Information Fusion 13 (2012) 285–295

Contents lists available at ScienceDirect

Information Fusion

journal homepage: www.elsevier.com/locate/inffus

# Asymmetric homomorphisms for secure aggregation in heterogeneous scenarios

Alexandre Viejo [*], Qianhong Wu, Josep Domingo-Ferrer

*Universitat Rovira i Virgili, Departament d'Enginyeria Informàtica i Matemàtiques, UNESCO Chair in Data Privacy, Av. Països Catalans 26, E-43007 Tarragona, Catalonia, Spain*

ARTICLE INFO

*Article history:*
Received 23 April 2010
Received in revised form 8 November 2010
Accepted 11 March 2011
Available online 21 March 2011

*Keywords:*
Asymmetric homomorphisms
Data aggregation
Many-to-one communication
Scalability
Security

ABSTRACT

In multicast communication, a single source transmits the same content to a large amount of receivers. This kind of communication is usually represented following a tree model where the root of the tree is the multicast source and the leaves are the receivers. Scalability problems arise when the root needs to collect data (sensor information, metering data, etc.) from the leaves. This results in a many-to-one (leaf-to-root) communication. The matter is further complicated if there are security requirements on the leaf-to-root traffic. In this paper we present a method for secure and scalable many-to-one lossy transmission based on asymmetric homomorphisms which enables the root of the tree to compute any mathematical function (*e.g.* minimum, maximum, average, ... ) on the data sent by the leaves. Our proposal preserves the confidentiality of those data. Authentication is guaranteed in the sense that only authorized nodes can participate in the protocol. Integrity against compromised leaves is also achieved. In the case of a compromised intermediate node which colludes with a compromised leave, they can only cause a limited deviation in the final aggregate value.

## 1. Introduction

Network communications are commonly based on point-to-point unicast channels and point-to-multipoint multicast channels. The unicast model is used when sending messages from a source to a single receiver. However, if a sender wants to transmit the same message to a large group of receivers, this model is inefficient due to the high number of point-to-point channels that must be established.

Multicast is a communication paradigm where a single sender transmits the same data to a group of receivers (one-to-many communication). This model provides scalability in the sense that the number of receivers can be increased without increasing the workload nor the bandwidth needs at the source. Some multicast applications may require the source to collect data from receivers. This situation results in *many-to-one* communication [1].

Both one-to-many and many-to-one communications are based on a tree topology. In one-to-many, the root is the source which transmits the data, the leaves are the receivers, and the intermediate nodes are the multicast routers receiving the content from their parent node and retransmitting it to their child nodes. In many-to-one, the leaves act as senders and the root is the only receiver. Some examples of the latter scenario are:

- *Environment monitoring.* Nodes with sensing capabilities send data upon reception of a request multicast by the base station (*e.g.* telemetering electric power consumption in a neighborhood).
- *Reverse multicast.* For example, to perform pay-per-view of multicast content delivery (*e.g.* cable TV, Internet live transmissions, pay-per-view video on demand). When performing payment or content selection, subscribers become senders and the content source is the only receiver.
- *Mobile ad hoc networks (MANET) and vehicular ad hoc networks (VANET)*. In those networks, there are applications in which a single node requests a certain data to the rest of the network (*e.g.* traffic status data in VANETs). Many-to-one communication takes place when the network nodes respond to the requesting node.

These different scenarios involve nodes with different computing and power capabilities, which limits the protocols and applications that can be implemented in each situation. Our work intends to cope with that diversity of scenarios. This includes networks of full-fledged devices (reverse multicast and MANET/VANET applications) but also environment-monitoring applications where light sensor nodes are scattered around a certain area (*e.g.* by a plane throwing them) while intermediate nodes can be other lightweight devices or even special vehicles (*e.g.* 4 × 4 cars/trucks in a forest) located at strategic points of the area.

Whatever the scenario is, many-to-one applications suffer from the implosion problem [2]. Implosion happens when too many leaves transmit simultaneously. Under this situation, the root of the communication tree is likely to be swamped with incoming messages.

[*] Corresponding author.
*E-mail addresses:* alexandre.viejo@urv.cat (A. Viejo), qianhong.wu@urv.cat (Q. Wu), josep.domingo@urv.cat (J. Domingo-Ferrer).

1566-2535/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.inffus.2011.03.002

In [3], a solution to implosion in many-to-one scenarios is presented. It is a general framework where intermediate nodes collect messages from their children, aggregate them and send a single aggregated message up to their parent. In this way, the base station (root of the tree) receives a single aggregated message. This solution is scalable as long as aggregated data do not grow in size. There are two types of aggregation:

- *Lossless aggregation*. This situation occurs when no information loss is tolerable during aggregation. This is the case of applications where the root multicasts a data request to the leaves and the leaves react by sending one $q$-ary symbol each (data sent by each leaf can be modeled as an integer ranging from 0 to $q-1$). At the end of the process, the root knows which symbol was transmitted by each leaf. It can be proven that symbols sent by $n$ leaves cannot be aggregated in a message whose length is below $O(n)$ when all symbols have the same probability of being sent. This cost limits the scalability of the system.
- *Lossy aggregation*. In this case, the message output by the aggregation operator contains less information than the set of messages input to the aggregation operator. Thus, the size of the output can stay the same as the size of each input. A drawback of this approach is that such information loss implies that in general the root does not know the specific data sent by each leaf. Some examples of lossy data aggregation are:
  - If data consist of temperatures, several temperatures can be aggregated by computing their average. The loss of information is clear: the base station does not know the temperature reading at each node, but just the average of all readings.
  - If data are counts, different counts can be aggregated by addition. Information loss occurs because the base station does not know the exact contribution by each node.

In addition to implosion resistance, leaf-to-root communication may also require security, understood as *confidentiality* (an intruder should not be able to learn the content being transmitted), *integrity* (it should not be possible to alter that content) and *authentication* (the receiver can make sure that the content comes from an authentic source). The framework presented in [3] does not address security. This fact represents a major drawback which disqualifies it when security requirements arise.

Cryptography permits secure data transmission over insecure communication channels. However, the addition of cryptography to many-to-one symbol transmissions is not straightforward. Secure communication protocols intended for this paradigm must be carefully designed to provide scalable transmission cost.

Some good schemes for secure many-to-one *lossless* communication exist in the literature [4–6], although they suffer from the scalability limitations inherent to lossless aggregation.

Regarding *lossy* communication, there are some proposals that enable the root to obtain the result of any mathematical function applied to the data sent by leaves. Examples of these functions are certain statistical measurements, (*e.g.* the average or variance of all measured data) or measurements of boundary values (*e.g.* min and/or max). However, no current scheme offers a good utility and security trade-off, enabling computation of a wide set of functions while providing confidentiality, authentication and perfect integrity against colluding adversaries.

### 1.1. Previous work on secure many-to-one lossy communication

Several protocols for secure many-to-one *lossy* communication can be found in the literature. These proposals follow two different approaches:

- *Unsafe aggregation:* In these proposals, intermediate nodes decrypt messages before aggregating them. Since intermediate nodes see the data they are aggregating, they can compute any mathematical function on them. However, when an intermediate node is compromised, the confidentiality of messages traversing that node becomes compromised too. Therefore, this kind of schemes only offer protection against external attackers eavesdropping transmitted messages. Schemes following this approach are [7–9].
- *Secure aggregation:* All proposals in this category are based on *homomorphic encryption schemes* which use encryption transformations to encrypt and aggregate data directly, without requiring decryption at intermediate nodes. This fact preserves the confidentiality of the data traversing the network towards the base station.

The latter approach is the only one offering protection against both external and internal attackers. Hence, we focus on this category.

As explained in [10], homomorphic encryption schemes can be symmetric or asymmetric. We next detail both classes of schemes.

#### 1.1.1. Symmetric homomorphic encryption

Symmetric schemes are orders of magnitude faster than their asymmetric counterparts. Therefore, the former are currently the only relevant ones for securing real-time applications in scenarios where all nodes are lightweight. Protocols that use symmetric homomorphic encryption can be found in [11–16].

