

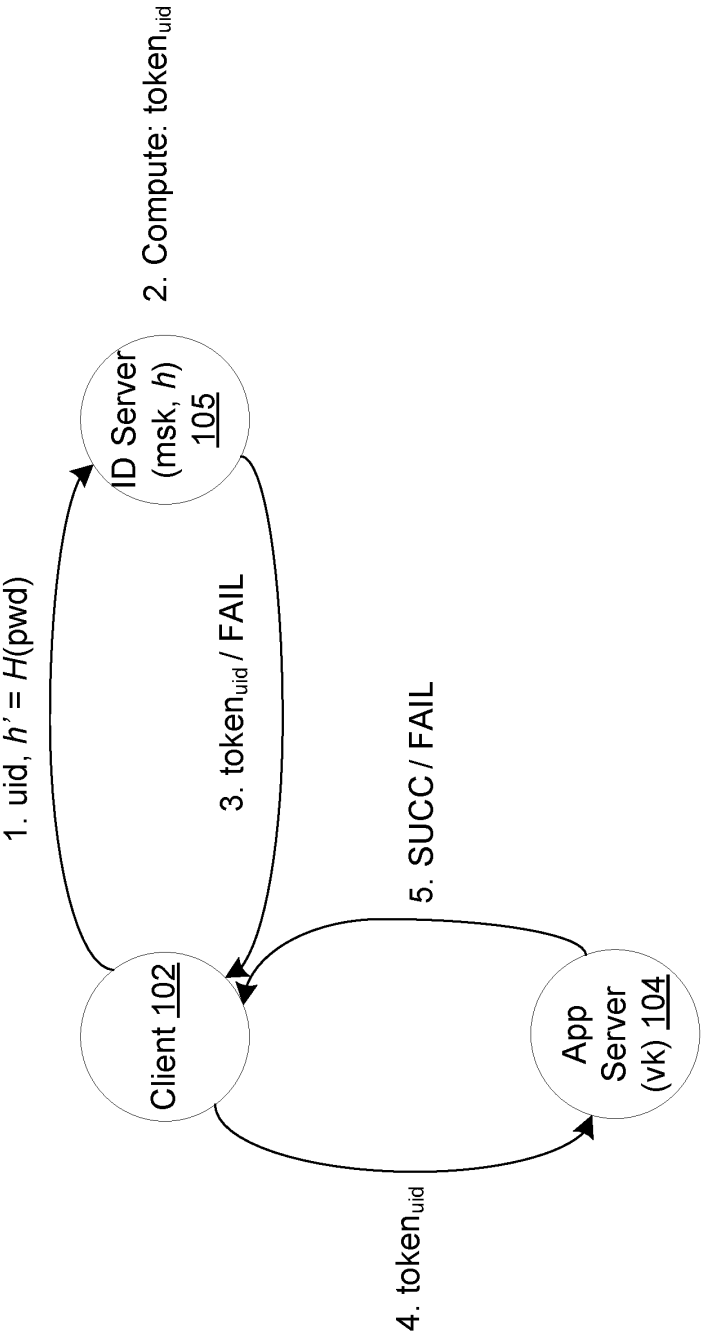


FIG. 1

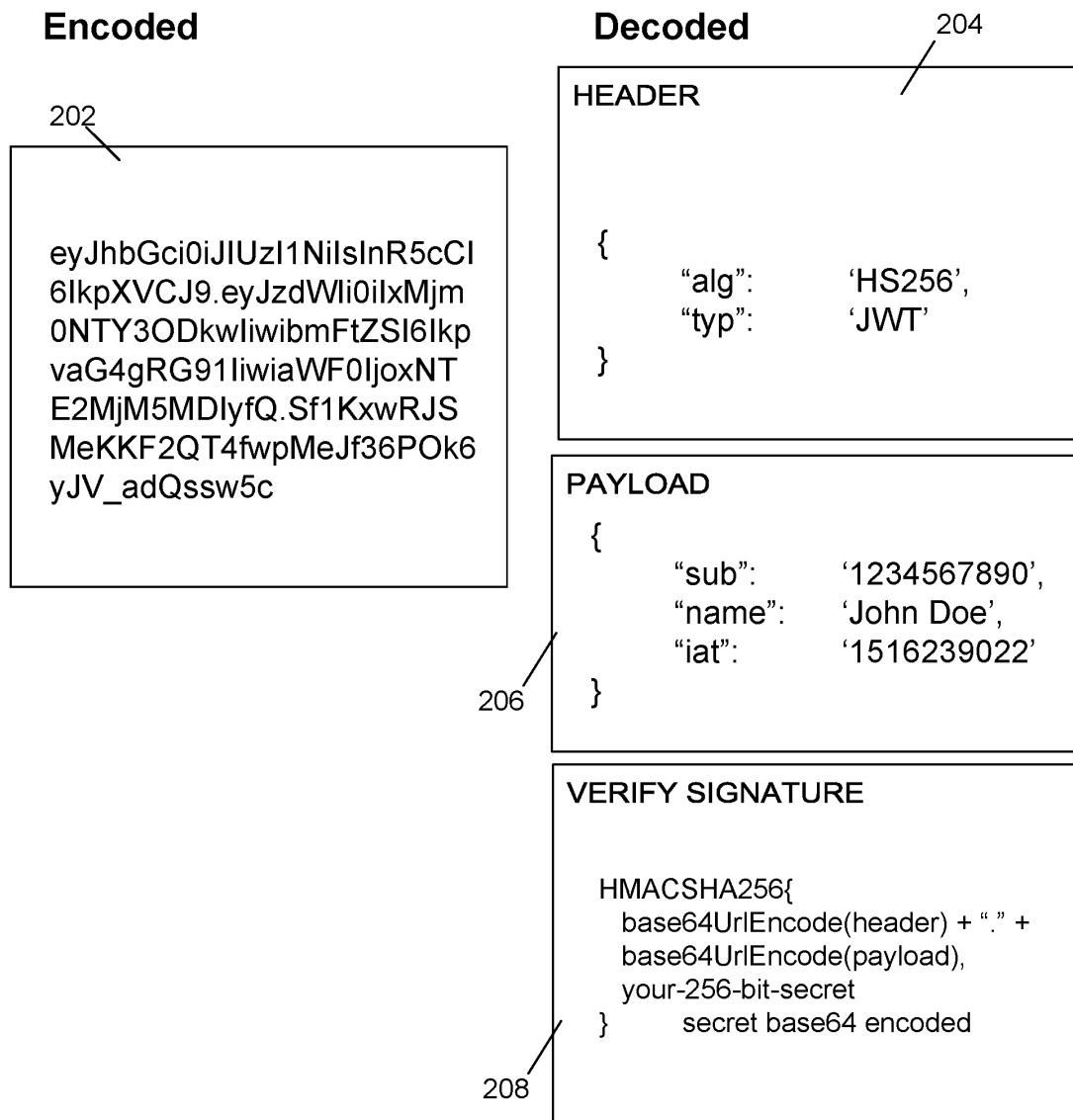
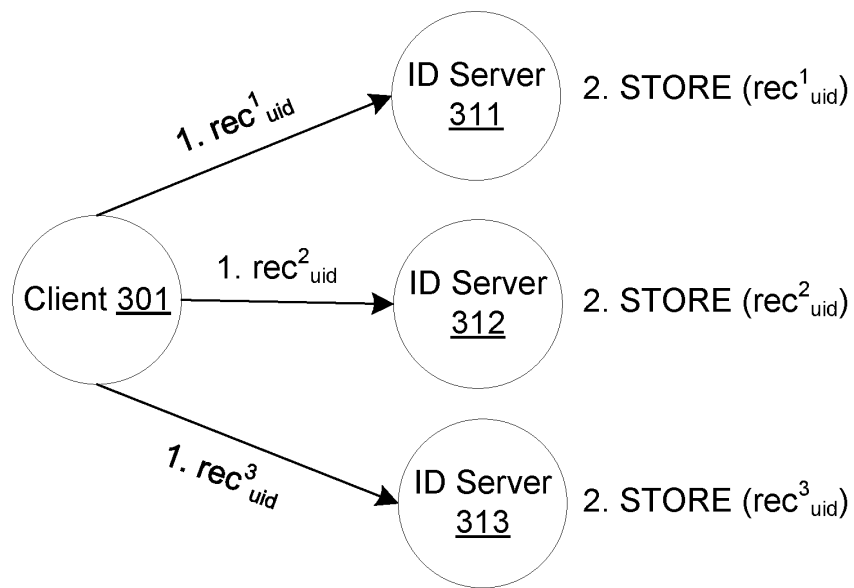
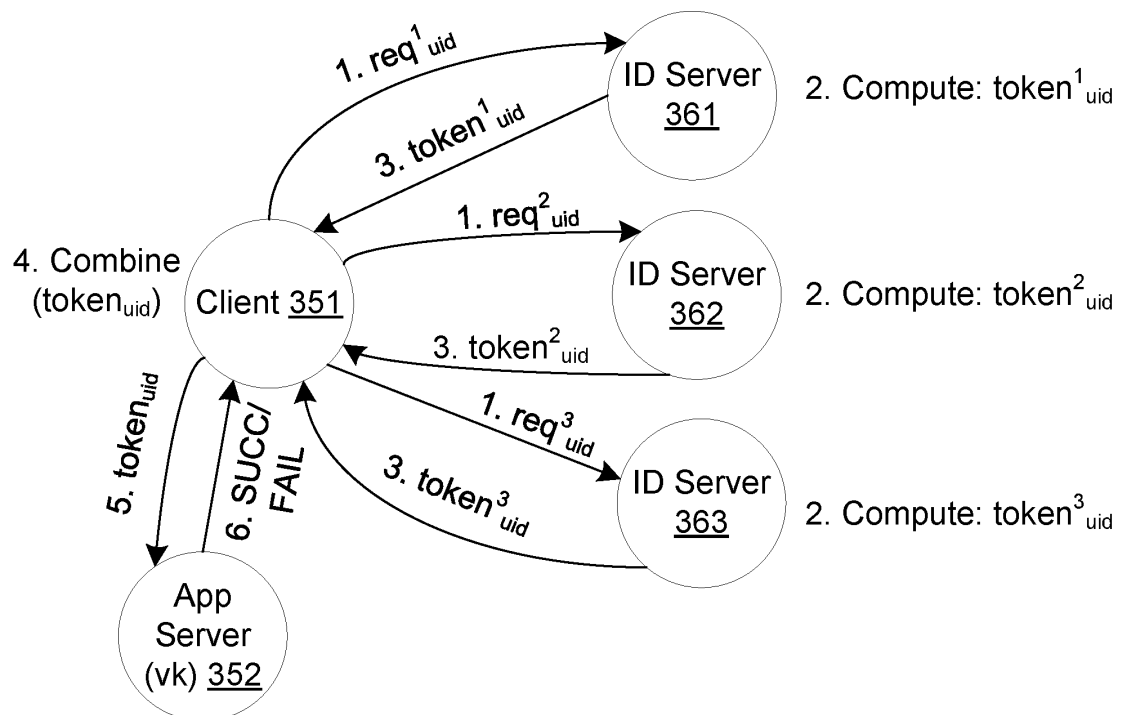
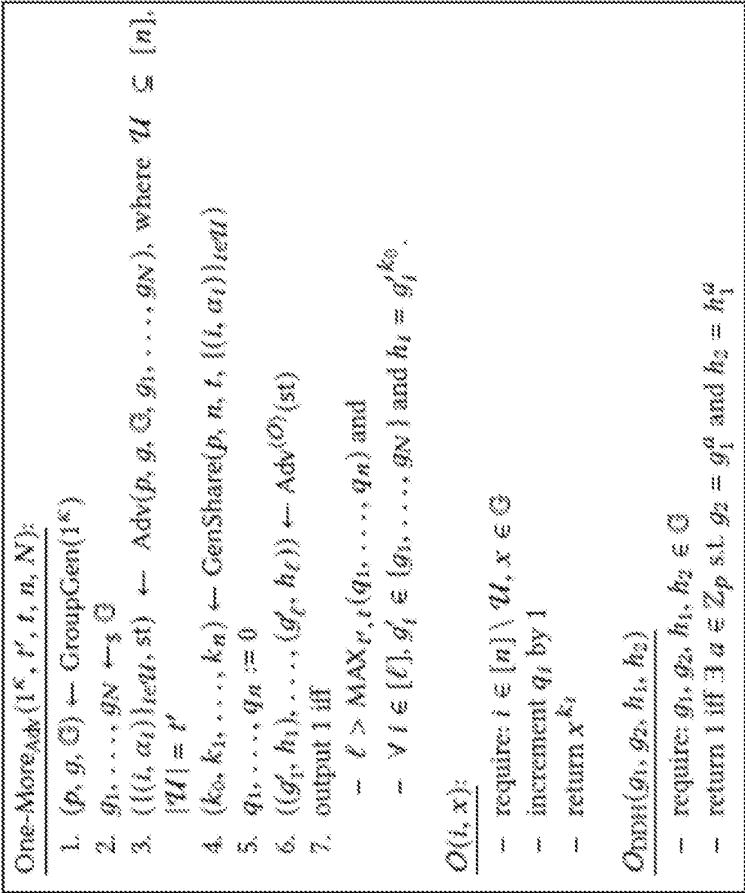


FIG. 2

3/18

**FIG. 3A****FIG. 3B**



```

UnpredictabilityTOP,Adv(1κ, n, t):
- ( $\llbracket \text{sk} \rrbracket$ , pp)  $\leftarrow$  Setup(1κ, n, t)
-  $\mathcal{U} \leftarrow \text{Adv}(\text{pp})$ 
-  $\hat{x} \leftarrow_{\mathcal{S}} \mathcal{X}$ 
-  $q_1, \dots, q_n := 0$ 
-  $h^* \leftarrow \text{Adv}^{(O)}(\{\text{sk}_i\}_{i \in \mathcal{U}})$ 
- output 1 iff TOP(sk,  $\hat{x}$ ) =  $h^*$ 

Oenc&eval():
-  $c := \text{Encode}(\hat{x}, \rho)$  for  $\rho \leftarrow_{\mathcal{S}} \mathcal{R}$ 
- for  $i \in [n] \setminus \mathcal{U}$ ,  $z_i \leftarrow \text{Eval}(\text{sk}_i, c)$ 
- return  $c, \{z_i\}_{i \in [n]}$ 

Oeval(i, c):
- require:  $i \in [n] \setminus \mathcal{U}$ 
- increment  $q_i$  by 1
- return Eval(ski, c)

Ocheck(h):
- return 1 if  $h = \text{TOP}(\text{sk}, \hat{x})$ ; else return 0

