

1/8

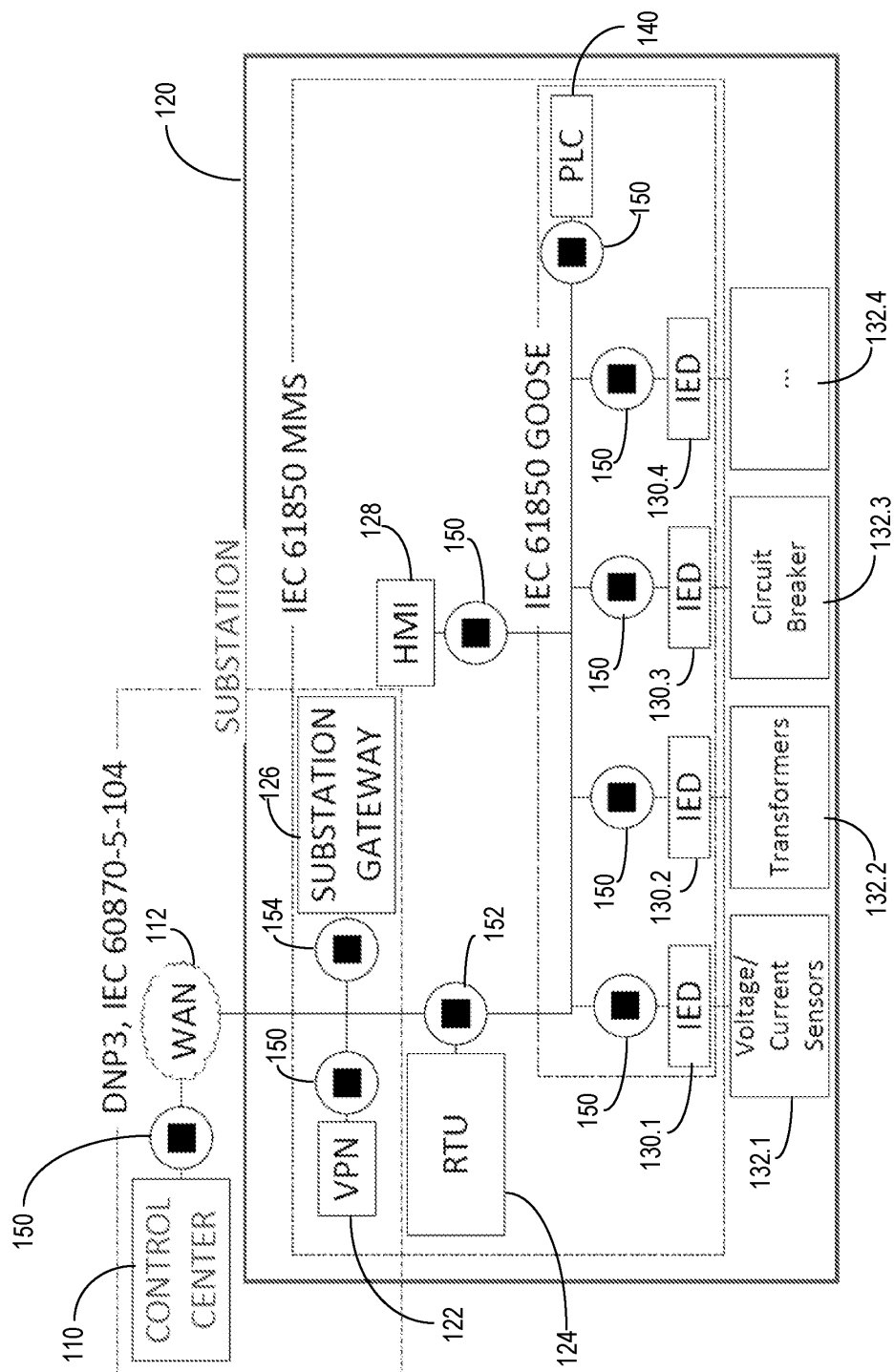


Figure 1

2/8

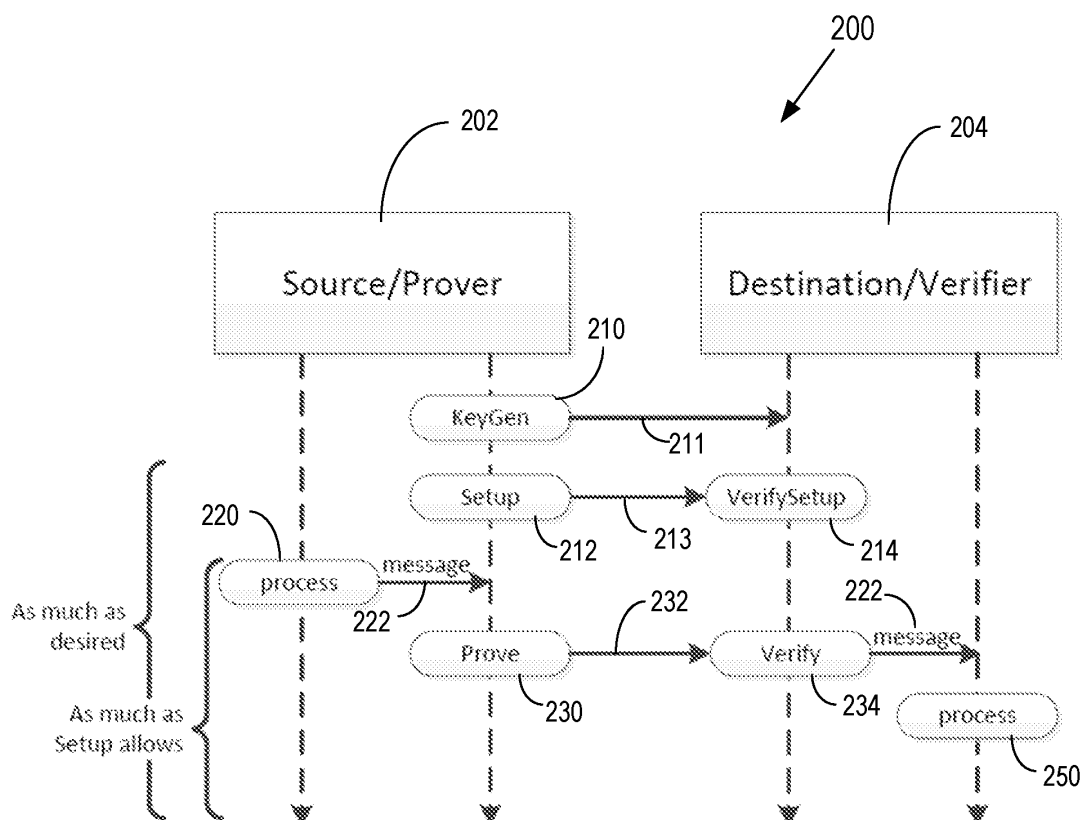


Figure 2

3/8

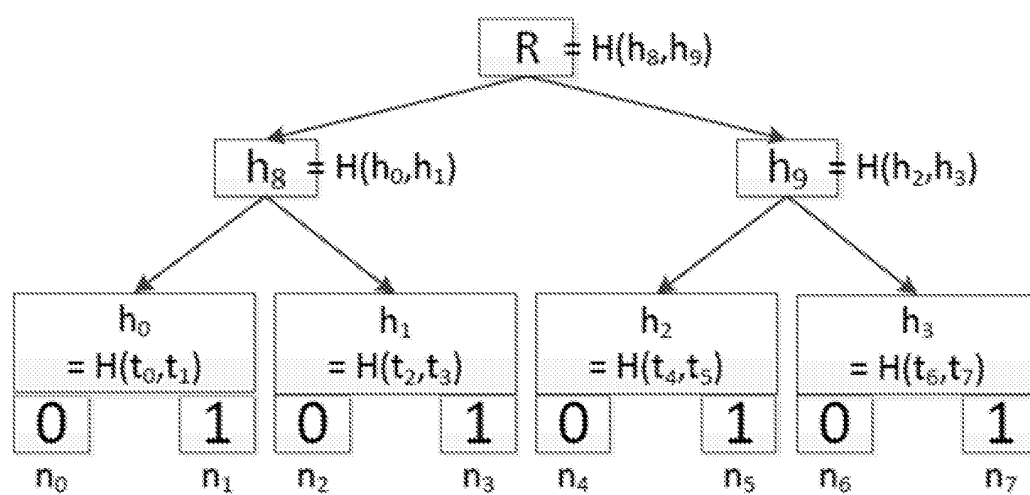


Figure 3

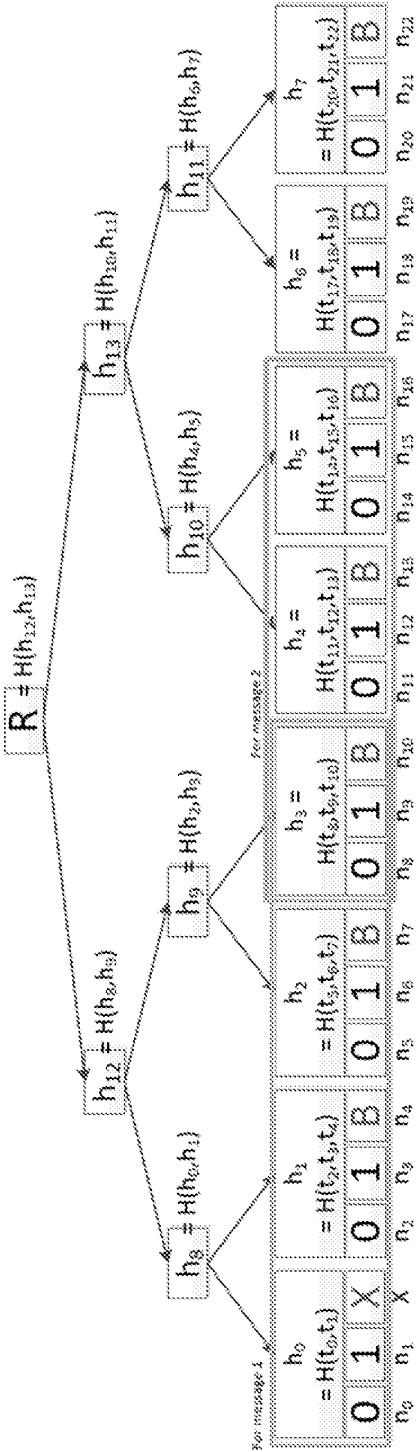


Figure 4

5/8

```

input : Leaf level length  $length$ 
output: A tri-leaf tree  $T$ 
1  $levelSize \leftarrow length$ 
2 for  $i = 1$  to  $levelSize$  do
3    $concat \leftarrow []$ 
4   for  $j = 0$  to 3 do
5      $T[0][i].items[j] \leftarrow random()$ 
6      $T[0][i].hashedItems[j] \leftarrow hash(T[0][i].items[j])$ 
7      $concat \leftarrow concat \parallel T[0][i].hashedItems[j]$ 
8    $T[0][i] \leftarrow hash(concat)$ 
9
10  $levelSize \leftarrow levelSize \gg 1$ 
11  $i \leftarrow 1$ 
12 while  $levelSize > 0$  do
13    $j \leftarrow 0$ 
14   while  $levelSize > j$  do
15      $T[i+1][j \gg 1] \leftarrow hash(T[i][j] \parallel T[i][j+1])$ 
16      $j \leftarrow j + 2$ 
17    $i \leftarrow i + 1$ 
18    $levelSize \leftarrow levelSize \gg 1$ 

```

Algorithm 1: Tri-leaf Tree Build Algorithm

Figure 5

6/8

```