Among the above, the proposals in [11–14] use additively homomorphic encryption techniques to perform the aggregation of data. These approaches suffer from two shortcomings:

- The set of mathematical functions that can be applied to the data stored in leaves is limited by the plaintext operation provided by the privacy homomorphism in use (addition in this case).
- Integrity is not properly addressed in scenarios with compromised nodes.

The protocols presented in [15,16] also use additively homomorphic encryption but they solve the first issue using a different representation of the data which is transmitted from the leaves towards the root of the tree. The main idea beneath both schemes is to use a bucket with a fixed number of slots, where each slot represents an interval of the values that can be sensed in the network. Each sensor, instead of reporting its sensed value, increases by one the value of one bucket slot, similarly to voting systems where different bins represent different choices and a user must express her preference by inserting a ball in a bin. This more disaggregated representation allows more flexibility in the operations computable on the leaf data.

Nevertheless, none of these two proposals solves the second issue. The authors of [15] do not consider integrity threats in their adversary model. Regarding [16], this scheme provides a certain level of integrity against a single compromised node: a single node is able to alter the final aggregated value received by the root but she can only cause a limited deviation of this value. Collusion attacks are not considered.

#### 1.1.2. Asymmetric homomorphic encryption

Asymmetric homomorphic encryption provides better system security at the expense of a higher computational cost. According to that, schemes based on those cryptosystems [17,18] are preferable in scenarios with full-fledged devices (like the ones used in VANETs, reverse multicast, etc.) or in applications where real time is not critical (*e.g.* encrypted and distributed long-term storage of data within a wireless sensor network).

The authors of [19] present a survey of secure in-network data processing. A section of this survey is devoted to the applicability of asymmetric homomorphic encryption in lightweight scenarios. The general consensus on this kind of privacy homomorphisms is that they need very long keys that imply long messages and a high computational effort that does not suit pure lightweight applications.

Even though these primitives are quite resource-demanding, we argue that they can be successfully used in several environments with high-end nodes (*e.g.* the aforementioned examples of VANETs, reverse multicast, etc.). Furthermore, asymmetric homomorphic encryption might be used in lightweight environments if applied selectively and wisely. In return, the use of this kind of encryption would improve the security properties achieved by the protocols based on it. This approach is underresearched in the literature and, in our opinion, it deserves more attention.

### 1.2. Contribution and plan of this paper

In this paper we propose a secure and scalable protocol designed to provide secure many-to-one *lossy* transmission of $q$-ary symbols in networks following a tree communication topology. It allows the base station to compute any mathematical function (*e.g.* the average, variance, min and/or max, … ) on the data sent by leaves. This proposal follows the same approach of [16]. However, asymmetric homomorphic encryption techniques are applied to provide better integrity properties than the protocol presented in [16]. These primitives are used selectively in order to allow implementation of the protocol in a wide range of environments.

Our scheme provides confidentiality because intermediate nodes aggregate encrypted data directly (without decrypting them). Perfect integrity is achieved against individual dishonest nodes trying to alter transmitted data. Colluding nodes may modify the final result of a mathematical function applied to the received data, but they can only cause a limited deviation from the correct result. Besides, at least one intermediate node needs to be in the collusion in order to achieve this deviation. In scenarios where intermediate nodes are safe from compromise by an adversary (*e.g.* an environment-monitoring application where the intermediate nodes are special vehicles and the leaves are sensor nodes), our scheme provides perfect integrity against collusions too.

Regarding the transmission cost of the protocol, our proposal offers a message length that grows logarithmically with the number of leaves of the tree. According to that, scalability is guaranteed, so that very large trees are manageable.

Section 2 describes the new protocol. Section 3 is a security analysis. Section 4 is a performance analysis. Finally, Section 5 is a conclusion.

## 2. The protocol

Our protocol assumes a tree communication model where the root corresponds to the base station which receives data from the leaves. For the sake of simplicity, we assume that only the leaves send data. According to that, intermediate nodes simply act as routers which aggregate messages. Extending the proposed scheme to accommodate data transmission from intermediate nodes is straightforward.

Let $U_i$, $1 \leqslant i \leqslant n$ be the leaf nodes. These nodes transmit symbols from a $q$-ary alphabet to the base station. Each symbol is represented by a different integer from the set $\{0,\ldots,q-1\}$.

This section is written in an evolutionary way. We start with a trivial, non-scalable protocol offering security. We then move to a scalable protocol without security. Finally we describe how to make the second protocol secure using cryptography.

### 2.1. Protocol 0: many-to-one security without scalability

Consider the following trivial, non-scalable protocol:

(1) Each leaf $U_i$ encrypts its data under a symmetric key $SK_i$ shared with the base station (root). Then, $U_i$ sends the result to its parent (intermediate node).
(2) Internal nodes aggregate the received values by concatenation.
(3) The base station receives a concatenation of encrypted values from the leaves. Then, it decrypts them and computes the desired value.

This trivial solution provides confidentiality and authentication, and it can easily offer integrity if some redundancy is added. Besides, after decryption, the base station can compute any mathematical function on the received data. The main shortcoming of this solution is that it is a lossless approach instead of a lossy one. This situation implies that the base station receives a final message with $O(n)$ length, $n$ being the number of leaves of the tree. This cost does not scale well for large values of $n$.

In short, *Protocol 0* is not scalable because it follows a lossless approach. Our aim is to come up with a lossy protocol which permits the same functionality as this trivial protocol without requiring $O(n)$ message length.

### 2.2. Protocol 1: many-to-one lossy transmission without security

A protocol execution consists of the following steps:

(1) CHALLENGE. The base station sends to the leaves a challenge consisting of an array $I$ of $q$ integers, indexed from 0 to $q-1$. Each integer $I[a]$ is linked to a unique $q$-ary symbol $a$ and will encode the number of times that symbol $a$ has been transmitted by the leaves. All integers contained in $I$ are initially set to zero.
(2) MESSAGE GENERATION.
    (a) Let $z$ be the integer representation of the $q$-ary symbol to be transmitted by a certain leaf $U_i$. Upon receiving $I$, $U_i$ computes $I_i[z] = I[z] + 1$.
    (b) $U_i$ sends $I_i$ up to its parent node.
(3) MESSAGE AGGREGATION. An intermediate node (or the base station) receives messages from its $j$ child routers/leaves and does the following:
    (a) Once all expected $I_j$ have been received, they are aggregated in a final $I$ array by computing:

```
• for $s$ in $0,\ldots,q-1$ loop
    · $I[s] = 0$
    · for $b$ in $0,\ldots,j-1$ loop
        $I[s] = I[s] + I_b[s]$
```

    (b) If the aggregating node is not the base station, it sends $I$ up to its parent node. Else, this is the final aggregated message.
(4) SYMBOL EXTRACTION. The base station receives a final $I$ that encodes exactly the number of leaves that have transmitted each symbol in the current protocol execution. The following step is to compute any mathematical function (*e.g.* minimum, maximum, average, … ) on the received data. Transmission is *lossy* since the base station has no way to know which symbol was sent by a certain leaf. Section 4 shows that the message length achieved by this protocol grows logarithmically with the number of leaves of the tree. Thus,

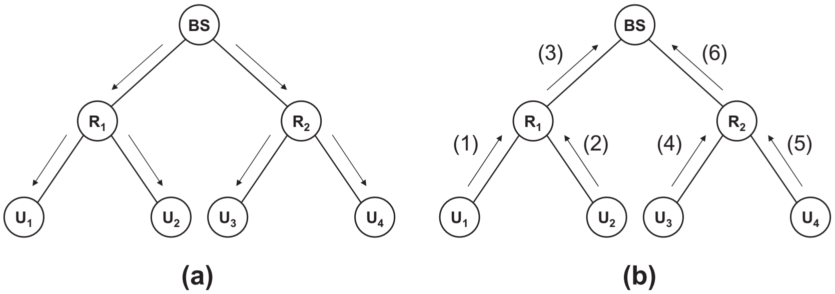

**Fig. 1.** Message flow in a protocol execution.

it scales well for large values of $n$. However, this scheme becomes inefficient when $q$ grows ($q$ represents the size of the alphabet used by leaves). Note that the size of the $q$-ary alphabet determines whether our proposal can be used for a certain application. Section 4.1.1 details for which values of $q$ our proposal outperforms *Protocol 0*.