```

FIG. 5

Obliviousness_{TOP, Adv}($1^k, n, t$):

- $(\llbracket \text{sk} \rrbracket, \text{pp}) \leftarrow \text{Setup}(1^k, n, t)$
- $\mathcal{U} \leftarrow \text{Adv}(\text{pp})$
- $\hat{x} \leftarrow_{\$} \mathcal{X}$
- $q_1, \dots, q_n \leftarrow 0$
- $x^* \leftarrow \text{Adv}^{(O)}(\{\text{sk}_i\}_{i \in \mathcal{U}})$
- output 1 iff $x^* = \hat{x}$

$O_{\text{enc&eval}}()$:

- $c := \text{Encode}(\hat{x}, \rho)$ for $\rho \leftarrow_{\$} \mathcal{R}$
- for $i \in [n]$, $z_i \leftarrow \text{Eval}(\text{sk}_i, c)$
- $h := \text{Combine}(\hat{x}, \{(i, z_i)\}_{i \in [n]}, \rho)$
- return $c, \{z_i\}_{i \in [n] \setminus \mathcal{U}}, h$

$O_{\text{eval}}(i, c)$:

- require: $i \in [n] \setminus \mathcal{U}$
- increment q_i by 1
- return $\text{Eval}(\text{sk}_i, c)$

FIG. 6

SecGame_{SI,Adv}($1^K, n, t, P$):

- $(\{sk_i\}, vk, pp, (SK_1, \dots, SK_n)) \leftarrow \text{GlobalSetup}(1^K, n, t, P)$
- $(\mathcal{U}, C^*, st_{adv}) \leftarrow \text{Adv}(pp)$ *$\mathcal{U}$: corrupt servers, C^* : targeted client*
- $\mathcal{V} := \emptyset$ *set of corrupt clients*
- $\text{PwdList} := \emptyset$ *list of (C, pwd) pairs, indexed by C*
- $\text{ReqList}_{C,i} := \emptyset$ for $i \in [n]$ *token requests C makes to S_i*
- $ct := 0, \text{LiveSessions} := \{\}$ *LiveSessions is indexed by ct*
- $\text{TokList} := \emptyset$ *list of tokens generated through O_{final}*
- $Q_{C,i} := 0$ for all C and $i \in [n]$
- $Q_{C,x} := 0$ for all C and x
- $\text{out} \leftarrow \text{Adv}^{O_{\text{SI}}}((sk_i)_{i \in \mathcal{U}}, (SK_i)_{i \in \mathcal{U}}, st_{adv})$

$O_{\text{corrupt}}(C)$.

- $\mathcal{V} := \mathcal{V} \cup \{C\}$
- if $(C, \star) \in \text{PwdList}$, return $\text{PwdList}(C)$

$O_{\text{signup}}(C)$.

- require: $\text{PwdList}(C) = \perp$
- $\text{pwd} \leftarrow_s P$
- add (C, pwd) to PwdList
- return $\text{SignUp}(C, \text{pwd})$

$O_{\text{req}}(C, x, \mathcal{T})$.

- require: $\text{PwdList}(C) \neq \perp$
- $(st, \{req_i\}_{i \in \mathcal{T}}) \leftarrow \text{Request}(C, \text{PwdList}(C))$
- $\text{LiveSessions}[ct] := st$
- add req_i to $\text{ReqList}_{C,i}$ for $i \in \mathcal{T}$
- increment ct by 1
- return $\{req_i\}_{i \in \mathcal{T}}$

$O_{\text{server}}(i, \text{inst}, m)$.

- require: $i \in [n] \setminus \mathcal{U}$
- if $\text{inst} = \text{store}$:
 - parse m as (C, msg_i)
 - $\text{rec}_{i,C} := \text{Store}(SK_i, C, \text{msg}_i)$
 - update rec_i with $(C, \text{rec}_{i,C})$ if no record for C exists
- else if $\text{inst} = \text{respond}$:
 - parse m as (C, x, req_i)
 - $\text{res}_i \leftarrow \text{Respond}(sk_i, \text{rec}_i, C, x, req_i)$
 - increment $Q_{C,x}$ by 1
 - if $req_i \notin \text{ReqList}_{C,i}$, increment $Q_{C,i}$ by 1
 - return res_i

$O_{\text{final}}(ct, \{res_i\}_{i \in S})$.

- $st := \text{LiveSessions}[ct]$
- $tk := \text{Finalize}(st, \{res_i\}_{i \in S})$
- add tk to TokList
- return tk

$O_{\text{verify}}(vk, C, x, tk)$.

- return $\text{Verify}(vk, C, x, tk)$

FIG. 7

Ingredients.

- A threshold authentication scheme $TA \equiv (TA.Setup, TA.PartEval, TA.Combine, TA.Verify)$.
- A threshold oblivious PRF $TOP \equiv (TOP.Setup, TOP.Encode, TOP.Eval, TOP.Combine)$.
- A CPA secure symmetric-key encryption scheme $SKE \equiv (SKE.Encrypt, SKE.Decrypt)$.
- An IND-CCA2 secure public-key encryption scheme $PKE \equiv (PKE.KeyGen, PKE.Encrypt, PKE.Decrypt)$.
- A hash function $\mathcal{H}$.

GlobalSetup($1^K, n, t, P$) $\rightarrow$ ($\llbracket sk \rrbracket, vk, pp, (SK_1, \dots, SK_n)$).

- Run $TA.Setup(1^K, n, t)$ to get $(\llbracket tsk \rrbracket, tpp)$.
- Run $PKE.KeyGen(1^K)$ n times to get $(PK_1, SK_1), \dots, (PK_n, SK_n)$.
- Set $sk_i \equiv tsk_i$ for all $i \in [n]$ and $pp \equiv (\kappa, n, t, P, \{PK_1, \dots, PK_n\}, tpp)$.

SignUp(C, pwd) $\rightarrow$ ($(C, msg_1), \dots, (C, msg_n)$).

- Run $TOP.Setup(1^K, n, t)$ to get $(\llbracket k \rrbracket, opp)$.
- Compute $h \equiv TOP(k, C \parallel pwd)$ and $h_i \equiv \mathcal{H}(h \parallel i)$ for $i \in [n]$.
- Set $msg_i \equiv PKE.Encrypt(PK_i, k_i \parallel h_i)$ for $i \in [n]$.

Store(SK_i, C, msg_i) $\equiv$ $rec_{i,C}$.

- Run $PKE.Decrypt(SK_i, msg_i)$ to get k_i, h_i .
- Set $rec_{i,C} \equiv (k_i, h_i)$.

