A lifetime-optimized end-to-end encryption scheme for sensor networks allowing in-network processing

Frederik Armknecht, Dirk Westhoff*, Joao Girao, and Alban Hessler
NEC Europe Ltd.
Kurfürsten Anlage 36
69115 Heidelberg, Germany
phone +49 6221 4342 0, fax +49 6221 4342 155
{frederik.armknecht, joao.girao, alban.hessler, dirk.westhoff}@netlab.nec.de

Abstract
The use of converge-cast traffic and in-network processing to minimize the amount of transmitted data is a frequently used approach to increase the lifetime of a wireless sensor network (WSN). Consequently, when aiming at security for WSNs, one has to focus primarily on protecting this type of traffic. Some recent proposals support the encryption of converge-cast traffic with in-network processing. However, they either require the transmission of the sensors’ IDs, creating additional data overhead linear in the number of sensors, or require an elaborate key pre-distribution mechanism. In this paper we propose a solution for end-to-end encryption of converge-cast traffic with a simple key pre-distribution scheme causing additional data only logarithmic in the number of sensors. The scheme is robust with respect to unreliable channels, exhausted nodes and routing flexibility. It supports refreshing the keys at the nodes, which has so far been fully neglected.

Keywords: Wireless sensor networks, concealed data aggregation, key pre-distribution

1 Introduction
The work at hand addresses security mechanisms for large scaled wireless sensor networks (WSNs). WSNs deployed in a volatile, untrusted or even hostile environment must be equipped with security mechanisms which preferably should support self-healing aspects, and be robust against exhausting nodes, unreliability of the wireless broadcast medium as well as a multitude of external attacks.

*Corresponding author
Established solutions like TinySec (the security module of TinyOS) or standardised security protocols like AES which is recommended to be used in the RF standard IEEE 802.15.4 [7] are based on hop-by-hop encryption. However, as the sensor nodes’ hardware in envisioned large scaled sensor networks will for a considerable amount of applications most likely not be tamper-resistant, the overall system security is very limited when applying the hop-by-hop security paradigm to WSNs.

In previous work, some of the authors of this article presented solutions for the concealment of monitored and aggregated data within a WSN, e.g., the average, the variance, movement detection, or the sum of sensed values. The so called concealed data aggregation (CDA) approach aims at end-to-end encryption for converge-cast traffic in the presence of in-network processing. In [18, 2, 13] the focus is on end-to-end encryption of real-time traffic in synchronous WSNs whereas in [11, 5] the WSN is considered to be asynchronous and to act as a distributed database to persistently store sensed data in an encrypted and condensed manner. Both approaches use a specific class of encryption transformations to encrypt data, namely homomorphic encryption schemes. Such encryption transformations support basic operations on encrypted data which is of high value in the context of a WSN where multi-hop data transmission with in-network processing needs to be supported. Nodes can perform simple operations on incoming encrypted data to aggregate and to forward only a representative value of them. Homomorphic encryption schemes can be symmetric or asymmetric. The schemes from Domingo-Ferrer [4] and from Castelluccia, Mykletun and Tsudik [2] are symmetric. Since they are orders of magnitude faster than their asymmetric counterparts they are currently the only relevant for securing real-time responsive traffic in WSNs. Note that even a nested arrangement of these schemes has recently been proposed [16] with the aim to reduce the weakness of malleability. In [11], some of the authors of this article give an overview on relevant asymmetric homomorphic encryption transformations [12, 14, 15] which provide better system security at the price of much higher computation costs. Nevertheless, we argue that asymmetric homomorphic encryption transformations are preferable in real-time uncritical applications, e.g., for an encrypted and distributed long term storage of data within the WSN.

All the existing schemes have in common that either the overall system security is low [6], the identities of the nodes involved in the aggregation need to be recorded and transmitted [2], or that the key-distribution is elaborate [13, 1]. It is the contribution of this work to provide a CDA scheme that overcomes the above cited obstacles. It deploys a specific symmetric encryption scheme that is homomorphic both in the data and the keys, which we term as bihomomorphic. In similarity to the recently proposed approach Topology Aware Group Keying (TAGK) [18], we name the key pre-distribution scheme itself Topology Aware Unique Keying (TAUK).
2 Related Work

Only a few schemes have been proposed that support the encryption of convergecast traffic with in-network processing. All the proposed schemes are closely linked to the concrete encryption instantiation and its combination with a suited key pre-distribution scheme (KPD). The basic principles behind the CDA concept will more thoroughly be introduced in Section 5.

2.1 Key Pre-distribution for CDA with unique symmetric Key

In [18], the KPD Topology Aware Group Keying (TAGK) has been proposed for the usage of a symmetric privacy homomorphic encryption transformation for securing convergecast traffic with in-network processing. TAGK can be classified as a groupwise KPD with keys distributed per “routable” region. The scheme is robust against exhausting nodes and it provides a higher system security compared to single-hop based encryption approaches. However, TAGK is designed to support a symmetric homomorphic encryption transformation which requires the same key for all the encrypting parties like for example the scheme [4]. Therefore, in particular for WSN applications requesting highest system security there is a strong need for conceptual enhancements. We will refer to this CDA approach as Do-Fe/TAGK.

2.2 Key Pre-distribution for CDA with multiple symmetric Keys

In [2], Castelluccia, Mykletem and Tsudik proposed a key stream generator based CDA scheme which allows to use different symmetric keys per plaintext operand and which is additively homomorphic. By applying this scheme for securing convergecast traffic in WSNs each sensing unit encrypts with a different key. Under the assumption that one-time keys are used the scheme itself is provably secure. In addition, since pairwise keys are used, the proposed scheme provides the highest achievable system security. Also, the required KPD is as simple as possible: keys can be randomly distributed to the nodes and only the sink node needs to store all the keys. Since the storage of keys on the nodes is independent of the final position of the nodes, the KPD reduces to a simple storage of different unique keys before the nodes deployment. Such a simple KPD is nearly perfect for highly self-organizing distributed environments. Also, from the viewpoint of security, a pairwise keying model for convergecast traffic is preferable since it provides a higher system security (see Section 4). We will refer to this CDA approach as CMT.

2.3 Remaining Problems

The benefit regarding the overall system security of a KPD supporting CDA with multiple symmetric keys like proposed in [2] comes at the cost of additional
overhead. Firstly, the nodes’ configuration before node deployment requires a pairwise pairing between each sensor node and the sink node to agree on the shared key. Secondly, although the sink node is considered to be equipped with much more memory than a sensor node, since we are aiming at large scaled sensor networks, the storage of thousands or hundreds of thousands of pairs (ID, key) may turn out to be a serious problem. Thirdly, since we are aiming at security solutions for a highly unreliable medium one cannot ignore the impact of packet loss over the wireless broadcast medium. Revealing per data transmission the key IDs respectively node IDs of all the currently involved nodes becomes mandatory. To fully judge the above requirement, recall that for a static transmission radius the energy consumption for transmission linearly increases with the number of transmitted bits. Since transmission is the predominant energy consuming operation of a sensor node one can roughly say that doubling the data to be transmitted approximately halves the lifetime of a sensor nodes. For example, consider an 4-degree tree shaped WSN of height 4 consisting of \( n = \sum_{i=1}^{4} 4^i = 340 \) sensor nodes where each node aggregates and forwards the data coming from its children. Let the sink be at aggregation level 0, its children at aggregation level 1, and so on. Each node has its own ID which has a bit size of \( 9 = \lceil \log_2(340) \rceil \). That is the nodes at aggregation level 4, where actually no aggregation happens, have to send in addition to the ciphertext 9 bits for the ID. At level 3, each node has to transmit up to additional \( 4 \cdot 9 = 36 \) bits, 9 bits for each of its 4 children, plus its own ID, yielding 45 bits which have to be transmitted besides the ciphertext. One sees easily that the amount of data increases, the closer (in terms of hops) a node is to the sink. The nodes at aggregation level 2 have to send up to \( 4 \cdot 45 + 9 = 189 \) bits for all IDs and at aggregation level 1 up to \( 4 \cdot 189 + 9 = 765 \) bits. Let us assume that the sensed values stem from a limited value space of 8 bits size, so that the aggregation of 340 measurements requires \( \lceil \log_2(340) + 8 \rceil = 17 \) bits. A node at aggregation level 1 has to send 765 + 15 = 782 bits in total, which is almost 46 times bigger than the size of the ciphertext. Even if we take into account that the transmission overhead on average will be slower in practice as not all nodes might have send some data (see network model in Section 3), this example illustrates that a CDA scheme that requires to send the node ID significantly reduces the lifetime of the WSN. Observe that this effect depends only on the size of the WSN and not on the size of the measurements. Therefore, we believe that large scale WSNs require other CDA approaches which work without sending any IDs or sending only few IDs.

### 2.4 Authenticated Interleaved Encryption and Multiple Encryption

Similar observations may have stimulated one of the authors of [2] to propose an approach named Authenticated Interleaved Encryption (AIE) [1]. Recently

---

1Alternatively, one could consider an approach where only the IDs of non-participating nodes are transmitted. However, this would presuppose a rigid network structure, limiting the routing flexibility.
in [13] a comparable homomorphic encryption scheme with multiple encryption
has independently been proposed. The basic idea is as follows:

If using a keystream generator for encryption, the only reason for the sink
node to know the identifiers of the participating nodes is to be able to "subtract"
the keys. The authors propose to store the key of a sensor node a second time
on another node on the path to the sink node so that when the data passes this
node on its way to the sink, it can easily take off the key without compromising
its security as it has been aggregated with other values in the meantime.

However, bounding the amount of data to be transmitted for the IDs comes
at the costs of a reduced security (with respect to node capturing) and a rigid
network structure due to the routing of convergecast traffic. Together with a
structured distribution of keys the approach overcomes some of the shortcomings
of [2]. However, authors simply ignore the problem of KPD and assume that
keys have initially been stored on the nodes in a proper way. Moreover, these
savings come with the costs of a weaker system security. We will refer to this
CDA approach as ME.