Fig. 1 shows the message flow generated by a protocol execution in a simple scenario with a base station ($BS$), two intermediate nodes ($R_1$ and $R_2$) and four leaves ($U_1, \ldots, U_4$). Let us assume that the protocol works with a binary alphabet ($q = 2$). Leaves in the tree store the following values: $U_1 \leftarrow 1$, $U_2 \leftarrow 0$, $U_2 \leftarrow 1$ and $U_4 \leftarrow 1$.

In Fig. 1a the base station broadcasts a challenge $I = \{0,0\}$ to all leaves (Step 1 in *Protocol 1*). In Fig. 1b, message (1) sent by $U_1$ corresponds to $I_1 = \{0,1\}$ while message (2) sent by $U_2$ represents $I_2 = \{1,0\}$ (Step 2 in *Protocol 1*). Node $R_1$ constructs message (3), which corresponds to $I_{R_1} = \{1, 1\}$, by aggregating messages (1) and (2) (Step 3 in *Protocol 1*). The same process occurs in the subtree rooted by $R_2$. The latter node constructs message (6) by aggregating messages (4) and (5), which correspond to the pairs $I_3 = \{0,1\}$ and $I_4 = \{0,1\}$, respectively. Eventually, the base station $BS$ aggregates messages (3) and (6) to get the final aggregated message $I = \{1,3\}$. After that, $BS$ extracts the data from $I$ (Step 4 in *Protocol 1*) and learns the following: (i) there is one leaf which sent symbol 0; (ii) there are three leaves which sent symbol 1. The base station does not know which leaf sent each symbol but it is able to compute any mathematical function on the received data.

### 2.3. Protocol 2: making Protocol 1 secure

*Protocol 1* above avoids the scalability problems inherent to a large set of senders in reverse multicast scenarios. However, *Protocol 1* does not offer any security against attackers inside or outside the communication tree. We next explain how to add cryptography to this scheme to deal with possible adversaries wishing to eavesdrop, alter or disrupt the communications between leaves and the base station. We assume that the base station is the only trusted node in the network.

In order to provide security, now array $I$ is filled with ciphertexts instead of plain integers. Each ciphertext encodes the number of leaves having transmitted a certain symbol, as in *Protocol 1*. Due to the use of ciphertexts, leaves and intermediate nodes must be able to aggregate encrypted data directly. We use an additive privacy homomorphism to allow aggregation of encrypted data. More specifically, we use the Okamoto–Uchiyama [17] cryptosystem which is probabilistic, public-key and additively homomorphic. Note that other cryptosystems with similar properties would work in our protocol as well.

A strong requirement needed to make *Protocol 1* secure is that the base station must know which leaves are active in the tree and such active nodes must answer each query from the base station. Hence, non-transmittal is disallowed, and it is necessary to define a $q$-ary symbol to represent the *null* value to be sent by leaves which do not wish to answer a certain query.

With the purpose of discovering which leaves are active and which ones have stopped their operation, the base station periodically uses any protocol providing integrity and authentication to acknowledge multicast messages in tree-based scenarios, for example [21]. The base station will not expect any transmission from leaves which fail to acknowledge messages.

We next give a brief description of the Okamoto–Uchiyama homomorphic cryptosystem. Then, we explain our secure protocol in two phases: *set-up* and *execution*.

#### 2.3.1. The Okamoto–Uchiyama cryptosystem

*Key generation.* Choose two large primes $p$, $e$ ($|p| = |e| = k$) and let $x = p^2e$. Choose $g \in \mathbb{Z}_x^*$ randomly such that the order of $g_p = g^{p-1} \bmod p^2$ is $p$ (note that gcd($p$, $e - 1$) = 1 and gcd($e$,$p - 1$) = 1). Let $h = g^x \bmod x$. The cryptosystem keys are as follows:

- Public-key: ($x$, $g$, $h$, $k$)
- Secret-key: ($p$, $e$)

*Encryption.* Let $m$ ($0 < m < 2^{k-1}$) be a plaintext. Select $r \in \mathbb{Z}_x^*$ randomly, and compute the ciphertext

$$C = E(m, r) := g^m h^r \bmod x$$

In some cases, if the randomness $r$ is not important, we will write $E(m) := g^m h^r \bmod x$ for simplicity.

*Decryption.* Let $C_p := C^{p-1} \bmod p^2$ and

$$m := \frac{L(C_p)}{L(g_p)} \bmod p$$

where the $L$-function is

$$L(x) = \frac{x-1}{p}$$

The Okamoto–Uchiyama cryptosystem has some useful properties which are briefly reviewed as follows:

- *Homomorphic property.* Denote $E(m_0) = g^{m_0} h^{r_0} \bmod x$ for a message $0 \leqslant m_0 < p$ encrypted by some randomly chosen integer $r_0 \in \mathbb{Z}_x^*$, and similarly, $E(m_1) = g^{m_1} h^{r_1} \bmod x$ for a message $0 \leqslant m_1 < p$ encrypted by some randomly chosen value $r_1 \in \mathbb{Z}_x^*$. The encryption function has the property of allowing computations on the encrypted values without knowing their corresponding plaintexts. More exactly, $E(m_1 + m_2) = E(m_1) \cdot E(m_2)$ and consequently $E(k \cdot m) = E(m)^k$. Here, $m_1 + m_2$ and $k \cdot m$ are computed modulo $p$. Specifically, if $0 \leqslant m_1 + m_2 < p$ and $0 \leqslant k \cdot m < p$, the computation modulo $p$ is the same as that without the modulus $p$.

- *Semantic security*. This cryptosystem is proven to be semantically secure [17]. This fact implies that the Okamoto–Uchiyama ciphertexts of different messages are indistinguishable. Therefore, given two ciphertexts $E(m_0)$, $E(m_1)$ and their corresponding plaintexts $m_0$ and $m_1$, a polynomial-time attacker with access to the public parameters of the Okamoto–Uchiyama cryptosystem is able to randomly guess the correct ciphertext linked to $m_0$ with probability 1/2.
- *Public randomizability*. From the homomorphic property and the semantic security, it is straightforward to see that the Okamoto–Uchiyama ciphertexts have public randomizability. This property states that: (i) ciphertexts can be randomized by merely using the public key; and (ii) given two randomized ciphertexts $E(m_0, r_0)$, $E(m_1, r_1)$ and their corresponding plaintexts $m_0$ and $m_1$, an attacker is able to guess the randomized ciphertext linked to $m_0$ with probability 1/2. In order to randomize a ciphertext $C = E(m, r)$, it is only required to randomly select $r' \in Z_x^*$ and compute the following:

  $$C' = E(m,r) \cdot E(0,r') \bmod x$$

  According to this computation, the resulting randomized ciphertext is $C' = E(m+0, r+r') = g^m h^{r+r'}$.

### 2.3.2. Protocol set-up

The base station generates the parameters of an Okamoto–Uchiyama cryptosystem: $x$, $g$, $h$, $k$, $p$, $e$. All those values but $x$ are kept secret by the base station. Note that $x$ must be known by all nodes in the tree since the protocol requires intermediate nodes and leaves to compute the ciphertext operations of the privacy homomorphism.

Each leaf $U_i$ shares a secret key $K_i$ with the base station. Besides, the base station gives to each $U_i$ a ciphertext with $\omega_i = gh^{ID_i}$ encrypted under $K_i$, where $ID_i$ is a random integer used by the base station to identify a certain $U_i$. Note that only the base station knows the values $g$, $h$ and $ID_i$. Both $K_i$ and $\omega_i$ must be kept secret by $U_i$. Furthermore, each node including the leaf node shares a secret key with its parent and the communication between them is encrypted with their shared key.

### 2.3.3. Protocol execution

We next explain the steps performed in a protocol execution. For simplicity, we omit the description of the symmetric encryption operation in the sequel. Compared with the modular exponentiations, symmetric encryption only introduces very slight additional overhead.

