1, \dots, \text{emsk}_n$  from  $\{p_1, \dots, p_n\} \setminus \{p_i\}$ 
     $\text{HW.Run}(e, (\text{"loadMskShares"}, \text{emsk}_1[i] \mid \dots \mid \text{emsk}_n[i]))$ 
} else {
    recv  $\text{msg}_1, \dots, \text{msg}_t$  from  $\{p_1, \dots, p_t\}$ 
    let  $(\text{epk}_j, \text{quote}_j) \leftarrow \text{msg}_j$ , and assert  $\forall j. \text{HW.QuoteVerify}(\text{quote}_j)$ 
    let  $\text{emsk}_i \leftarrow \text{PKE.Enc}(\text{epk}_1, \text{msk}_i), \dots, \text{PKE.Enc}(\text{epk}_t, \text{msk}_i)$ 
    send  $\text{emsk}_i$  to  $\{p_1, \dots, p_t\}$ 
}

(1) TEE parties get signatures that all parties got encrypted output
send  $S.\text{Sig}(\text{sk}_i, \text{nonce})$  to  $\{p_1, \dots, p_n\} \setminus \{p_i\}$  if got encrypted output
if  $i \leq t$  {
    recv  $\{\text{sig}_j\}_{j \in \{1, \dots, n\} \setminus \{i\}}$  from  $\{p_1, \dots, p_n\} \setminus \{p_i\}$ 
    assert  $\forall j \in \{1, \dots, n\}: S.\text{Verify}(\text{pk}_j, \text{sig}_j, \text{nonce})$ 
}

