Privacy-preserving Robust Data Aggregation in Wireless Sensor Networks

Mauro Conti[⋆,1,2], Lei Zhang[2], Sankardas Roy[2], Roberto Di Pietro[3,4], Sushil Jajodia[2], and Luigi Vincenzo Mancini[1]

[1] Dipartimento di Informatica, Università di Roma "La Sapienza", 00198 - Roma, Italy, {conti, mancini}@di.uniroma1.it

[2] Center for Secure Information Systems, George Mason University, Fairfax, VA, USA - 22030, {mconti1, lzhang8, sroy1, jajodia}@gmu.edu

[3] Dipartimento di Matematica, Università di Roma "Tre", 00146 - Roma, Italy, dipietro@mat.uniroma3.it

[4] UNESCO Chair in Data Privacy, Universitat Rovira i Virgili - DEIM, E-43007 Tarragona, Catalonia-Spain, roberto.dipietro@urv.cat

Abstract. *In-network data aggregation* in Wireless Sensor Networks (WSNs) is a technique aimed at reducing the communication overhead—sensed data are combined into partial results at intermediate nodes during message routing. However, in the above technique, some sensor nodes need to send their individual sensed values to an aggregator node, empowered with the capability to decrypt the received data to perform a partial aggregation. This scenario raises privacy concerns in applications like personal health care and the military surveillance. A few other solutions exist where the data is not disclosed to the aggregator (e.g. using privacy homomorphism), but these solutions are not robust to node or communication failure.

The contributions of this paper are two-fold: First, we design a private data aggregation protocol that does not leak individual sensed values during the data aggregation process. In particular, neither the base station nor the other nodes are able to compromise the privacy of an individual node's sensed value. Second, the proposed protocol is robust to data-loss; if there is a node-failure or communication failure, the protocol is still able to compute the aggregate and to report to the base station the number of nodes that participated in the aggregation. To the best of our knowledge, our scheme is the first one that efficiently addresses the above issues all at once.

1 Introduction

Wireless Sensor Networks (WSNs) have proved to be useful in many applications [17–19], such as military surveillance and environmental monitoring. One operating model for this technology is as follows: After nodes are deployed in the area of interest, the data sensed by these nodes are routed to a Base Station (BS). To have these data delivered to the BS, nodes typically organize themselves into a multi-hop network. One possibility for the network manager to collect the sensed information is to allow each sensor node to propagate its sensed value to the BS. However, this approach is prohibitively wasteful due to excessive communication overhead. Indeed, a typical sensor node is severely constrained in communication bandwidth and energy reserve. Hence, network designers have advocated alternative approaches for data collection.

An *in-network* data aggregation algorithm combines partial results at intermediate nodes en route to the base station, reducing the communication overhead of an intermediate node. As a result, less energy is consumed on a node, increasing the lifetime of the network. To route data to the BS, typically, a spanning tree is constructed with the BS as the root [13, 22] and then aggregation is performed along the tree via an in-network data aggregation algorithm. Partial results propagate level by level up the tree, with each intermediate node awaiting messages from all of its children (or till when a time-out expires), before sending a new partial result to its parent. Several energy-efficient in-network algorithms [13, 22] are proposed in the literature to compute aggregates such as Count (i.e., the number of sensor nodes present in the network), Sum (i.e., the summation of the readings of all of the nodes), Average, etc.

Inspired by the low-cost, flexibility and ubiquitousness of this technology, researchers [16] are

⋆ Corresponding Author

envisioning more sophisticated applications of WSNs including sensors being installed in personal environment, such as houses, and human body. However, one issue raised by this range of applications is the privacy of the data being collected. In [16], a future application is cited, which involves sensing power or water usage of private households to compute the average trend of a region. People might not agree to allow these applications to intrude their personal world if the privacy of the collected information is not protected. The goal of this paper is to design a scalable, efficient, data-loss resilient, privacy-preserving data aggregation algorithm for WSNs.

To achieve this goal, one might suggest to adapt the existing privacy-preserving algorithms designed for data mining applications [1, 12], but, unfortunately, these algorithms are too computational expensive to meet the severe resource constraint of sensor nodes. Exploring alternative paths, researchers [2, 14, 16, 23] have already presented a few proposals to solve the privacy problem in data aggregation. However, to the best of our knowledge, none of the existing algorithms satisfy the following three requirements at the same time, which is the goal of our paper:

1. To prevent the sensed value of an individual node from being disclosed to other nodes during the aggregation process.
2. To prevent the sensed value of an individual node from being disclosed to the BS, i.e, the BS will have access only to the data aggregate.
3. The possibility of a node becoming off-line during the aggregation process, or a message being lost before reaching the BS, should not affect the correctness of the aggregate computed based on the nodes that participated in the aggregation.

Rationales of our proposal The main idea behind the design of our algorithm is as follows. The nodes in the network divide themselves into clusters. The aggregate of the nodes within a cluster is computed in such a way that no individual sensor reading is leaked during this process. To *obfuscate* the individual sensor readings, we make use of *twin-keys* shared by node pairs within a cluster, which are established in an onetime setup phase. After the cluster aggregates are computed, they are sent in clear text to be further aggregated to compute the final aggregate—usually via a tree-based aggregation algorithm.

Contributions This paper is the first one, to the best of our knowledge, that achieves both the following properties: First, it provides a mechanism that preserves the privacy of the data contributed by a sensor to the aggregate value. That is, the individual values as well as the identity of the contributing nodes cannot be derived by any node in the network, as well as by the BS. Second, the protocol is robust against communication and node failures. In Table 2 we summarize the features of our proposed protocol compared with other protocols in literature.

Organization The rest of the paper is organized as follows. We present the related work in Section 2. Section 3 presents the assumptions and the threat model considered in this paper. In Section 4 we give the overview of our proposed solution, while a detailed presentation is discussed in Sections 5 and 6. In Section 7, we present the security and performance analysis of our protocol. We finally conclude in Section 8.

2 Related Work

Researchers [13, 22] proposed in-network aggregation algorithms which fuse the sensed information en route to the BS to reduce the communication overhead. Several algorithms are designed to compute aggregates such as Count, Sum, and Average.

The research community also examined a few security issues related to aggregation algorithms. Wagner [27] addressed the problem of resilient data aggregation in the presence of false data injection attack by a few malicious nodes, and provided guidelines for selecting appropriate aggregation functions in a sensor network. Yang et al. [29] proposed SDAP, a secure hop-by-hop data aggregation protocol using a tree-based topology to compute the correct Average in the presence

of a few compromised nodes. Chan et al. [3] designed a novel verification algorithm by which the BS could detect if the computed aggregate was falsified. Another approach for computing Count and Sum, even if a few compromised nodes inject false values, was proposed by Roy et al. [26]. A recent solution for the secure Median computation has also been proposed [25]. However, none of the above algorithms address the privacy issues of data aggregation.

There exists a body of work that addresses privacy issues in data mining applications. In [1, 12], the authors proposed data perturbation techniques to protect the private values, whereas a few secure multi-party computation schemes were designed in [6, 15, 30]. However, these privacy-preserving algorithms are too much computation expensive to be applicable for the low-end nodes in a sensor network.

Privacy Homomorphism (PH) proposed by Rivest et al [24] allows to aggregate encrypted data. PH is an encryption transformation that enjoys the following property, related to an operation "$\circ$". Given an encryption function, $E : S \times R_1 \rightarrow R_2$, and a decryption function, $D : S \times R_2 \rightarrow R_1$, where R_1, R_2 are rings and S is the key-space, for $a, b \in R_1$ and $s \in S$, the following equation holds: $a \circ b = D_s(E_s(a) \circ E_s(b))$. More recently, Domingo-Ferrer [10] proposed a PH that preserves both addition and multiplication ("$\circ$" is "+" and "$\cdot$", respectively). The proposal has been proven to be secure against known-cleartext attack (as long as the ciphertext space is much larger than the cleartext space).

In Girao et al.'s work [14], the PH is used to allow the aggregator node to compute the correct encrypted aggregate from the encrypted values coming from sensor nodes. This allows the protocol to guarantee the privacy of the sensor nodes against a passive eavesdropper. However, as all of the nodes share the same encryption key with the base station, the protocol does not guarantee the privacy of individual sensed data against other nodes or the BS.

In [2], the authors propose a solution for data aggregation that protects the privacy against other nodes. The authors assume that each node n_i shares a key k_i with the BS. Basically, each node n_i adds a random number to its sensed value where the random number is determined by k_i. After receiving the encrypted aggregate, the BS filters out the correct aggregate by subtracting all the random numbers added by the nodes. We observe that this scheme does not protect privacy of individual sensed values if the BS eavesdrops over the network. Moreover, this scheme is critically vulnerable to message loss, which is very common in a WSN. If just one message is lost, the BS obtains a bogus aggregate, without suspecting any problem. The author propose how to cope with this last issue, adding the list of contributing nodes or the list of nodes that did not contribute (whichever is the shorter one). However, note that this solution does not prevent this list from being $O(N)$ in length, where N is the number of sensors in the network.

In [16], the authors propose two different solutions: CPDA and SMART. The former gives a solution for data aggregation preserving node-privacy against other nodes. We observe that CPDA can be extended to provide privacy against BS and furnish a solution against data-loss as well. However, as stated by the authors, the overhead of this protocol is high. Indeed, they use the anonymization sets, where each node out of the C nodes within a cluster has to send (and receive) $C - 1$ messages, resulting in $O(C^2)$ sent (and received) messages within each cluster, for each aggregation phase. Furthermore, each single node has to encrypt and decrypt $O(C)$ messages, and the cluster head has to compute the inverse of a $C \times C$ matrix, for each aggregation phase. The latter proposal, SMART, is more efficient than the first proposal. However, it does not protect privacy against the BS, and suffers from the same problem of message-loss as in [2].