(1) CHALLENGE. The base station publicly sends to the leaves a challenge $(v, I) = (v, I[0],\ldots,I[q-1])$ consisting of a random value $v$ and an array $I$ containing $q$ ciphertexts. The initial ciphertext $I[s]$ linked to each symbol $s$ is computed as $I[s] = g^{\alpha_s} h^{\delta_s} \bmod x$ (note that $I[s] = E(\alpha_s)$ where $s = 0,1,\ldots, q-1$). Value $\alpha_s$ is a large random integer such that $\alpha_s + n < p$, being $n$ the number of leaves and $p$ a parameter from the Okamoto–Uchiyama cryptosystem (see Section 4.1 for details about this inequality). Regarding $\delta_s$, it is a random integer without any restriction. The whole message is signed by the base station to prevent adversaries from modifying the challenge. The public key for verification is known and accepted as valid by all nodes in the tree.

(2) MESSAGE GENERATION. Upon receiving the challenge and verifying its signature, each leaf $U_i$ does the following:

(a) Compute a pseudo-random value $a_i \leftarrow \mathscr{H}(v \| K_i)$, where $\mathscr{H}$ is a one-way hash function.

(b) Use $a_i$ to alter each ciphertext contained in $I$ by computing:

- for $s$ in $0,\ldots,q-1$ loop

  · $I_i[s] = I[s]^{a_i} \bmod x$ (this represents $E(\alpha_s)^{a_i} = E(a_i \cdot \alpha_s)$).

After this homomorphic operation, the ciphertext linked to symbol $s$ is equal to $I_i[s] = g^{a_i \cdot \alpha_s} h^{a_i \cdot \delta_s} \bmod x$ ($I_i[s]$ represents $E(a_i{\cdot}\alpha_s, a_i{\cdot}\ \delta_s)$).

(c) Transmit the $q$-ary symbol —denoted by $\bar{s}$ $(0 \leqslant \bar{s} \leqslant q-1)$— that the leaf wants to send by multiplying $\omega_i$ with the ciphertext contained in $I_i$ corresponding to $\bar{s}$:

$$I_i[\bar{s}] = I_i[\bar{s}] \cdot \omega_i \bmod x$$

($I_i[\bar{s}] \cdot \omega_i$ represents $E(a_i \cdot \alpha_{\bar{s}}, a_i \cdot \delta_{\bar{s}}) \cdot E(1, ID_i)$, hence the resulting $I_i[\bar{s}]$ represents $E(a_i \cdot \alpha_{\bar{s}} + 1, a_i \cdot \delta_{\bar{s}} + ID_i)$). After this homomorphic operation, the resulting array $I_i$ is as follows:

$$I_i[s] = \begin{cases} E(a_i \cdot \alpha_s + 1, a_i \cdot \delta_s + ID_i) & \text{if } s = \bar{s} \\ E(a_i \cdot \alpha_s, a_i \cdot \delta_s) & \text{if } s \neq \bar{s} \end{cases}$$

(d) Send $I_i$ up to $U_i$'s parent node. The ciphertexts transmitted by $U_i$ are $I_i = (I_i[0], I_i[1],\ldots,I_i[q-1])$.

(3) MESSAGE AGGREGATION. An intermediate node (or the base station) receives messages from its $j$ child routers/leaves and does the following:

(a) Once all expected $I_j$ have been received, aggregate them in a final array $I$ by computing:

- for $s$ in $0,\ldots,q-1$ loop
  $I[s] = \prod_j I_j[s] \bmod x$

($\prod$ is the homomorphic ciphertext operation corresponding to plaintext addition in the Okamoto–Uchiyama cryptosystem.)

(b) If the aggregating node is not the base station, send $I$ up to the aggregating node's parent node. Else, this is the final aggregated message.

(4) SYMBOL EXTRACTION. The base station decrypts and processes the final aggregated messages as follows:

(a) Decrypt each ciphertext contained in $I$:

$$\varphi_s = Decrypt(I[s]) \quad \text{for } s \in \{0,\ldots,q-1\}$$

(*Decrypt* corresponds to the Okamoto-Uchiyama function used to decrypt ciphertexts.)

(b) Then, obtain the number $\mu_s$ of times that each $q$-ary symbol $s$ has been sent by computing: $\mu_s = \varphi_s - (\alpha_s \cdot \sum_{i=1}^{n} a_i)$, for $s \in \{0,\ldots,q-1\}$. (Note that only $a_i$ values from active leaves must be counted in this operation.)

(c) Finally, check the received values using the procedure described in Section 2.4. If everything is correct, multicast a signed acknowledgment message to all leaves and operate values $\mu_0,\ldots,\mu_{q-1}$ to compute any required mathematical function (*e.g.* median, average, min/max, … ). Otherwise, multicast a corrupt message alert and call the tracing protocol detailed in Section 2.5.

## 2.4. Data checking

The base station performs three checks to verify the integrity of the message which contains the aggregation of all the data sent by the leaves of the tree.

(1) *Consistency check*. The base station receives $q$ ciphertexts contained in $I$. Each ciphertext can be represented as $g^x h^y$. The base station obtains the $h$ components of all ciphertexts and aggregates them in a final component $h' = h^{y_0 + \cdots + y_{q-1}}$. Since the base station knows which leaves will send symbols, it is able to compute $h'$ locally and compare it with

the received value. If both values are the same, the received message passes the consistency check.

(2) *Sum check.* Since each active leaf must send a symbol, $\sum_{s=0}^{q-1}\mu_s$ must be equal to the number of leaves which are active in the tree.

(3) *Range check.* The base station checks that $0 \leqslant \mu_s \leqslant n_a$ for $s \in \{0,\ldots,q-1\}$, where $n_a$ is the number of active leaves.

If all these checks hold, the base station accepts the collected data. Else, it rejects them.

### 2.5. Tracing message corruption

During symbol extraction, the base station checks the integrity of the received message. If this verification fails, the base station identifies this message as corrupt and raises an error. Then, the base station must run a procedure to exclude the dishonest node(s) from the network.

Even though our scheme is not linked to any specific tracing protocol, we propose to use the procedure introduced in [4] to remove the dishonest nodes from the tree. This protocol requires the following assumptions to work:

- Each intermediate node must store incoming messages until the corresponding aggregated message is acknowledged by the base station.
- Each intermediate node or leaf must have a public/private key pair (e.g. RSA keys [22]). The corresponding public key must be accepted by all its parent nodes and the base station.
- Each node timestamps and signs the message it sends to its parent. The signature is also sent. This is done at the link level: those signatures are not forwarded to the base station under normal operation. When an intermediate router receives a message from one of its child nodes, it must check the child's signature. If it is not correct, then the message should not be accepted.

The tracing protocol described in [4] permits to identify the nodes where message corruption happened so that they can be removed from the multicast tree. It is a recursive protocol initially called as *Tracing (base station, final aggregated message)*, whose pseudo-code is detailed below.

#### 2.5.1. Tracing(explored router, explored corrupt message)

The base station does:

(1) Obtain from the "explored router" the messages and signatures it has received from its child nodes (before aggregation).

(2) Check the signatures on the "explored corrupt message" and on the messages obtained from the "explored router". This step is omitted when the explored router is the base station.

(3) If any of the signature checks fails, mark the "explored router" as guilty and go to Step (6).

(4) Check whether the aggregation of the obtained messages matches the "explored corrupt message". If not, mark the "explored router" as guilty and go to Step (6).

(5) Check each stored message using the procedure described in Section 2.4.

  (a) For corrupt messages sent by leaves, mark those leaves as guilty.

  (b) For each intermediate router for which the correctness check has failed call *Tracing(intermediate router, aggregated message sent by router).*

(6) Remove any guilty nodes from the network.

## 3. Security analysis

Our proposal achieves implosion resistance by aggregating data, which causes some information loss. The aggregation process is performed at the intermediate nodes, which aggregate the received ciphertexts. Hence, the ciphertexts sent by the leaves towards the base station are different when they reach their destination. The transmitted data are modified at each hop in the network as part of the aggregation process. Each aggregated ciphertext is considered to be a new message. This new message is completely valid and its source is the intermediate node which has generated it. According to that, it is not possible to provide strict end-to-end integrity and authentication, that is, the base station cannot directly check the integrity or identify the true source of a certain symbol originated by some leaf. Accepting this point, we show that our proposal provides practical security, instead of ideal security. Our scheme is robust against the typical attacks launched in many-to-one scenarios. We refer to such attacks to prove the security properties achieved by our scheme: *confidentiality*, *authentication* and *integrity*.