Our Contribution

It is the contribution of this work to propose a solution for encrypted converge-
cast traffic which is

- almost optimal regarding the data overhead. Aggregated and encrypted
  convergecast traffic should be nearly of the same size than sending aggre-
gated convergecast traffic in plaintext,
- provides reasonable security with respect to an attacker’s rational cost
  window she is willing to invest for breaking the security architecture, and
- is robust against exhausted nodes and an unreliable broadcast medium.

The contribution is especially valuable for large scale WSNs as it does not require
the transmission of any node IDs.

3 Network model and traffic pattern

We assume that the WSN has the following properties: There is a single node,
called the sink node, with strong to moderate power and storage capacity. The
other nodes are sensor nodes and/or aggregator nodes. Once the network is
deployed, the sensor nodes continuously perform measurements and transmit
them via multi-hop to the sink node. For the sake of simplicity, we divide the
overall transmission time into discrete time frames such that each sensor makes
(at most) one measurement per time frame. To reduce the data overhead,
the measurements are not simply concatenated but aggregated to a shorter
representative value. That is at each time frame, a dedicated subset of all
nodes, the aggregator nodes, receive some data, aggregate them, and forward
the result into the direction of the sink. Eventually, the sink node receives the
aggregation of (some of) the measurements made within the network during this time frame. The nodes in the network including the sink node are stationary. However, their roles can change from time to time. Nodes can act as sensing nodes, aggregating nodes and/or as forwarding nodes. Also, to save energy, nodes may sometimes become idle. After their initial deployment, the nodes are left unattended. Since nodes might be idle, exhausted, or even stolen, it is rather unlikely that all nodes contribute within one time frame $t$ with a sensed value. Furthermore, the wireless communication channel is not fully reliable, such that messages might not have reached the destination. In the remaining text we refer to nodes from which no data reached any other node, independently of the exact reason, as *silent* nodes.

To model the network and this specific traffic pattern, we represent the WSN as a directed graph $T(V, E)$ where $V$ is the set of vertices and $E$ the set of edges. For this purpose, the set of edges is derived from the potential traffic flow which is identified during the initial deployment phase. A tuple $(N, N')$ with $N, N' \in V$ is an edge in $E$ if $N'$ might directly transmit some data during a time frame to $N$ with the intention that $N$ aggregates the data and forwards it to the sink. This yields a tree-like structure on the graph where the root is the sink node and the leaves are the sensor nodes which never aggregate other data. Consequently, we define for a node $N$ its *successors* by $\text{Succ}(N) := \{N' \mid (N, N') \in E\}$ and its *predecessors* by $\text{Pred}(N) := \{N' \mid (N', N) \in E\}$. In a similar manner, we define a *subtree* rooted at a node $N$ by

$$\text{ST}(N) := \{N' \mid \exists N_1, \ldots, N_s : (N, N_1), (N_1, N_2), \ldots, (N_s, N') \in E\}. \quad (1)$$

Observe that the model does not limit the routing flexibility and also covers node role changes. However, we assume that within one time frame, each node sends its data to only one another node. Thus, to model this kind of snapshot, we assume from now on that each node has exactly one well defined predecessor. We want to stress that this assumption is only introduced to make the description of the proposed scheme easier. It does not imply any limitations of the discussed scheme. In fact, it can be modified such that they support multiple predecessors, that is some flexibility in the routing.

### 4 Attacker Model and Costs

For a sound security evaluation a clear attacker model is necessary which specifies the expected goal, knowledge, and capabilities of an attacker.

**Goal:** We assume that an attacker is interested into not more than is required for the underlying application, i.e., to gain knowledge of the aggregated value which reaches the sink node. Out of scope are attacks which hinder the functionality like replay attacks, forging data or DoS attacks.

**Knowledge:** We apply Kerckhoff’s principle which says that an attacker knows principally everything about the system except of the secret keys. In
this case, she knows the network structure, the key distribution, and the deployed mechanisms.

Capabilities: An attacker fully controls the communication channel. She can eavesdrop, catch, destroy, modify, and send data anytime at her wish. For this purpose she is limited to a subregion of the network per time frame. Additionally she can physically capture nodes to read out the sensitive information stored at this node. Consequently we will distinguish for the security evaluation in Section 10.2 between cryptographic attacks, which purely aim at the security mechanism, and physical attacks which additionally compromise nodes.

We note that the WSN system security is (at best) a minimum function over all its security components, like e.g. cryptographic primitives, security protocol, key storage, etc. We further note that a rational attacker is never willing to pay more for an attack on the WSN as it would cost to buy and roll out another WSN in the same environment. Consequently a rational attacker’s effort to break the weakest system component will never exceed the price the attacker needs to pay for rolling out his own WSN. This qualitative attacker cost estimation significantly shrinks the attacker’s rational cost window which surely impacts the design decisions for the chosen security components.

5 Concealed Data Aggregation

Concealed data aggregation (CDA) has originally been introduced in [6, 8]. CDA is an approach that allows to combine in-network processing for reducing energy consumption on the one hand and end-to-end encryption for providing security on the other hand. This requires the use of an encryption mechanism which allows to aggregate different ciphertexts in such a way that the aggregation of the underlying plaintexts can be easily retrieved if the corresponding keys are known. Let \((\mathbb{P}, +), (\mathbb{C}, \oplus)\), and \((\mathbb{K}, \otimes)\) denote the plaintext space, the ciphertext space, and the key space, respectively, together with some binary operations. Furthermore, the encryption transformation is referred by \(E: \mathbb{K} \times \mathbb{P} \rightarrow \mathbb{C}\) and the corresponding decryption transformation by \(D: \mathbb{K} \times \mathbb{C} \rightarrow \mathbb{P}\). For CDA, one requires for each integer \(\alpha \geq 1\) the existence of a subset \(\mathbb{K}_\alpha \subseteq \mathbb{K}\) and an efficient function \(\kappa_\alpha: \mathbb{K}_\alpha \rightarrow \mathbb{K}\) such that

\[
D_{\kappa_\alpha}(k_1, \ldots, k_\alpha) \left( \bigoplus_{i=1}^{\alpha} E_{k_i}(v_i) \right) = \sum_{i=1}^{\alpha} v_i
\]

for all \((k_1, \ldots, k_\alpha) \in \mathbb{K}_\alpha\) and \((v_1, \ldots, v_\alpha) \in \mathbb{P}_\alpha\). Observe that the essential requirements are such that the function \(\kappa_\alpha\) is efficient and independent of the plaintexts. We call such an encryption function plaintext-homomorphic to distinguish it from the function we use in our proposal. Under the above setting an aggregator node \(A\) is not required to perform decryption and subsequent encryption operations to aggregate incoming data from sensing nodes like it is required
when using conventional hop-by-hop encryption. Instead, it can simply combine
the incoming ciphertexts and forward the results towards its predecessor. If the
sink knows the used keys, it can easily decrypt (using $\kappa$) the aggregated value
of the ciphertexts. The concept is illustrated in figure 1 for a simple WSN with
one aggregator and one level of sensor nodes. Note that CDA also supports a
hierarchy of aggregating nodes as long as the aggregation function itself sup-
ports such a cascaded adjustment. Consequently, the approach is best suited
for large scaled WSNs. Furthermore, no intermediate node has the knowledge
to decrypt the incoming ciphertexts. This increases the overall system security
since there is no lack of security at the aggregating nodes.

Although all currently proposed CDA schemes are based on the same princi-
ple, they significantly differ in terms of encryption work load, key distribution,
and communication overhead. The CDA mechanism originally introduced in
[6] uses the probabilistic\(^2\) encryption scheme from Domingo-Ferrer [4]. Each
encrypting party encrypts with the same symmetric key, which is subsequently
also used for decryption. More formally, it holds that $\mathbb{K}_\alpha = \{(k, \ldots, k) | k \in \mathbb{K}\}$
and $\kappa_\alpha(k_1, \ldots, k_\alpha) := k_1$.

The authors of [2] proposed the use of keystream generators $kg : \mathbb{K} \times
\mathbb{N} \to \mathbb{P}$, which produce deterministically from a seed $k \in \mathbb{K}$ a keystream
$kg(k, 1), kg(k, 2), \ldots$ which is then simply "+"-combined with the plaintexts
to encrypt them. Let $n$ denote the number of sensor nodes within the network
amongst which some keys $k_1, \ldots, k_n$ are distributed. We define $k_i^t := kg(k_i, t)$.
Under the assumption that the sensor nodes are synchronized, they commonly
use the same parameter $t$ to encrypt the measured value $v_i$ to $E_{k_i^t}(v_i) := k_i^t + v_i$.
The aggregation of the ciphertexts yields to

$$\bigoplus_{i=1}^{n} E_{k_i^t}(v_i) = \sum_{i=1}^{n} k_i^t + \sum_{i=1}^{n} v_i$$

That is encrypting the same plaintext twice under the same key leads to different cipher-
texts with high probability.

\(^2\)That is encrypting the same plaintext twice under the same key leads to different cipher-
texts with high probability.
6 Bihomomorphic Encryption Transformation

Similar to the previously described CDA solutions, our proposal relies on the use of a homomorphic encryption function, which however fulfills the following additional requirements for any $\alpha$:

$$K_\alpha := K^\alpha \text{ and } \kappa_\alpha(k_1, \ldots, k_\alpha) := k_1 \circ \ldots \circ k_\alpha.$$  

(4)

This yields that the following is true for any $k_1, \ldots, k_\alpha \in K$ and $v_1, \ldots, v_\alpha \in P$:

$$E_{k_1}(v_1) \circ \ldots \circ E_{k_\alpha}(v_\alpha) = E_{k_1 \circ \ldots \circ k_\alpha}(v_1 + \ldots + v_\alpha).$$  

(5)

Thus, the encryption is homomorphic both on the plaintext space and on the key space. Consequently, we term such an encryption as bihomomorphic.