Researchers also looked into source privacy issues in sensor network. In [28], Yang et al. prevent a global adversary from identifying a node as the event source.

3 Network Assumptions and Threat Model

In this section, we describe the network assumptions and the threat model considered in the rest of this paper. We consider a static multihop WSN of N sensor nodes and a single base station (BS). We consider sensor nodes similar to the current generation of sensors (e.g., Berkeley MICA2 Motes [20]) in their computational and communication capabilities and power resources, while the BS is a laptop-class device supplied with long-lasting power.

A pair-wise key mechanism, like the ones in [5, 7, 11, 21], is used to enable secure communications among the network nodes. Our protocol is independent from the particular mechanism used. We further assume that a set of K keys (key-ring), is pre-loaded in each node, using the set-up procedure of Eschenauer and Gligor's protocol [11]: The K keys are randomly chosen from a larger key-pool of size P. The key-pool is known by the network administrator; i.e. the nodes manifacturer that stores the keys in the nodes memory. The BS that collects the data—that can be a third party—does not know any information about the key-pool. Compromising the key-pool would result in a threat to nodes privacy. However, coherently with other works in literature [5, 7, 11, 21] considering a manifacturer compromision is out of the threat model that we consider. Finally, note that this set of keys is not related to the pair-wise key establishment while it is exclusively used in our "twin-key" establishment protocol.

In our proposal, we use a clustering mechanism to group the nodes in several clusters. It is out of the scope of this paper to give a detailed description of the cluster formation algorithm. Our protocol makes use of a cluster formation algorithm such as the one in [4]. The basic building block for cluster formation in [4] is as follows: Each node applies an hash function, $H : (seed|x) \rightarrow y \in [1..deg]$, where deg is the average degree of a node and x is a node ID. The node having the largest result among neighbors (y_{max}) becomes the *leader* and announces its status. Nodes with value smaller than y_{max} wait to hear from a leader to set that node as the cluster head. Special cases, conflict resolution and the security of this protocol are discussed in [4].

In this paper, we focus on in-network computation of the Sum aggregate. Note that it is possible to extend the Sum aggregate to implement other aggregation function as well, such as Count and Average.

We assume that the aim of the adversary is to compromise the privacy of a node. In particular, it will not drop or modify messages if this does not help him to violate the privacy of a node. We consider the adversary to be able to:

1. Eavesdrop all of the network communications;
2. Control a fraction of nodes;
3. Obtain any information from the BS.

Table 1 summarizes the notation used in this work.

Symbol	Meaning
P	Key Pool
K	size of the node key-ring
$\mathcal{K}_{ID}$	key-ring set of the node ID
e	indicates the id of the executing node in the procedures
k_i	i-th key in the key pool for twin-key
C	Number of nodes in each cluster
A	the number of twin-keys each node needs to establish
V	the number of alive twin-keys required for active participation
R	the size of message in terms of declared keys
r	the number of twin-keys each node can initiate in each round
H	a hash function
d_{ID}	sensed (private) value of the node ID

Table 1. Notations

4 Protocol overview

The key elements of our protocol are the following: First, we establish *twin-keys* for different pairs of nodes in the network. We require the twin-key establishment to be anonymous—each node in a pair cannot derive the identity of

the other node (*twin-node*) it is sharing a twin-key with. Second, for each aggregation phase, we use an anonymous liveness announcement protocol to declare the liveness of each twin-key—each node becomes aware of whether a twin-key it possesses will be used by the anonymous twin-node. Finally, during the aggregation phase, each node encrypts its own value by adding *shadow values* computed from the alive twin-keys it holds. As a result, the contribution of the shadow values for each twin-key will cancel out each other.

Our protocol consists of three major steps as follows:

1. Local cluster formation: In this step, we require the network nodes to group themselves into clusters. In literature, there are different solutions for the cluster formation. The description of a detailed cluster formation algorithm is out of the scope of this paper. In the following, we assume that a cluster algorithm such as the one proposed in [4] is used. For ease of exposition, from now on we also assume that each cluster has a fixed number of nodes, C. In the following steps of the protocol, we further require each cluster to form different logical Hamiltonian circuits. Note that an extension of the cluster formation in [4] can be used for the Hamiltonian circuit formation: e.g. the hash function, H, described in the previous Section returns a value for each node; the order of these values define an order of the corresponding nodes that can be used to determine the next and previous neighbour of each node. As another example, a node, say n_i, can start setting a new circuit by randomly choosing one cluster node, say n_j, as its next (right) neighbor in a circuit. Node n_i communicates its choice to node n_j. The selected node, n_j, will choose its own next (right) neighbor within the nodes in the same cluster, that are not yet selected. Eventually, the last node will select node n_i as its next (right) neighbor, to complete the Hamiltonian circuit. We observe that, we require to build just a logical circuit that it is not based on any physical information (as for example a right nodes being actually on the right of a node) but on the fact that the considered nodes are in the same neighbourhood (i.e. they are in the communication range of each other). As a result, no GPS equipments of any triangulation task is required. Finally, we require for each pair of nodes that are neighbors in the circuit to share a pair-wise key.
2. Twin-key establishment: This step is performed independently within each local cluster. We recall that we assume that each node contains a pre-deployed key-ring of K symmetric keys, randomly chosen from a larger common key-pool of size P. In this step, each node n_i anonymously checks which ones of its K keys are also shared with other nodes in the same cluster. In particular, a node is required to have at least A out if its K keys shared within its local cluster. This step will be further discussed in Section 5.
3. Data aggregation: This is the actual aggregation step of our protocol. Note that, other than this step, all the previous steps are performed only once during the set-up phase. The data aggregation step can be further divided into two main parts:
   3.1. First, each cluster computes the aggregated value of its nodes, together with a twin-key liveness announcement procedure. During this phase an aggregate is routed twice along the Hamiltonian circuit. Each node adds to the aggregate its own sensed value. At the same time, for each alive twin-key it adds (or removes, in accordance with the liveness announcement) a corresponding shadow value. As a result, the cluster head obtains the correct aggregate for the cluster.
   The liveness announcement guarantees that any shadow value, computed from a twin-key, that is added in the aggregation by one node, will be removed by another node that shares the same twin-key.
   3.2. At the end of step 3.1, there will be several nodes in the network that acted as cluster heads. These nodes own the corresponding cluster aggregates. Now, we want to further aggregate all of these values and

to pass the final aggregate to the BS. In this step, we use a tree-aggregation hierarchical structure, commonly discussed in literature. In particular, we assume to use the TAG algorithm [22] with the following modification. The cluster head nodes will contribute to the aggregate with the cluster aggregate computed in step 3.1. All of the other nodes do not contribute to the aggregate while they forward the aggregate computed from the received sub-aggregates. As a result, the BS will receive the sum of the values owned by all of the cluster heads.

Figure 1 illustrates this procedure with an example. The cluster heads n_1, n_8, n_{11} and n_{14} possess the corresponding cluster aggregates 10, 13, 8 and 9, respectively. All of the other nodes contribute to the final aggregate with value 0. Finally, the BS will obtain the aggregate result 40.

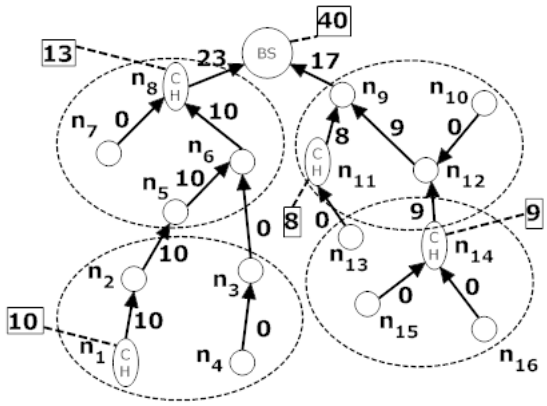


Fig. 1. Aggregating the individual cluster aggregates: A tree aggregation protocol is used to compute the final aggregate. In this example we have four clusters. Each cluster head possesses the aggregate of the corresponding cluster (computed in the step 3.1 of our protocol). After the hierarchical aggregation along the tree the BS receives the final aggregate.

We discuss the step 3.1 in Section 6, while we refer to [22] for the TAG tree-aggregation of the step 3.2.

5 Twin-key Agreement

In this section we describe the set-up phase. The aim of this step is, for each node, to establish a number of twin-keys with other nodes. In particular, we say that node n_i established a twin-key with another node (twin-node) in the cluster if n_i is aware of the fact that there is a node in the cluster sharing a key with it. Note that n_i does not know the identity of its twin-node. The requirements of a twin-key establishment are:

- The twin-keys are only known to the owners. They cannot be eavesdropped by other nodes or by the outside attackers.
- The twin-nodes (nodes that agree on a twin-key) cannot determine each other's identity, i.e., the twin key is established anonymously.

Furthermore, to improve on the level of anonymity, we require each node to establish at least a given number, A, of twin-keys.

The twin-key agreement is a relay-based protocol. The twin-keys are initiated by nodes, passed through the circuit of the local cluster, and accepted by other nodes. Our protocol assumes that a key-ring of symmetric keys are pre-deployed in each node using the same approach as in Eschenauer et al.'s scheme [11]. For each sensor node, the manufacturer stores in the node's memory K keys randomly selected from a pool of $P >> K$ symmetric keys. As a result, two nodes in the same cluster will share a given key with a probability depending on K and P as studied in [9].