    input  : A tri-leaf tree  $T$ , message  $M$ 
    output: The proof  $P$ 
1   $P \leftarrow [T.offset]$ 
2   $pl \leftarrow T.offset$ 
3   $pr \leftarrow T.offset + M.length - 1$ 
4
5  for  $i \leftarrow 0$  to  $M.length - 1$  do
6     $P.append(\text{extractNonce}(M[i], T[0][T.offset + i]))$ 
7
8  for level  $\leftarrow 0$  to  $T.height - 1$  do
9    if  $pl \wedge 1$  then
10      $P.append(T[level][pl \oplus 1])$ 
11
12    if  $(\neg pr) \wedge 1$  then
13      $P.append(T[level][pr \oplus 1])$ 
14
15     $pl \leftarrow pl \gg 1$ 
16     $pr \leftarrow pr \gg 1$ 
17
18   $T.offset \leftarrow T.offset + M.length - 1$ 
19  return  $P$ 

```

Algorithm 2: Batch Proving Algorithm

Figure 6

7/8

```

    input  : proof  $P$ , message  $M$ , root  $R$ 
    output: accept, reject
1  offset  $\leftarrow P[0]$ 
2   $p \leftarrow 1$ 
3  levelSize  $\leftarrow M.length$ 
4  buffer  $\leftarrow []$ 
5
6  for  $i \leftarrow 0$  to levelSize do
7      val0  $\leftarrow (M[i] == '0') ? \text{hash}(P[p]) : P[p]$ 
8      val1  $\leftarrow (M[i] == '1') ? \text{hash}(P[p + 1]) : P[p + 1]$ 