A simple example for a bihomomorphic encryption transformation is the modulo integer addition. That is for $K \equiv P \equiv \mathbb{Z} \pmod{n}$ with an integer $n \geq 1$, $E_k(m) := k + m \pmod{n}$ is a bihomomorphic encryption where both $\circ$ and $+$ being the addition modulo $n$.

Observe that the encryption with a keystream generator as defined in Section 5 is not bihomomorphic according to the definition. As shown in (3), the encryption is homomorphic in the plaintext and in the keystream elements. However, to be homomorphic in the keys would require for all $k_1, \ldots, k_\alpha \in k$ the existence of a key $k \in K$ such that $kg(k_1, t) \circ \ldots \circ kg(k_\alpha, t) = kg(k, t)$ for all $t$ which is very unlikely in the general case.

7 TAUK and Bihomomorphic Encryption in a Nutshell

Next, we describe the complete key management architecture of the Topology Aware Unique Keying (TAUK) and its application to a bihomomorphic encryption function. Compared to previous solutions like [2], [13] we see substantial advantages in that:

1. an (almost) optimal data overhead during the aggregation phase is ensured. No list of node IDs or, more sophisticated, complementary list of node IDs respectively subset of node IDs needs to be transmitted in each aggregation phase,

2. robustness in respect of silent nodes is provided during the aggregation phase,

3. key refreshment is provided as an essential pre-requisite to support any deterministic bihomomorphic encryption function.

The KPD TAUK supports any bihomomorphic encryption function for CDA. It impacts the initialization phase, each aggregation phase and eventual refreshment phases:
to run TAUK in the initialization phase, it is assumed that each sensor node $N$ already knows its direct neighbors $\text{Pred}(N)$ and $\text{Succ}(N)$. Subsequently, TAUK ensures that each node receives a single symmetric key, whereas all keys from the sensor nodes are derived from a master key, which is solely stored at the sink node. In addition to its key, each node stores encrypted default values. Each of such ciphertexts corresponds to a $N' \in \text{Succ}(N)$. We will see that they provide robustness during the aggregation phase. During the initialization phase the system is vulnerable, even to passive attacks. No attacker is assumed to be in place during this phase.

- during an aggregation phase, convergecast traffic is encrypted end-to-end from the sensing nodes to the sink node. Each node $N$ applies the bihomomorphic encryption function by encrypting its monitored value with its own unique key and by subsequently summing up the resulting ciphertexts from its children $\text{Succ}(N)$. Since each node purely stores its own key it can not decrypt the incoming ciphertexts from its children. Only the sink node is enabled to decrypt the final aggregated value by applying the master key to the received ciphertexts. Each intermediate node adds those stored default ciphertexts to the aggregation value which correspond to its silent successors. During an aggregation phase, the system provides reasonable security against passive and active attacks.

- during a refreshment phase, each node’s key is updated. Unlike during the initialization phase where no attacker is assumed to be in place, any information exchanged in this phase might be eavesdropped. Therefore, it must be ensured that no data is revealed that might compromise the security. Note that in particular for any deterministic encryption scheme it is essential to refresh keys, ideally after each single aggregation phase. However, for a realistic setting we propose to find a good balance between the frequency of key refreshment and the required security level.

We will substantiate the statements from this birdsview in the following Section.

8 Topology Aware Unique Keying and bihomomorphic Encryption

8.1 Basic idea

We give a general description of our new CDA scheme before we discuss concrete instantiations in Section 9. Let in time frame $t$ denote $K_t^N$, the key used by node $N$ and $v_t^N$, the measured value. When the transmitted values are encrypted with
a bihomomorphic encryption scheme, $S$ receives at time frame $t$ the value
\[ \bigoplus_N E_{K'_N}(v'_N) = E_{\bigodot_N K'_N} \left( \sum_N v'_N \right). \] (6)

Thus, $S$ is able to get the sum of all the measurements by decrypting the received value with the key
\[ K'_S := \bigodot_N K'_N. \] (7)

Interestingly, $S$ actually does not need to know the concrete values of the "summands" $K'_N$ as long as it knows the "sum" $K'_S$.

This observation is the basic motivation for the CDA scheme we propose. The deployment of a bihomomorphic encryption algorithm allows the aggregation of data encrypted under different keys where only the aggregation of the keys needs to be known to the sink node. For this purpose we assume that for each time frame $t$ every node $N$ has a key $K'_N$ and the sink node knows $K'_S := \bigodot_N K'_N$.

At the end of an aggregation process, the sink node receives the encryption of the aggregated sensed values. Due to the bihomomorphic property this corresponds to a ciphertext encrypted with the aggregation of the keys of the responding nodes. However, the keys of the silent nodes are missing. Thus, decrypting with $K'_S$ would lead to a false result. To handle this problem, we propose that each node $N$ additionally stores some values $E_{K'_N}(R'_N)$ for all $N' \in \text{Succ}(N)$. The purpose of the values $R'_N$ is that, whenever a node $N$ does not receive the data from all its successors $\text{Succ}(N)$, it replaces the missing ciphertexts by the ciphertexts of dummy values $R'_N$. Depending on how the dummy values are generated, the sink node has to treat them differently. We discuss this aspect in Section 9.3.

In principle, the CDA schemes is divided into two different phases:

1. the initialization phase when the data, e.g., keys, are installed on the nodes within the network,
2. the aggregation phase which is divided into time frames where the measurements are taken, aggregated, and forwarded to the sink node.

In the following, we describe both phases in general terms.

### 8.2 Initialization phase

In the initialization phase, the keys and the dummy values are distributed within the network. We assume that each node knows its predecessor and its successors. Furthermore, we assume that no attackers are present in this time period as all secrets are exchanged in clear. Alternatively the nodes may be equipped with

\[ \text{For the sake of simplicity, we assume for the moment that all nodes have participated in the aggregation. The case of silent nodes will be addressed in our scheme later.} \]
a network wide key which is used to secretly exchange the information in this phase and delete it subsequently from the memory. This would protect the initialization phase against passive attackers.

The initialization proceeds top-down from the sink node to the leaf nodes. The sink node chooses one (or, more sophisticated, several) master keys, which will be the aggregation of all individual keys distributed in the WSN. In a nutshell, every node receives at some point in time during the key distribution his share of the master key which it distributes to its successors in a controlled way. At the same time, the dummy values are created and distributed.

**Initialization:** The sink node $S$ selects a number $r$ which represents how many keys each node will store at the same time. $S$ chooses randomly $r$ master keys $\hat{K}_S^1, \ldots, \hat{K}_S^r \in K$ and dummy values $R_N^1, \ldots, R_N^r \in P$.

**Distribution:** Let $N$ be a node which is not a leaf node and $\text{Succ}(N) = \{N_1, \ldots, N_m\}$.

1. If $N$ is not the sink node, it receives during the key distribution from its predecessor some data $(\hat{K}_N^1, \ldots, \hat{K}_N^r)$ and $(R_N^1, \ldots, R_N^r)$. If $N$ is the sink node, it created these values on its own. Whereas the first is its share of the master keys, the second determines which dummy values will be installed for the successors of $N$.

2. If $N$ is a leaf node, it solely deletes $(R_N^1, \ldots, R_N^r)$ from its memory and defines $K_N^i := \hat{K}_N^i$. Otherwise, the following steps are performed:
   
   (a) $N$ splits the key $\hat{K}_N^t$ into $\hat{K}_N^t = \left( \bigodot_{N' \in \text{Succ}(N)} \hat{K}_{N'}^t \right) \odot K_N^t$ for $t \in \{1, \ldots, r\}$ where the splitting is randomly chosen.
   
   (b) Furthermore, $N$ derives from $(R_N^1, \ldots, R_N^r)$ appropriate dummy values $(R_N^1, \ldots, R_N^r)$ and stores $E_{\hat{K}_{N'}^i}(R_{N'}^t)$ for all $N' \in \text{Succ}(N)$. The derivation mechanism depends on the concrete instantiation.

   (c) Subsequently, $N$ sends to all successors $N' \in \text{Succ}(N)$:
   
   $$ N \rightarrow N' : (\hat{K}_N^1, \ldots, \hat{K}_N^r), (R_N^1, \ldots, R_N^r) $$

   (d) Finally, $N$ deletes all values from its memory except of its own keys $K_N^1, \ldots, K_N^r$ and the encrypted dummy values $E_{\hat{K}_{N'}^i}(R_{N'}^t)$ for $i = 1, \ldots, m$ and $N' \in \text{Succ}(N)$. The only exception is in the case of the sink node since $S$ does not discard the master keys $\hat{K}_S^1, \ldots, \hat{K}_S^r$.

Observe that for each node $N$, it holds that

$$ \hat{K}_N^i = \left( \bigodot_{N' \in \text{Succ}(N)} K_{N'}^i \right) \odot K_N^i. $$

That is the key $\hat{K}_N^i$ is the aggregation of all keys from the subtree rooted in $N$. However, as $N$ keeps $K_N^i$ but deletes $\hat{K}_N^i$, it has no information on the keys in
the lower levels. This is in particular true for the case $N = S$ where

$$K'_S = \bigodot_N K'_N. \quad (10)$$

$S$ knows the aggregation of all keys distributed within the network which is all it needs to decrypt the aggregated data as discussed above. The distribution phase is illustrated in Figure 2.

8.3 Aggregation phase

The aggregation phase is subdivided into discrete time frames in which data is measured, encrypted, aggregated, and forwarded from the leaf nodes to the sink node. Each node receives the encrypted data from (some of its) successors, aggregates them together with its own ciphertext, and subsequently forwards the result to its predecessors. For a detailed description, one has to distinguish between leaf nodes, aggregator nodes, and the sink node.