5.1 Twin-key Agreement: Protocol Description

At the beginning of the twin-key agreement protocol, each node runs *PREPARE* procedure (Procedure 1). Variable a is used to keep track of the number of twin-keys that the node still needs to establish. *List* is a list of $< seed, key >$ pairs used to keep track of the twin-keys waiting to be agreed by other nodes, where each key is identified by a seed which is a random number. $TKList$ is a list of already established twin-keys. $\mathcal{K}'$ is the list of keys not yet used for twin-key

agreement. Finally, $Valid$ indicates whether the executing node will participate in the subsequent aggregation procedure.

Procedure 1

1: **procedure** PREPARE
2: $a = A$; //Number of Twin-key needs to be agreed
3: $List = empty$; //Seed-Key pairs sent out
4: $TKList = empty$; //Agreed Twin-keys
5: $\mathcal{K}' = \mathcal{K}_e$; //Twin-keys can be initiated
6: $Valid = true$; //Will participate in aggregation
7: **end procedure**

Procedure 2

1: **procedure** INITIATE_AGREEMENT
2: $M \leftarrow \langle C \times A, \langle 0, \_, \rangle_1, \ldots, \langle 0, \_ \rangle_R \rangle$;
3: randomly select $i_1, \ldots, i_r$ from $\{1, 2, \ldots R\}$;
4: randomly select and remove $k_1, \ldots, k_r$ from $\mathcal{K}'$;
5: randomly select key seeds (random number) $s_1, \ldots, s_r$;
6: $List = List \cup \{\langle s_1, k_1 \rangle, \ldots, \langle s_r, k_r \rangle\}$;
7: replace $\langle 0, \_ \rangle_{i_1}, \ldots, \langle 0, \_ \rangle_{i_r}$ with $\langle s_1, H(k_1) \rangle_{i_1}, \ldots, \langle s_r, H(k_r) \rangle_{i_r}$ in M;
8: send M to the next node; //encrypted with pair-wise key
9: **end procedure**

PREPARE procedure is executed by each single node. After this procedure is carried out, each cluster head executes procedure *INITIATE_AGREEMENT* (Procedure 2). First, the CH creates a message M with R empty seed-key pairs (line 2), where R is a design parameter. The message format is: $\langle S, \langle s_1, h_1 \rangle, \ldots, \langle s_R, h_R \rangle \rangle$, where S is the total number of twin-keys to be established, and $\langle s_i, h_i \rangle, 1 \leq i \leq R$, denote the twin-keys declared in the message waiting to be agreed on. The cluster head initializes the message by setting $S = A \cdot C$. This is used to meet the requirement that each of the C nodes in the cluster shares at least A twin-keys.

After generating M, CH randomly selects r positions out of the R ones in M (line 3) and randomly selects and remove r keys from $\mathcal{K}'$ (line 4). Then, for each selected key, it generates the pair $< s_i, H(k_i) >$ (line 5), where $H(k_i)$ is the hash of k_i and s_i is a random number associated with k_i. These pairs are copied in the r selected positions of M. Note that the r positions are randomly selected (line 3) in order to prevent the attacker to associate a node identity with a given position in M. Finally, CH sends M to one of its neighbor. This implicitly determines the direction of the message in the circuit.

Each node that receives the twin-key agreement message executes procedure *RECEIVE MESSAGE* (Procedure 3). This procedure consists of the following main steps performed by each node n_i:

(i) n_i checks the newly agreed keys which it had proposed before (lines from 3 to 16). That is, for each key declared in the previous round (line 4), n_i checks whether the key has been accepted. In particular, if the declaration of the key is still in the message M (line 6) it means that no other node agreed for that key. Otherwise, the key will be considered shared with someone (line 9). If the executing node has not yet established enough twin-keys (i.e. $a < A$), the newly agreed keys will be counted in the node's number of agreed keys, a, and the cluster's number of agreed keys, S. We recall that the declaration of a key is a pair $< s_i, H(k_i) >$, where s_i is a random number used to keep track of the particular instance of the key held by the node that declare that key. This is used in order to avoid confusion if the same key is proposed by more than two nodes in the cluster. As a result, if a node n_i declares key k_1 in M and k_1 is removed by n_j, a third node n_z re-declaring the same key, k_1, will use a different seed. If the message goes back to n_i, it can indeed understand that its key has been accepted and that the one declared in the message is just another instance of k_1 declared by some other node.

(ii) n_i checks for twin-keys proposed by other nodes that it can agree on (line from 17 to 29). For all keys' hashes in M (line 17), it checks if it has the corresponding key (line 18). If n_i never agreed on that key (line 19) it agrees on this key (line 20 and 21). Also, the corresponding counter is updated, if necessary (lines from 22 to 25). In any condition, when the key is agreed, the corresponding declaration is removed from the message M.

(iii) n_i proposes new keys to be agreed on (lines from 30 to 43). If the number of twin-keys established by n_i are not yet enough (line 30), it checks if it still has some keys to propose for agreement (line 31). If this condition is not satisfied, then the node will not participate in the aggregation phase (lines 32), because its privacy is not protected enough. In this case, the node also updates the variable S, in order to allow the CH to end the protocol without its own participation (line 33). If the node has other keys it did not try to share yet (i.e. condition in line 31 does not hold) it selects available keys from $\mathcal{K}'$ and declares these keys in the message M. The number of keys that can be declared in M will be bounded by $t = min\{t, r, |\mathcal{K}'|\}$—$t$ is the number of free positions in M (line 35), and r is the maximum number of keys declared in each round.

(iv) n_i sends the message or concludes the protocol (lines from 44 to 52). In particular, the only node that can end the protocol is the CH. The cluster head can terminate the protocol only if each node in the cluster either agreed on A keys (lines 11-12 and 23-24) or refused to participate in the aggregation (line 33). In all of the other cases, that is if $S \neq 0$ or the executing node is not the CH, it just sends the message to its neighbour.

An example of twin-keys agreement is shown in Figure 2. In this example, the CH (node n_1) initiates the protocol by sending message 1, where it proposes two keys (k_1, k_3) for the agreement. When the announcement of k_1, that is $< s_1, H(k_1) >$, reaches n_4, it agrees on this key and removes the announcement from the message. Eventually, when n_1 receives message 6, it knows that k_1 has been agreed with someone.

6 Data Aggregation

In this section we explain the data aggregation phase. In particular, for ease of exposition, we describe it in two consecutive steps:

- (i) Twin-keys liveness announcement;
- (ii) Data aggregation with shadow values.

Procedure 3

```
1: procedure RECEIVE_MESSAGE(M)
2:     ⟨S, ⟨s_1, h_1⟩_1, ..., ⟨s_R, h_R⟩_R⟩ ← M
3:     if !empty(List) then
4:         for all ⟨s', key'⟩ ∈ List do
5:             remove ⟨s', key'⟩ from List
6:             if ∃ s_i = s' then
7:                 replace ⟨s_i, h_i⟩_i with ⟨0, _⟩_i in M
8:             else
9:                 TKList = TKList ∪ {key'}
10:                if a > 0 then
11:                    a = a − 1
12:                    replace S with S − 1 in M
13:                end if
14:            end if
15:        end for
16:    end if
17:    for all i, i = 1 ... R, s_i ≠ 0 do
18:        if ∃ key' ∈ K_e, H(key') = h_i then
19:            if key' ∈ K' then
20:                TKList = TKList ∪ {key'};
21:                remove key' from K';
22:                if a > 0 then
23:                    a = a − 1;
24:                    replace S with S − 1 in M;
25:                end if
26:            end if
27:            replace ⟨s_i, h_i⟩_i with ⟨0, _⟩_i in M;
28:        end if
29:    end for
30:    if a > 0 then
31:        if K' = φ then
32:            Valid = false;
33:            replace S with S − a in M;
34:        else
35:            t = number of s_i = 0 in M;
36:            t = min{t, r, |K'|};
37:            randomly select and remove i_1, ..., i_t from {1, 2, ... R};
38:            randomly remove k_1, ..., k_t from K';
39:            randomly select key seeds (random number) s_1, ..., s_t;
40:            List = List ∪ {⟨s_1, k_1⟩, ..., ⟨s_t, k_t⟩};
41:            replace ⟨0, _⟩_{i_1}, ..., ⟨0, _⟩_{i_t} with ⟨s_1, H(k_1)⟩, ..., ⟨s_t, H(k_t)⟩_{i_t} in M;
42:        end if
43:    end if
44:    if executing is CH then
45:        if S = 0 then
46:            broadcast twin-key agreement over;
47:        else
48:            send M to the next node; //encrypted with pair-wise key
49:        end if
50:    else
51:        send M to the next node; //encrypted with pair-wise key
52:    end if
53: end procedure
```

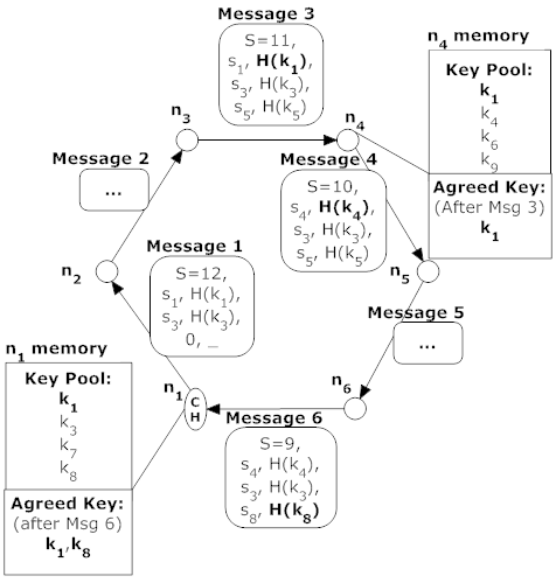


Fig. 2. Twin-key agreement example.