### 3.1. Adversary model

Our attacker model considers an adversary who can control nodes (which become compromised nodes) and can also access communication lines to capture, modify and retransmit messages. Her computational power does not permit the adversary to break current computationally secure cryptosystems.

#### 3.1.1. Possible attacks

An adversary can try to perform the following attacks:

- Eavesdrop messages and try to read the information they convey.
- Alter the number of times that a certain symbol has been sent by the leaves.
- Impersonate a certain leaf $U_i$.
- Do not aggregate messages received from some of her child nodes.

Note that only an adversary who has compromised an intermediate node of the network could try to perform the last attack.

### 3.2. Attacks and security properties

#### 3.2.1. Eavesdrop messages

This attack is against the *confidentiality* property. Participants in the protocol exchange a set of ciphertexts which have been previously computed by the base station. Intermediate nodes can only perform homomorphic aggregation on the ciphertexts; they can neither decrypt them nor generate new valid ciphertexts.

We deal next with the confidentiality of the symbols sent by the leaf nodes in front of an eavesdropper. These symbols are encrypted using the public key of the base station. The attacker cannot decrypt these messages and she cannot access the decryption done by the base station. According to that, she can only eavesdrop the ciphertexts sent by the leaves towards the base station. Since the aggregation performed by the intermediate nodes generates information loss in the transmitted data, the best option for the eavesdropper is to capture the ciphertexts received by the parent nodes of the leaves. Therefore, from now on, only the eavesdropper who controls the parent nodes of the leaves will be considered.

In the following, we show that such an eavesdropper cannot get any useful information from the messages sent by the leaves. This is due to the semantic security property of the underlying Okamoto–Uchiyama cryptosystem [17].

Let us consider the data that the eavesdropper can get from the public channel and the compromised intermediate nodes:

- In the protocol set-up (see Section 2.3.2), the attacker can get the system parameter $q$, the description of the public algorithms in use (hash function, key generation algorithm and encryption/decryption procedures of the Okamoto–Uchiyama cryptosystem) and the value $x$ (the public key of the base station). Nevertheless, the eavesdropper does not know the factorization of $x$. The security of the Okamoto–Uchiyama cryptosystem is based on the difficulty of factoring large composite integers. As long as there is no algorithm which efficiently factors an arbitrary integer, the eavesdropper cannot get any useful information from $x$. In this protocol step, the attacker also gets the ciphertext $\omega_i = E(1, ID_i)$. This element is encrypted using the symmetric key $K_i$ shared between the base station and the leaf $U_i$. Let us assume that the employed symmetric encryption is secure. Therefore, the encrypted $\omega_i$ is random from the point of view of the eavesdropper.
- The protocol execution (see Section 2.3.3) is divided in four steps. In the third one (message aggregation), the ciphertexts sent by the leaves towards the base station are aggregated/compressed by the intermediate nodes. In the fourth step (symbol extraction), the base station extracts the symbols sent by the leaves. The aggregated ciphertexts cannot contain more information than the original ciphertexts and the eavesdropper cannot access the decryption of the base station. As a consequence, only the first two steps are considered. In the first step (challenge), the eavesdropper can intercept the vector ($v$, $E(\alpha_0),\ldots,E(\alpha_{q-1})$) sent to $U_i$ by the base station, where $E(\alpha_s) = g^{\alpha_s} h^{\delta_s} \bmod x$ for $s = 0,1,\ldots,q-1$. The eavesdropper does not know $g$ or $h$, hence $\alpha_s$ and $\delta_s$ are random values from the attacker's viewpoint. In the second step (message generation), each leaf $U_i$ generates $I_i = (I_i[0], I_i[1],\ldots,I_i[q-1])$ and sends it to her parent node. The eavesdropper can obtain $I_i$ which contains

$$I_i[s] = \begin{cases} E(a_i \cdot \alpha_s + 1, a_i \cdot \delta_s + ID_i) & \text{if } s = \bar{s} \\ E(a_i \cdot \alpha_s, a_i \cdot \delta_s) & \text{if } s \neq \bar{s} \end{cases}$$

  where $a_i$ is the keyed hash output $\mathscr{H}(v \| K_i)$ and $K_i$ is the symmetric key shared between $U_i$ and the base station. Array ($I_i[0]$, $I_i[1],\ldots,I_i[q-1]$) encodes the $q$-ary symbol that $U_i$ wants to send, which we represent by $\bar{s}$. In this array, only the $\bar{s}$-th entry encrypts value *1* while the other entries encrypt value *0*. Note that $a_i$, $\alpha_s$, $\delta_s$, $ID_i$ are unknown and random in their respective sampling space from the point of view of the eavesdropper.

From all the information gathered by the attacker, the array $I_i$ is the only data structure that contains the symbol sent by $U_i$. Therefore, we focus on proving that this array of ciphertexts provides confidentiality in front of an eavesdropper.

Due to the semantic security and the public randomizability of the Okamoto–Uchiyama cryptosystem, for each array entry $I_i[s]$, the eavesdropper cannot distinguish $E(a_i \cdot \alpha_s + 1, a_i \cdot \delta_s + ID_i)$ from $E(a_i \cdot \alpha_s, a_i \cdot \delta_s)$. That is, for $0 \leqslant s \leqslant q-1$, the eavesdropper cannot decide whether $I_i[s] = E(a_i \cdot \alpha_s + 1, a_i \cdot \delta_s + ID_i)$ or $I_i[s] = E(a_i \cdot \alpha_s, a_i \cdot \delta_s)$. Here, $E(a_i \cdot \alpha_s + 1, a_i \cdot \delta_s + ID_i)$ can be viewed as the randomization of $E(a_i \cdot \alpha_s + 1, a_i \cdot \delta_s)$ using $E(0, ID_i)$. The eavesdropper does not know $ID_i$, hence the attacker cannot distinguish $I_i[\bar{s}]$ from $I_i[s]$ or deduce which index $s$ is the exact symbol $\bar{s}$ that $U_i$ is transmitting. As a result, our protocol provides confidentiality of the data sent by the leaves.

Last but not least, our proposal requires all active leaves to send a value to the base station after each query (an actual symbol or a *null* symbol). Instead of performing a non-transmittal, an active leaf with nothing to transmit sends a *null* symbol whose encryption is indistinguishable from the encryption of an actual symbol. Therefore, the eavesdropper cannot even ascertain which leaves transmit actual symbols.

### 3.2.2. Alter the number of times that a symbol has been sent

This attack is against the *integrity* property. An intruder willing to perform this attack for a certain symbol $z$ without being detected must be able to modify the component $g$ of $I[z]$ without leaving any trace in component $h$ and without damaging the ciphertext.

We distinguish two categories of attackers: (i) external attackers; (ii) internal attackers, *i.e.*, leaves and intermediate nodes (excluding the base station).

As explained by Chan and Castelluccia in [20], it is not possible to achieve perfect integrity for aggregated messages in scenarios with internal attackers. Nevertheless, partial integrity should be provided in this case, meaning that a compromised internal node should only be capable of introducing some (limited) deviation in the final aggregated value, rather than generating a new one.

Along this line, we show that: (i) our protocol preserves the integrity of the messages against external attackers; (ii) for internal attackers, our protocol provides integrity if the compromised nodes do not collude. The latter is a rational assumption in practice. If a compromised leaf and a compromised intermediate node collude, they can break the integrity of the system in some cases. However, in those situations, they can only achieve a limited deviation of the final result. We next explain how our scheme behaves against adversaries attacking integrity.

For an attacker with knowledge of $g$, it would be very easy to add or subtract any desired value $c$ to a certain ciphertext. For instance, a subtraction would be done as follows:

$$\frac{I_i[z]}{g^c} = g^{a_i \cdot \alpha_z + 1 - c} h^{a_i \cdot \delta_z + ID_i}$$

Note that to pass the sum check (see Section 2.4), an attacker needs to add the value $c$ to another ciphertext, which can be done in a similar way.

However, only the base station knows $g$. According to that, an attacker can only use the ciphertexts provided by the base station to perform her attack. These ciphertexts are always of the form $g^m h^r$. It is mandatory for the base station to always generate ciphertexts with different $r$ values to prevent an attacker from obtaining a $g^m$ alone (which could be achieved by dividing one ciphertext by another having the same $r$ value). Hence, neither internal nor external attackers can obtain the value $g$.