8.3.1 Leaf nodes

Let $N$ be a leaf node with key $K^t_N$ and $v^t_N$ denotes its sensed value which is encrypted to $C^t_N := E_{K^t_N}(v^t_N)$. The leaf node sends the ciphertext to its predecessor together with a counter set to zero:

$$N \rightarrow Pred(N) : (C^t_N, 0) \quad (11)$$

8.3.2 Aggregator nodes

Let $N$ be a node which is neither a leaf node nor the sink node. If $N$ is not a sensor node, we set $v^t_N := 0$, otherwise let $v^t_N$ denote the measurement made in frame $t$. Furthermore, it expects to receive data from its successors where it cannot be excluded that some of them remain silent in this frame. Let $Succ(N)^t_{\text{resp.}}$ denote the nodes from which $N$ got a transmission during frame $t$ and $Succ(N)^t_{\text{silent}}$ denote the nodes which remained silent. That is it holds

$$Succ(N) = Succ(N)^t_{\text{resp.}} \cup Succ(N)^t_{\text{silent}} \quad (12)$$
for all \( t \). For each \( N' \in \text{Succ}(N) \), the node \( N \) receives some data \((C^t_{N'}, \text{ctr}^t_{N'})\) where \( C^t_{N'} \) is the encrypted aggregated value of the subtree \( ST(N') \) and \( \text{ctr}^t_{N'} \geq 0 \) is a non-negative integer which is equal to the number of dummy values contained in the plaintext of \( C^t_{N'} \).

\( N \) computes the actual ciphertext by

\[
C^t_N := \bigoplus_{N' \in \text{Succ}(N) \text{ resp.}} C^t_{N'} \oplus \bigoplus_{N' \in \text{Succ}(N) \text{ silent}} E_{K^t_{N'}}(R^t_{N'}) \oplus E_{K^t_N}(v^t_N)
\]

and the actual counter

\[
\text{ctr}^t_N := \sum_{N' \in \text{Succ}(N) \text{ resp.}} \text{ctr}^t_{N'} + |\text{Succ}(N)_{\text{silent}}|.
\]

Let \( P^t_N \) denote the plaintext underlying \( C^t_{N'} \), that is \( E_{\hat{K}^t_N}(P^t_N) := C^t_{N'} \). Then, \( C^t_N \) can be rewritten to

\[
C^t_N = E_{\hat{K}^t_N} \left( \bigoplus_{N' \in \text{Succ}(N) \text{ resp.}} P^t_{N'} \oplus \bigoplus_{N' \in \text{Succ}(N) \text{ silent}} E_{K^t_{N'}}(R^t_{N'}) \oplus v^t_N \right)
\]

Thus, any node \( N \) sends ciphertext which is encrypted under its key \( \hat{K}^t_N \). The above also implies the following recursive definition for \( P^t_N \) where we set \( P^t_N := v^t_N \) if \( N \) is a leaf:

\[
P^t_N := \bigoplus_{N' \in \text{Succ}(N) \text{ resp.}} P^t_{N'} \oplus \bigoplus_{N' \in \text{Succ}(N) \text{ silent}} R^t_{N'} \oplus v^t_N.
\]

Subsequently it sends this tuple to its predecessor (see also Figure 3):

\[
N \rightarrow \text{Pred}(N) : (C^t_N, \text{ctr}^t_N).
\]

Observe that \( N \) knows only the key \( K^t_N \) during the initialization phase. Hence it is not able to decrypt the aggregated ciphertexts.

### 8.3.3 Sink node

At the end of each time frame, the aggregation of some encrypted measurements reached the sink node while others got lost due to silent nodes. Observe that even if a node \( N \) successfully reported its ciphertext to its predecessor \( \text{Pred}(N) \), it might be lost at the end if \( \text{Pred}(N) \) (or one of its predecessors) was silent in this time frame.

Let \( V^t_{\text{resp}} \) denote the set of all nodes whose ciphertexts are part of the aggregated data that reaches the sink node. As explained in Section 8.3.2, each
\( N \) forwards the encryption of \( P_{N}^{t} \), for all \( N' \in Succ(N)_{\text{resp.}} \) and the encryption of \( R_{N}^{t} \), for all \( N' \in Succ(N)_{\text{silent}} \). If we define

\[
\mathcal{V}^{t} := \bigcup_{N \in \mathcal{V}_{\text{resp.}}^{t}} Succ(N) \tag{18}
\]

\[
\mathcal{V}_{\text{silent}}^{t} := \mathcal{V}^{t} \setminus \mathcal{V}_{\text{resp.}}^{t} \tag{19}
\]

then the sink node receives the following data:

\[
\begin{align*}
C_{S}^{t} & = \bigoplus_{N \in \mathcal{V}_{\text{resp.}}^{t}} E_{K_{S}^{t}}^{t}(v_{N}^{t}) \oplus \bigoplus_{N' \in \mathcal{V}_{\text{silent}}^{t}} E_{K_{S}^{t}}^{t}(R_{N'}^{t}) \tag{20} \\
& = E_{K_{S}^{t}}^{t}\left( \sum_{N \in \mathcal{V}_{\text{resp.}}^{t}} v_{N}^{t} + \sum_{N \in \mathcal{V}_{\text{silent}}^{t}} R_{N}^{t} \right) \tag{21}
\end{align*}
\]

As opposed to the other aggregator nodes, \( S \) knows the key \( \hat{K}_{S}^{t} \) and can therefore decrypt the ciphertext \( C_{S}^{t} \) to get

\[
D_{K_{S}^{t}}(C_{S}^{t}) = \sum_{N \in \mathcal{V}_{\text{resp.}}^{t}} v_{N}^{t} + \sum_{N \in \mathcal{V}_{\text{silent}}^{t}} R_{N}^{t}. \tag{22}
\]

Observe that the counter \( \text{ctr}^{t}_{S} \) tells how many dummy values are contained in \( R^{t} \), that is \( \text{ctr}^{t}_{S} = |\mathcal{V}_{\text{silent}}^{t}| \). Depending on how the dummy values are created and handled, the sink node can cope with \( \sum_{N \in \mathcal{V}_{\text{silent}}^{t}} R_{N}^{t} \) to get (possibly only an approximation) of \( \sum_{N \in \mathcal{V}_{\text{resp.}}^{t}} v_{N}^{t} \). This will be discussed into more detail in Section 9.3.

**Remark:** As it is now, the sink node can recover the sum of the measured and reported values but it does not know how many values have been aggregated in total, that is the size of \( \mathcal{V}_{\text{resp.}}^{t} \). Since these information may be relevant for some WSN applications, one should incorporate a second counter, \( \text{resp}^{t} \), which computes the number of responded nodes. The counter \( \text{resp}^{t} \) can be handled like \( \text{ctr}^{t} \). The node \( N \) receives from each responding node \( N' \in Succ(N) \) a value \( \text{resp}^{t}_{N'} \) and forwards the sum of it to \( \text{Pred}(N) \).

## 9 Concrete instantiations

Observe that the previous Section provides only a high level description as the exact steps differ depending on the concrete instantiations. For a more concise description, the following building blocks need to be specified:

1. The bihomomorphic encryption mechanism,
2. the refreshing of the keys, and
3. the creation and handling of the dummy values.
9.1 Encryption scheme

Recall the definition of a bihomomorphic encryption given in Section 6. To the best of our knowledge, bihomomorphic symmetric encryption schemes have not been studied in open literature. However, a simple example has already been mentioned in Section 6, namely $E_k(v) := k + v$ if $k$ and $v$ both belong to the same algebraic structure, e.g., $\mathbb{Z}_{2^n}$.

9.2 Key refreshment

For security reasons, keys should be regularly refreshed, at least for deterministic encryption schemes. However, to ensure that the sink node can still decrypt the aggregated values, it is necessary that the sink node knows the aggregation of the modified keys. Thus, either the change of the keys is coordinated by the sink node itself, or it is done following a deterministic rule which allows the sink node to compute the new key autonomously. We refer to this two different approaches by coordinated key refreshment and autonomous key schedule.

9.2.1 Coordinated key refreshment

In this approach, the keys are refreshed on request of and coordinated by the sink node. This means that some data is distributed from the sink node to the individual nodes which determines how the keys have to be changed.

In the following we propose one concrete coordinated key refreshment mechanism. At this we assume that each node (including the sink node) stores only one key at one point in time, i.e. $r = 1$, which is changed according to data coming from the sink node. In principle the refreshment is similar to the initialization phase. The idea is to modify the keys such that the master key $\hat{K}_S^t$ is replaced by a new master key $\hat{K}_S^{t+1} := \hat{K}_S^t \circ \Delta^t_S$ where the sink node $S$ chooses (randomly) the difference $\Delta^t_S$. This difference is split and reported to the successors in the same way as the master key in the initialization phase.

More concretely, let $N$ be a node which is not a leaf node. During the key refreshment phase, $N$ receives a value $\Delta^t_N$ which will be used to update all keys.
in the subtree $ST(N)$. For this purpose $N$ splits the given value $\hat{\Delta}_N^t$ into

$$\hat{\Delta}_N^t = \bigodot_{N' \in Succ(N)} \hat{\Delta}_{N'}^t \odot \Delta_{N}^t$$ (23)

and sends $\hat{\Delta}_{N'}^t$ to each $N'$. The own key $K_{N}^t$ is changed to

$$K_{N}^{t+1} := K_N^t \odot \Delta_N^t.$$ (24)

Furthermore, $N$ computes for all $N' \in Succ(N)$:

$$E_{\Delta_{N'}^t}^t(0) \oplus E_{K_{N'}^t}^t(R_{N'}^t) = E_{K_{N'}^t \odot \Delta_N^t}^t(R_{N'}^t) = E_{K_{N'}^{t+1}}^t(R_{N'}^t)$$ (25)

to replace the encryption of $R_{N'}^t$, under the "old" key $K_{N'}^t$, to the encryption under the "new" key $K_{N'}^{t+1}$. At this, we assume that the dummy value $R_{N'}^t$ remains constant over all time frames, that is $R_{N'}^{t+1} = R_{N'}^t$. Observe that the described modifications are possible with neither knowing $R_{N'}^t$ nor $K_{N'}^t$. The whole mechanism is shown in Figure 4.