Despite we present them separately, these two procedures can run together as discussed at the end of this section.

In the liveness announcement procedure, all of the nodes anonymously declare the liveness of the twin-key they posses. Each node will check whether the number of currently alive twin-keys is enough to protect the privacy of the sensed value. With $V \leq A$ we indicate the number of twin-keys that a node requires to be used during the aggregation in order to reach a satisfactory level of privacy. Assume that at least V of the node's twin-key are announced alive. Then, in the aggregation phase a node will add to the aggregate value computed so far its private value and the sum of the shadow values computed based on its alive twin-keys. However, if less than V twin-keys are announced as alive by the corresponding twin nodes, the node will add to the aggregate only the shadow values without its own private value.

In Section 6.3, we show how to combine the twin-key liveness announcement and the aggregation together.

6.1 Twin-key Liveness Announcement: Protocol Description

In the twin-key liveness announcement, each node first executes the procedure *ANNOUNCE_PREPARE* (Procedure 4). $ATKList^+$ and $ATKList^-$ are used to record the alive twin-keys. Twin-keys in $ATKList^+$ will be used to compute positive shadow values and twin-keys in $ATKList^-$ will be used to compute negative shadow values. The data-type of $ATKList^+$ and $ATKList^-$ are *Set* and *Bag* respectively: A key appears at most once in $ATKList^+$ but could appears more than once in $ATKList^-$. Finally, *Avalid* is used to record whether the executing node will participate in the data aggregation.

Procedure 4

1: **procedure** ANNOUNCE_PREPARE
2: $ATKList^+ = \phi$;
3: $ATKList^- = \phi$;
4: $TempATKList^+ = \phi$;
5: $Avalid = true$;
6: **end procedure**

After the *ANNOUNCE_PREPARE* procedure has been executed by all of the nodes, CH initiates the twin-key liveness announcement, i.e. executes procedure *CH_ANNOUNCE* (Procedure 5). It initializes the number of participating node, $T = 0$ (line 2). Then, it will consider A of its twin-keys (line 3). For each key, k_i, it will generate a random seed, s_i (line 4), and it will write all the pairs $< s_i, H(k_i) >$ in the message it sends to its neighbour (line 7).

Procedure 5

1: **procedure** CH_ANNOUNCE
2: $T = 0$;
3: Select A twin-keys from $TKList$: $k_1, \ldots, k_A$;
4: Select A random numbers: $s_1, \ldots, s_A$;
5: $TempATKList = \{(s_1, k_1), \ldots, (s_A, k_A)\}$;
6: $List = \{(s_1, H(k_1)), \ldots, (s_A, H(k_A))\}$;
7: send $(T, List)$ to the next node; //encrypted with pairwise key
8: **end procedure**

Each node, except the CH, receiving the liveness announcement message for the first time executes the procedure *FIRST_ROUND* (Procedure 6). The CH never executes this procedure. In particular, for each h_i in the $List$ within the message M (line 4) it checks if it has a twin-key, k, such that $H(k) = h_i$ (line 5). If this is the case it adds the key, k, in its list $ATKList^-$ (line 6) and removes the corresponding key announcement from the message (line 7). Furthermore, the node declares the liveness of all its A' other keys not yet known as alive, where $A' = A - |ATKList^-|$ (line 10). To do so, as for the Procedure *CH_ANNOUNCE*, it selects a random seeds s_i for each key k_i in $TKList \setminus ATKList^-$ and adds the corresponding pair $< s_i, H(k_i) >$ in the $List$ in M. Finally, it sends the updated message M to its neighbour node.

Procedure 6

```
1: procedure FIRST_ROUND
2:     (T, List) ← M;
3:     (s_1, h_1), ..., (s_R, h_R) ← List;
4:     for h_i, (1 ≤ i ≤ R) do
5:         if ∃k ∈ TKList, H(k) = h_i then
6:             ATKList^- = ATKList^- ∪ {k};
7:             remove (s_i, h_i) from List;
8:         end if
9:     end for
10:    A' = A − |ATKList^-|;
11:    Select A' twin-keys from TKList \ ATKList^-:
       k_1, ..., k_A';
12:    Select A' random numbers: s'_1, ..., s'_A';
13:    TempATKList = {(s'_1, k_1), ..., (s'_A', k_A')};
14:    List = List ∪ {(s'_1, H(k_1)), ..., (s'_A', H(k_A'))};
15:    send (T, List) to the next node; // (T, List) encrypted
       with pair-wise key
16: end procedure
```

Each node, except the CH, executes the procedure *SECOND_ROUND* (Procedure 7) when it receives a liveness announcement message for the second time. The CH executes this procedure when it receives the message for the first time. For each key that the executing node announced in the previous round of the message (line 4), it checks if the corresponding declaration is still in the message M it just received (line 5). If it is so, this means that no other node has removed the declaration from M. The node will then remove the non-alive key from the message. Otherwise, i.e. the key is alive (someone stored the key in its $ATKList^-$), the executing node puts the corresponding key in its list $ATKList^+$. Furthermore, for each declaration in M (line 11), the node checks if it is storing a key (in $TKList$), not yet announced by itself (not in $ATKList^+$), that some other node is asking for liveness (line 12). Note that at this point the node has removed its own declared keys from the $List$ in M: the only keys in M are declared by other nodes, indeed. If the node has such a key k, it adds the key to its $ATKList^-$ (line 13) and removes the corresponding declaration from the $List$ (line 14). The node then checks the number of twin-keys it is using in this aggregation (line 17). If this number is smaller than V, the required number of keys deemed necessary to satisfy the node privacy, it will participate in the aggregation only with the shadow values but not with its own private value (line 19). Otherwise, it will participate also with its private value, and increases the number T of participating nodes (line 21). Finally, the executing node sends the message M to its next neighbour. Note that, when the CH receives the message for the second time, the cluster aggregation process terminates. Then, the tree aggregation is performed as described in point 3.2 of Section 4.

Figure 3 illustrates an example of the result of the procedures shown in this section applied to the same setting used in Figure 2. Keys in bold font are the alive keys while non-bold fonts indicates non-alive key. For example, node CH does not consider alive the key k_1, actually shared with node n_4 that is currently off-line.

6.2 Data Aggregation with Shadow Values: Protocol Description

Here we describe the aggregation phase, while each node executes the procedure *NODE_AGGREGATION* (Procedure 8). In particular, if a node does not have enough alive twin-keys to protect its own private value (line 3), it just does not participate in the aggregation. That is, it does not include its own value in the aggregate (line 3). Otherwise, it initializes the variable x with its own private value d (line

Procedure 7

```
1: procedure SECOND_ROUND
2:     (T, List) ← M;
3:     (s_1, h_1), ..., (s_R, h_R) ← List;
4:     for (s, k) ∈ TempATKList do
5:         if ∃(s_i, h_i), s = s_i ∧ H(k) = h_i then
6:             remove (s_i, h_i) from List;
7:         else
8:             ATKList^+ = ATKList^+ ∪ {k};
9:         end if
10:    end for
11:    for (s_i, h_i) ∈ List do
12:        if ∃k ∈ TKList \ ATKList^+, H(k) = h_i then
13:            ATKList^- = ATKList^- ∪ {k};
14:            remove (s_i, h_i) from List;
15:        end if
16:    end for
17:    Num_k = |ATKList^+ ∪ BagToSet(ATKList^-)|;
18:    if Num < V then
19:        Avalid = false;
20:    else
21:        T = T + 1;
22:    end if
23:    send (T, List) to the next node; //encrypted with pair-
       wise key
24: end procedure
```

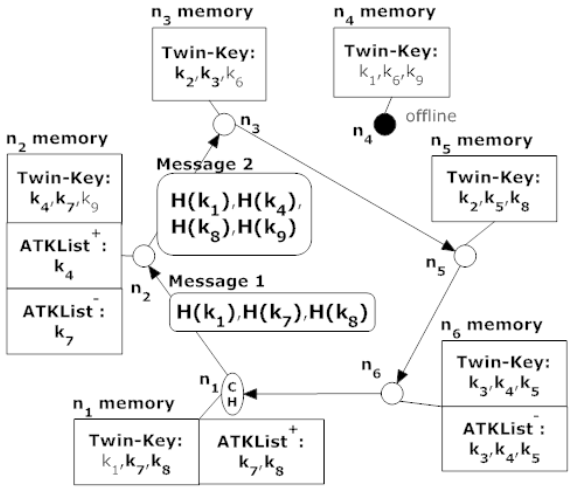


Fig. 3. Twin-key liveness announcement example. $A = 3$, $V = 2$.

Procedure 8

```
1: procedure NODE_AGGREGATION
2:     (y) ← M;
3:     if Avalid = false then
4:         x = 0;
5:     else
6:         x = d;
7:     end if
8:     for k ∈ ATKList^+ do
9:         x = x + H(Seed, k);
10:    end for
11:    for k ∈ ATKList^- do
12:        x = x − H(Seed, k);
13:    end for
14:    send (x + y) to the next node; //encrypted with pair-
       wise key
15: end procedure
```

6). Then, for each key in $ATKList^+$ (line 8), it adds $H(Seed, k)$ to x (line 9). Similarly, for each key in $ATKList^-$ (line 11) it removes $H(Seed, k)$ from x (line 12). Finally, the value $x + y$ is sent to the node's next neighbour. Note that $Seed$ is unique for each different data aggregation. For example, it can be a random number broadcast from the BS together with the aggregation request. Also, it could be a time sequence number when the data aggregation is executed, if executed at given interval of time without any request from the BS.

