Accountant: Protection of Data Integrity and Identification of Malicious Nodes in In-network Data Processing

David Jost and Mathias Fischer
Universität Hamburg, Germany

Keywords: Data Aggregation, Distributed Networks, Integrity, Attacker Identification, Sensor Networks.

Abstract: Data integrity in distributed data sensing and processing platforms or middlewares is an important issue, especially if those platforms are open to anyone. To leverage the resources of participating nodes and to enhance the scalability, nodes can be included in the data processing, e.g., in the aggregation of results. In an open system, it is also likely that some participating nodes are malicious and lie about their sensed values or about the results of data processed by them. Current approaches that preserve data integrity for in-network processing require expensive cryptographic operations. With Accountant we propose a new approach, which requires significantly less computation at the expense of slightly more signalling overhead. Furthermore, our approach cannot only preserve data integrity, but also allows to identify malicious nodes. For that, Accountant uses multiple inner node-disjoint trees for data dissemination and hash trees for preserving the data integrity. We compare it to existing solutions, showing that with only minor additional messaging overhead, Accountant can protect the data integrity and can identify attackers at the same time.

1 INTRODUCTION

Smart cities are in close reach. They benefit from the rise of the Internet of Things (IoT) that manifests in a multitude of deployed sensors that add to the knowledge base of cities. These sensors, e.g., raise large quantities of environmental data that is of interest to (city) governments, organisations, but also citizens (Bornholdt et al., 2019, Kumar and Madria, 2013).

To orchestrate sensors, and to store and process the resulting data volume, distributed data sensing and processing platforms or middlewares are deployed, e.g., SANE (Bornholdt et al., 2019, Villanueva et al., 2013, Mohamed et al., 2017). Some of these distributed platforms also come with in-network data processing to make efficient use of the resources of participating nodes, e.g., for data aggregation (Fasolo et al., 2007). Relaying nodes aggregate received data to forward only single values and to reduce the data to be exchanged.

However, especially in open data sensing and processing platforms, nodes can be malicious and might inject false data. The state of the art contains several approaches, e.g., (Mahimkar and Rappaport, 2004, He et al., 2008, Chen et al., 2012, Kumar and Madria, 2013), that can protect the data integrity during such distributed data processing. However, all of them use expensive cryptographic operations.

The main contribution of this paper is *Accountant*, an *integrity* preserving data processing scheme that can *identify* malicious nodes injecting either faked values or suppressing the results of other nodes. The approach focuses on data sensing and processing platforms, like the SANE platform (Bornholdt et al., 2019), in which in theory every node could communicate with any other node. Another assumption is that nodes contribute to the measurements, directly when the node is a sensor or indirectly when the node is acting on behalf of attached sensors. For this reason, it cannot be applied directly to Wireless Sensor Networks (WSNs), but might be used to interconnect the base stations of different WSNs. Every node joining the system is contributing its resources for data sharing and processing. To ensure that the in-network processing produces the correct outcome, Accountant uses multiple *inner node-disjoint overlay trees*. As a result, malicious nodes cannot lie about the values of their children anymore. They can only influence the result in the tree they are actually forwarding information. In all other trees they are in a leaf position. Accountant even goes one step further and allows to identify malicious nodes. For that, the integrity of exchanged information is protected by

Jost, D. and Fischer, M.
Accountant: Protection of Data Integrity and Identification of Malicious Nodes in In-network Data Processing.
DOI: 10.5220/0008974405610568
In *Proceedings of the 6th International Conference on Information Systems Security and Privacy (ICISSP 2020)*, pages 561-568
ISBN: 978-989-758-399-5; ISSN: 2184-4356

mapping exchanged message values to *hash trees*. We evaluate the performance of Accountant by comparing its message overhead and cryptographic overhead to existing approaches. Our analysis indicates that Accountant requires less computational power at the expense of slightly more communication overhead.

The remainder of this paper is structured as follows: Section 2 describes the considered attacker model, followed by related work in the field of integrity preservation in distributed networks in Section 3. Accountant is described in detail in Section 4 and analyzed in Section 5. Finally, Section 6 concludes the paper.