At the end, any node $N$ has changed its key $K_{N}^t$ to $K_{N}^{t+1} = K_N^t \odot \Delta_N^t$. The new keys can be used for the next frame $t + 1$. As each node knows only shares of the difference between the old secret master key and the new one, the confidentiality of the forthcoming data is not compromised.

However, it is important that an outsider cannot eavesdrop the differences. Otherwise he could easily compare the values $v_N^t$ and $v_N^{t'}$ from different time frames $t < t'$ by comparing $E_{K_{N'}^{t'}}(v_N^t)$ with

$$\bigoplus_{i=t}^{t'-1} E_{\Delta_N^t}^i(0) \oplus E_{K_{N'}^t}^i(v_N^t) = E_{K_{N'}^{t'}}(v_N^t).$$ (26)

This would render the whole key refreshment needless.

One possibility to conceal these values from outside parties could be that the nodes establish pairwise keys between the nodes and each of their successors during the initialization phase which will be used to encrypt the differences. That is, each node has then two different keys, one used for encrypting the aggregated values, and one for encrypting the data for the key refreshment. For the latter, any encryption scheme can be used, and the keys do not have to sum up to a value known by the sink node.

9.2.2 Key schedule

The previously described approach requires the regular transmission of additional data to determine the key change. Depending on the frequency of the key updates and the network structure, this may impose impractical workload and energy consumption. Therefore, we describe as an alternative a key schedule which allows each node to update its key by itself without any further communication with other devices. Recall that during the initialization phase, $r$ different
keys $K^1_N, \ldots, K^r_N$ have been installed at each node $N$ which are used in the first $r$ time frames. They can be used in a key schedule to determine all subsequent keys. Let $\varphi^t_N$ denote the operation a node $N$ applies at the end of time frame $t$ to generate the next key $K^t_{N+1}$. In a similar manner, we refer to the mechanism $S$ uses to update its master key $\hat{K}^t_S$ by $\tilde{\varphi}^t_S$. Then it holds

$$K^t_{N+1} = \varphi^t_N(K^1_N, \ldots, K^r_N) \quad (27)$$

for all $t \geq 1$. However, recall Equation (10) which states the following basic condition for our scheme:

$$\hat{K}^t_S = \bigotimes_N K^t_N \quad \forall t \geq 1. \quad (28)$$

Thus, the key derivation function must be chosen such that this relation remains true for all time frames. This implies

$$\tilde{\varphi}^t_S\bigl(\bigotimes_N K^1_N, \ldots, \bigotimes_N K^r_N\bigr) = \varphi^t_N\bigl(\hat{K}^t_S, \ldots, \hat{K}^t_S\bigr) = \hat{K}^t_{N+1} \quad (10)$$

and

$$= \bigotimes_N \varphi^t_N(K^1_N, \ldots, K^r_N).$$

An obvious choice is to set all functions $\varphi^t_N$ and $\varphi^t_S$ to be equal and to let them be linear in $K$ with respect to $\otimes$. That is for each $t$ exist coefficients $c^1_1, \ldots, c^t_r \in \mathbb{N}$ such that

$$\varphi^t_N(x_1, \ldots, x_r) := c^1_1 \cdot x_1 \otimes \ldots \otimes c^t_r \cdot x_r. \quad (29)$$

where $c \cdot x$ is defined as $x \otimes \ldots \otimes x$ (repeated $c$ times). Observe that the coefficients $c^t_i$ are the same for each node $N$ and should be determined by some deterministic algorithm. For example one could envision the vector $(c^1_1, \ldots, c^t_r)$ to be the actual state of a linear feedback shift register (e.g., see [10]).

Figure 4: Refreshment phase
would ensure that all possible vectors of coefficients (except the all-zero vector) occur.

Recall that a node must be able to autonomously update (the encryption of) the dummy values \( R_{N'}^{t+1} \). This can be achieved by using the same mechanism for key refreshment to modify the encrypted values. For example in the choice described above, one can easily show that

\[
\hat{K}_N^{t+1} = \bigotimes_{N' \in \text{ST}(N)} K_{N'}^{t+1} \odot K_N^{t+1}
\]

(30)

\[
= \bigotimes_{N' \in \text{ST}(N)} \left( \bigotimes_{i=1}^r c_i^t \cdot K_{N'}^{t+1} \right) \odot \bigotimes_{i=1}^r c_i^{t+1} \cdot K_N^{t+1}
\]

(31)

\[
= \bigotimes_{i=1}^r c_i^t \cdot \left( \bigotimes_{N' \in \text{ST}(N)} K_{N'}^{t+1} \odot K_N^{t+1} \right)
\]

(32)

\[
= \bigotimes_{i=1}^r c_i^{t+1} \cdot \hat{K}_N^{t+1}
\]

(33)

Thus, if one defines\(^4\)

\[
R_N^{t+1} = \sum_{i=1}^r c_i^t \cdot R_N^t
\]

(34)

then a node \( N \) would compute for each \( N' \in \text{Succ}(N) \):

\[
E_{\hat{K}_N^{t+1}}(R_{N'}^{t+1}) := \bigoplus_{i=1}^r c_i^t \cdot E_{\hat{K}_N^{t+1}}(R_N^t) = E_{\bigotimes_{i=1}^r c_i^{t+1} \cdot K_{N'}^{t+1}} \left( \sum_{i=1}^r c_i^t \cdot R_N^t \right) = R_{N'}^{t+1}
\]

(35)

Thus, this approach allows the autonomous update of both the keys and the encryption of the dummy values.

### 9.3 Robustness in respect to silent nodes

As explained above, the (encryption of the) values \( R_N^t \) will be used to replace the (encryption of the) missing measurements from silent nodes \( N \). Also here, different possibilities exist on how to implement these which are mainly influenced by the following two issues:

1. How the dummy values are handled by the sink node at the end when the aggregated ciphertext is decrypted (see also Eq. (22)).
2. How the keys are refreshed as each node \( N \) needs to store the encrypted values \( E_{\hat{K}_N^{t+1}}(R_{N'}^t) \) for every \( N' \in \text{Succ}(N) \).

\(^4\)Again, we define \( \epsilon \cdot R \) by \( R + \ldots + R \).
In the previous Section, we distinguished between coordinated key refreshment triggered by the sink node and a key schedule which can be done autonomously by any node. Regarding the first approach, we have shown how the encryption of the dummy values can be updated as well even if the key itself is unknown. Although we assumed that the dummy values themselves remain constant, that is \( R^t_N = R^1_N \) for all \( t \geq 1 \), this is not mandatory. For a more general approach, consider that besides the value \( \Delta^t_N \), which specifies the key update, a second value \( \rho^t_N \) exists such that \( R^{t+1}_N = R^t_N + \rho^t_N \). Then, similar to the description given in Section 9.2.1, the encryption of the dummy values can be updated without knowing the keys by computing

\[
E^\Delta^t_{K^t_N} (\rho^t_N) \oplus E^\Delta_{K^t_N} (R^t_N) = E^\Delta_{K^t_N} \circ \Delta^t_N (\rho^t_N + R^t_N) = E^\Delta_{K^t_N} (R^{t+1}_N).
\]

This shows that the update of the dummy values is completely independent of the key update and can be freely chosen. Although a variety of different possibilities is imaginable, we will describe only two of them. The first is that all dummy values are equal and remain constant. That is a constant \( R \in \mathbb{P} \) exists such that \( R^t_N = R \) for all time frames \( t \) and all nodes \( N \). The advantage of this approach is twofold. Firstly, no additional operations are required to update the dummy value as it remains constant. Secondly, and probably more important, according to (22), the sink node decrypts

\[
D^{K_S}_S (C^t_S) = \sum_{N \in V_t^{\text{resp.}}} v^t_N + \text{ctr}^t_S \cdot R.
\]

If we require that \( S \) knows \( R \) and as \( \text{ctr}^t_S \) is reported to \( S \) anyway, the sink node can easily determine the value \( \sum_{N \in V_t^{\text{resp.}}} v^t_N \). Observe that this approach is efficient and provides utmost accuracy of the result. However, we will see in Section 10.2 that this compromises the security to some extent.

Therefore, we describe a second alternative approach in combination with the coordinated key refreshment. In this, the dummy values are no random values anymore but defined by the last successfully reported value. More formally, assume that a node \( N \) received during time frame \( t \) the ciphertext \( C^t_N = E^K_{N'} (P^t_{N'}) \) from one of its successors \( N' \in \text{Succ}(N) \) where \( P^t_{N'} \) is the aggregation of one or several values. However, in the next time frame \( t+1 \), node \( N' \) remained silent, for whatever reason. Then, in this approach \( N \) simply updates the encryption of \( (\text{the unknown value}) v^t_{N'} \), to re-use it for the actual time frame. More precisely, \( N \) computes

\[
E^\Delta_{K^t_N} (0) \oplus C^t_N = E^\Delta_{K^t_N} (0) \oplus E^K_{N'} (m^t_{N'}) = E^K_{N'} \circ \Delta^t_N (m^t_{N'}) = E^K_{N'} (m^{t+1}_{N'}) = C^t_{N'}.
\]

and uses the result for the aggregation. This means in principal that \( N \) pretends that \( N' \) has re-send the same value \( P^t_{N'} \) which it did send in the previous frame \( t \). Of course the final result is not accurate anymore as \( N' \) might have measured another value in this time frame. Therefore, we assume that this approach has
value in dedicated use cases. However it illustrates the flexibility our scheme provides.