We first consider how integrity is achieved against *external attackers*. Since the communications between each node and its parent are secured using symmetric encryption, an external attacker cannot know $I$ (see Section 2.3.3). Any modification of these communications results in the base station receiving a random value. Then, after decryption, the base station obtains a random $h'$. Hence, the consistency check $h' = h^{y_0 + \cdots + y_{q-1}}$ (see Section 2.4) holds with only a negligible probability $1/Order(h)$, where $Order(h)$ is the order of $h$ in the Okamoto–Uchiyama group. Note that $h = g^x \bmod x$, where $x = p^2 e$ and $g$ is randomly chosen from $\{1,\ldots,x-1\}$. Therefore, $g$ has order $p(p-1)(e-1)$ with probability $\frac{p(p-1)(e-1)}{x} = \frac{(p-1)(e-1)}{p \cdot e} \approx 1$. Besides, value $h$ has order $(p-1)(e-1)$ with the same probability. Accordingly, the probability that the consistency check $h' = h^{y_0 + \cdots + y_{q-1}}$ passes is about $\frac{1}{(p-1)(e-1)}$. This probability is negligible, hence our proposal provides integrity against external attackers.

Now we consider how the system behaves against *internal attackers*. This kind of adversaries can attempt to modify the $\mu_s$ values which are received by the base station without being detected. An internal attacker can be a leaf or an intermediate node:

- *The attacker is a leaf.* This adversary can try to contribute more than once with her own $\omega_i$. In this case, the final aggregated message will not pass the consistency check (see Section 2.4). Also, if she alters the encrypted communication between other nodes, the base station will output a random $h'$ that will pass the consistency check with only negligible probability. As a result, a dishonest leaf working alone is only able to modify her own contribution, and this is not considered an attack.
- *The attacker is an intermediate node.* This adversary receives messages from her children and knows the challenge $I$ broadcast by the base station. According to that, she can launch the following attacks:
  - She can try to modify the communications between other nodes. In this case, the base station will output a random $h'$ that will pass the consistency check with only negligible probability.
  - She can try to modify the symbols sent by her children using the challenge $I$. In this way, she can use the array $I$ to add/subtract the random value $\alpha_s$ encrypted in each ciphertext $I[s] = g^{\alpha_s}h^{\delta_s}$. She could choose some integers $\gamma_0,\ldots,\gamma_{q-1}$, perform a subtraction by computing $I_i[0]/I[0]^{\gamma_0}, \ldots, I_i[q-1]/I[q-1]^{\gamma_{q-1}}$, and then send the message towards the base station as usual. After receiving the final message, the base station performs the consistency check over the data. The base station expects a final $h' = h^c$ where $c = \delta_0(a_1 +\cdots+ a_n) +\cdots+ \delta_{q-1}(a_1 +\cdots+ a_n) + (ID_1 +\cdots+ ID_n)$. However, due to the subtraction, what it really gets is $h^{c'}$ where $c' = c - (\delta_0^{\gamma_0} + \ldots + \delta_{q-1}^{\gamma_{q-1}}) \bmod \mathit{Order}(h)$. Since the $c'$, $c$, $\delta_j$ are unknown to the attacker and look like random integers to her, by choosing $\gamma_j$, she has only a probability of $1/\mathit{Order}(h) < 1/p$ to pass this consistency check. In addition to that, the base station will output the modified values $\mu'_s = \mu_s - \alpha_s^{\gamma_0} \bmod p$. Since the attacker does not know $\alpha_s$, and $\mu'_s$ will be a random integer in $\mathbb{Z}_p$, $\mu'_s$ can pass the range check $0 \leqslant \mu'_s \leqslant n_a \leqslant n$ with at most a probability $n/p$. Notice that $n$ is the number of leaves and, in practice, it should be less than $2^{30}$. On the other hand, $p$ is typically greater than $2^{300}$ in order to guarantee the hardness of factoring of $x = p^2e$ [17]. Therefore, the attacker's success probability $n/p < 2^{-270}$ is negligible in practice. Hence, this attack will be detected by both the consistency and the range checks.

The above analysis shows that our proposal provides integrity against external attackers and against internal attackers too if the latter do not collude. Now we examine the integrity of our scheme when internal attackers work together. In this scenario there are two different attacks:

- Two dishonest nodes $U_c$ and $U_d$ share their ciphertexts ($\omega_c$ and $\omega_d$ respectively) and operate them to isolate a component $h$:

  $$\frac{\omega_c}{\omega_d} = h^{ID_c - ID_d}$$

  The only use of this value is to exchange the trace ($ID_i$) of one user for the other user's trace. For example, it means that $U_c$ can select her own symbol and she can also select the one from $U_d$. Note that, this is only harmful for $U_d$, who loses her own contribution.
- A dishonest leaf and a dishonest intermediate node collude. In this case, our proposal fails to provide integrity. However, the adversaries can only deviate the final result by a certain integer $\lambda$. We next detail how this attack affects our system. Without loss of generality, we assume that the leaf node $U_1$ colludes with an intermediate node which is close to the base station. Both nodes want to change the values of $\mu_0$ and $\mu_1$. Their attack works as follows:
  - The malicious intermediate node operates as usual and obtains $I[s]$ for $s \in \{0,\ldots,q-1\}$.
  - Since the intermediate node knows $\omega_1$ from $U_1$, she can choose a number $\lambda > 0$ ($\lambda = 0$ does not produce any deviation) and compute:

  $$I[0]' = I[0]\cdot\omega_1^{\lambda} = g^{*+\lambda}h^{*+ID_1\cdot\lambda}, I[1]' = I[1]/\omega_1^{\lambda} = g^{*-\lambda}h^{*-ID_1\cdot\lambda}$$

  where $*$ represents the original values if the attack had not occurred.
  - The intermediate node follows the rest of the protocol as usual and sends $I[s]'$ to the root, where $I[s]' = I[s]$ except for $I[0]'$ and $I[1]'$.

It is straightforward to verify that the above attack can pass both the consistency and the sum checks. However, the base station will output the values of the symbols:

$$\mu'_0 = \mu_0 + \lambda, \mu'_1 = \mu_1 - \lambda, \mu'_2 = \mu_2, \ldots, \mu'_{q-1} = \mu_{q-1}.$$

The values of $\mu_0$ and $\mu_1$ are modified to $\mu'_0$ and $\mu'_1$, respectively. Hence, if $\mu'_0 \leqslant n_a$ and $0 \leqslant \mu'_1$, this attack passes the range check and the base station cannot detect it. Since $\mu_0 \leqslant n_a$, $0 \leqslant \mu_1$ and $\mu_0 +\ldots+ \mu_{q-1} = n_a$, if $\lambda \leqslant n_a$ then:

$$\begin{aligned} Pr(\mu'_0 > n_a) &= Pr(n_a \geqslant \mu_0 \leqslant n_a - \lambda + 1) \\ &= \sum_{i=n_a-\lambda+1}^{n_a} \binom{n_a}{i} (1/q)^i (1-1/q)^{n_a-1} \end{aligned}$$

$$Pr(\mu'_1 < 0) = Pr(0 \leqslant \mu_1 \leqslant \lambda - 1) = \sum_{i=0}^{\lambda-1} \binom{n_a}{i} (1/q)^i (1-1/q)^{n_a-1}$$

Hence, the collusion attack can succeed with probability

$$\begin{aligned} Pr(\mu'_0 \leqslant n_a, 0 \leqslant \mu'_1) &= \left[\sum_{i=0}^{n_a-\lambda} \binom{n_a}{i} (1/q)^i (1-1/q)^{n_a-1}\right] \\ &\quad \times \left[\sum_{i=\lambda}^{n_a} \binom{n_a}{i} (1/q)^i (1-1/q)^{n_a-1}\right] \end{aligned}$$

which decreases as $\lambda$ grows; in particular, the above attacker's success probability is *0* for $\lambda \geqslant n_a$. According to that, the $\lambda$ value that the adversaries can use is constrained because a large $\lambda$ cannot pass the range check. Thus, colluding adversaries can only achieve a certain deviation of the final result. This result is consistent with the conclusions presented in [20].