2 ATTACKER MODEL

In an open and distributed data sensing and processing platform, data is contributed by most nodes in the network and processed in a distributed fashion. Nodes that forward data for others can modify, aggregate, and relay data of other nodes. This results in different attacker models, which attempt to modify the final aggregation result, described in the following:

- A node may *lie* about its own sensed values, but does not lie consistently across different aggregation trees. This misinformation propagates in the aggregation process and results in a wrong overall result.
- The attacker may *alter* the data it receives from other nodes, which it is supposed to aggregate. The forwarded data then does not represent the children's original data anymore.
- A node may *deny* or suppress the data it received from child nodes.

Furthermore, these three attacker types can come at different numbers, and can either operate independently or in a colluding manner.

3 RELATED WORK

There exists a large body of work in the area of sensor networks, and distributed data processing in such networks (Fasolo et al., 2007). These approaches are enabling aggregation over arbitrary networks, but cannot preserve the integrity of the aggregated data. Certain concepts also take data integrity into account (Mahimkar and Rappaport, 2004,He et al., 2008,Chen et al., 2012, Kumar and Madria, 2013). Even though these integrity preserving approaches are considered for low power environments, they require either relatively complex asymmetric cryptography, or a lot of additional messaging. The following describes some of those works, and highlights their problems in regard to our goal of providing efficient integrity protection and liar identification in distributed networks.

SecureDAV (Mahimkar and Rappaport, 2004) uses clusters to group nodes. Each cluster has special node, the *cluster head*, which aggregates the data of all the other nodes in the cluster. Afterwards, the cluster head sends the aggregate back to its children, so that they confirm the legitimacy of the aggregation. The child nodes verify that the aggregate lies within a certain threshold. If so, they sign the aggregate with their part of a *secret share*, which the cluster head subsequently combines, to obtain a full signature for the base station. Should not enough sensors confer with the aggregation of the cluster head, the resulting signature would be invalid. Additionally, the sensors sign their values with a private key, from which the cluster head constructs a *Merkle Hash Tree* (Merkle, 1990), verifiable by the base station. This avoids over-reliance on the cluster heads. SecureDAV requires asymmetric cryptographic operations, resulting in a lot of computational load on sensors. Even though the Merkle Hash Tree provides integrity, it is not used to identify an attacker.

Sensors in iPDA (He et al., 2008) split their data into multiple *slices*, encrypt them with pre-shared keys, and send them to different aggregators. With this, a sensor hides its value from a single aggregator, preserving its data privacy. Intermediate nodes can (additively) aggregate the received values and send their aggregate towards the base station. Integrity is provided by using multiple *node-disjoint trees*, where nodes send their results over both trees, which do not share any common nodes (except root). iPDA as well does not provide any means to identify the source of the mismatching values.

RCDA (Chen et al., 2012) provides data integrity and raw data recoverability by encoding sensor data into the stream of messages, allowing the base station to recover all data and verify the result. The encoding is effectively the original data stringed together. It is encrypted with *Elliptic Curve ElGamal (EC-EG)* and signed with the *homomorphic signature* scheme by Boneh et al. (Boneh et al., 2003), both of which allow for additive aggregation. In the end, the base station is able to decode and verify all measurements. This approach sends the entire raw data up to the base station for it to verify. Additionally, it uses asymmetric cryptography and does not identify the lying node.

Lastly, PIP (Kumar and Madria, 2013) describes a scheme where nodes first encrypt their data with *EC-EG*, before *splitting* it and distributing it to neighbouring nodes. This provides a certain anonymity towards

the cluster head, which will eventually receive all parts from all its children, without being able to attribute it to one in particular. Integrity is provided via an embedded *integrity key*, which can only be verified by the base station. Aggregation is performed as of the homomorphic properties of Shamir's Secret Sharing (SSS) and Recursive Secret Sharing (RSS, (Parakh and Kak, 2010)) and a scrambling key. Similar to the other approaches, PIP uses asymmetric cryptography and does not detect which node actually lied.

These approaches show different ways on employing data integrity for in-network aggregation. Neither of them takes attribution into account and most require rather intensive cryptographic operations. Thus a new approach must integrate the identification of attackers into the integrity preservation while staying lightweight.