This flexibility might no longer be provided if a key schedule is used. As we explained for the concrete proposal, if we define $R_{N}^{t+1} = \sum_{i=1}^{r} c_{i}^{t} \cdot R_{N}^{t}$, a node is able to update the encryption of the dummy values on its own, even without knowing the keys. However, the dummy values cannot longer be freely modified but have to follow the given rule. This prohibits for the example the approach described above of re-using previous (unknown) values $P_{N}^{t}$. Yet, what still works is to let the dummy values in one time frame to be equal to allow the sink node to take them off from the final decryption. More precisely, assume that there exist some constant values $R_{1}, \ldots, R_{r} \in M$ such that $R_{N}^{t} = R_{i}^{t}$ for any node $N$ and $1 \leq i \leq r$. This immediately implies $R_{1}^{t} = \sum_{i=1}^{r} c_{i}^{t} \cdot R_{i}^{t}$ which can be computed by the sink node if it knows the values $R_{1}^{t}, \ldots, R_{r}^{t}$. Furthermore, the sink node $S$ decrypts at the end of a time frame the following:

$$D_{K_{S}}^{t}(C_{S}^{t}) = \sum_{N \in V_{resp}}^{t} v_{N}^{t} + ctr_{S}^{t} \cdot R_{1}^{t}.$$  \hspace{1cm} (40)

Thus, again $S$ is able to derive the accurate result $\sum_{N \in V_{resp}}^{t} v_{N}^{t}$.

10 Evaluation

In this section, we are going to evaluate the proposed scheme and compare it to previous proposals in respect to communication costs, security, and flexibility. As already pointed out, a variety of different instantiations are thinkable which may differ in the communication costs and/or the provided security. In practice the choice should be made depending on the aimed use scenario. However, we will concentrate on only one concrete variant to make the comparison more simple and illustrative. Stimulated by the fact that the lifetime of a WSN has probably the highest market relevance, we have chosen a variant which has minimum communication costs and still a reasonable security level. This means that only the aggregated ciphertexts and few additional data need to be forwarded whereas still a reasonable security level is provided for a huge class of applications. More concretely, we focus on the following instantiation:

**Encryption:** The message space $P$ is $\{0, \ldots, 2^{\mu} - 1\}$ and both the ciphertext space $C$ and the key space $K$ are equal to $(\mathbb{Z}_{p}, +)$ where $p$ is a prime number $\geq 2^{\mu+b_{\text{sensor}}}, b_{\text{sensor}}$ the logarithm of the number of sensors in the network taken to the base of 2, and “+” denotes the usual modulo addition. Observe that $\mathbb{P}$ can be easily embedded in $\mathbb{K}$ (resp. $\mathbb{C}$). The encryption is defined by $E_{k}(m) := m + k$.

**Key refreshment:** For the key encryption we consider the key schedule where the keys are defined by the first $r$ keys. The coefficients $c_{r}^{t} = (c_{r}^{t}, \ldots, c_{1}^{t}) \in \{0,1\}^{r} \in \mathbb{Z}_{2}^{r}$ correspond to the internal state of a linear feedback shift register (LFSR) of length $r$ with a primitive minimal polynomial and a non-zero initial state.
Robustness: For handling silent nodes, we suppose the variant that at each time frame $t$, every node stores the same dummy value $R^t$.

For comparison, we consider the following CDA schemes:

- **HbH** (*Hop by Hop encryption*): we use the simple hop by hop encryption and aggregation scenario as the basis for our comparison.

- **TAGK** (*Concealed data aggregation, [18]*): encryption using the Domingo-Ferrer encryption scheme.

- **CMT** the scheme proposed by Castelluccia, et al. [2]: encryption by adding the output of a keystream generator.

- **ME** (*Multiple Encryption, [1, 13]*): an extension of CMT. This class refers to the proposals described in [1, 13] which minimize the node ID transmission overhead problem in schemes where unique keys are used.

- **TAUK** (*TAUK with modular addition and deterministic key schedule*): the proposal described in this paper.

10.1 Energy consumption

We analyze and compare the cost of our proposal with previously proposed solutions in terms of energy consumption. For this purpose, we differentiate between computational effort and transmission costs.

10.1.1 Computational effort

Regarding the computational effort, the schemes CDA, ME, and TAUK all encrypt by aggregating some keys to the plaintexts, which in most cases will mean modular addition. Thus, the effort for one encryption is comparatively the same. However, CDA and TAUK encrypt each plaintext only once, whereas in ME a purely sensing node encrypts $d$ times before transmission, and an aggregator node decrypts and encrypts $d+1$ times, resulting in a somewhat higher effort.

TAGK uses besides modular addition also modular multiplication, making it more elaborate than the previous ones. For HbH, any appropriate encryption scheme can be used. Thus, the encryption effort might be smaller compared to the other schemes. However, each aggregator node has to decrypt all incoming ciphertexts and to encrypt the result again. Therefore, we expect that the total cost are rather higher compared to CMT, ME, and TAUK.

10.1.2 Transmission costs

Compared to the energy consumption for CPU processing, the energy consumption for data sending or receiving is in the order of $10^5$ higher [9]. Therefore,
transmission of messages represents the pre-dominant energy consuming process. We will study only the sending costs although there is also a corresponding receiving cost associated to every transmission.

First of all, equal for all schemes is that every node (except the sink) forwards during each time frame some ciphertext. However, schemes may use different encryption schemes which vary in their data size. Another difference lies in the handling of the identifiers. Let \( b_{\text{data}} \) denote the bit size of the data sensed by the sensors and similarly let \( b_{\text{ID}} \) be the bit size of a node’s identity. For the sake of simplicity, we assume these values to be constant throughout the whole network. Furthermore, let \( b_{\#\text{sensors}} \) be the logarithm of the number of sensors within the network taken to the base 2. Thus, the aggregation of all \( 2^{b_{\#\text{sensors}}} \) measurements takes at least \( b_{\#\text{sensors}} + b_{\text{data}} \) bits, being a lower bound for the ciphertext size. Besides, we assume that the sink needs to know the number of aggregated values in the result it receives. Thus, if no identifiers are reported, at least one counter of size \( b_{\#\text{sensors}} \) has to be included within the transmissions.

In the (HbH) case, the aggregator simply decrypts any incoming ciphertexts, aggregates the resulting plaintexts, and forwards the encryption of the result. No identities need to be transmitted. For encryption, any encryption scheme can be used. Therefore we assume the minimal length of \( b_{\#\text{sensors}} + b_{\text{data}} \) for a ciphertext.

In CDA, an encryption function taken from [4] is considered where two parameters influence the size of the ciphertext: \#splits, the number of times the plaintext message is split, and \( m \), the module used which determines the size of each part. For this purpose, \( m \) has to be chosen such that there exists a divisor \( m' \) which needs to be at least as big as the plaintext range and \( s := \log_{m'}(m) \) specifies the security parameter. Thus, we can express the ciphertext length by \( \#\text{splits} \cdot s \cdot (b_{\#\text{sensors}} + b_{\text{data}}) \). As each node uses the same key, no identifiers need to be transmitted but one counter.

CMT must transport all the IDs which were involved in the encryption process. At each aggregator \( N \) this results in the concatenation of the ID list within \( ST(N) \). The usage of a keystream generator allows to the ciphertext to be of minimal size \( b_{\#\text{sensors}} + b_{\text{data}} \).

ME based mechanisms reduce the effect of transporting the IDs by performing \#hops-hop encryption and decryption. The problem of carrying the involved IDs is limited to the IDs of the nodes which are at most \#hops hops after an aggregator \( N \). We denote this subtree of \( ST(N) \) by \( ST_{\#\text{hops}}(N) \). The encryption is again done by a keystream generator. As the IDs are not transported up to the sink, a counter is necessary here.

The scheme proposed in this paper requires only the transmission of the aggregated ciphertexts and two counters. It is therefore almost as data efficient as simple hop-by-hop encryption while requiring less operations (only aggregation instead of encryption and decryption) and providing a higher security (aggregator nodes are no longer able to decrypt the incoming ciphertexts).

Table 1 shows the parametrized cost caused by a transmitting node during the aggregation phase. For the sake of simplicity, we assume that every node is a sensor node. While some schemes have a constant data overhead for every
aggregation level, the effort of other schemes like CMT heavily depends on the network structure.

To give an impression on how much the considered schemes differ in the message size, we illustrate this on some concrete examples. Similar to the example discussed in Section 2.3, we consider networks with a balanced tree structure. That is the network has $h$ aggregation levels and each node (except of the leaves) has an outdegree of $d$. To examine how much the message sizes can maximally vary, we assume that no nodes are silent.

In Figure 5, we display the results for the case $h = d = 3$ (39 sensor nodes). It shows that the size of the transmissions made in TAUK is only slightly bigger compared to HbH but about 4 times smaller at aggregation level 1 and about 3 times smaller at aggregation level 2 compared to CMT.

Figure 6 shows for the case of a tree of height $h = 5$ and outdegree $d = 4$ (1364 sensor nodes), that the difference in the transmission sizes are even more extreme. The number of bits which need to be send in CMT are of magnitudes bigger compared to TAUK.

Table 1: Comparison of the maximum node transmission size.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>$N \rightarrow \text{P}red(N)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>HbH</td>
<td>$(b_{\text{data}} + b_{#\text{sensors}}) + b_{#\text{sensors}}$</td>
</tr>
<tr>
<td>CDA</td>
<td>$(b_{\text{data}} + b_{#\text{sensors}}) \cdot s \cdot #\text{splits} + b_{#\text{sensors}}$</td>
</tr>
<tr>
<td>CMT</td>
<td>$(b_{\text{data}} + b_{#\text{sensors}}) + b_{ID} \cdot #\text{ST}(N)$</td>
</tr>
<tr>
<td>ME</td>
<td>$(b_{\text{data}} + b_{#\text{sensors}}) + b_{ID} \cdot #\text{ST}<em>{#}\text{hops}(N) + b</em>{#\text{sensors}}$</td>
</tr>
<tr>
<td>TAUK</td>
<td>$(b_{\text{data}} + b_{#\text{sensors}}) + 2 \cdot b_{#\text{sensors}}$</td>
</tr>
</tbody>
</table>

Figure 5: Comparison of the maximum transmission sizes in respect to the aggregation level for a $h = d = 3$ tree.
Concluding, we see that the use of CMT or other schemes which need to send the node IDs are rather impractical for large scale WSNs. In these cases, schemes with a low data overhead like our proposed scheme are more promising.