Output $\perp$ if decryption fails.

Request($C, pwd, x, \mathcal{T}$) $\rightarrow$ ($\{(C, x, req_i)\}_{i \in \mathcal{T}}, st$).

- If $|\mathcal{T}| < t$, output $\perp$.
- Pick a ρ at random. Run $TOP.Encode(C \parallel pwd, \rho)$ to get c .
- Set $req_i \equiv c$ for all $i \in [n]$ and $st \equiv (C, pwd, \rho, \mathcal{T})$.

Respond(sk_i, anc_i, C, x, req_i) $\rightarrow$ res_i .

- If $rec_{i,C} \notin anc_i$, output $\perp$. Else, parse $rec_{i,C}$ as (k_i, h_i) .
- Run $TOP.Eval(k_i, req_i)$ to get z_i .
- Run $TA.PartEval(tsk_i, x)$ to get y_i .
- Set $res_i \equiv (z_i, SKE.Encrypt(h_i, y_i))$.

Finalize($st, \{res_i\}_{i \in S}$) $\rightarrow$ tk .

- Parse res_i as $(z_i, ctxt_i)$ and st as $(C, pwd, \rho, \mathcal{T})$.
- If $S \neq \mathcal{T}$, output $\perp$.
- Run $TOP.Combine(C \parallel pwd, \{(i, z_i)\}_{i \in \mathcal{T}}, \rho)$ to get h .
- For all $i \in \mathcal{T}$, compute $h_i \equiv \mathcal{H}(h \parallel i)$ and $y_i \equiv SKE.Decrypt(h_i, ctxt_i)$.
- Finally, set tk to be $TA.Combine(\{(i, y_i)\}_{i \in \mathcal{T}})$.

(If any of $TOP.Combine$, $SKE.Decrypt$ or $TA.Combine$ fail, $\perp$ is output.)

Verify(pp, sk, x, tk) $\rightarrow$ $\{0, 1\}$.

- Output $TA.Verify(tpp, tsk, x, tk)$.

FIG. 8

Simple Token Generation

Any t valid responses can recover the token.
Any $t - 1$ servers cannot recover the password (by offline attack) or forge a token.
In this example, $t = 3$.