4 SYSTEM MODEL

To perform aggregation in a structured manner, Accountant utilises *overlay trees* to organise message routing. It uses multiple *inner node-disjoint* trees, so that data aggregation is performed over all trees in parallel. The differently structured trees allow to disclose lying nodes and prevent denying nodes to suppress results of any of its children. By using enough trees, a majority of valid results will be established, exposing any incorrect behaviour. To identify the attacker, cryptographic hashes in the form of a *hash tree* are used. This allows Accountant to follow mismatching hashes and pinpoint the lying node in case the aggregation results of different trees do not match. Sections 4.2 and 4.3 describe the overlay trees and hash tree in more detail.

Accountant operates in two major phases. The first one is *data aggregation* (cf. Section 4.4), where the nodes organized in overlay trees answer the request for data that is flooded in the trees starting from the root node r. The data aggregation starts at the leaf nodes that forward their values to their parent nodes. Intermediate nodes aggregate their value with the ones obtained from their children, and forward the result to their parents in all trees again. In addition, each node adds a unique hash to its message. Aggregating nodes combine those hashes according to the rules of the hash tree computation. Reaching the root node, the aggregated data of each overlay tree can be compared with the others. If there are differences in the results, a node has suppressed or lied about values of its children. This triggers the second phase.

In the second phase (cf. Section 4.5), *lying nodes are identified* by using the information of overlay trees and the hash tree. As the overlay trees provide a majority of correct values, the erroneous trees can easily be detected. The reported hashes of the erroneous trees are reproduced with hashes from correct ones, and the procedure follows level by level the mismatching hashes. In the end, Accountant requires only a subset of all hashes to actually identify the attacker.

4.1 Assumptions

For our approach, we use the following assumptions:

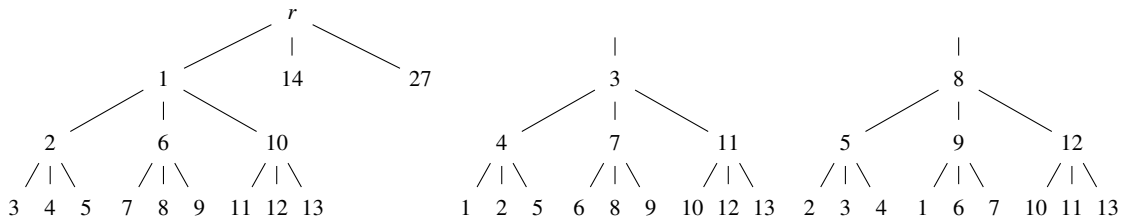
- All nodes share a unique (symmetric) key $(K_{i,r} \mid \forall i \in V)$ with the root node r, to allow them to authenticate their hashes without having to use asymmetric cryptography. We assume that every node has obtained a certificate once during an initial bootstrapping from the root r. Both derive the symmetric key from this as well as the unique request id, allowing them to share the secret without directly exchanging it.
- Each node is an intermediate node in at most one overlay tree and a leaf node in all others, as described in Section 4.2.
- All nodes in the overlay can reach each other if required.

4.2 Overlay Tree Construction

For Accountant to note differences in the aggregation, it requires comparable aggregation results. These are received via different overlay trees. All trees share the same nodes, they are just differently ordered, such that any node is an intermediate node in at most one of those trees and a leaf node in all others (except the root node that always stays the same). These *inner node-disjoint trees*, as proposed by (Brinkmeier et al., 2009), allow do disclose a malicious node that lies about or drops the value of its children.

Figure 1 displays three trees with the same nodes, rearranged accordingly to exhibit the property of inner node-disjointness. The root node stays fixed, as it is the initiator of the requests. The graphic also illustrates why those trees are necessary in the first place: Assuming node 2 is malicious, then it is able to lie about the values of its children 3, 4, and 5 in T_1, without being noticed. But taking the other trees into account, 2 has no longer the ability to influence or dismiss the values any of its children sends to their parent without being noticed. Thus, at the top of the tree, the difference in values is visible.

For Accountant actually being able to identify in which tree the error occurred, a majority of valid trees



(a) T_1 (b) T_2 (c) T_3

Figure 1: Three trees, constructed as described in Section 4.2, where each node is intermediate in at most one tree. As r is fixed, it is enough to only regard one subtree here to show the constraints.

with valid results is required. Thus, with m being the number of attackers, $n = 2m+1$ trees (Lamport, 2006) are necessary to obtain a correct result.