10.2 Security Analysis

As explained in Section 4, we consider attackers who aim to gain knowledge of the aggregation of as many measurements as possible made within the network. Attackers can control the communication in the network. She can eavesdrop, stop, and alter the outgoing messages of any node but not for all nodes at the same time. We refer to this kind of attackers as cryptographic attackers. In addition to controlling the communication, an attacker might physically capture some nodes to read out the confidential data stored on them. We name such attackers physical attackers.

For a security analysis, one can ignore the attacker’s capability of changing messages as this does not yield any new information. More formally, assume that a node $N$ sends at time frame $t$ a message $C_N^t$, that is a ciphertext, towards its predecessor $N'$ but this message is changed to $\tilde{C}_N^t$ by the attacker. $N'$ receives $\tilde{C}_N^t$ (instead of $C_N^t$), aggregates it with the other ciphertexts, and forwards the result $\tilde{C}_N^t$ (instead of $C_N^t$) to its predecessor. Because of $\tilde{C}_N^t - C_N^t = C_N^t - C_N^t$, the impact of changing $C_N^t$ has on other messages yields no new information to the attacker.

For the comparison we assume that the deployed cryptographic primitives like the encryption function in HbH and the keystream generators used in CMT and ME are secure.
10.2.1 Cryptographic attackers

In the following we argue that for all schemes the analysis can be restricted to the case that the attacker controls the messages coming to or from exactly one node $N$ although possibly over several time frames. For HbH, CMT, and ME different nodes use independent keys so that attacking one node provides no information on secrets stored at other nodes. In the case of TAUK, the keys are not really independent but are shares of one master key. However, as the master key is secret for an attacker, she cannot distinguish two keys stemming from one master key from two random values. Thus, for her the keys are independent as well. For the case of TAGK, several nodes within one routable region share the same key. But for the security analysis it makes formally no difference whether one considers several of these nodes within one time frame or one node over several time frames. In both cases, several plaintexts are encrypted using the same key.

Therefore, we focus on an attacker who eavesdrops from one node $N$ the ciphertexts $C_{t_1}^N, \ldots, C_{t_s}^N$ from time frames $t_1, \ldots, t_s$ while possibly blocking some messages from its successors. We differentiate between the ciphertext only and known plaintext cases:

10.2.2 Ciphertext only

In a ciphertext-only attack, the attacker only eavesdrops the ciphertexts $C_{t_1}^N, \ldots, C_{t_s}^N$. First of all, all schemes can be used in such a way that encrypting the same plaintext twice does not necessarily lead to the plaintext. Thus, a comparison of the ciphertexts provides no information on the equality of the plaintexts.

As we assumed that the deployed cryptographic primitives are secure, HbH, CMT, and ME are secure in the ciphertext-only scenario. To the best of our knowledge, no ciphertext-only attack has been published on the Domingo-Ferrer encryption scheme. Thus, we assume TAGK to be secure in this case.

For the considered instantiation of TAUK with a bihomomorphic encryption scheme, some information on the plaintexts can be leaked. By definition, each ciphertext is the sum of the plaintext plus the key derived from the key schedule, i.e. $C_N^t = P_N^t + \hat{K}_N^t$ where $P_N^t$ refers to the plaintext as defined in Equation (16). As the key schedule is linear (Eq. (29)), there exist for each key $\hat{K}_N^t$ known coefficients $c_i^t$ such that $\hat{K}_N^t = \sum_{i=1}^r c_i^{t-1} \hat{K}_N^i$ (Eq. (33)). If an attacker eavesdrops $s$ ciphertexts $C_{t_1}^N, \ldots, C_{t_s}^N$, she can set up the following system of linear equations:

\begin{align}
C_{t_1}^N + P_{t_1}^N &= c_{t_1}^{t_1-1} \cdot \hat{K}_N^1 + \ldots + c_{t_r}^{t_1-1} \cdot \hat{K}_N^r \\
C_{t_1}^N + P_{t_1}^N &= c_{t_1}^{t_1-1} \cdot \hat{K}_N^1 + \ldots + c_{t_r}^{t_1-1} \cdot \hat{K}_N^r \\
&\vdots \\
C_{t_s}^N + P_{t_s}^N &= c_{t_1}^{t_s-1} \cdot \hat{K}_N^1 + \ldots + c_{t_r}^{t_s-1} \cdot \hat{K}_N^r 
\end{align}

\[ (41) \]

In the case of HbH, this can be achieved by either using a probabilistic encryption algorithm, defining a key schedule between two nodes or by appending a random value to the plaintext before encrypting.
In the case that the equations on the right hand side are linearly dependent, e.g., if $s > r$, she can build linear combinations of the rows to cancel out the initial keys $\hat{K}_1^N, \ldots, \hat{K}_r^N$. More formally, the attacker can compute coefficients $\lambda_1, \ldots, \lambda_s$ such that

$$\sum_{i=1}^{s} \lambda_i \cdot (c_{i,t}^{-1} \hat{K}_{i,t}^N + \ldots + c_{i,r}^{-1} \hat{K}_{r,t}^N) = 0.$$  \hspace{1cm} (42)

This implies

$$\sum_{i=1}^{s} \lambda_i P_{i,t}^N = \sum_{i=1}^{s} \lambda_i (-C_{i,t}^N)$$  \hspace{1cm} (43)

where the values on the right side are known by assumption. To what extent this information is valuable for an attacker might depend on the use case. Observe that (43) yields the aggregation of plaintexts over different time frames for one node. This differs from an attackers goal of figuring out the aggregation in one time frame of plaintexts from several nodes.

It might also depend on how many plaintexts are present in the linear combination. Let $\Lambda := \{i | \lambda_i \neq 0\}$ be the indices of the non-zero coefficients. Equation (43) yields the value of a linear combination of $|\Lambda|$ plaintexts which might provide some information on the individual plaintexts. It seems to be reasonable to assume that this information is more valuable the lower $|\Lambda|$. Observe that finding $t_1, \ldots, t_s$ and coefficients $(\lambda_1, \ldots, \lambda_s)$ such that $s$ is not too big and $|\Lambda|$ is small corresponds to find a multiple of the LFSR’s minimal polynomial with a not too high degree and of low weight. This corresponds to a well-known presumably hard problem in cryptography, e.g., see [3].

**Known plaintext:** In this scenario, the attacker knows additionally to the ciphertexts $C_{t_1}^N, \ldots, C_{t_s}^N$ the corresponding plaintexts $P_{t_1}^N, \ldots, P_{t_s}^N$. Again, HbH, CMT, and ME are secure if the deployed encryption scheme is secure against known-plaintext attacks.

Regarding the Domingo-Ferrer encryption scheme used in TAGK, it has been shown in [17] that it can be broken under certain parameter settings by solving a system of linear equations. Similar is true for the proposed TAUK scheme. If an attacker knows $s$ ciphertexts $C_{t_1}^N, \ldots, C_{t_s}^N$ and the corresponding plaintexts $P_{t_1}^N, \ldots, P_{t_s}^N$, she can set up the system of equations as displayed in (41). If $r$ rows on the right side are linearly independent, the attacker can derive the initial keys $\hat{K}_1^N, \ldots, \hat{K}_r^N$ by solving a system of linear equations as the values on the left side are known. Once these are known, the attacker can easily calculate every frame key $\hat{K}_{t,N}$ on her own.

However, if this poses a serious threat in practice or not depends on the use case and how likely an attacker knows the plaintexts. Recall that the plaintext $P_{N,t}^t$ is equal to

$$\sum_{Succ(N)_{\text{resp.}}} P_{N,t}^t + \sum_{Succ(N)_{\text{silent}}} R^t + v_{N,t}^t.$$  \hspace{1cm} (44)

The larger the subtree $ST(N)$ the more values stemming from different nodes are included in $P_{N,t}^t$ and the more difficult it is probably for an attacker to
<table>
<thead>
<tr>
<th>Scheme</th>
<th>Ciphertext only</th>
<th>Known plaintext</th>
</tr>
</thead>
<tbody>
<tr>
<td>HbH</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>TAGK</td>
<td>None</td>
<td>Key by solving system of linear eqs.</td>
</tr>
<tr>
<td>CMT</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ME</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>TAUk</td>
<td>$\sum_i \lambda_i P^i_N$</td>
<td>Key by solving system of linear eqs.</td>
</tr>
</tbody>
</table>

Table 2: Information recovery of a cryptographic attacker in respect to the different attack scenarios.

figure out this value. While this might be easily achieved if $N$ is a leaf, it should be much more difficult if $N$ is the root of a large subtree. A possible but probably elaborate approach could be to derive first the keys at the leaves, then at their predecessors, and so on. Another approach could be to block all messages coming from the successors to indirectly force the plaintext to be $\sum_{N' \in \text{Succ}(N)} R^t + v^t_N$. However, $v^t_N$ would still remain secret if no further actions are taken. Observe that this attack can easily be thwarted by initiating a coordinated key refreshment every $r'$ time frames where $r' < r$. This would prevent an attacker from setting up enough linearly independent equations for a key recovery.