9      val2  $\leftarrow (M[i] == 'B') ? \text{hash}(P[p + 2]) : P[p + 2]$ 
10     buffer.append(val0||val1||val2)
11      $p \leftarrow p + 3$ 
12
13 while levelSize > 1 do
14     ind  $\leftarrow 0$ 
15     if (ind + offset)  $\wedge$  1 then
16         buffer[0]  $\leftarrow \text{hash}(P[p] || \text{buffer}[0])$ 
17          $p \leftarrow p + 1$ 
18         levelSize  $\leftarrow \text{levelSize} + 1$ 
19         ind  $\leftarrow 1$ 
20     while ind < levelSize do
21         buffer[ind  $\gg$  1]  $\leftarrow \text{hash}(\text{buffer[ind]} || \text{buffer[ind} + 1])$ 
22         ind  $\leftarrow \text{ind} + 2$ 
23     if ind == levelSize then
24         buffer[ind  $\gg$  1]  $\leftarrow \text{hash}(\text{buffer[ind]} || P[p])$ 
25          $p \leftarrow p + 1$ 
26         levelSize  $\leftarrow \text{levelSize} + 1$ 
27     offset  $\leftarrow \text{offset} \gg 1$ 
28     levelSize  $\leftarrow \text{levelSize} \gg 1$ 
29
30 if  $R == \text{buffer}[0]$  then
31     return accept
32 else
33     return reject