To meet the requirement of any node being in an intermediate position in at most one tree, for all n trees, it is necessary to use trees with enough leaf positions available. Equation 1 shows that this is solved by setting the node degree d to at least the number of trees n. l represents the required amount of leaf nodes and N the number of total nodes in a tree.

$$l \geq n \cdot \lfloor N/d \rfloor \tag{1}$$

4.3 Hash Tree

To be able to identify misbehaving nodes, Accountant uses a Merkle Hash Tree (Merkle, 1990). As can be seen in Figure 2, each node's hash represents a leaf in the hash tree. They are then combined pairwise, leading to a final hash (h_{1-8} in Figure 2). This reduces the number of required hashes per level to represent the same data. A combined hash can be reproduced later on by combining the hashes it stems from.

Aggregating nodes of the overlay tree follow the schema of the hash tree to combine hashes of their descendants as soon as possible. As the overlay trees differ in their order, nodes may not pass on the same hashes in one tree as they do in another. Nonetheless hashes are always combined in the same order as dictated by the hash tree, which may result in aggregating nodes passing on more than one hash, before they are combined later on.

To indicate a hash of node i, generated in a specific overlay tree t, h_i^t is used.

4.4 Phase 1: Data Aggregation

The first phase of Accountant describes the request and aggregation of data. It may be divided into those two stages of *request* and *aggregation*. All actions are performed for each overlay tree individually. This

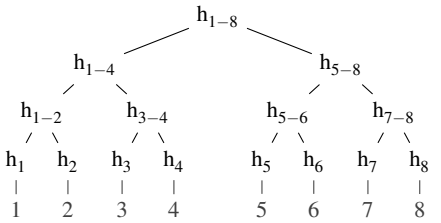


Figure 2: Design of a hash tree for eight sources.

allows us to focus on operations in one tree for now, though they are applied for each tree.

In the stage of *request*, the root node r floods a request for data on all available trees. This request is then answer via a reverse-multicast starting at the leaf positions in all overlay trees that are constructed as described in Section 4.2. This allows r to gather multiple views on the same data.

Aggregation: Leaf nodes answer directly with their data v_i and the hash h_i of that data, which is combined with the shared secret $K_{i,r}$ and a nonce ω (supplied by r during the request) of that request (cf. Equation 3) to make it unique. Intermediate nodes aggregate the values received by their children together with their own value (cf. Equation 2). Additionally, they extend the list of hashes with their own and combine any hashes according to the hash tree as given in Section 4.3 (cf. Equation 4). As of the fixed nature of the hash tree, it will not correspond to the structure of the overlay tree. Thus, nodes may send more than just one hash, though combining as many as possible.

$$\mathrm{agg}(i) = v_i \circ \left(\bigcirc_{j \in \mathrm{succ}(i)} \mathrm{agg}(j)\right) \tag{2}$$

$$\mathrm{h}_i = \mathrm{hash}(v_i, \omega, K_{i,r}) \tag{3}$$

$$\mathrm{data}_i = (v_i, \mathrm{agg}(i), \{\mathrm{h}_i, \ldots\}) \tag{4}$$

To provide an example, Figure 3 shows an excerpt of the sent data in T_1 (cf. Figure 1a). Here, node 3 sends it data up to its parent 2. Node 2, being an intermediate node, then combines all received values with its own, producing the aggregate $\mathrm{agg}(2,3,4,5)$. Additionally, the corresponding hashes are constructed: the hashes of nodes 3 and 4 can be combined to h_{3-4},

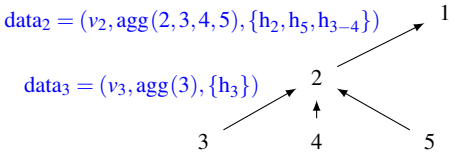


Figure 3: Example of data sent in phase 1.

so that node 2 sends the set of h_2, h_5, and h_{3-4} to its parent 1.

Upon receiving the aggregations from the multiple overlay trees, r can compare them with each other. If not all off them match, an inconsistency has been detected and phase 2 of identifying the liar can begin.