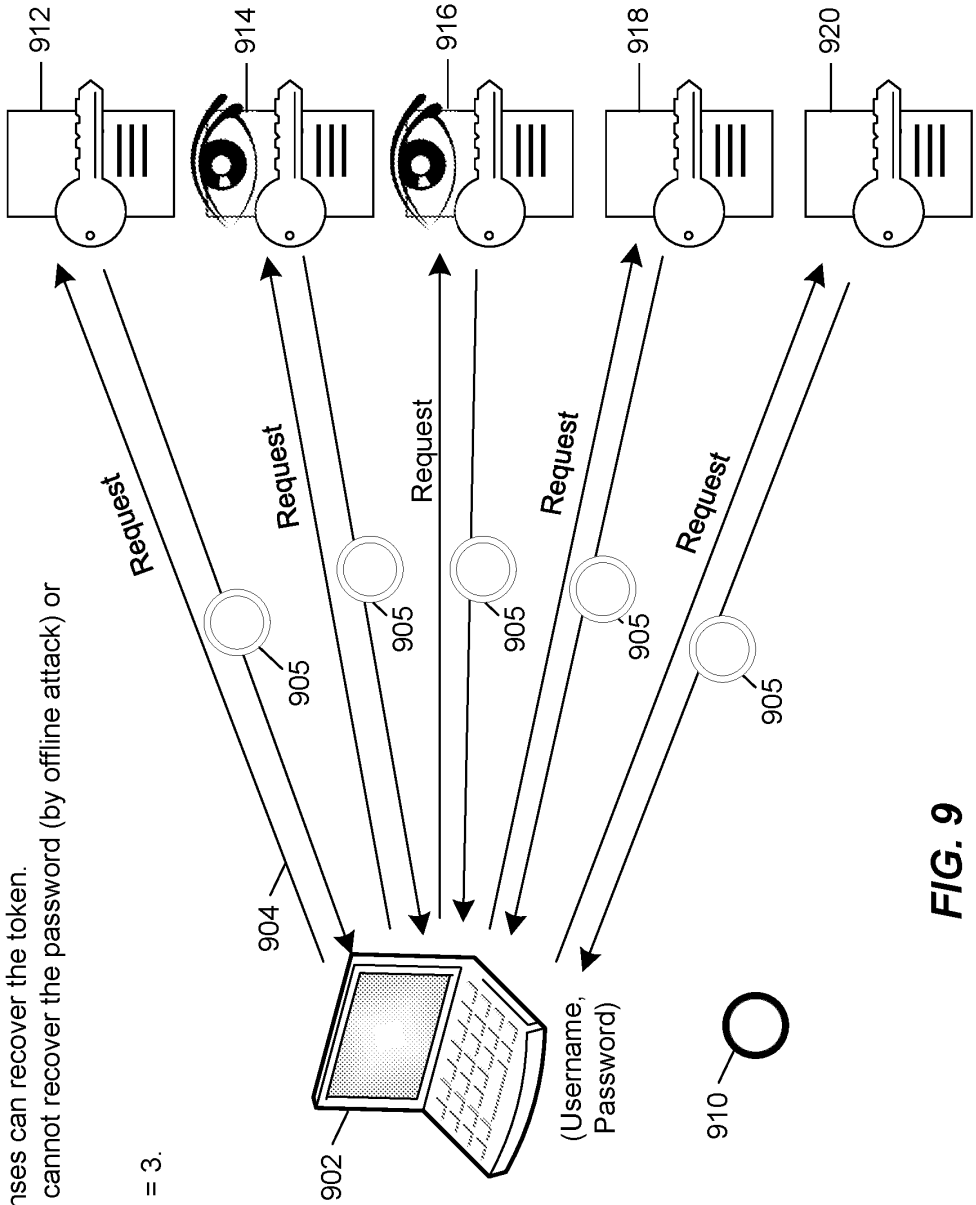


FIG. 9

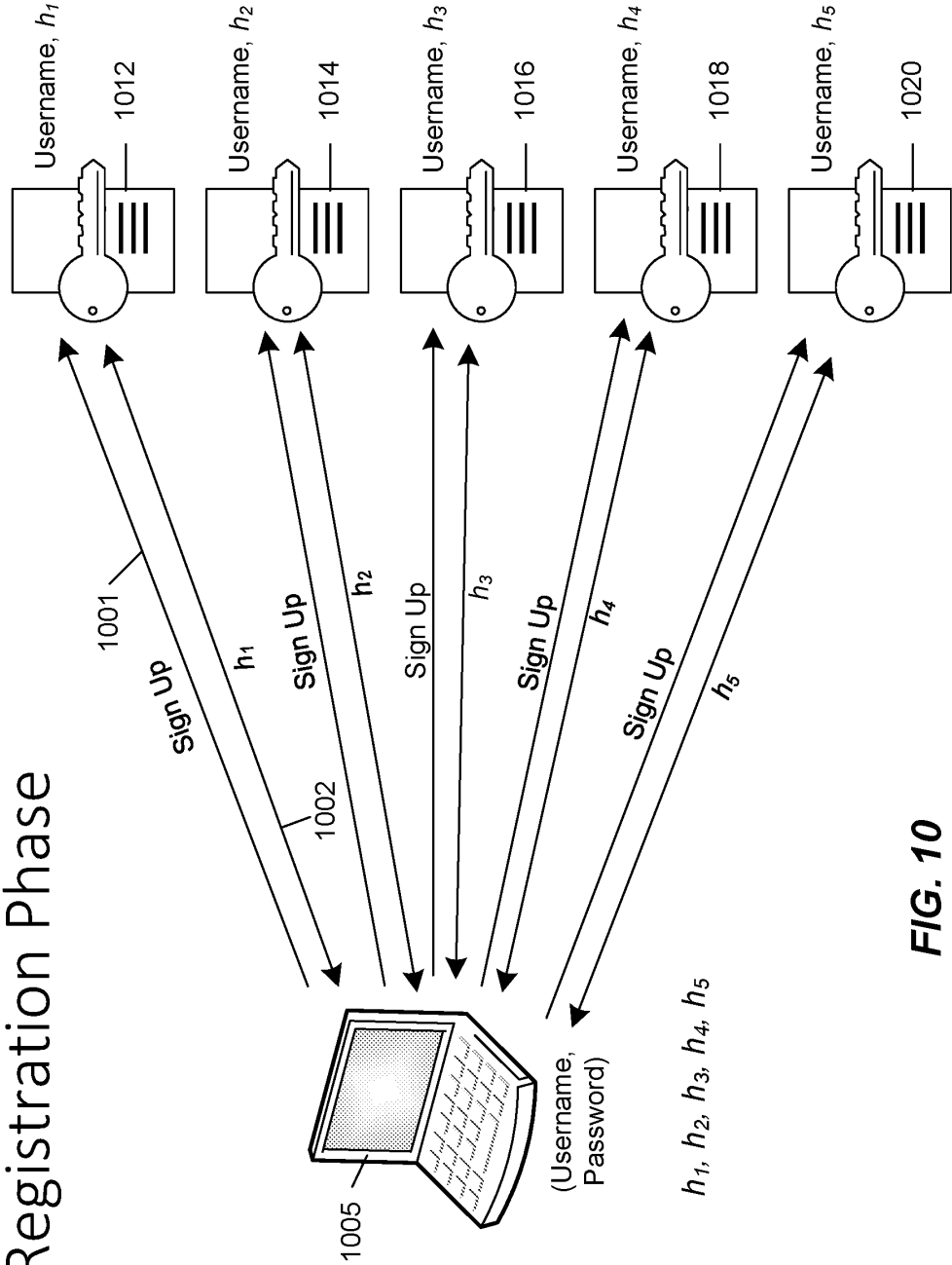


FIG. 10

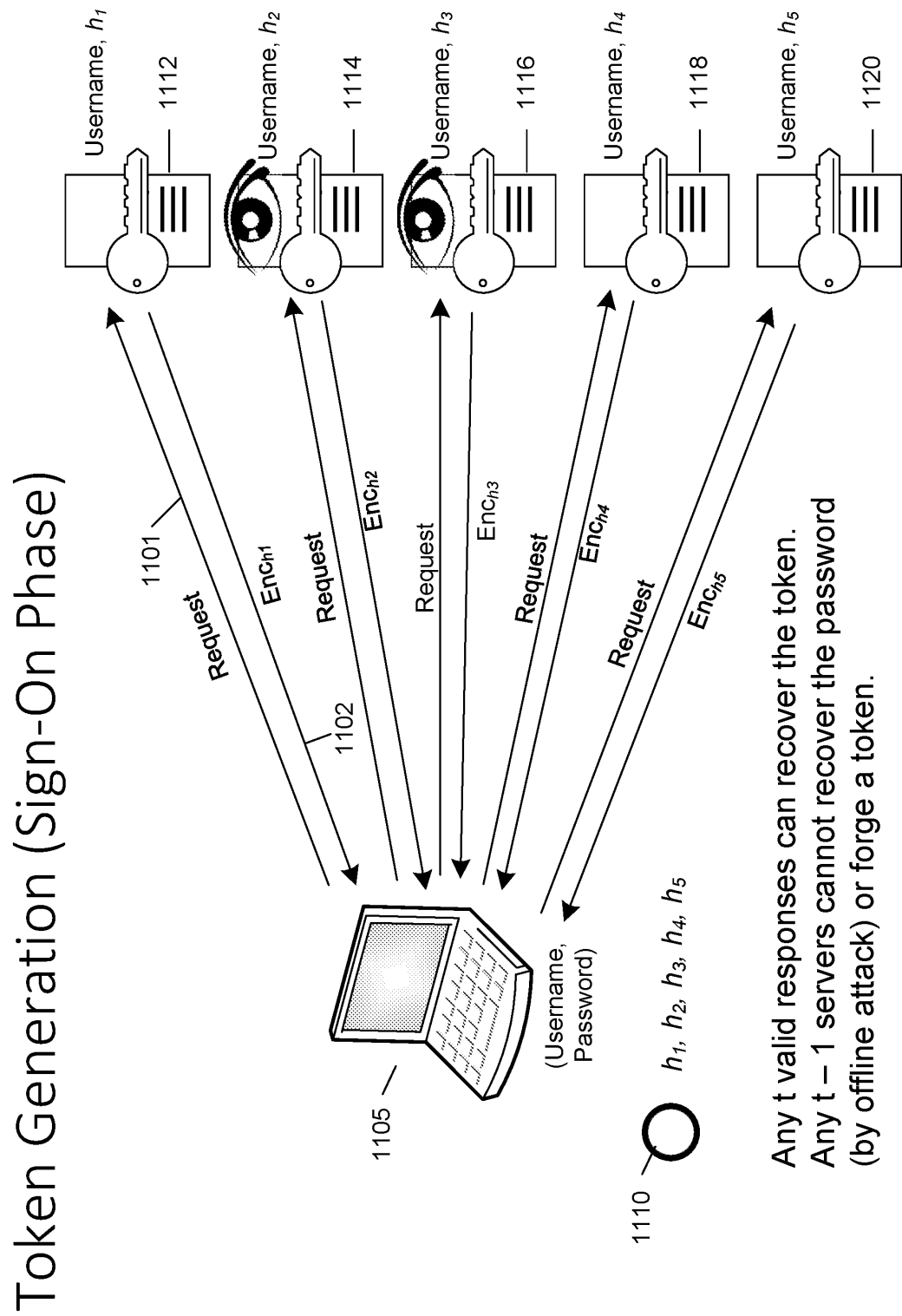


FIG. 11

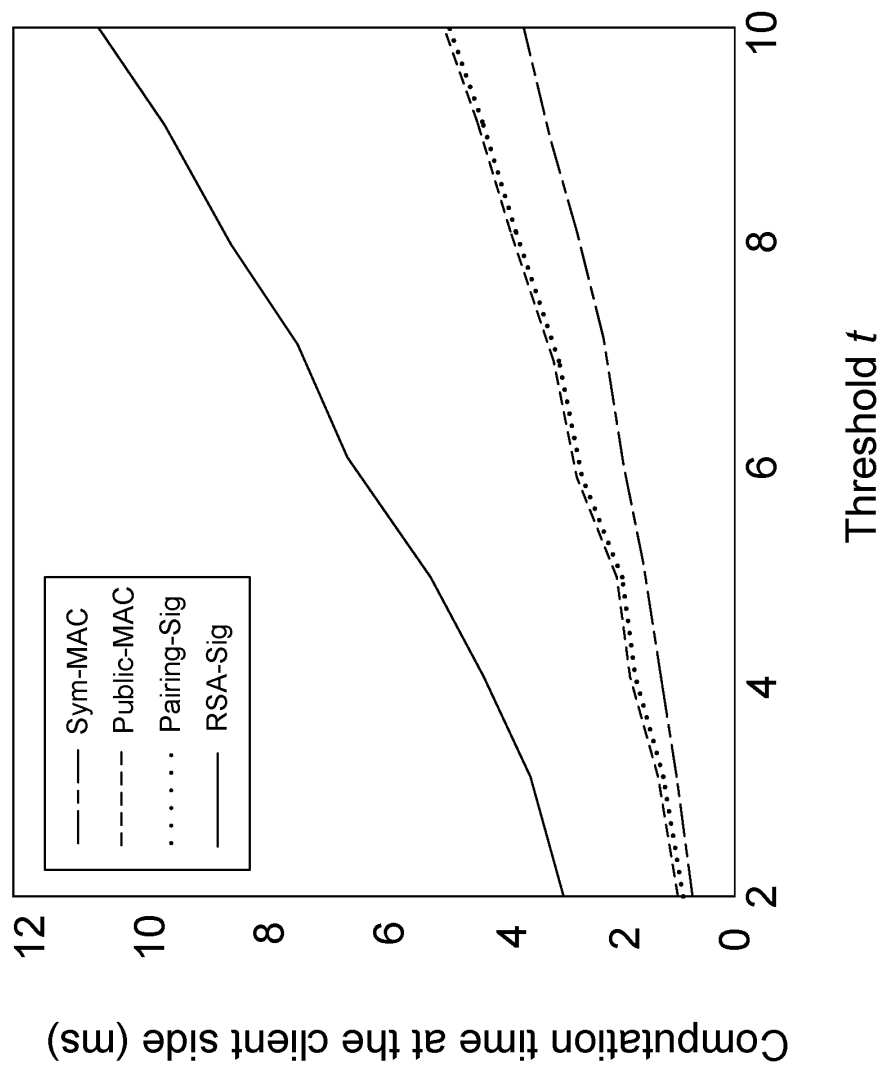


FIG. 12

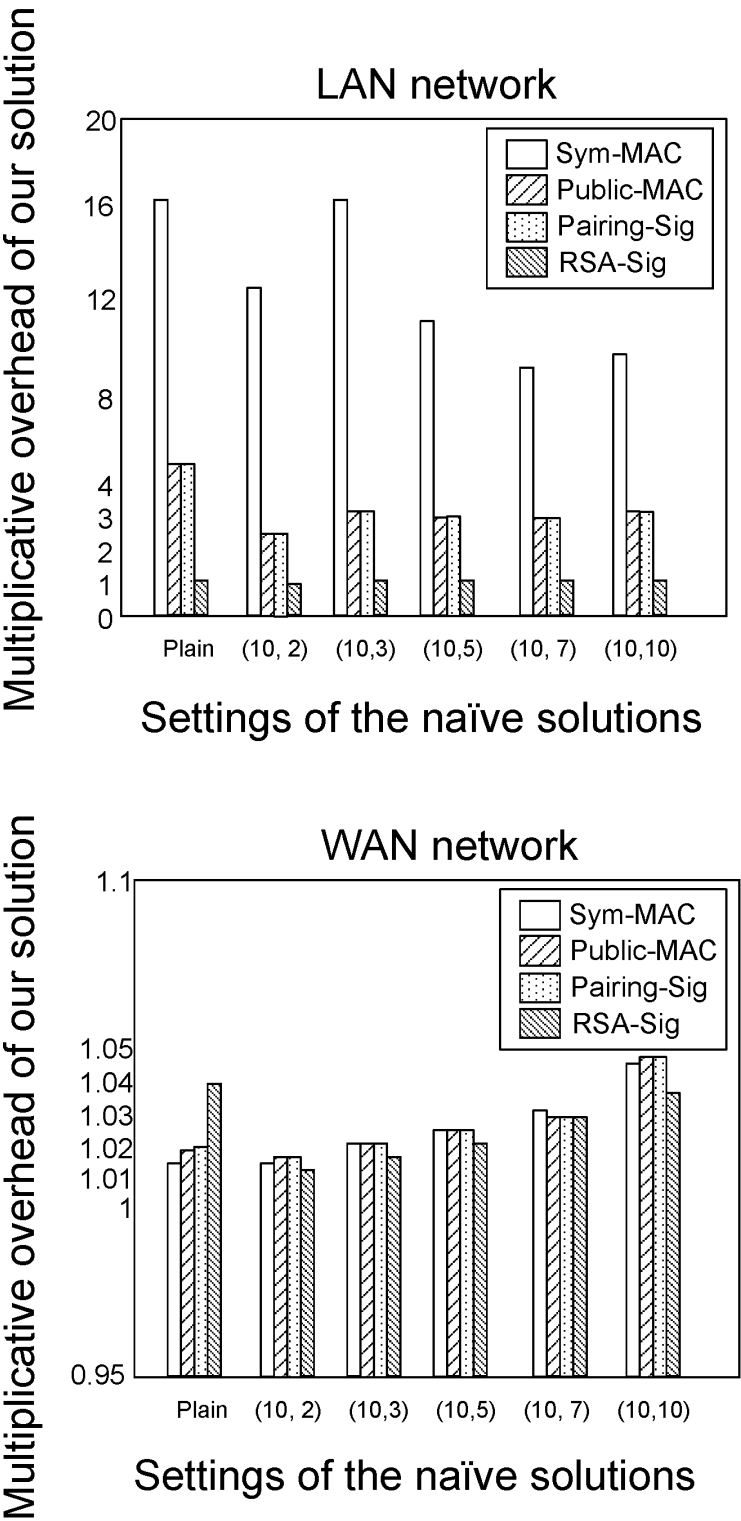


FIG. 13

Ingredients.

- An IND-CCA2 public-key encryption scheme $\text{PKE} \equiv (\text{PKE.KeyGen}, \text{PKE.Encrypt}, \text{PKE.Decrypt})$.

$\text{GlobalSetup}(1^\kappa, n, t, P) \rightarrow (\llbracket \text{sk} \rrbracket, \text{vk}, \text{pp}, (\text{SK}_1, \dots, \text{SK}_n))$.

- Run $\widetilde{\text{GlobalSetup}}(1^\kappa, n, t, P) \rightarrow (\llbracket \text{sk} \rrbracket, \text{vk}, \widetilde{\text{pp}})$.
- Run $\text{PKE.KeyGen}(1^\kappa)$ n times to get $(\text{PK}_1, \text{SK}_1), \dots, (\text{PK}_n, \text{SK}_n)$.
- Set $\text{pp} \equiv (\widetilde{\text{pp}}, \{\text{PK}_1, \dots, \text{PK}_n\})$.