```

Algorithm 3: Batch Verification Algorithm

Figure 7

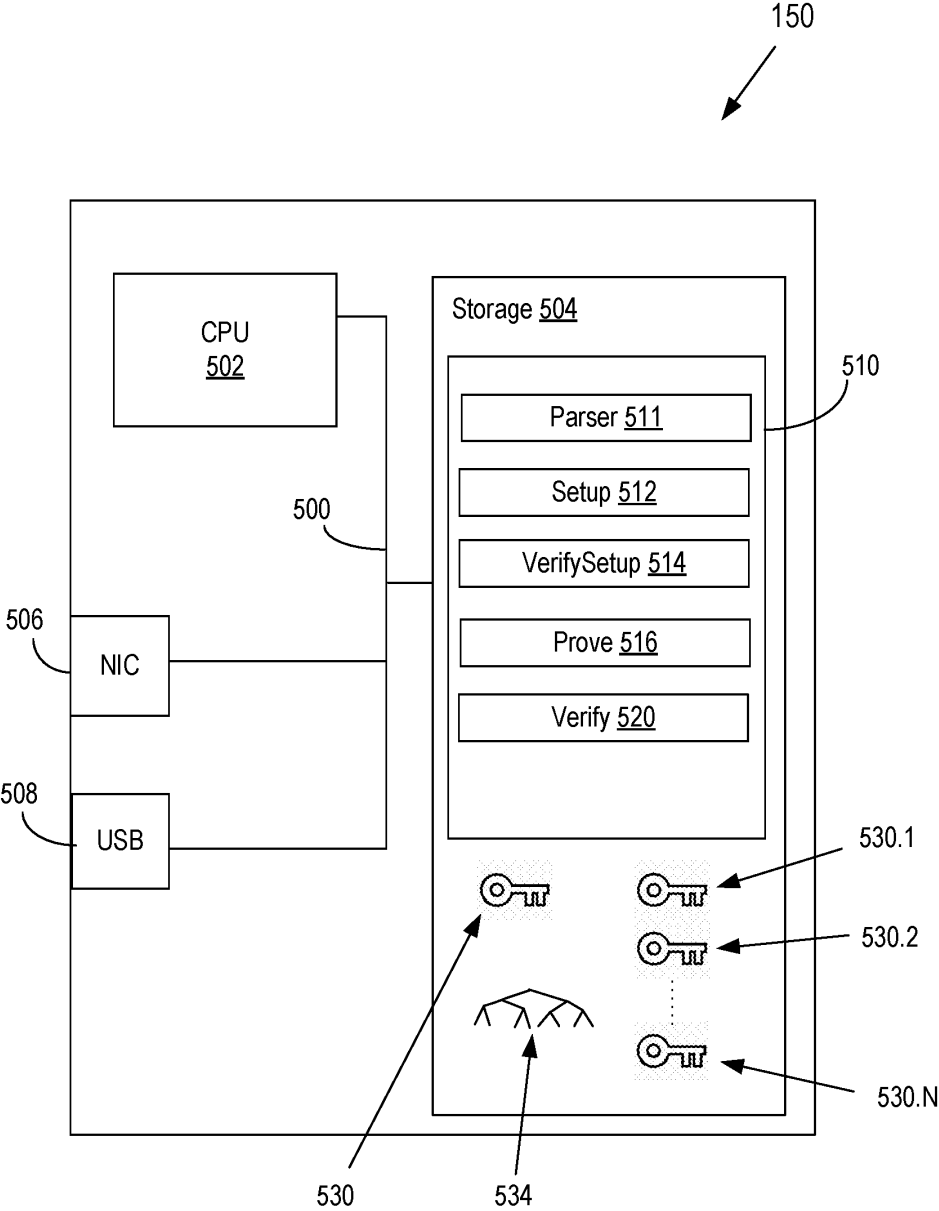


Figure 8