4.5 Phase 2: Identification of Lying Nodes

The second phase tries to iteratively narrow down the lying node. Starting at the root it follows unmatching hashes down the corresponding subtrees, eventually leading to the lying node, i.e., the attacker.

Due to the majority of well-behaving trees (cf. Section 4.2), we know which trees provided truth and can limit the search to the erroneous trees. In the example of one attacker ($m = 1$), this would also lead to only one tree being investigated, as the other two would match with each other.

Starting at the root node of the tree in question (this is repeated for all other erroneous trees), the delivered hashes from the direct children are verified to find the unmatching subtree. Would we now follow the normal style of a Merkle Hash Tree and verify sent hashes by combining the children's hashes, we would not be able to detect lower/earlier manipulations, as the henceforth combined hashes would verify the parent, though they actually stem from a hash of false data. Thus, we retrieve the hashes from other nodes, which also calculated those, but in another (correct) tree. This allows us to verify hashes in one tree with values and/or hashes from another. As an example, using the notation introduced in Section 4.3, the hash of h^1_{3-4} may be checked against $h_{3-4} = \text{hash}(h^2_3 \,\|\, h^3_4)$.

Is a mismatch detected, the procedure is repeated within that subtree in which the inconsistency has been identified. On the other hand, if the hashes of all children have successfully been verified, it indicates that the parent/current node is the actual liar.

Algorithm 1 displays the described logic. Functions `lefthash` and `righthash` return the child hashes required to construct the hash of i. The actual identification function `identify` checks the hashes of each subtree and continues to investigate into mismatching subtrees (cf. Line 7). If no mismatch has been found, this subtree is correct and the parent p is identified to be the malicious node (cf. Line 9).

Algorithm 1: Algorithm to detect a liar used in the second phase of Accountant.

```
1 func identify(p, hashes):
      Input: p = parent
      hashes = hashes {h_i^{t∈T}, ...}
      Data: T = erroneous trees
      Data: G = correct trees
2     for h_i ∈ hashes do
3         l ← lefthash(h_i)
4         r ← righthash(h_i)
5         if l = r = NIL and h_i ≠ h_i^{g∈G} then
              return i                    // i is a leaf
6         x ← hash(h_l^{p∈G} || h_r^{q∈G})
7         if x ≠ h then                   // erroneous subtree
8             return identify(i,
                  {h_j^t | j ∈ i^t.children})
9     return p                            // parent node is the liar
```

In conclusion, Accountant describes a protocol consisting of the normal *data aggregation phase* and the additional *liar identification phase*. It requests data over multiple aggregation trees, where the inner nodes are only part of the aggregation at most once. This allows Accountant to detect differentiations performed by *altering* and *denying* attackers. Using the simultaneously created hash trees, it *identifies* the attacker in a second phase.

5 EVALUATION

In this section, we will evaluate our presented approach, by comparing it to selected scenarios. The following analyses each of the selected approaches and compares them in regard to scalability. We use the extended notation shown in Table 1.

Number of Messages. First, we analyse the number of sent messages of one complete routine run, comparing simple, intuitive methods with existing work and our approach.

Direct connection: Each node sends its data directly to r, resulting in N messages sent. This is used as a baseline for comparison with the other approaches.

$$c_{A.1} = N \tag{5}$$

Table 1: Notation used in the evaluation.

m	Number of attackers
n	Number of trees (i.e. $n = 2m+1$)
N	Number of nodes per tree (i.e. $N = \|V\|$)
d	Node degree (i.e. $d \geq n$ for Accountant)
H	Height of a tree (i.e. $H = \log_d N$)
c_p	Costs of asymmetric encryption
c_a	Costs of homomorphic operations
c_h	Costs of (secret) hashing

Without Aggregation Over the Tree: Each node sends its data over the overlay tree to r. A message from a leaf node has to traverse also its parental links, resulting in l messages per node. This accumulates to $\sum_{i=0}^{H-1} id^i$ (cf. Equation 6) messages in total.

$$\begin{aligned} c_{A.2} &= \sum_{i=0}^{H-1} id^i \\ &= \frac{d - Hd^H + (H-1)\cdot d^{H+1}}{(1-d)^2} \\ &= \frac{d - \log_d N \cdot d^{\log_d N}}{(1-d)^2} \\ &\quad + \frac{(\log_d N - 1)\cdot d^{\log_d N + 1}}{(1-d)^2} \end{aligned} \tag{6}$$