$\text{SignUp}(C, \text{pwd}) \rightarrow ((C, \text{msg}_1), \dots, (C, \text{msg}_n))$.

- Run $\widetilde{\text{SignUp}}(C, \text{pwd}) \rightarrow (m_1, \dots, m_n)$.
- Set $\text{msg}_i \equiv \text{PKE.Encrypt}(\text{PK}_i, m_i)$.

$\text{Store}(\text{SK}_i, C, \text{msg}_i) \equiv: \text{rec}_{i,C}$.

- Run $\text{PKE.Decrypt}(\text{SK}_i, \text{msg}_i)$ to get m_i .
- Set $\text{rec}_{i,C} \equiv m_i$.

$\text{Request}(C, \text{pwd}, x, \mathcal{T}) \rightarrow (\{(C, x, \text{req}_i)\}_{i \in \mathcal{T}}, \text{st})$.

- Run $\widetilde{\text{Request}}(C, \text{pwd}, x, \mathcal{T}) \rightarrow (\{(C, x, \text{req}_i)\}_{i \in \mathcal{T}}, \text{st})$.

$\text{Respond}(\text{sk}_i, \text{REC}_i, C, x, \text{req}_i) \rightarrow \text{res}_i$.

- Run $\widetilde{\text{Respond}}(\text{sk}_i, \text{REC}_i, C, x, \text{req}_i) \rightarrow \text{res}_i$.

$\text{Finalize}(\text{st}, \{\text{res}_i\}_{i \in \mathcal{T}}) \rightarrow \text{tk}$.

- Run $\widetilde{\text{Finalize}}(\text{st}, \{\text{res}_i\}_{i \in \mathcal{T}}) \rightarrow \text{tk}$.

$\text{Verify}(\text{pp}, \text{sk}, x, \text{tk}) \rightarrow \{0, 1\}$.

- Output $\widetilde{\text{Verify}}(\widetilde{\text{pp}}, \text{sk}, x, \text{tk})$.

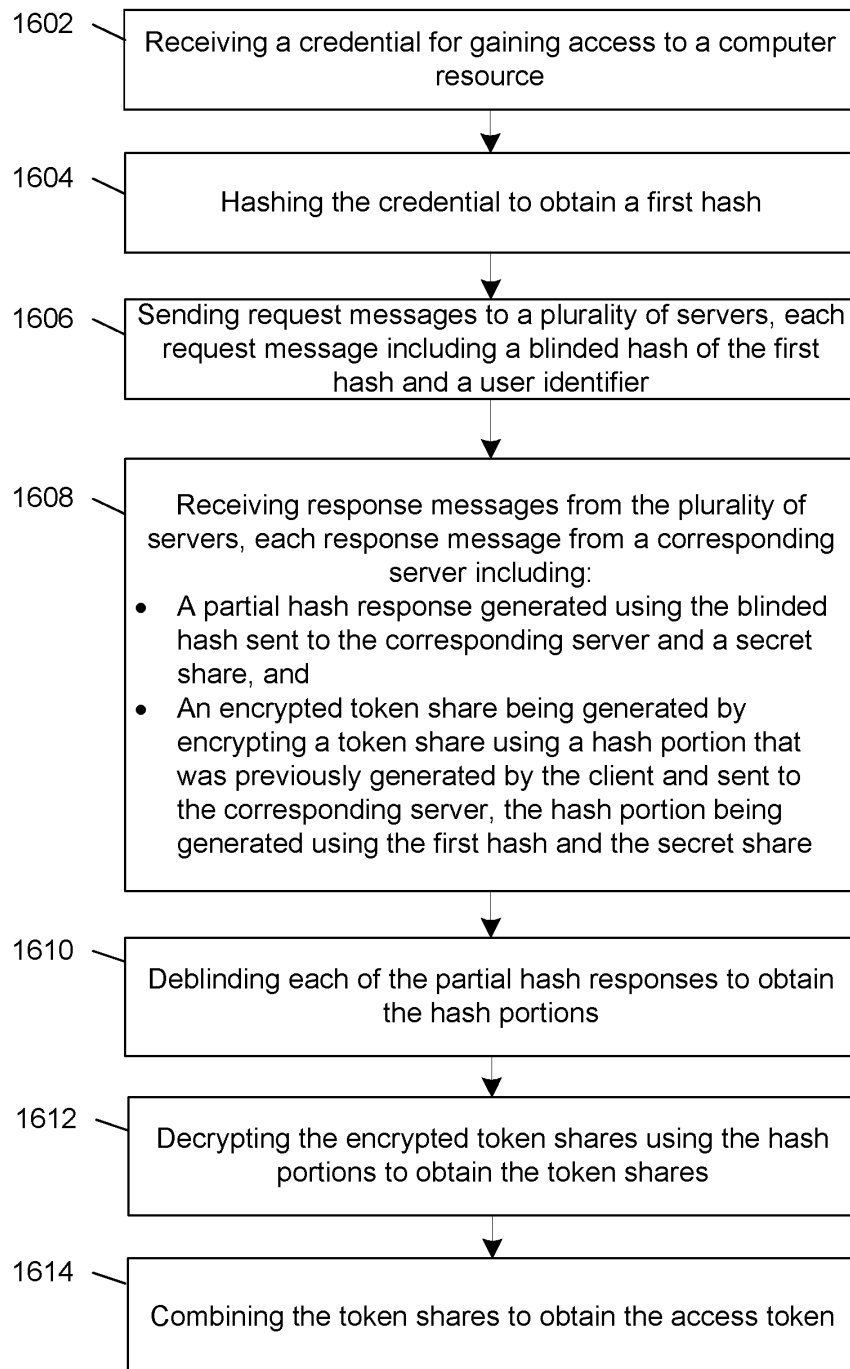
FIG. 14

Let the password space be $P := \{\text{pwd}\}$, set $n := 2, t := 2$, let $\mathcal{T}$ be the set of all servers in Π_{sym} , and set $x := 0$.

