Wireless Sensor Network Security in ProFuN

Volkan Cambazoglu
Wireless Sensor Networks

1. Formal verification of a secure aggregation protocol

2. Trust establishment for secure communication in the demonstrator
Secure Hierarchical In-Network Aggregation in Sensor Networks

Spanning tree

Figure 1: Aggregation and naive commitment tree in network context

(a) Example network graph.
(b) Naive commitment tree, showing derivations of some of the vertices. For each sensor node G to the root.
Secure Hierarchical In-Network Aggregation in Sensor Networks

**Aggregation**

Label = \langle\text{count}, \text{value}, \text{complement}, \text{hash-value}\rangle

\[
R = \langle 12, v_R, \overline{v_R}, H[N||12||v_R||\overline{v_R}||H_0||A_1||I_0]\rangle
\]

\[
A_1 = \langle 9, v_{A_1}, \overline{v_{A_1}}, H[N||9||v_{A_1}||\overline{v_{A_1}}||A_0||B_1||C_1||D_0]\rangle
\]

\[
C_1 = \langle 4, v_{C_1}, \overline{v_{C_1}}, H[N||4||v_{C_1}||\overline{v_{C_1}}||C_0||E_0||F_1]\rangle
\]

\[
F_1 = \langle 2, v_{F_1}, \overline{v_{F_1}}, H[N||2||v_{F_1}||\overline{v_{F_1}}||F_0||G_0]\rangle
\]

\[
G_0 = \langle 1, a_G, r - a_G, G\rangle
\]
Secure Hierarchical In-Network Aggregation in Sensor Networks

Figure 3: Process of node aggregation in sensor networks.
Secure Hierarchical In-Network Aggregation in Sensor Networks

In the naive commitment tree, each sensor node always computes the result along with all the trees in the forest, which they then communicate to their parent sensor nodes. Each sensor node creates at most \( \log n \) trees in the forest. Hence, each sensor node keeps track of every vertex that it created, as well as the labels of the commitment subtrees along with the path vertex to the root of its tree is shaded grey. In Figure 1 is an example of the structure of the aggregation tree. The main observation is that, since the aggregation trees are a subgraph of the commitment tree vertices for each tree in the forest, they may be arbitrarily unbalanced. In such a commitment tree, instead of having \( n \) trees where \( n \) is the number of nodes, the commitment tree would consist of \( \log n \) trees where \( n \) is the total number of nodes and the diameter of the network, which in turn depends on the node density of hops) of the network. For an aggregation tree constructed with TaG, the vertex needs to receive all their labels to verify its contribution to the aggregation. In irregular topologies the diameter of the network may be higher, and the congestion cost to communicating a vertex could be a function of the diameter. In Figure 1(b) suppose that \( q \) is a vertex that is the parent of both the roots of the forests to form a new forest as follows.

Definition 5. Sensor node origins a single-vertex commitment forest, which they then communicate to their parent sensor nodes. Each sensor node included in the forest, up to a maximum of \( \log n \) count of commitment trees such that there is at most one commitment tree of any given height. The algorithm terminates in \( O(n \log n) \) steps since each step reduces the number of trees in the forest by one, and there are at most \( \log n \) trees in the forest. Hence, each sensor node creates at most \( \log n \) trees in the forest. Note that since all commitment trees are complete binary trees, tree merging is efficient. In the following discussion, we will for brevity make reference to "communicating a vertex" to another sensor node, or "communicating an aggregate" to another sensor node.
Secure Hierarchical In-Network Aggregation in Sensor Networks

Authentication

MAC_{K_u}(Nonce, OK)
The Goal

Formally verify that the security property of SHIA indeed holds

**Definition 1** A *direct data injection* attack occurs when an attacker modifies the data readings reported by the nodes under its direct control, under the constraint that only legal readings in $[0, r]$ are reported.