We next discuss the maximum deviation which can be achieved in different scenarios and how it affects the applicability of the

**Table 1**
Success probability for different $\lambda$ values in three different scenarios.

| $\lambda$ | $n_a$ = 100, $q$ = 14 (%) | $n_a$ = 500, $q$ = 76 (%) | $n_a$ = 1000, $q$ = 153 (%) |
|---|---|---|---|
| 1 | 99.9 | 99.9 | 99.9 |
| 4 | 93.2 | 89.5 | 89.1 |
| 6 | 72.6 | 64.3 | 63.6 |
| 8 | 42.3 | 33.8 | 33.2 |
| 10 | 17.6 | 12.8 | 12.5 |
| 12 | 5.3 | 3.5 | 3.4 |
| 15 | 0.4 | 0.3 | 0.3 |

proposed protocol. Table 1 presents the attacker's success probability for several $\lambda$ values in three different scenarios. Each scenario has a selected combination of $n_a$ and $q$ values. Each selection is consistent with the results presented in Section 4.1.1. It can be seen from the table that it is quite difficult for an attacker to achieve a deviation greater than 10. More specifically, the success probability is practically negligible for deviations greater than 12. On the other hand, a deviation between 1 and 4 succeeds with a probability of 89% or more. Besides, it can be observed that the success probability for a certain $\lambda$ value does not vary significantly across scenarios. Therefore, those results are also useful when considering other scenarios with a different number of active nodes.

Let us illustrate the effect of the deviation in two different applications of our protocol, where leaf nodes are sensors measuring the temperature of chemical tanks in a petrochemical plant. Both applications are implemented in a scenario with 1000 leaves-sensors and a 153-ary alphabet. Each sensor node monitors a certain tank which is independent from the rest of tanks. The use of a 153-ary alphabet allows the sensor nodes to transmit temperatures from −76 to +76°C. We have simulated the temperatures in tanks by drawing from the uniform distribution in the [−76,+76] interval.

In the first application, the base station wants to retrieve the average temperature of all tanks. In the second application, the base station wants to know the maximum temperature of all tanks. We next discuss both applications:

- *Average temperature.* We have simulated this scenario with and without attackers. The simulations without attackers show that the average temperature is a value between −2.6 and +1.5°. Let us consider an attacker who decreases by 4 the frequency of the symbol linked to −76 and increases by 4 the frequency of the symbol linked to +76. The third column of Table 1 shows that a deviation $\lambda = 4$ can pass the data checking procedure with a high probability (89.1%). The simulations show that this attack increases the average temperature by 0.6. This increment is practically negligible in this scenario.
- *Maximum temperature.* In our simulations, temperatures are uniform: as a result, there are several sensors with reading +76° or close to this temperature. Therefore, altering the frequencies of a few sensors is unlikely to change the maximum temperature significantly. However, if temperatures are distributed in such a way that high temperatures are very rare, the attacker will succeed easily in misleading the base station about the maximum sensed value. In order to achieve that, the attacker only needs to contribute once to the maximum symbol of the alphabet (+76) and decrease by one the frequency of any other symbol. Table 1 shows that such a deviation with $\lambda = 1$ has a success probability of 99.9%.

### 3.2.3. Impersonate a certain leaf $U_i$

This attack refers to the *authentication* property. An intruder willing to impersonate a certain leaf $U_i$ without being detected needs knowledge of $\omega_i$ and $K_i$. Both values are only known by the base station and by $U_i$. As long as these values are not compromised, the system provides authentication.

Note that ciphertexts sent by leaves at each protocol execution are modified using a fresh pseudo-random value $a_i$ which depends on the current challenge. According to that, an attacker cannot perform replay attacks using old valid messages from authentic leaves.

### 3.2.4. Do not aggregate a message received from a child node

A dishonest intermediate node can decide not to aggregate a message received from some of its child nodes. Since non-transmittal is not allowed, the base station will execute the tracing protocol and punish the guilty node.

## 4. Performance analysis

We evaluate the protocol performance in terms of message length and computational cost at the nodes.

### 4.1. Message length

Our scheme keeps the length of messages constant on their way from the leaves towards the base station. When $n$ leaves send their encrypted symbols with our scheme, all symbol transmissions are eventually aggregated into a single array of $q$ integers. Each integer can be at most $x$ (this is a parameter of the cryptosystem in use), so its length is $\log_2 x$. Equivalently, the bitlength $\Omega$ of each integer in that array is

$$|\Omega| = \log_2 x = \log_2(p^2 e) = 2\log_2 p + \log_2 e = 3\log_2 p$$

where we have used that, in the Okamoto–Uchiyama cryptosystem, $x = p^2 e$ with $|p| = |e|$.

If $\alpha := \max_s \alpha_s$ —that is, the maximum value of the $\alpha_s$ used for all symbols $s$ at the beginning of *Protocol 2* (see Section 2.3.3)—, then parameter $p$ should be large enough to allow a single ciphertext to contain $\alpha$ plus all the contributions from the leaves (the worst case arises when all the leaves send the same symbol towards the base station). As a result, a lower bound on $p$ is:

$$p > \alpha + n$$

This implies that $|\Omega| \approx 3\ \log_2(\alpha + n)$. Thus, the total message length is $q \cdot 3 \cdot \log_2(\alpha + n)$. This expression is dominated by $\log_2 n$ as the number of users grows. So the message length in our scheme grows logarithmically with the number of leaves in the tree.

### 4.1.1. Comparison with Protocol 0

In order to use *Protocol 0*, we must assume that each user $U_i$ shares with the root of the tree a key $K_i$ corresponding to a block cipher (*e.g.* AES, [23]). The message $M$ containing the symbol $b$ will look like

$$M = E_{K_i}(b \| ts \| ck), U_i$$

where $E_{K_i}()$ stands for the encryption function of the block cipher, $ts$ is a timestamp, $ck$ is a checksum and $U_i$ is the identity of user $U_i$. Integrity is ensured by $ck$ and $ts$ (the time-stamp prevents replacing future transmissions with past transmissions).

When $n$ users simultaneously send their encrypted symbols using *Protocol 0*, $n \cdot (B + \log_2 n)$ bits are received by the root, assuming that $B$ is the block bitlength of the block cipher and $\log_2 n$ is the bitlength of the user identifier $U_i$. We assume also that the bitlength of $b \| ts \| ck$ is less than or equal to $B$. For a block cipher such as AES, $B$ should be equal to *128* bits, so the previous assumption is reasonable.

The message length in our scheme is $q \cdot 3 \cdot \log_2(\alpha + n)$. The bitlength of the message received by the root in *Protocol 0* is $n \cdot (128 + \log_2 n)$. The values of $q$ for which our proposal outperforms *Protocol 0* are

$$q < \frac{n \cdot (128 + \log_2 n)}{3 \cdot \log_2(\alpha + n)}$$

Table 2 shows the maximum size of the $q$-ary alphabet which can be used by our scheme (above this value it no longer outperforms *Protocol 0*). Results are given as a function of the number of leaves $n$. The bitlength of $\alpha$ is fixed to *300* bits.

The size of the $q$-ary alphabet determines whether our proposal can be used for a certain application. If a large $q$-ary alphabet is

**Table 2**
Maximum size of the $q$-ary alphabet which can be used by our scheme. Results are given as a function of the number of leaves $n$. The bitlength of $\alpha$ is fixed to 300 bits.

| $n$ | Max $q$ allowed |
|---:|---:|
| 100 | 14 |
| 500 | 76 |
| 1000 | 153 |
| 10,000 | 1569 |
| 100,000 | 16,067 |
| 1,000,000 | 164,368 |

used, our scheme is better than *Protocol 0* only for a large number of leaves (thousands of them). This is good news for our proposal in the sense that this kind of scenario (with a huge $n$) is the usual one in many-to-one scenarios (*e.g.* sensor networks, reverse multicast, etc.).

### 4.2. Computational cost

Next, we analyze the time complexity of the protocol in five operations: message generation, message aggregation, symbol extraction, data checking and message corruption tracing:

*Message generation:* Leaf $U_i$ generates a pseudo-random value $a_i$, which costs $O(1)$ (this operation corresponds to a hash function). Then it uses $a_i$ to modify the challenge array received from the root. This represents $q \cdot a_i$ multiplications of ciphertexts.