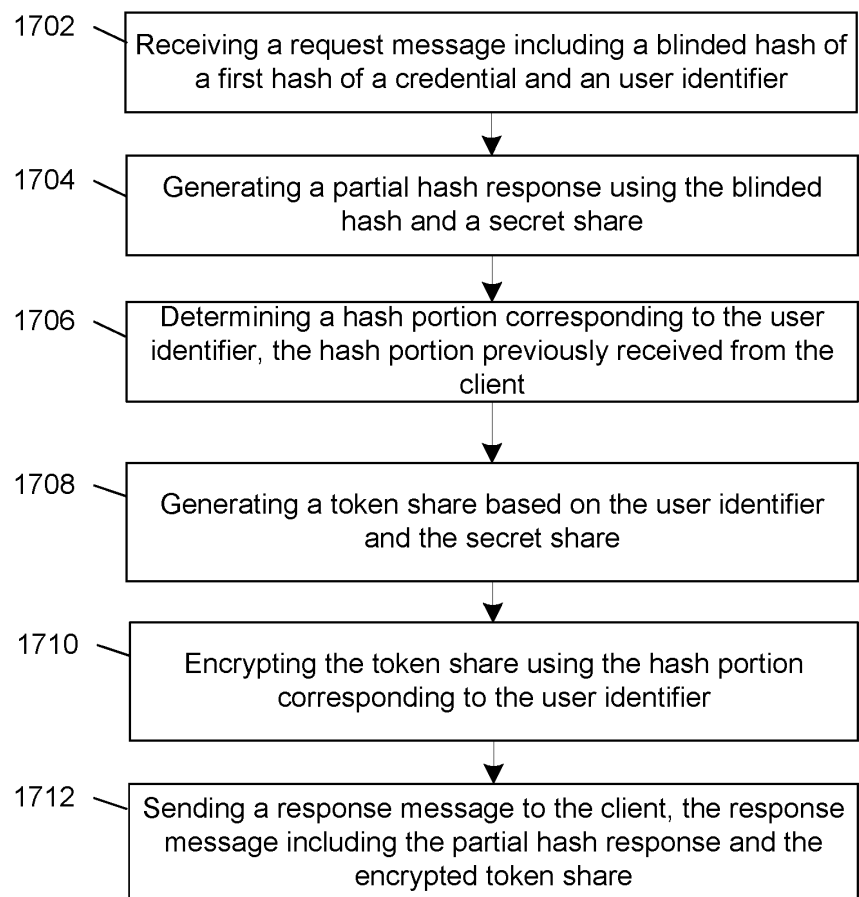
1. P_2 executes the following:
 - Run $\text{GlobalSetup}(1^\kappa, n, t, P) \rightarrow (\llbracket \text{sk} \rrbracket, \text{vk}, \widetilde{\text{pp}})$.
 - Run $\text{SignUp}(C, \text{pwd}) \rightarrow (m_1, \dots, m_n)$.
 - Set $\text{rec}_{i,C} := m_i$.
 - Send $(\widetilde{\text{pp}}, C)$ to P_1 .
2. On receiving the first message from P_2 , P_1 does the following:
 - Run $\text{Request}(C, \text{pwd}, x, \mathcal{T}) \rightarrow (\{(C, x, \text{req}_i)\}_{i \in \mathcal{T}}, \text{st})$.
 - Send $\{\text{req}_i\}_{i \in \mathcal{T}}$ to P_2 .
3. On receiving the message from P_1 , P_2 does the following:
 - Run $\text{Respond}(\text{sk}_i, \text{REC}_i, C, x, \text{req}_i) \rightarrow \text{res}_i$ for $i \in \mathcal{T}$.
 - Send $\{\text{res}_i\}_{i \in \mathcal{T}}$ to P_1 .
 - Emulate the protocol:
 - a. $\text{Request}(C, \text{pwd}, x, \mathcal{T}) \rightarrow (\{(C, x, \text{req}'_i)\}_{i \in \mathcal{T}}, \text{st}')$.
 - b. $\text{Respond}(\text{sk}_i, \text{REC}_i, C, x, \text{req}'_i) \rightarrow \text{res}'_i$ for $i \in \mathcal{T}$.
 - c. $\text{Finalize}(\text{st}', \{\text{res}'_i\}_{i \in \mathcal{T}}) \rightarrow \text{tk}'$.
 - Output tk' .
4. On receiving the second message from P_2 , P_1 executes the following:
 - Run $\text{Finalize}(\text{st}, \{\text{res}_i\}_{i \in \mathcal{T}}) \rightarrow \text{tk}$.
 - Output tk .

FIG. 15

16/18

**FIG. 16**

17/18

**FIG. 17**

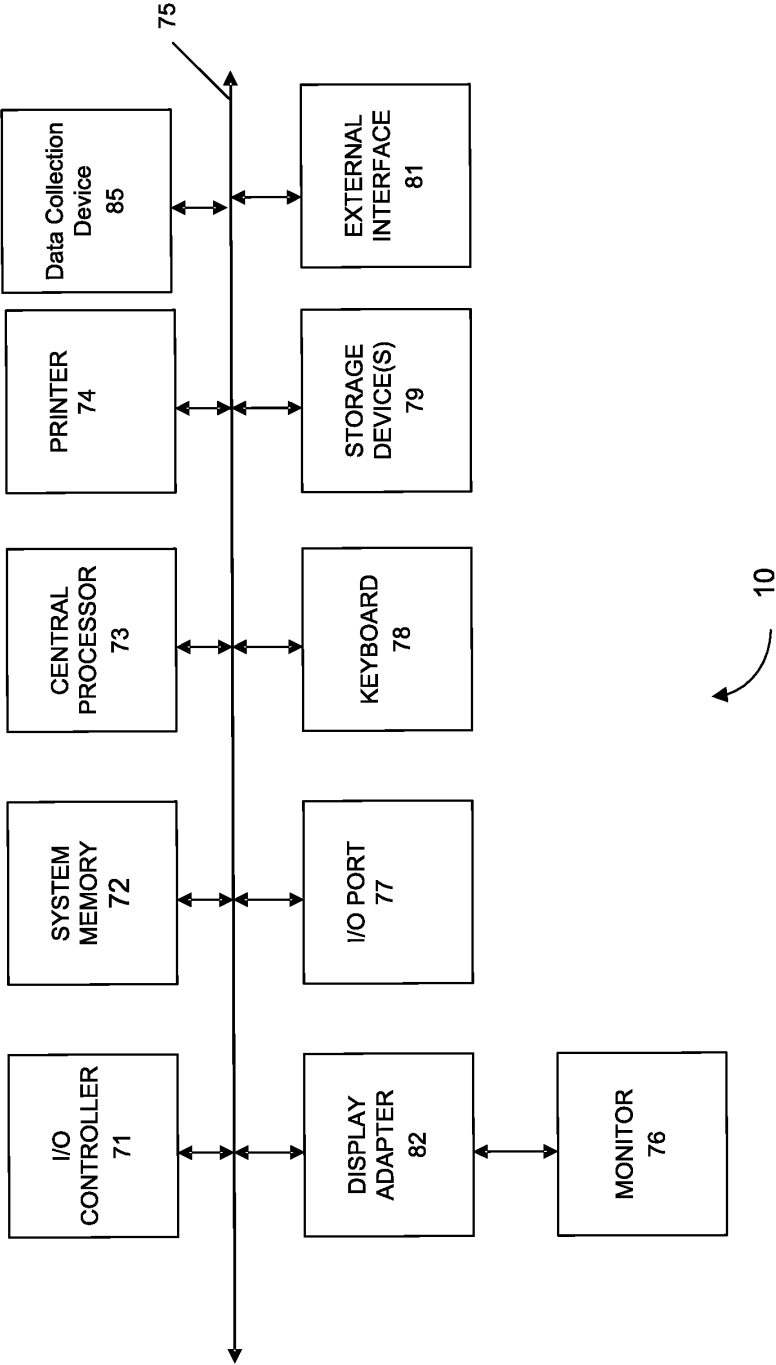


FIG. 18