SecureDAV (Mahimkar and Rappaport, 2004): In SecureDAV, each node sends its data to its parent, which aggregates it, asks all children for verification and then forwards it. This results in three messages per node. Additionally, integrity verification is performed over a Merkle Hash Tree, adding another $\log_2 2^{H+1}$:

$$c_{A.3} = 3 \cdot N + \log_d N + 1 \tag{7}$$

PIP (Kumar and Madria, 2013): Messages are split into x several pieces and distributed to a node's x neighbours. After that, nodes send the data to their cluster head/parent. This gives $2x-1$ messages per node, which can be summed up to $2N-3$, as the base station does not forward any messages.

$$c_{A.4} = 2 \cdot N - 3 \tag{8}$$

Accountant: Aggregating the data in the tree results in constant size messages, so that the level of the sender in a tree does not have any impact. We send $e = N-1$ messages up the overlay tree. Due to the necessary integrity preservation, this is done for all n trees. We do not add the overhead of the rarely performed liar identification, as this functionality is not part of the normal operation mode. It would otherwise add a less impactful amount of $m \cdot d \cdot \log_2 2^{H+1} = m \cdot n \cdot (\log_n N + 1)$ messages to the total.

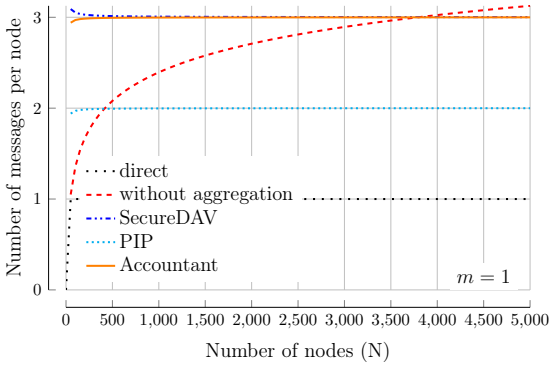


Figure 4: Number of messages a single node sends during a normal routine with increasing number of participating nodes for each of the described approaches.

Table 2: Scalability in regard to amount of messages given in big O notation.

Direct	$O(N)$
Without aggregation	$O\left(\log_d N \cdot d^{\log_d N}\right)$
SecureDAV	$O(N)$
PIP	$O(N)$
Accountant	$O(n \cdot N)$

$$c_{A.5} = n \cdot (N-1) \tag{9}$$

Figure 4 compares the costs of Equations 5 to 9. It shows that Accountant is comparably efficient as SecureDAV. The communication over a tree without aggregation performs, as expected, worst, as the tree is not leveraged in any way. PIP performs best, as it does not use back-and-forth communication like SecureDAV or multiple overlay trees like Accountant.

Table 2 shows the above results in comparable big O notation. Again, the tree communication without aggregation performs worst, all others are classified as $O(N)$, whereby Accountant depends on the number of trees, meaning effectively the number of attackers that are to be identifiable. Accountant performs worse, because it supplies this feature of attacker identification. If it would only provide integrity, the number of trees could be reduced to 2, leaving it to be better than SecureDAV, though still less efficient than PIP.

Cryptographic Load. In our second analysis, we highlight the differences in cryptographic overhead of the different approaches. We assume costs for asymmetric encryption (c_p), homomorphic operations (c_a), and hashing (c_h), to distinguish their impact on the

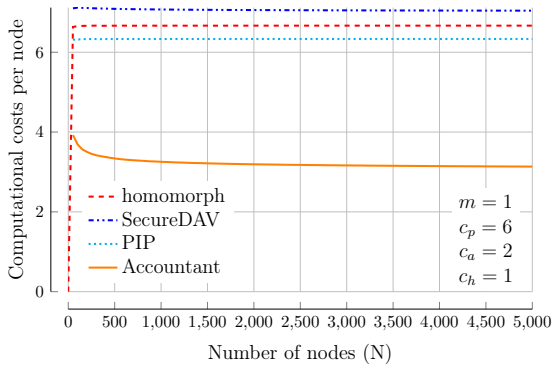


Figure 5: Costs of cryptographic operations a node has to perform during a single routine in comparison to increasing network size. The discussed approaches are shown, assuming asymmetric cryptography to be six times more expensive than hashing operations.