(2) Each TEE-possessing party sends share to all non-TEE parties
if  $i \leq t$  {
    for  $j \in [t+1, \dots, n]$ : send  $\text{PKE.Enc}(\text{pk}_j, k_i)$  to  $p_j$ 
} else if  $i > t$  {
    recv  $\text{msg}_1, \dots, \text{msg}_t$  from  $p_1, \dots, p_t$ 
    let  $k_1, \dots, k_t \leftarrow \text{PKE.Dec}(\text{sk}_1, \text{msg}_1), \dots, \text{PKE.Dec}(\text{sk}_t, \text{msg}_t)$ 
}

```

FIG. 11

12/17

- (3) **Each non-TEE party sends share to all TEE-possessing parties**
 if $i \leq t$ {
 recv $ek_{t+1}, \dots, ek_n$ from $p_{t+1}, \dots, p_n$
 let $k_{t+1}, \dots, k_n \leftarrow \text{PKE.Dec}(sk_i, ek_{t+1}), \dots, \text{PKE.Dec}(sk_i, ek_n)$
 } else if $i > t$ {
 if any of $p_1, \dots, p_t$ aborted in step (2) { **goto** step (4) }
 send $\text{PKE.Enc}(pk_j, k_i)$ to all $p_j \in \{p_1, \dots, p_n\}$
 recv one of $k_{t+1}, \dots, k_n$ from each $p_j \in \{p_{t+1}, \dots, p_n\}$
 }
 (4) **If got n shares, then send them to all parties; else wait for n shares**
 if got n shares { **send** $k \doteq k_1 \oplus \dots \oplus k_n$ to $p_1, \dots, p_n$ }
 else { if **recv** all n shares from some $p_j \in \{p_1, \dots, p_n\}$ }
 if got all n shares, then **output** k
 (5) **If we still didn't get all n shares, then enter fallback mechanism**
 if $i \leq t$ { **post** $sig_1, \dots, sig_n$ to L }
 else if $i > t$ { **sleep** in this round, and **goto** step (5) }
 (6) **Perform step 2 of the fallback mechanism**
 if $i \leq t$ {
 get (signatures, σ) from L, where signatures $\doteq sig_1 \parallel \dots \parallel sig_n$
 let $keyenc \leftarrow \text{HW.Run}(e, (\text{"ProofOfPublication1"}, \text{signatures}, \sigma))$
 post $keyenc$ to L
 } else if $i > t$ { **sleep** in this round, and **goto** step (6) }
 (7) **Complete the fallback mechanism**
 get ($keyenc, \sigma$) from L
 if $i \leq t$ {
 $k \leftarrow \text{HW.Run}(e, (\text{"ProofOfPublication2"}, keyenc, \sigma))$; **output** k
 } else if $i > t$ {
 $k \leftarrow \text{PKE.Dec}(sk_i, keyenc)$; **output** k
 }

FIG. 12

13/17

We assume $t \cdot n$ instances of $\mathcal{F}_{\text{SyX}}$, and denote them $\mathcal{F}_{\text{SyX}}^{i,j}$, where $i \in [1, \dots, t]$ and $j \in [1, \dots, n]$. We use $\mathcal{F}_{\text{SyX}}^{i,j}$ and $\mathcal{F}_{\text{SyX}}^{j,i}$ interchangeably to denote the same instance.

- (0) **Pre-processing (performed once for multiple MPC executions)**
 - if $i \leq t$ { edges = n } else { edges = t }
 - foreach $j \in [1, \dots, \text{edges}]$ {
 - send (load, f_1, f_2, ϕ_1, ϕ_2) to $\mathcal{F}_{\text{SyX}}^{i,j}$
 - get ($k_i^{i,j}, c_i^{i,j} \doteq \text{com}(k_i^{i,j}), c_j^{i,j} \doteq \text{com}(k_j^{i,j})$) from $\mathcal{F}_{\text{SyX}}^{i,j}$
- (1) **Run unfair MPC protocol for function f using agreed-upon id**
 - invoke $\pi_{\text{MPC}}(x_i, \text{id}, \{k_i^{i,j}, c_i^{i,j}, c_j^{i,j}\}_{j \in [n]})$, where x_i is p_i 's input.
 - π_{MPC} emits $\text{AE.Enc}(k, f(x_1, \dots, x_n)) \parallel \{\text{PKE.Enc}(pk_j, k_j)\}_{j \in [n]}$
 - $\parallel \{\text{PKE.Enc}(pk_j, k_{\text{id}}^{i,j})\}_{j \in [n]} \parallel \{\text{AE.Enc}(k_{\text{id}}^{i,j}, k_j)\}_{i \in [t], j \in [n]}$,
 - where $k = k_1 \oplus \dots \oplus k_n$, and $k_{\text{id}}^{i,j} \doteq \text{H}(k_i^{i,j} \oplus k_j^{i,j}, \text{id})$.
- (2) **Each TEE-possessing party sends share to all non-TEE parties**
 - if $i \leq t$ {
 - for $j \in [t+1, \dots, n]$: send $\text{PKE.Enc}(pk_j, k_i)$ to p_j
 - } else if $i > t$ {
 - recv $\text{msg}_1, \dots, \text{msg}_t$ from $p_1, \dots, p_t$
 - let $k_1, \dots, k_t \leftarrow \text{PKE.Dec}(sk_i, \text{msg}_1), \dots, \text{PKE.Dec}(sk_i, \text{msg}_t)$
 - }

FIG. 13

14/17

- (3) **Each non-TEE party sends share to all TEE-possessing parties**
 if $i \leq t$ {
 recv $ek_{t+1}, \dots, ek_n$ from $p_{t+1}, \dots, p_n$
 let $k_{t+1}, \dots, k_n \leftarrow \text{PKE.Dec}(sk_i, ek_{t+1}), \dots, \text{PKE.Dec}(sk_i, ek_n)$
 } else if $i > t$ {
 if any of $p_1, \dots, p_t$ aborted in step (2) { **goto** step (4) }
 send $\text{PKE.Enc}(pk_j, k_i)$ to all $p_j \in \{p_1, \dots, p_n\}$
 recv one of $k_{t+1}, \dots, k_n$ from each $p_j \in \{p_{t+1}, \dots, p_n\}$
 }
 (4) **If got n shares, then send them to all parties; else wait for n shares**
 if got n shares { **send** $k \doteq k_1 \oplus \dots \oplus k_n$ to $p_1, \dots, p_n$ }
 else { if **recv** all n shares from some $p_j \in \{p_1, \dots, p_n\}$ }
 if got all n shares, then **output** k
 (5) **If we still didn't get all n shares, then enter fallback mechanism**
 if $i \leq t$ {
 for $j \in [i-1]$ { **send** (trigger) to $\mathcal{F}_{\text{SyX}}^{i,j}$; **recv** $k_{\text{id}}^{i,j}$ to get k_i, k_j }
 for $j \in [i+1 \dots n]$ { **send** (trigger) to $\mathcal{F}_{\text{SyX}}^{i,j}$; **recv** $k_{\text{id}}^{i,j}$ to get k_j }
 send k to all parties
 } else { **receive** k from some party after n rounds }

FIG. 14

15/17

- On input (“genKey”)
 - let (epk, esk) $\leftarrow$ PKE.Keygen(1^λ)
 - ret HW.Quote(epk)
- On input (“genProtKey”, in)
 - let $q_1, \dots, q_t \leftarrow \text{in}$
 - let (μ_1, epk_1) $\leftarrow q_1, \dots, (\mu_t, \text{epk}_t) \leftarrow q_t$
 - assert $\mu_1 = \mu(\mathcal{E}_{fct}) \wedge \dots \wedge \mu_t = \mu(\mathcal{E}_{fct})$
 - assert HW.QuoteVerify(q_1) $\wedge \dots \wedge$ HW.QuoteVerify(q_t)
 - let (pk, sk) $\leftarrow$ PKE.Keygen(1^λ)
 - let out $\leftarrow$ pk || PKE.Enc(epk₁, sk), . . . , PKE.Enc(epk_t, sk)
 - ret HW.Quote(out)
- On input (“loadProtKey”, in)
 - let ($\mu, \text{pk} \parallel \text{ct}_1, \dots, \text{ct}_t$) $\leftarrow$ in
 - assert $\mu = \mu(\mathcal{E}_{fct}) \wedge$ HW.QuoteVerify(in)
 - let sk $\leftarrow$ PKE.Dec(esk, ct_i)
- On input (“revealKey1”)
 - assert round == 3 * t + 4
 - ret HW.Quote(PKE.Enc(pk_{t+1}, sk), . . . , PKE.Enc(pk_n, sk))
- On input (“revealKey2”)
 - assert round == 4 * t + 6
 - ret sk

FIG. 15

16/17

- (1) **Launch the $\mathcal{E}_{fct}$ enclave, and generate key material**
 - if $i > t$ then **goto** step (2)
 - let $e \leftarrow \text{HW.Load}(\text{prog}_{fct})$
 - let $\text{msg} \leftarrow \text{HW.Run}(e, (\text{"genKey"})$
 - let $[q_1, \dots, q_t] \leftarrow \text{Prot}_{DS}[\text{msg}, \{p_1, \dots, p_t\}]$ in parallel
 - let $\text{generator} \leftarrow 1$
- (2) **Establish Protocol Instance Key**
 - if $\text{generator} = i$ {
 - let $\text{msg} \leftarrow \text{HW.Run}(e, (\text{"genProtKey"}, q_1, \dots, q_t))$
 - invoke $\text{Prot}_{DS}[\text{msg}, \{p_1, \dots, p_n\}]$ as commander
 - } else if $\text{generator} \leq t$ {
 - let $\text{msg} \leftarrow \text{Prot}_{DS}[\perp, \{p_1, \dots, p_n\}]$
 - if $\text{msg} = \perp$ then increment generator and **retry** step (2)
 - $\text{pk} \parallel _ \leftarrow \text{msg}$
 - if $i \leq t$ { $\text{HW.Run}(e, (\text{"loadProtKey"}, \text{msg}))$ }
 - } else {
 - invoke $\text{Prot}_{HFCT}[p_{t+1}, \dots, p_n]$ and **output** result
- (3) **Broadcast individual coins**
 - let $\text{msg} \leftarrow \text{PKE.Enc}(\text{pk}, \text{S.Sig}(\text{sk}_{p_i}, b_i))$
 - let $[e_{b_1}, \dots, e_{b_n}] \leftarrow \text{Prot}_{DS}[\text{msg}, \{p_1, \dots, p_n\}]$ in parallel

FIG. 16

17/17

(4) Sharing Secret Key