Figure 4 illustrates an example of data aggregation on the same setting considered in Figure 2, with the alive twin-keys shown in Figure 3. In this example, the CH node adds its own values d_1, $H(Seed, k_7)$, and $H(Seed, k_8)$ (it has both k_7 and k_8 in its $ATKList^+$). The following node (n_2) adds its own value d_2, adds $H(Seed, k_4)$, and subtracts $H(Seed, k_7)$ (k_4 and k_7 are in $ATKList^+$ and $ATKList^-$ respectively). The resulting message sent by node n_2 has a value that now corresponds to $d_1 + H(Seed, k_4) + d_2 + H(Seed, k_8)$. When the message reaches CH again, it will contain the exact sum of the private values of all the participating nodes.

6.3 A complete protocol run

Here, we show how to integrate the alive announcement round and the aggregation round together to have a more efficient solution: The two phases can run together using one single message containing both liveness announces and aggregated value. Note that the twin-key liveness announcement requires the message M to be relayed two times along the circuit. Also, note that the second time each node receives the message knows exactly which of the possessed twin-keys it should use in the aggregation. Then, we can

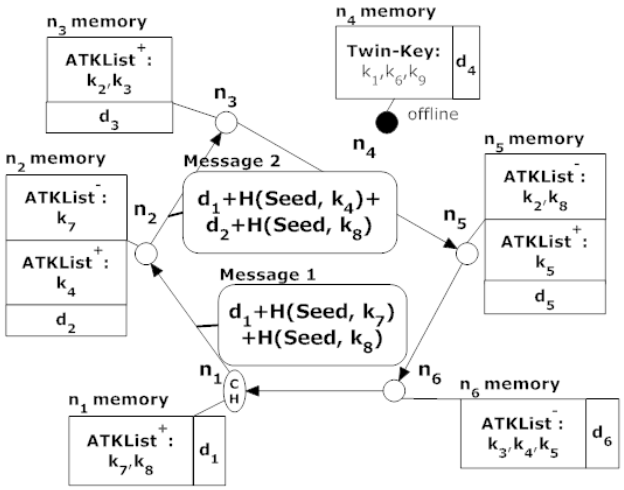


Fig. 4. Data aggregation with shadow values. $A = 3$, $V = 2$.

think that in the same message relayed in this second round the aggregated value is also routed and updated according to Procedure 8.

7 Security and Complexity Analysis

In this section, we present a thorough analysis of our protocol. In next subsection we study the problem's parameters; in Section 7.2 we discuss the security features of our proposed protocol; finally, in Section 7.3, performance analysis is shown. Finally, in Section 7.4 we compare our protocol with other solutions in literature.

7.1 Parameter Study

In this section we provide guidelines for selecting the values to assign to the parameters of our protocol, that is P, K, C, and A. We start reminding that the necessary condition for a node to participate in a cluster-based aggregation is to share A keys with other nodes in the same cluster. Note that, if the above condition does not hold for a node, this node can attempt to join another (neighboring) cluster it shares enough keys with. In practice, we expect that for a given set of nodes in a cluster there is a reasonably high probability, p_s, that all these nodes share A keys. Set the desired probability p_s, assignment for P, C, A, K satisfying p_s should be found.

For example, we can set $p_s = 0.99$. Then, we can choose the key-pool size $P = 10,000$, the cluster size $C = 20$, and require that each node shares $A = 5$ keys with the other $C - 1$ nodes in the same cluster. Then, the only other parameters we can tune to satisfy p_s is the variable K. To simplify the analysis, without making it less rigorous, we state the following assumptions:

- The K keys assigned to each node are chosen from P with replacement.
- The probability of sharing any of the A keys is independent for the nodes in the cluster. Note that, as observed in [8], this is a feasible approximation.

Using the previous assumptions, the probability for a given node to share at least A keys with the others $C - 1$ nodes is equal to:

$$p_s = 1 - \sum_{i=0}^{A-1} \binom{K(C-1)}{i} \left(\frac{K}{P}\right)^i \left(\frac{P-K}{P}\right)^{K(C-1)-i} \quad (1)$$

In Figure 5 we plot the analytical result (Equation 1) and the simulation result for $P = 10,000$, $C = 20$ and $A = 5$. From this graph we can observe that $p_s > 0.99$ is achieved for any $K \geq 65$. Note that, once the required p_s is guaranteed, choosing a smaller K increases security. Indeed, an adversary capturing a node will acquire a smaller number of pool's keys.

Then, to satisfy $p_s > 0.99$ for the selected $P = 10,000$, $C = 20$, and $A = 5$, we chose $K = 65$. This choice of parameters will be used for the following sections.

To show how this parameters choice affects the number of nodes participating in the aggregation phase, in Figure 6 we simulate the protocol, reporting (y-axis) the number of nodes actively participating in the aggregation, while increasing the number of the off-line nodes (x-axis). A non-off-line node actively participates in the aggregation adding its own value and the hashes of its alive twin-keys, if the number of these hashes are at least V. Otherwise, it passively participates adding only the hashes of its alive twin-keys. In Figure 6 we consider different number of agreed twin-keys, $A = 5, 6, 7$. For each of these,

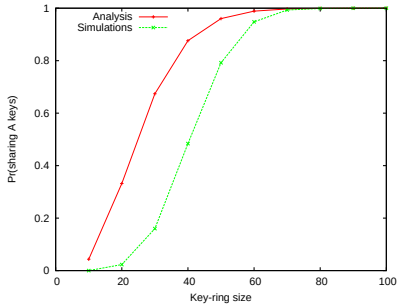


Fig. 5. Probability for a node to share at lest A twin-keys in the cluster, varying the key-ring size, K. $P = 10,000$, $C = 20$, $A = 5$.

we also consider different number of alive twin-keys required for the node active participation ($V = 3, 4, 5$). Note that, for $A = 5$, $V = 5$, the number of off-line nodes significantly affects the active participation of the other nodes. We expect that a similar behaviour could be observed when $A = V$. However, if we set A to be greater than V, such negative effects are seriously reduced. For example, for $C = 20$, $A = 5, V = 3$, if 7 nodes are off-line, on average 10 out of the 13 on-line nodes can actively participate in the aggregation.

7.2 Security Analysis

In this section, we analyze the security of our protocol based on the threat model discussed in Section 3. That is, the aim of the attacker is just to compromise the privacy of the nodes. To reach this goal, We assume that an attacker can: (1) eavesdrop all of the communications in the network, (2) steal information from the BS, and (3) compromise a fraction of the network nodes.

For ease of exposition, we explain the security features of our protocol considering an increasingly powerful attacker. First, we assume that the attacker can just eavesdrop the exchange of messages. Due to the pair-wise encryption between nodes, the attacker cannot obtain any useful information to compromise a single node's privacy.

Then, we consider that the attacker can also steal information from the BS. In this case, we observe that in our protocol, the BS only acts as a receiver of the final aggregation result. There is no other information that can be gathered when compromising the BS. Therefore, the attacker obtains no useful information from the BS to compromise a single node's privacy.

A major threat appears when we assume that the attacker can also capture some nodes. In fact, all of the information stored in the captured nodes become known to the attacker, including pre-distributed keys and the agreed twin-keys. While it is not possible to protect the privacy of the captured nodes, our aim is to protect the privacy of the non-captured nodes. Therefore, we are interested in assessing the probability that the attacker can compromise the privacy of a non-captured node leveraging the information acquired having captured a certain number of nodes.

We assess this probability in two scenarios, hypothesising two different adversary behaviors: the *passive* and *active* one, described below.

Passive attack In this scenario, we assume that the attacker can only elaborate on the information extracted from the captured nodes (i.e. from the memory) and the received messages.

Each node value is protected by one or more shadow values. The secrecy of the shadow value, in turn, is protected by the secrecy of the twin-keys. To compromise the privacy of non-captured node, n_i, the attacker has to obtain the keys used to generate the shadow values that n_i uses to protect its own privacy. For a node n_i, it will send out a value computed by the following expression (see Procedure 8):

$$d_i + \sum_{k \in ATKList_i^+} H(k, Seed) - \sum_{k \in ATKList_i^-} H(k, Seed) \quad (2)$$

We refer to this value as the *coated value* of the node n_i. We recall that $Seed$ is a one-time-use number broadcasted by the BS together with the aggregation request.

Before starting our analysis we use an example to illustrate how the privacy of non-captured nodes is protected, even after the attacker compromised a significant number of other nodes. Figure 7 shows an example with $A = 3$ and

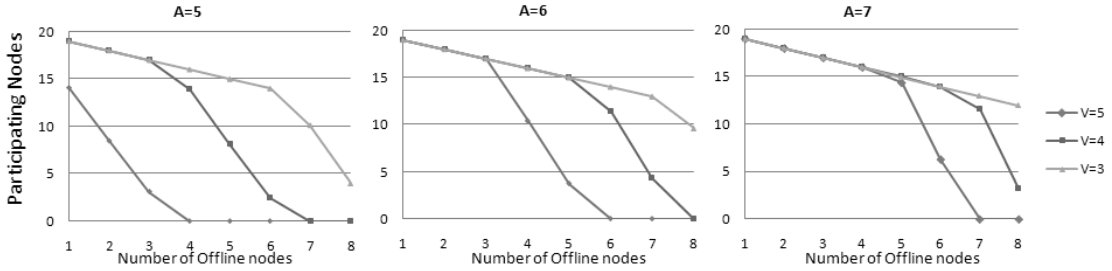


Fig. 6. On-line nodes actively participating in the aggregation, while increasing the number of the off-line nodes. $P = 10,000$, $C = 20$, $K = 65$.