Table 3: Computational costs of cryptographic operations in big O notation.

Hom. Enc.	$O(c_p \cdot N + c_a \cdot N)$
SecureDAV	$O(c_p \cdot N + c_a/N + c_h \cdot N)$
PIP	$O\left(c_p \cdot N + c_a \cdot d^{\log_d N}\right)$
Accountant	$O(c_h \cdot n \cdot N)$

computational costs of the individual approach. We assume asymmetric encryption to be the most the resource intensive operation, and homomorphic operations to be only bit more intensive than ordinary hashing.

Homomorphic Encryption: We assume a system, where each node encrypts its data with an asymmetric, homomorphic cipher, e.g., ElGamal, for r. Nodes pass data to their parent, which in turn aggregates the data of its children and its own together as of the homomorphic properties of the cipher. This continues until the root node is reached. As a result, there will be N messages encrypted, while $\lfloor N/d \rfloor$ nodes also perform the aggregation. This will result in cryptographic costs of $c_{B.1}$, seen in Equation 10.

$$c_{B.1} = N \cdot c_p + \lfloor N/d \rfloor \cdot c_a \tag{10}$$

SecureDAV: Nodes do not encrypt their data, but partially sign the query of their cluster head for verification, which then combines the signatures. Both are asymmetric operations, yielding costs of $x \cdot c_p + c_a/x$ per node, resulting in $c_p \cdot (N-1) + c_a/(N-1)$. In addition, hashes for the Merkle Hash Tree are generated, costing $c_h \cdot \left(N + \lfloor (2^{H+1}-1)/2 \rfloor\right)$. This concludes with costs $c_{B.2}$ as given by Equation 11.

$$\begin{aligned} c_{B.2} = {} & c_p \cdot (N-1) + \frac{c_a}{N-1} \\ & + c_h \cdot \left(N + \left\lfloor (2^{\log_d N+1}-1)/2 \right\rfloor\right) \end{aligned} \tag{11}$$

PIP: In PIP, nodes firstly encrypt their data, before splitting and distributing it. The aggregation uses homomorphic properties, which together returns costs of $N \cdot c_p + c_a \cdot \lfloor (d^{H+1}-1)/2 \rfloor$.

$$c_{B.3} = c_p \cdot N + c_a \cdot \left\lfloor \left(d^{\log_d N+1} - 1\right)/2 \right\rfloor \tag{12}$$

Accountant: Each node hashes its data including a unique secret, shared with r. Thus we have N hashing operations and an additional $\lfloor (2^{H+1}-1)/2 \rfloor$ hashes (without secret) for the binary hash tree, combining existing messages to reduce message size. Including the hash operations required for verifying hashes in the second phase, this results in costs of $c_{B.2}$ as given in Equation 13.

$$\begin{aligned} c_{B.4} = {} & c_h \cdot \left(n \cdot \left(N + \left\lfloor \left(2^{\log_n N+1} - 1\right)/2 \right\rfloor\right)\right. \\ & \left. + m \cdot n \cdot (\log_n N + 1)\right) \end{aligned} \tag{13}$$

Figure 5 shows the above equations in comparison. We set asymmetric encryption to be six times more expensive than normal hashing (based on *EC-EG* and *SHA3* instruction costs, as approximated from (Gamal, 1985, Bertoni et al., 2009, Granlund, 2019, Fog, 2019)). As Accountant uses no asymmetric cryptographic operations at all, it performs best here. Even with more trees, it requires less computational power than the other approaches. The other three approaches scale in the same manner and their costs are mainly determined by the number of homomorphic operations.

Table 3 shows the corresponding abstraction of the above results. It is visible, that Accountant performs better, as long as $c_p \geq c_h \cdot n$ is valid, meaning it can use more trees to be able to detect multiple attackers, and still require less computational power than the other approaches.

6 CONCLUSION

In this paper we proposed *Accountant* to efficiently protect the integrity during data aggregation in distributed sensing and processing platforms. Our approach can detect attackers that either modify the values of their children or suppress them. Moreover, Accountant is able to detect multiple attackers in the network. It uses several inner node-disjoint aggregation

trees to utilise redundant aggregation paths to compare the outcome of the aggregation and uses hash trees to be able to determine attackers.