```

if  $i \leq t$  {
  let  $\text{msg} \leftarrow \text{HW.Run}(e, (\text{"revealKey1"}))$ 
  invoke  $\text{Prot}_{\text{DS}}[\text{msg}, \{p_1, \dots, p_n\}]$  as commander
} else {
  let  $\text{msg}_1, \dots, \text{msg}_t \leftarrow \text{Prot}_{\text{DS}}[\perp, p_1, \dots, p_n]$  in parallel
  let  $\text{msg}$  be any valid message from  $\{\text{msg}_1, \dots, \text{msg}_t\}$ 
  if no such  $\text{msg}$  {
    invoke  $\text{Prot}_{\text{HFCT}}[p_{t+1}, \dots, p_n]$  and output result
  }
  let  $[e'_{b_{t+1}}, \dots, e'_{b_n}] \leftarrow \text{msg}$ , and let  $\text{sk} \leftarrow \text{PKE.Dec}(\text{pk}_{p_i}, e'_{b_i})$ 
  let  $[b_1, \dots, b_n] \leftarrow [\text{PKE.Dec}(\text{sk}, e_{b_1}), \dots, \text{PKE.Dec}(\text{sk}, e_{b_n})]$ 
  output  $b_1 \oplus \dots \oplus b_n$ 
}

```

(5) Obtain Local Output

```

if  $i > t$  then skip
let  $\text{sk} \leftarrow \text{HW.Run}(e, (\text{"revealKey2"}))$ 
let  $[b_1, \dots, b_n] \leftarrow [\text{PKE.Dec}(\text{sk}, e_{b_1}), \dots, \text{PKE.Dec}(\text{sk}, e_{b_n})]$ 
output  $b_1 \oplus \dots \oplus b_n$ 

```

FIG. 17