10.2.3 Physical capture of nodes

Next, we quantify an active attacker’s gain when corrupting multiple sensor nodes. As no cryptographic mechanism can provide any further security once the keys have been compromised, it is no longer the question if the attacker can gain some information on the plaintext but rather at which cost. Consequently we define the gain function on a subset $S \subseteq V$ to be the maximum set of measurements for which the attacker can recover the aggregation if the nodes in $S$ are compromised:

$$G(S) := \max_{|S'|, S \subset S' \subseteq V} \left\{ |S'| \sum_{N \in S'} v^t_N \right\} \text{ known if } S \text{ compromised} \right\} \quad (45)$$

In other words, if the information stored in $S$ allows an attacker to determine $\sum_{N \in S'} v^t_N$ (without any further physical captures but maybe other actions like eavesdropping or blocking messages) and if this is not true for any superset of $S'$ then we say that $S'$ is the gain of the attacker when corrupting $S$. Corrupting another node $N \notin S$ might increase the gain such that $G(S) \subset G(S \cup \{N\})$. The best an attacker can achieve is to find out the aggregation of all measurements made within the network. Therefore we call a set $S$ a total gain set if $G(S) = V$, that is compromising the nodes in $S$ reveals all measured values. A total gain set $S$ is called minimal if any other total gain set is as least as big as $S$. 

28
**HbH:** Each node $N$ decrypts all incoming plaintexts, aggregates them, and forwards the result to its predecessor. Thus, an attacker compromising $N$ can easily decrypt the outputs of $N$ to gain knowledge of the aggregation made in $ST(N)$. In other words, it holds $G(S) = \bigcup_{N \in S} ST(N)$. In particular, corrupting $\text{Succ}(S)$ reveals all measurements made in the network.

**CMT:** In both schemes, each node has its own individual key which is completely unrelated to the keys of the other nodes. Compromising a set $S$ therefore reveals no information about any other nodes. Therefore, it holds that $G(S) = S$ for all $S \subseteq V$. In particular, $V$ is the only total gain set.

**TAGK:** The authors proposed to divide the network into routable regions which are further subdivided in groups of nodes which share the same subset of keys. That is we can define sets of nodes $\Gamma_1, \ldots, \Gamma_s$ such that two nodes from the same set have the same key. Let $\Gamma(N)$ denote for a node $N$ its group, i.e., $\Gamma(N) = \Gamma_i$ if and only if $N \in \Gamma_i$. Obviously it holds for any set $S$ that $G(S) = \bigcup_{N \in S} \Gamma(N)$. A minimum total gain set is any set $S = \{N_1, \ldots, N_s\}$ such that $\bigcup_{N \in S} \Gamma(N) = V$.

**ME:** Similar to CMT, pairwise keys are installed on the nodes. The difference is that a node does not necessarily shares its key with the sink but with its $d$th predecessor $N'$. That is $N$ and $N'$ are $d$ hops away. If $N$ is closer to the sink than $d$ hops then the key is shared with the sink. As a ciphertext $C_N^t$ can contain the keys of all successors of $N$ which are at most $d$ hops away the gain of compromising $ST_d(N)$ would be $ST(N)$. In particular $ST_d(S) \setminus \{S\}$ is the unique minimum total gain set. Observe that other gains are possibly if the attacker blocks dedicated messages. For example let $N$ be a node with $\text{Succ}(N) = \{N_1, \ldots, N_s\}$. Then the gain of compromising $\{N\} \cup ST_{d-1}(N_1)$ is $\{N\} \cup ST(N_1)$ if the attacker continuously blocks all messages coming from $N_2, \ldots, N_s$.

**TAUK:** In principle, the keys stored at the nodes are unrelated as they are randomly chosen shares of the secret master key. Thus, $G(S) = S$ would hold in a completely reliable network. However, the concrete instantiation with the network wide dummy values $R^1, \ldots, R^r$ brings a security weakness as follows. Assume that an attacker compromises two nodes $N$ and $N'$ such that $N' \in \text{Succ}(N)$ and $N'$ is a leaf, then the attacker can find out $E_{K_{N'}^{r}}(R^t), E_{K_{N'}^{r+1}}(R^{t+1}), \ldots, E_{K_{N'}^{r+1}}(R^{t+r-1})$ (stored at $N$) and $K_{N'}^{r}, \ldots, K_{N'}^{r+r-1}$ (stored at $N'$). Both information can be used to find out the network wide dummy values $R^t, \ldots, R^r$. With this information, compromising any other node $\tilde{N}$ immediately reveals the values $K_{N'}^{t}$ and $\tilde{K}_{N'}^{t}$ for $\tilde{N'} \in \text{Succ}(\tilde{N})$. Because of

$$\tilde{K}_{N'}^{t} = \sum_{\tilde{N'} \in \text{Succ}(\tilde{N})} \tilde{K}_{N'}^{t} + K_{N'}^{t}$$

(see Eq. (9)), an attacker can compute $\tilde{K}_{N'}^{t}$ and decrypt any ciphertext coming from $\tilde{N}$ from now on. The gain would be $ST(\tilde{N})$ in this case. Thus a minimal

---

9If $N'$ is not a leaf it has stored $K_{N'}^{t}$, and not $\tilde{K}_{N'}^{t}$, so that the information gain is not immediate clear.
\[ \mathcal{G}(?) = ST(N) \]

Minimum total gain set

<table>
<thead>
<tr>
<th>Scheme</th>
<th>{N_1, \ldots, N_s} s.t.</th>
<th>{N_1, \ldots, N_s} s.t.</th>
</tr>
</thead>
<tbody>
<tr>
<td>HbH</td>
<td>\bigcup_i \Gamma(N_i) \supseteq ST(N)</td>
<td>\bigcup_i \Gamma(N_i) = V</td>
</tr>
<tr>
<td>TAGK</td>
<td>ST(N)</td>
<td>\bigcup_i \Gamma(N_i) = V</td>
</tr>
<tr>
<td>CMT</td>
<td>ST_d(N)</td>
<td>ST_d(S) \cup {S}</td>
</tr>
<tr>
<td>ME</td>
<td>ST_d(N) \cup {N}</td>
<td>ST_d(S) \cup {S}</td>
</tr>
<tr>
<td>TAUk</td>
<td>{N, N'} with</td>
<td>{N} \cup Succ(S) with</td>
</tr>
<tr>
<td></td>
<td>\exists N' \in Succ(S) s.t.</td>
<td>N \in Succ(S')</td>
</tr>
</tbody>
</table>

Table 3: Information gain of a physical attacker.

total gain set is given by \( Succ(S) \cup \{N\} \) with \( N \) being a successor of one node in \( Succ(S) \).

10.2.4 Security upshot

The security analysis shows that the considered instantiation is vulnerable against some attacks which do not apply for the other schemes. Obviously it does not reach the same security level as its competitors. However, recall that the primary goal for the considered instantiation was to increase the lifetime while still providing some reasonable security. Some of the weaknesses are consequences from these choices, e.g., to use a network-wide dummy value \( R_t \) to allow key refreshment without any additional data exchange.

If the provided security level meets the requirements of the concrete scenario depends on the attacker cost model (Section 4). Recall the arguments that an attacker will likely not invest more time and resources into attacking the network than it would require to build her own. In this context, we stress that none of the attacks provide a total break "for free". The ciphertext-only attack is certainly the cheapest one but provides only the aggregation of plaintexts over several time frames, making the value for an attacker unclear. The known-plaintext attack requires to eavesdrop each node at least \( r \) times to reveal the corresponding plaintexts. In particular the second condition could have the practical implication that an attack at reasonable costs requires to attack the network level by level, starting from the leaves. The physical attack requires the corruption of only the highest level (plus one successor) to break the whole network. Thus, in scenarios where corrupting nodes is feasible and the specified nodes are easily reachable, we would not recommend to use TAUk without any further protection. In other scenarios, the provided security level might be enough to fulfill the requirements, especially when the user is aiming for a long-lived WSN.
10.3 Routing Flexibility

Next, we discuss the flexibility and robustness properties of the available CDA solutions and their recommended key pre-distribution schemes.

A WSN representing the network model which we introduced in Section 3 requires some flexibility with respect to changing routes for the transmission of convergecast traffic. This comes mainly due to the need of i) exhausting nodes, as well as the conceptual mean of ii) aggregator node election per time frame.

In case of using HbH we differentiate between HbH with pairwise keys and HbH with groupwise keys. In the first case the architecture provides no routing flexibility at all. With groupwise keys HbH encryption achieves routing flexibility at the cost of almost no system security. Regarding the degree of routing flexibility we observe that CMT with a randomized pairwise key pre-distribution as well as the single symmetric key based TAGK are the most promising ones. However, they either come at the costs of reduced system security or an unacceptable data overhead during the aggregation phase.

Since both ME and TAUK require some keys stored on the path to the sink, this approach looses routing flexibility and ends up in the requirement of static routing. These restrictions can be relaxed to some extent by storing the particular keys at several nodes at the same aggregation level. However, a more detailed analysis is out of scope of this work.

10.4 Key distribution

Key distribution is most simple in the case of TAGK where only few keys are distributed on the nodes. In the case of CMT, each node needs to pair with the sink before deployment, but no further actions are needed. Thus, key distribution is likewise simple here. However, we observe that the key distribution of the CMT-extension ME looses its original simplicity and evolves to a very complex approach requiring a structured key distribution approach. To the best of our knowledge, there is a lack of specifying in which manner keys can be distributed like it is required. Although TAUK requires a similar structured key distribution, an appropriate simple and autonomous key predistribution mechanism is described in Section 8.2 which works after the nodes have been laid out. In the case of HbH, the probably easiest key predistribution approach is to start pairwise keying after the WSN roll out. This has an effort comparable to the case of TAUK.

11 Conclusion

We introduced a CDA derivate named TAUK that provided end-to-end encryption of convergecast traffic. We introduced the concept of bihomomorphic encryption transformation and proposed to use such a primitive together with the key pre-distribution scheme topology aware group keying. Compared to other currently available CDA derivates our approach reduces the overhead in the aggregation phase to a minimum. We show that with the usage of encrypted
dummy values the proposed solution is still robust and flexible due to exhausting nodes and changing routes. Our recommendation is to apply TAU for large scale WSNs when a long lifetime and a reasonable security are required.

References