Due to its efficient cryptographic operations, Accountant's additional messaging overhead is compensated by the lower computational power required to perform. Accountant does only need to initiate the second phase of identification, if an attack has been recognised, lowering its output of messages and required work in the normal case.

Future work will investigate the reduction of the number of required trees, so that the overall network footprint can be reduced.

REFERENCES

Bertoni, G., Daemen, J., Peeters, M., and Assche, G. V. (2009). The road from panama to keccak via radiogatún. In Handschuh, H., Lucks, S., Preneel, B., and Rogaway, P., editors, *Symmetric Cryptography, 11.01. - 16.01.2009*, volume 09031 of *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany.

Boneh, D., Gentry, C., Lynn, B., and Shacham, H. (2003). Aggregate and verifiably encrypted signatures from bilinear maps. In Biham, E., editor, *Advances in Cryptology — EUROCRYPT 2003*, pages 416–432, Berlin, Heidelberg. Springer Berlin Heidelberg.

Bornholdt, H., Jost, D., Kisters, P., Rottleuthner, M., Bade, D., Lamersdorf, W., Schmidt, T. C., and Fischer, M. (2019). SANE: Smart networks for urban citizen participation. In *Proceedings of the IEEE 26th International Conference on Telecommunications (ICT 2019)*.

Brinkmeier, M., Schäfer, G., and Strufe, T. (2009). Optimally dos resistant p2p topologies for live multimedia streaming. *IEEE Transactions on Parallel and Distributed Systems*, 20(6):831–844.

Chen, C.-M., Lin, Y.-H., Lin, Y.-C., and Sun, H.-M. (2012). RCDA: Recoverable concealed data aggregation for data integrity in wireless sensor networks. *IEEE Transactions on Parallel and Distributed Systems*, 23(4):727–734.

Fasolo, E., Rossi, M., Widmer, J., and Zorzi, M. (2007). In-network aggregation techniques for wireless sensor networks: a survey. *IEEE Wireless Communications*, 14(2):70–87.

Fog, A. (2019). 4. instruction tables. https://www.agner.org/optimize/instruction_tables.pdf. Accessed on 2019-12-17.

Gamal, T. E. (1985). A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Information Theory*, 31(4):469–472.

Granlund, T. (2019). Instruction latencies and throughput for amd and intel x86 processors. https://gmplib.org/ tege/x86-timing.pdf. Accessed on 2019-12-17.

He, W., Nguyen, H., Liu, X., Nahrstedt, K., and Abdelzaher, T. (2008). iPDA: An integrity-protecting private data aggregation scheme for wireless sensor networks. In *Proceedings - IEEE Military Communications Conference MILCOM*.

Kumar, V. and Madria, S. (2013). PIP: Privacy and integrity preserving data aggregation in wireless sensor networks. *2013 IEEE 32nd International Symposium on Reliable Distributed Systems*, pages 10–19.

Lamport, L. (2006). Lower bounds for asynchronous consensus. *Distributed Computing*, 19(2):104–125.

Mahimkar, A. and Rappaport, T. (2004). SecureDAV: a secure data aggregation and verification protocol for sensor networks. In *IEEE Global Telecommunications Conference, 2004. GLOBECOM '04.*, volume 4, pages 2175–2179. IEEE.

Merkle, R. C. (1990). A certified digital signature. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 435 LNCS, pages 218–238.

Mohamed, N., Al-Jaroodi, J., Jawhar, I., Lazarova-Molnar, S., and Mahmoud, S. (2017). Smartcityware: a service-oriented middleware for cloud and fog enabled smart city services. *Ieee Access*, pages 17576–17588.

Parakh, A. and Kak, S. C. (2010). Recursive secret sharing for distributed storage and information hiding. *CoRR*, abs/1001.3331.

Villanueva, F. J., Santofimia, M. J., Villa, D., Barba, J., and Lopez, J. C. (2013). Civitas: The smart city middleware, from sensors to big data. In *IMIS, 2013 Seventh int. conf. on*, pages 445–450. IEEE.