$V = 3$. The attacker controls half of the nodes in the cluster: n_1, n_4 and n_5. The following twin-keys are then known to the attacker: k_1, k_2, k_5, k_6, k_7, k_8 (indicated with non-bold font in Figure 7). Besides, k_3 and k_4 remain unknown to the attacker. Also, in the aggregation phase, the attacker is able to obtain the content of the messages received or sent by controlled nodes. In this example, from these messages, the attacker can derive the following equations containing the private values of non-captured nodes:

$$Observed\_v_1 = d_6 - H(K_3, Seed) - H(K_4, Seed) \quad (3)$$

$$Observed\_v_2 = d_2 + d_3 + H(K_3, Seed) + H(K_4, Seed). \quad (4)$$

In particular, $Observed\_v_1$ is computed as the difference of the content of the messages sent and received by node n_6; $Observed\_v_2$ is computed as the difference between the message sent by n_3 and the one received by n_2. With five unknown variables, this system of equations cannot be solved by the attacker. The privacy of the non-captured nodes: n_2, n_3, n_6, is still protected against the attacker.

In general, the following lemma holds.

Lemma 1. *By executing the algorithm described in Section 6, each private value counted in the aggregation is protected by at least V keys, where $V \leq A$ and A is the number of twin-keys possessed by each node.*

Proof. By construction, each node executing the protocol in Section 6 will participate in the aggregation phase, Procedure 8, if and only if it finds out at least V alive twin-key it shares with other nodes, Procedure 7, line 18. □

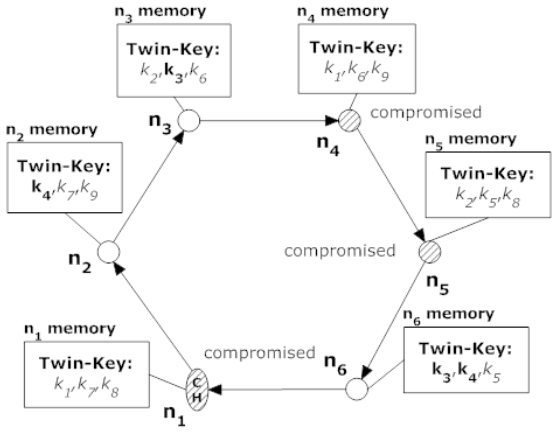


Fig. 7. Attack Example: Not Compromised Twin-key. $A = 3$, $V = 3$.

Next, we give a formal analysis for the probability that the privacy of a non-captured node can be compromised by the attacker. In order to compromise the private value d_i of node n_i, the attacker has to obtain both the coated value and the sum of the shadow values, that is:

$$\sum_{k \in ATKList_i^+} H(k, Seed) - \sum_{k \in ATKList_i^-} H(k, Seed). \quad (5)$$

For the coated value we have the following lemma.

Lemma 2. *The attacker can compromise the coated value of a node, n_i, if and only if it compromises the two neighbours of n_i in the circuit used during the aggregation (Procedure 8).*

Proof. ($\Rightarrow$) First of all, notice that if the attacker compromises the two neighbours of n_i in the circuit, it can observe the content of the following

messages: i) the aggregation message received by n_i, and ii) the message sent out from n_i. The difference between the values in these messages corresponds to the coated value of node n_i.

($\Leftarrow$) Assume only one of the neighbours of n_i is compromised. Also, assume the worst scenario where all of the other nodes in the cluster are compromised but node n_i and one of its neighbour, n_{i+1}. Without loss of generality we can assume that n_i appears before n_{i+1} in the circuit. In this case, the attacker would be able to observe: (i) the message received by n_i, and (ii) the messages sent out by n_{i+1}. By the difference of the values in the messages (i) and (ii) the attacker can observe the result of the following expression:

$$d_i + \sum_{k \in ATKList_i^+} H(k, Seed) - \sum_{k \in ATKList_i^-} H(k, Seed) + d_{i+1} + \sum_{k \in ATKList_{i+1}^+} H(k, Seed) - \sum_{k \in ATKList_{i+1}^-} H(k, Seed). \quad (6)$$

Once the aggregated value is sent out in the message (ii) by node n_i, neither the value d_i nor d_{i+1} will be removed by other nodes. Hence, the attacker cannot obtain the value of any expression that contains only one of d_i or d_{i+1}. □

Assume the attacker obtained the coated value of a node, n_i, by controlling its two neighbours. Next step is to obtain the n_i's shadow value. There are two kinds of useful information for the attacker to reconstruct the shadow value:

- *Type 1* knowledge. Twin-keys obtained from the captured nodes.
- *Type 2* knowledge. Sum of set of shadow values $H(k, Seed)$, for a key k. The attacker can obtain these values by capturing the neighbors of a node that participate in the aggregation without contributing its own value. Recall that a node, n_i, under the condition that less than V of its twin-keys are alive in the current aggregation phase (Procedure 7, lines 18-19 and Procedure 6, lines 3-6.), will add to the aggregate just the shadow values computed from its alive twin-keys (without its own value d_i).

For these two types of knowledge the following two observations hold.

Observation 1 *For a node n_i, Type 2 knowledge is a subset of Type 1 knowledge.*

On one hand, if the attacker knows all the keys of a node (*Type 1* knowledge), it can compute any possible subset of shadow values, including *Type 2* knowledge. On the other hand, *Type 2* knowledge does not reveal the secret twin-keys.

Observation 2 *Given the attacker can compromise $w \geq 2$ nodes, its best attack strategy for compromising the privacy of node n_i is the following: (1) By Proposition 2, the attacker has to compromise the two neighbors of the target node; (2) The attacker captures the remaining $(w-2)$ nodes selecting every other nodes, in a circuit, following one of the n_i's neighbours.*

In fact, by capturing nodes in this way, the attacker will have *Type 1* knowledge over the w captured nodes and will also have the chance to obtain *Type 2* knowledge from the $(w-2)$ nodes between two close captured nodes.

For example, to compromise node n_1 in Figure 8, the best strategy for an attacker that can capture $w = 4$ nodes is: (1) to capture the two neighbors n_2, n_c; (2) to capture the nodes n_4 and n_6 as above described. In this way, the attacker will have *Type 1* knowledge over n_c, n_2, n_4 and n_5. Also, it will have *Type 2* knowledge for n_1, n_3 and n_5.

Let us assume that $w \geq 2$ nodes are compromised using the strategy in Observation 2. Based on Observations 1 we can consider an upper bound on the attacker's knowledge as has captured also all of the nodes, n_{i+1}, between two consecutive compromised nodes, n_i and n_{i+2}. (except the privacy attack target node). In the above example, when the attacker captures nodes n_2 and n_4, we assume that the node n_3 has also been captured. That is, we bound the *Type 2* knowledge of the attacker by the *Type 1* knowledge.

In conclusion, if the attacker controls w nodes, we assume it has *Type 1* knowledge over $2+2(w-2)$ nodes. Referring to the Example in Figure 8,

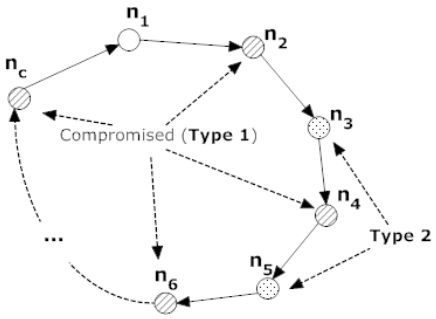


Fig. 8. Best attack strategy.

we assume the attacker has *Type 1* knowledge of the nodes n_c, n_2, n_3, n_4, n_5 and n_6.

To ease exposition, in the following analysis, we assume that each twin-key is shared by only two nodes in a cluster. In Figure 9 we report the simulation results for the probability, $p_i(k)$; that is, the probability that the same symmetric key k is considered as twin-key by i nodes in the cluster, assuming the parameters identified in Section 7.1. From this figure, we can notice that the probability that a key is actually shared between more than two nodes is quite small, i.e. less than 4% in this example.

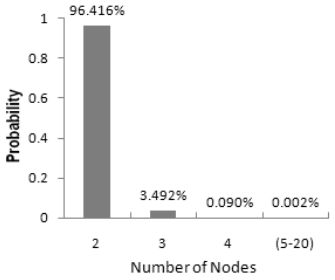
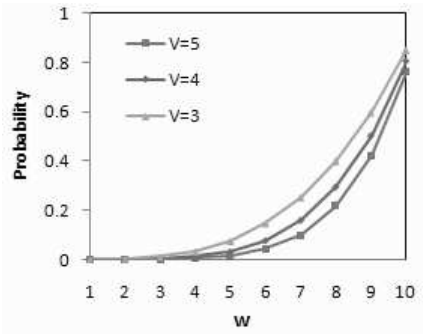


Fig. 9. Probability, $p_i(k)$, that a twin-key k is shared among i nodes in the cluster.