**Definition 2** An aggregation algorithm is *optimally secure* if, by tampering with the aggregation process, an adversary is unable to induce the querier to accept any aggregation result which is not already achievable by direct data injection.
Progress

1. Extract the algorithm from the paper
2. Take the algorithm to Psi-calculus specification
   - Focus on process communication
   - Abstract away from the details (helper functions and computations)
3. Define the terms, the conditions and the assertions
   - Revise several times to simplify
4. Write the rules for parsing and printing the specification
Some examples

Terms

```scala
val sgnSpecification = " Sorts
  " ch, tch, i, nonce, key, hash, mac, lbl, llist, dir
```

Conditions

```
  " LT : (i,i) => bool,
  " and : (bool,bool) => bool,
  " not : (bool) => bool,
  " iEq : (i,i) => bool,
  " dEq : (dir,dir) => bool
```

Other functions

```
  " XOR : (mac,mac) => mac,
  " Log2 : (i) => i,
  " Dec : (i) => i,
  " Add : (i,i) => i,
  " Sub : (i,i) => i,
  " Sort : (llist) => llist,
  " dLeft : () => dir,
  " dRight : () => dir,
```
Some examples

NodeVerify(chParent, chLeft, chRight, chFail, iMinVal, nonceQ, iKey, iLeftID, iRightID, iCountLeft, iValLeft, iCompLeft, hashLeft, iCountRight, iValRight, iCompRight, hashRight, iCountInHere, iCountOwn, iValOwn, iCompOwn, hashOwn) <= 
"Verify(chParent)"(iCountRoot, iValRoot, iCompRoot, hashRoot).
case "not(and(iEq(iLeftID, ∅), iEq(iRightID, ∅)))" :
  "Verify(chLeft)"<iCountRoot, iValRoot, iCompRoot, hashRoot>.
  "Verify(chRight)"<iCountRoot, iValRoot, iCompRoot, hashRoot>.
  "Offpath(chLeft)"<iCountRight, iValRight, iCompRight, hashRight, "dRight()">.
  "Offpath(chRight)"<iCountLeft, iValLeft, iCompLeft, hashLeft, "dLeft()">.
ForwardOffpathLabels<chParent, chLeft, chRight, nonceQ, iKey, iLeftID, iRightID, "Sub(Log2(iCountRoot), Log2(iCountInHere))">
[ ] "and(iEq(iLeftID, ∅), iEq(iRightID, ∅))": 
ReceiveOffpathLabels<chParent, chLeft, chRight, chFail, iMinVal, nonceQ, iKey, iLeftID, iRightID, iCountOwn, iValOwn, iCompOwn, hashOwn, iCountRoot, iValRoot, iCompRoot, hashRoot, "Log2(iCountRoot)", "LNil()">;
Next step

1. Implement abstracted details (helper functions and computations)

2. Implement constraint solver to handle the specification
   1. the properties that we need to check
      1. off-path labels
      2. boundaries
Encryption & Authentication

- **Node**: Zolertia Z1
- **OS**: Contiki
- **Chip**: CC2420 (2.4 GHz IEEE 802.15.4 Compliant and ZigBee™ Ready RF Transceiver)
- **AES-CCM** (Counter with CBC-MAC) 128 bits
  - Link layer software solution from Thingsquare Mist
Setting

1. Data Aggregation in a tree-based WSN
2. A node has to know at most 3 neighbours
   - Parent
   - Left child
   - Right child
3. Problem: Securely introduce a new node to the aggregation tree as a
   - Leaf (sensing) node
   - Aggregating node
Introduce a new node

1. Bring initialized node to the network. (known UID, net address and cryptographic keys)
2. Scan RFID/NFC tag with smartphone. (the tag has new node's UID)
3. Securely transmit the scanned value to the central system from the smartphone.
4. Central system validates the value and if it is valid, locate the associated network address in the network.
Introduce a new node

5. When the node receives message from the central system, it has instructions to update neighbour data.

6. The new node confirms that the neighbour data is applied

7. Central system sends update requests to affected neighbours

8. Central system collects replies from neighbour nodes that the update is done!
Introduce a new node

9. If successful, the role of the new node can be selected from the central system so that the necessary code is sent to the new node securely via the WSN.

10. Otherwise, the existing nodes neglect the new node.
Key Management

1. Base to node

2. Features:

• Backward secrecy - new member should not be able to decrypt old messages.

• Forward secrecy - old member should not be able to decrypt new messages.

• Group re-keying - group keys have to be re-arranged so that previous two features are supported.
DEMO!
Next step

1. Implementing µTesla in Contiki
2. Dynamic addition of a new node and/or re-location of an existing node
3. Different key management techniques
   - asymmetric
   - zero-knowledge