*Message aggregation:* The aggregation process is executed by the intermediate nodes and it requires at most $O(n)$ ciphertext operations of the privacy homomorphism (this operation is a multiplication). This is done for each symbol in the $q$-ary alphabet. The total cost for this step is $O(q \cdot n)$.

*Symbol extraction:* This step requires $q$ decryptions and $n$ multiplications which are computed only once. As a result, the total cost is $O(q) + O(n)$. The base station performs this step.

*Data checking:* Three checks are needed to verify the integrity of the received data. These three checks require $q$ additions each. According to that, the total cost is $O(q)$. The base station performs this step.

*Message corruption tracing:* This process is only executed when a corrupt message is detected. The base station analyzes each intermediate node of the tree looking for the source of the corrupt message (see Section 2.5 for details about this). Let us assume a $k$-ary tree (each node has $k$ children). For each intermediate node, the base station does the following:

- Check $k$ signatures. This cost is $O(k)$.
- Perform a message aggregation of the stored messages. As explained previously, the cost for this step is $O(q \cdot n)$.
- Check whether the aggregation of the stored messages matches the "explored corrupt message". This cost is $O(1)$.
- Check the $k$ stored messages using the data checking procedure. As shown previously, the cost for each message is $O(q)$. Therefore, the cost for this step is $O(k \cdot q)$.

According to that, the cost for analyzing each intermediate node is $O(k) + O(q \cdot n) + O(q) + O(k \cdot q)$. In a $k$-ary tree with $n$ leaves, the total number of intermediate nodes is $\tau = \frac{n \cdot k - 1}{k - 1} - (n + 1)$. Hence, the total cost for the tracing operation is $O(\tau \cdot (k + q \cdot n + q + k \cdot q))$.

The base station carries out all the encryption and decryption operations of the homomorphic cryptosystem in use. This device also performs the tracing operation. The rest of the nodes of the tree (leaves and intermediate nodes) only perform multiplications of the ciphertexts which are generated by the base station. Thus, the computational cost of our protocol is affordable in a wide range of environments.

## 5. Conclusion

A scalable protocol for secure lossy many-to-one transmission in tree-based networks has been presented. This proposal offers security to the data transmitted from the leaves towards the root while allowing the base station to compute any mathematical function (*e.g.* min/max, average, median, etc.) on the received data. Confidentiality is achieved by intermediate nodes aggregating encrypted data directly. Perfect integrity is provided only against individual dishonest nodes trying to alter transmitted data. Colluding nodes can modify the final result of a mathematical function applied to the received data, but they can only cause a limited deviation from the correct result. In scenarios where intermediate nodes are safe from compromise by an adversary, perfect integrity is provided too.

Messages transmitted during a protocol execution keep a constant bitlength on their way from the leaves to the root. This length depends logarithmically on the number of leaves in the tree. Hence, very large trees can be managed.

## Disclaimer and acknowledgments

The authors are with the UNESCO Chair in Data Privacy, but they are solely responsible for the views expressed in this paper, which do not necessarily reflect the position of UNESCO nor commit that organization. This work was partly supported by the Spanish Ministry of Science and Innovation through projects TSI2007-65406-C03-01 "E-AEGIS", CONSOLIDER CSD2007-00004 "ARES" and PT-430000-2010-31 "Audit Transparency Voting Process", and by the Government of Catalonia under grant 2009 SGR 1135. The third author is partly supported by the Government of Catalonia as an ICREA-Acadèmia researcher.

## References

[1] C.K. Miller, Multicast Networking and Applications, Addison Wesley, Reading MA, 1999.

[2] B. Quinn, K. Almeroth, IP Multicast Applications: Challenges and Solutions, Internet RFC 3170. <http://www.ietf.org>, 2001.

[3] T. Wolf, S.Y. Choi, Aggregated hierarchical multicast – a many-to-many communication paradigm using programmable networks, IEEE Transactions on Systems, Man and Cybernetics – Part C 33 (3) (2003) 358–369.

[4] F. Sebé, J. Domingo-Ferrer, Scalability and security in biased many-to-one communication, Computer Networks 51 (1) (2007) 1–13.

[5] F. Sebé, A. Viejo, J. Domingo-Ferrer, Secure many-to-one symbol transmission for implementation on smart cards, Computer Networks 51 (9) (2007) 2299–2307.

[6] A. Viejo, F. Sebé, J. Domingo-Ferrer, Secure and scalable many-to-one symbol transmission for sensor networks, Computer Communications 31 (10) (2008) 2408–2413.

[7] B. Przydatek , D. Song, A. Perrig, SIA: Secure information aggregation in sensor networks, in: Proceedings of the 1st International Conference on Embedded Networked Sensor Systems, 2003, pp. 255–265.

[8] T. Dimitriou, Efficient mechanisms for secure inter-node and aggregation processing in sensor networks, in: Proceedings of the 4th International Conference on Ad-Hoc, Mobile, and Wireless Networks, 2005, pp. 18–31.

[9] T. Dimitriou, Securing communication trees in sensor networks, in: Proceedings of the 2nd International Workshop on Algorithmic Aspects of Wireless Sensor Networks, 2006, pp. 47–58.

[10] F. Armknecht, D. Westhoff, J. Girao, A. Hessler, A lifetime-optimized end-to-end encryption scheme for sensor networks allowing in-network processing, Computer Communications 31 (4) (2008) 734–749.

[11] P. Jadia, A. Mathuria, Efficient secure aggregation in sensor networks, in: Proceedings of the 11th International Conference on High Performance Computing, 2004, pp. 40–49.

[12] C. Castelluccia, E. Mykletun, G. Tsudik, Efficient aggregation of encrypted data in wireless sensor networks, in: Proceedings of 2nd IEEE Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services, 2005, pp. 1–9.

[13] D. Westhoff, J. Girao, M. Acharya, Concealed data aggregation for reverse multicast traffic in sensor networks: encryption, key distribution, and routing adaption, IEEE Transactions on Mobile Computing 5 (10) (2006) 1417–1431.

[14] H.-M. Sun, Y.-C. Hsiao, Y.-H. Lin, C.-M. Chen, An efficient and verifiable concealed data aggregation scheme in wireless sensor networks, in: Proceedings of the 2nd International Conference on Embedded Software and Systems, 2008, pp. 19–26.

[15] W. Zhang, C. Wang, T. Feng, GP$^2$S: generic privacy-preservation solutions for approximate aggregation of sensor data, in: Proceedings of the 6th IEEE International Conference on Pervasive Computing, 2008.

[16] C. Castelluccia, C. Soriente, ABBA: a balls and bins approach to secure aggregation in WSNs, in: Proceedings of the 6th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks, 2008.

[17] T. Okamoto, S. Uchiyama, A new public-key cryptosystem as secure as factoring, in: Advances in Cryptology – EUROCRYPT'98, 1998, pp. 308–318.

[18] P. Paillier, Trapdooring discrete logarithms on elliptic curves over rings, in: Advances in Cryptology – ASIACRYPT'00, 2000, pp. 573–584.

[19] A. Sorniotti, L. Gomez, K. Wrona, L. Odorico, Secure and trusted in-network data processing in wireless sensor networks: a survey, Journal of Information Assurance and Security 2 (2007) 189–199.

[20] A.C.-F. Chan, C. Castelluccia, On the (im)possibility of aggregate message authentication codes, in: Proceedings of the IEEE International Symposium on Information Theory, 2008, pp. 235–239.

[21] A. Nicolosi, D. Mazieres, Secure acknowledgment of multicast messages in open peer-to-peer networks, in: Proceedings of the 3rd International Workshop on Peer-to-Peer Systems, 2004.

[22] R.L. Rivest, A. Shamir, L.M. Adleman, A method of obtaining digital signatures and public-key cryptosystems, Communications of the ACM 21 (2) (1978) 120–126.

[23] Advanced Encryption Standard (AES), Federal Information Processing Standards Publication 197, 2001. <http://www.csrc.nist.gov/publications/fips/fips197/fips-197.pdf>