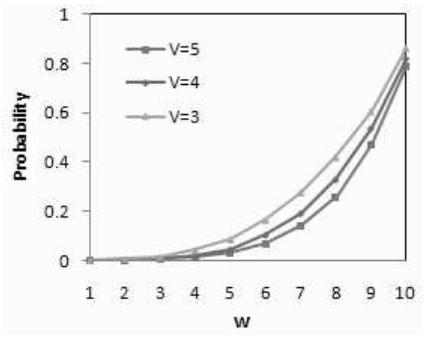
Furthermore, we assume that the attacker controls w nodes. From the previous upper bound on the attacker knowledge, the adversary has *Type 1* knowledge for $2 + 2(w-2) = 2w-2$ nodes; the probability that the adversary has knowledge of a single key of n_i is $\left(\frac{2w-2}{C-1}\right)$. Then, the upper bound probability that the adversary knows all the V keys a node uses in a data aggregation is:

$$\left(\frac{2w-2}{C-1}\right)^V \tag{7}$$

Figure 10(a) shows the above probability for the attacker to compromise the privacy of a node by capturing w nodes (x-axis). As an example, for $w = 5$ (the attacker has knowledge of the key ring of $2+2(w-2) = 8$ nodes, each one composed of $K = 65$ symmetric keys) we have that the attack's success probability is some $0.01, 0.03$, and 0.07 for $V = 5, 4$, and 3 respectively. The analytical result shown in Figure 10(a) is confirmed by the simulation result shown in Figure 10(b). Note that the simulation is done without the assumption of any constraint on the number of nodes sharing a twin-key.



(a) Analysis result



(b) Simulation result

Fig. 10. Probability of privacy compromising, varying the number of nodes captured by the attacker. $P = 10,000$, $K = 65$, $C = 20$, $A = 5$.

Active attack In the following we analyze an *active* attacker, that is an attacker that leverages the w controlled nodes to push forward the privacy compromising of a node n_i.

A necessary condition to compromise the privacy of n_i is to control the neighbors of n_i, as previously discussed. Assume the attacker actively controls these nodes during the twin-key liveness announcement phase (described in Section 6.1). Then, it can be able to enforce n_i not to participate in the aggregation (described Section 6.2). In fact, using the neighbor of n_i, the attacker can let n_i believe that all of the twin-keys it agreed on in the twin-key agreement (described in Section 5), are not alive during the current aggregation phase. We observe that this does not imply any privacy violation of node n_i.
However, if the attacker controls the neighbors of n_i during the twin-key agreement (note that the set-up phase is performed only once) it can do something more. Actually, controlling n_i's neighbors the attacker can try to let n_i agree on twin-keys only with the n_i's neighbors (nodes controlled by the attacker). If the attacker does not have enough keys (V), n_i will simply not participate in the aggregation for this cluster. However, if the attacker has enough keys to let n_i agree on V keys known by the attacker, this can pose a serious threat to the protocol: The privacy of the node could be violated. To solve this problem, and also to increase the resilience of our protocol against other kind of attacks, we propose an extension of our protocol by using multiple logical circuits.

We finally observe that our analysis related to the nodes compromision (passive and active attacker cases) is generic in terms of the type of node (CH or non CH) being compromised. In fact, an hijack of a CH cannot cause more serious threats than the hijack of any other node in the cluster. The CH does not have any privileged informations or duties compared to the other nodes in the cluster but to participate in the final aggregation (point 3.1 of Section 4). We remind that in this paper we are addressing the privacy (not the security) of the data aggregation. The participation in the final aggregation does not help the CH to violate the privacy of the nodes that already participated in the cluster aggregation. Furthermore, as for the security of the aggregation (that is out of the scope of this paper) we observe that any node (not just the CH) can inject an arbitrary amount of error. However, to address this problem other aggregation verification protocols such as [3, 25, 26] can be used, as discussed in Section 2.

Using multiple logical circuit to improve the protocol resilience To address the problem exposed in the previous section, we extend our protocol using multiple logical circuits in the twin-key agreement phase, as shown in Figure 11. Then, when a node initiates a twin-key agreement, it randomly selects one of the available circuits. This requires the attacker to control an higher number of nodes to achieve the same goal as in the single circuit scenario. In general, to compromise the privacy of a given node n_i, the attacker must capture all the other $C-1$ nodes, if $\frac{C!}{2}$ logical circuit are used.

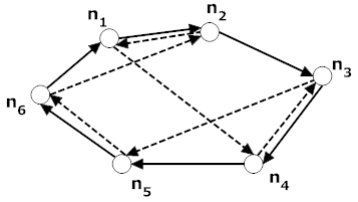


Fig. 11. Using multiple circuits.

Managing off-line nodes and message loss In Section 6.1 we described the twin-key liveness announcement mechanism that allows our protocol to be resilient to a node failure. In fact, a correct aggregate can be computed also if some nodes are off-line. For easy of exposition, in Section 6.1 we described our solution considering only the alive nodes. However, a technique for traversing a circuit with off-line nodes must be used. Here, we briefly explain a simple solution to achieve this goal. We can assume that each node broadcasts the ids of its next (right) and previous (left) neighbor, for each logical circuit built. In this way, each node can autonomously know the sequence of nodes for each circuit. Then, after sending a message to its neighbor, the sender

node waits for an acknowledge message, *ack*. If the ack is not received, the sender sends the message to the next node in the circuit. For example, in Figure 12 node n_4 sends a message to n_5 and waits for an ack. If the ack is not received within an expected time interval, n_4 sends the message to the next node in the circuit, n_6.

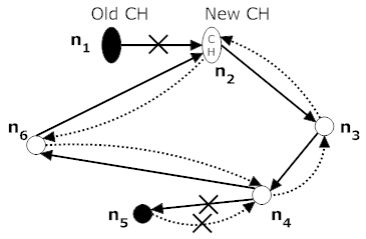


Fig. 12. Managing off line nodes.

In a similar way, if the CH does not start the protocol within a given time interval, the next node (n_2 in Figure 12) can assume the role of CH sending the first message. Note that message loss problem can be treated in the same way. If the message or the corresponding ack is lost, the destination node is regarded as off line.

The mechanisms described here require some time-outs for the nodes to react. As the aim of this paper is to provide a generic solution we are not assuming any specific setting. We leave to the network user the setting of these time-outs as a trade off between the following two factor: (i) promptness of the aggregation, (ii) number of nodes actually participating in the aggregation. In fact, if a prompt aggregation is required the time-out will be small but some nodes will be excluded from the aggregation—not because actually off-line but because they will not be able to react within the time-out. On the other side, if the time-outs are set to higher value, the nodes will have more time to react but this will require more time for the overall aggregation.

7.3 Complexity Analysis

In this section, we analyze the complexity of our proposed algorithm from both the communication and the computation point of view.

First, we analyze the complexity of the algorithm in the data aggregation phase (described in Section 6). Note that this is the major part of the overhead. In fact, such complexity is associated to each single aggregation. Finally, we discuss the complexity of the set-up phase (described in Section 5) that, instead, is performed only once.

Complexity in Data Aggregation Phase First, we analyze the computation complexity of the data aggregation. We consider the per node worst case overhead. For each agreed twin-key, k, each node has to compute two hash values. One hash is computed for the verification of the liveness announcement of k. The other hash is executed to compute the k's corresponding shadow value added in the aggregated value. Furthermore, each node has to compute two symmetric key encryption and two symmetric key decryption to receive and to send out the required message. Therefore, the computation of each node in the worst case is $2A$ hash computations, 2 symmetric key encryption and 2 symmetric key decryption, considering A agreed twin-keys. The overall computational complexity is in general $O(1)$, with respect to the size of the cluster, C, as reported in Table 2.

From the communication complexity point of view, each node only needs to receive and to send out two messages. That is, the number of messages is $O(1)$. However, we should also considered the size of these messages. Each node has to send out the hash values corresponding to its twin-keys that have not yet been declared alive by other nodes (Procedure 6, lines 10-15). For example, consider the message received by n_2, in Figure 3, during the twin-key liveness announcement. CH sent a message announcing k_1, k_7 and k_8. Now, the set of twin-key for n_2 contains k_4, k_7, and k_9, therefore it will remove k_7 from the message and will add all of its twin-keys not yet declared alive, k_4 and k_9. Consider each node established A twin-keys. In the worst case, $AC/2$ different twin-keys have to be declared in the twin-key liveness announcement message. We recall that we considered a declared (or established)

twin-key as the symmetric key shared between a pair of alive nodes. For example, if the same symmetric key, k, is used in two different pairs of nodes it is considered as two different established twin-keys. Then, the worst case message size is $AC/2$. That is, the size of message is $O(C)$. Considering the $O(1)$ number of messages, the overall communication overhead is $O(C)$, as reported in Table 2.

Next, we estimate the average message size $M_{avg-size}$. Let L_k be the average number of messages each alive twin-key, k, appears in. Assuming each node is alive, the liveness of all the twin-keys is checked when the message comes back to CH for the first time (when CH receives the liveness message in Procedure 7 the *List* structure is empty). Each twin-key k, out of the $AC/2$ twin-keys, will be in L_k messages out of the C messages sent in the circuit. Then, we have that:

$$M_{avg-size} = \frac{\frac{AC}{2} L_k}{C} = \frac{AL_k}{2}. \quad (8)$$

Furthermore, given a twin-key k, we define the function $m(k)$ as the number of nodes in the cluster that use the symmetric key k as a twin-key. Then, we have that $L_k = C/m(k)$. We define $p_i(k)$ as the probability that $m(k) = i$, where $i \in [2 \dots C]$. Equation 8 can be rewritten as:

$$M_{avg-size} = \frac{A}{2} \sum_{i=2}^{C} p_i(k) \frac{C}{i}. \quad (9)$$

Finally, rewriting $p_i(k)$ in terms of pool size, P, and key-ring size, K, we have:

$$M_{avg-size} = \frac{A \sum_{i=2}^{C} \binom{C}{i} \left(\frac{K}{P}\right)^i \left(\frac{P-K}{p}\right)^{c-i} \left(\frac{C}{i}\right)}{2 \sum_{i=2}^{C} \binom{C}{i} \left(\frac{K}{P}\right)^i \left(\frac{P-K}{p}\right)^{c-i}}. \quad (10)$$

Assuming each twin-key shared by exactly two nodes, a more tight average message size applies: $AC/4$—that is, $O(C)$ if A is a constant.

Complexity in Set-up Phase We consider first the communication complexity. Each node has to test, in the worst case, each of its pre-distributed keys to find out the required A twin-keys shared with other nodes. Therefore, up to CK keys will be declared in the message passing through the circuit, where K is the number of the pre-distributed keys in each node and C is the cluster size. Remember that our protocol binds to r the number of keys that a node can declare each time it has the declaration message (Procedure 2, lines 3-4 and Procedure 3, lines 37-38). As an upper bound, we can also assume that R is the overall maximum message size. Then, the upper bound for the total number of messages transferred by each node is CK/R.

Finally, we consider the computation complexity under the same assumptions. Each node, in the worst case, has to compute the hash for each of its pre-distributed keys and to encrypt each message it sends out. Altogether, we have K hash computations and CK/R encryptions.

7.4 Comparison

In Table 2 we summarize the features of our proposal compared with other relevant algorithms present in the literature. The feature *aggregation type* indicates who is responsible for the aggregation: *hop-by-hop* means that each node adds its own value to the aggregate while *CH* means that the local aggregation is performed by the cluster head. The column *encryption type* indicates who are the peers of the encryption considered in the protocol. As an example, *node-to-BS* means that each node encrypts some data that cannot be decrypted until it reaches the BS. Table 2 also indicates if the protocol protects privacy against outside eavesdropper, other network nodes or the BS, in columns 3, 4, and 5 respectively. The last two columns denote the per node computational and communication complexity. By *data-loss resilience* we refer whether the BS fails to compute the correct aggregate if a few nodes do not participate in the protocol or if a message is lost. Note that in this paper, we consider that the aggregation protocol cannot send neither the list of the nodes participating in the current aggregation, nor the list of those who do not participate. We renounced to formulate such an assumption since it can be well the case where such lists are $O(n)$, hence vanishing the savings that an aggregation protocol is supposed to deliver.

	Aggregation type	Encryption type	Privacy vs. outsiders	Privacy vs. other nodes	Privacy vs. BS	Data-loss resilience	Node comput. complexity	Node comm. complexity
Girao et al. CDA protocol [14]	hop-by-hop	CH-to-BS	Yes	No	No	**Yes**	$O(1)$	$O(1)$
Castelluccia et al. protocol [2]	hop-by-hop	node-to-BS	Yes	**Yes**	No	No	$O(1)$	$O(1)$
He et al. CPDA protocol [16]	CH	node-to-CH	Yes	**Yes**	**Yes***	**Yes***	$O(C^2 \log C)$	$O(C)$
He et al. SMART protocol [16]	hop-by-hop	node-to-node	Yes	**Yes**	**Yes**	No	$O(1)$	$O(C)$
Mlaih et al. protocol [23]	hop-by-hop	node-to-BS	Yes	**Yes**	No	No	$O(1)$	$O(1)$
Our solution	CH	node-to-node	Yes	**Yes**	**Yes**	**Yes**	$O(1)$	$O(C)$

(* the original protocol can be extended to achieve this property)

Table 2. Private Data Aggregation protocols: Comparing the security features and complexities.

8 Conclusion

In-network data aggregation is commonly used in sensor network for efficiency reason. However, in privacy sensitive applications, a single node can be interested in contributing its sensed value to compute an aggregate, but it would like to have neither other nodes nor the BS to know the value it contributed. Further, data aggregation has to be resilient to data loss. In this work, we proposed an efficient solution for data aggregation that protect the node privacy, according to the above requirement, against a powerful attacker. In particular, the attacker can eavesdrop the exchanged messages, control a fraction of the network nodes, and also acquire some information from the base station.

Our analysis supports the feasibility of our private aggregation protocol, showing that it is secure, scalable, resilient to data loss, and efficient. To the best of our knowledge, our scheme is the first one that provides the above properties all at once.

References

1. R. Agrawal and R. Srikant. Privacy-preserving data mining. In *SIGMOD Conference*, pages 439–450, 2000.
2. C. Castelluccia, E. Mykletun, and G. Tsudik. Efficient aggregation of encrypted data in wireless sensor networks. In *The Second Annual International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous '05)*, pages 109– 117, 2005.
3. H. Chan, A. Perrig, and D. Song. Secure hierarchical in-network aggregation in sensor networks. In *Proceedings of the 13th ACM conference on Computer and communications security (CCS '06)*, pages 278–287, 2006.
4. H. Choi, S. Zhu, and T. F. La Porta. SET: Detecting Node Clones in Sensor Networks. In *Proceedings of IEEE 3rd International Conference on Security and Privacy in Communication Networks (SecureComm '07)*, 2007.
5. M. Conti, R. Di Pietro, and L. V. Mancini. ECCE: Enhanced cooperative channel establishment for secure pairwise communication in wireless sensor networks. *Ad Hoc Networks*, 5(1):49–62, 2007.
6. R. Cramer, I. Damgard, and S. Dziembowski. On the complexity of verifiable secret sharing and multiparty computation. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing (STOC '00)*, pages 325–334, 2000.
7. R. Di Pietro, L. V. Mancini, and A. Mei. Energy efficient node-to-node authentication and communication confidentiality in wireless sensor networks. *Wireless Networks*, 12(6):709–721, 2006.
8. R. Di Pietro, L. V. Mancini, A. Mei, A. Panconesi, and J. Radhakrishnan. Sensor networks that are provably resilient. In *Proceedings of IEEE 2nd International Conference on Security and Privacy in Communication Networks (SecureComm 2006)*, pages 1–10, 2006.
9. R. Di Pietro, L. V. Mancini, A. Mei, A. Panconesi, and J. Radhakrishnan. Redoubtable sensor networks. *ACM Transactions on Information and System Security (TISSEC)*, 11(3):1–22, 2008.
10. J. Domingo-Ferrer. A provably secure additive and multiplicative privacy homomorphism. In *Proceedings of the 5th International Conference on Information Security (ISC '02)*, pages 471–483, 2002.
11. L. Eschenauer and V. D. Gligor. A key-management scheme for distributed sensor networks. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS '02)*, pages 41–47, 2002.
12. A. Evfimievski, R. Srikant, R. Agrawal, and J. Gehrke. Privacy preserving mining of association rules. In *Proceedings of 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '02)*, 2002.
13. W. F. Fung, D. Sun, and J. Gehrke. Cougar: the network is the database. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data (SIGMOD '02)*, pages 621–621, 2002.

14. J. Girao, D. Westhoff, and M. Schneider. CDA: concealed data aggregation for reverse multicast traffic in wireless sensor networks. In *2005 IEEE International Conference on Communications, 2005. (ICC 2005)*, pages 3044–3049, 2005.
15. J. Halpern and V. Teague. Rational secret sharing and multiparty computation: extended abstract. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing (STOC '04)*, pages 623–632, 2004.
16. W. He, X. Liu, H. Nguyen, K. Nahrstedt, and T. Abdelzaher. Pda: Privacy-preserving data aggregation in wireless sensor networks. In *26th Annual IEEE Conference on Computer Communications (INFOCOM 2007)*, pages 2045–2053, 2007.
17. http://firebug.sourceforge.net. The firebug project, 2008.
18. http://www.cens.ucla.edu. James reserve microclimate and video remote sensing, 2008.
19. http://www.greatduckisland.net/. Habitat monitoring on great duck island, 2008.
20. http://www.xbow.com. Crossbow technology inc., 2008.
21. D. Liu and P. Ning. Establishing pairwise keys in distributed sensor networks. In *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS '03)*, pages 52–61, 2003.
22. S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: a tiny aggregation service for ad-hoc sensor networks. In *Proceedings of the 5th symposium on Operating systems design and implementation (OSDI '02)*, pages 131–146, 2002.
23. E. Mlaih and S. A. Aly. Secure hop-by-hop aggregation of end-to-end concealed data in wireless sensor networks. In *arXiv:0803.3448v1 [cs.CR]*, 2008.
24. R. Rivest, L. Adleman, and M. Dertouzos. On data banks and privacy homomorphisms. *Foundations of Secure Computation*, pages 169–179, 1978.
25. S. Roy, M. Conti, S. Setia, and S. Jajodia. Securely computing an approximate median in wireless sensor networks. In *Proceedings of the Forth International Conference on Security and Privacy in Communication Networks (SecureComm 2008)*, 2008.
26. S. Roy, S. Setia, and S. Jajodia. Attack-resilient hierarchical data aggregation in sensor networks. In *Proceedings of the fourth ACM workshop on Security of ad hoc and sensor networks (SASN '06)*, pages 71–82, 2006.
27. D. Wagner. Resilient aggregation in sensor networks. In *SASN '04: Proceedings of the 2nd ACM workshop on Security of ad hoc and sensor networks*, pages 78–87, 2004.
28. Y. Yang, M. Shao, S. Zhu, B. Urgaonkar, and G. Cao. Towards event source unobservability with minimum network traffic in sensor networks. In *Proceedings of the First ACM Conference on Wireless Network Security (WiSec 2008)*, pages 77–88, 2008.
29. Y. Yang, X. Wang, S. Zhu, and G. Cao. SDAP: a secure hop-by-hop data aggregation protocol for sensor networks. In *Proceedings of the 7th ACM international symposium on Mobile ad hoc networking and computing (MobiHoc '06)*, pages 356–367, 2006.
30. A. Yao. Protocols for secure computations. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS '82)*, pages 160–164, 1982.