DTU MAT report no. 2009-01

Analysis of the End-by-Hop Protocol for Secure Aggregation in Sensor Networks

Erik Zenner

Technical University of Denmark
Department of Mathematics
e.zenner@mat.dtu.dk

Abstract. In order to save bandwidth and thus battery power, sensor network measurements are sometimes aggregated en-route while being reported back to the querying server. Authentication of the measurements then becomes a challenge if message integrity is important for the application.
At ESAS 2007, the End-by-Hop protocol for securing in-network aggregation for sensor nodes was presented [4]. The solution was claimed to be secure and efficient and to provide the possibility of trading off bandwidth against computation time on the server.
In this paper, we disprove these claims. We describe several attacks against the proposed solution and point out shortcomings in the original complexity analysis. In particular, we show that the proposed solution is inferior to a naive solution *without* in-network aggregation both in security and in efficiency.

1 Introduction

Sensor nodes are resource-restricted processors that are combined with sensors to measure environmental data (e.g. light, noise, temperature, smoke, pressure...) and radio equipment to send and receive data over a radio interface. Sensor networks typically consist of a large number of such nodes, with the individual node being low-cost and thus computationally restricted.

Sensor networks are often controlled by a server who can ask queries and obtain responses that are forwarded through the sensor network. In order to save bandwidth, it often makes sense to aggregate the result on the way. To give an example from a wildfire alarm network, if the server is only interested in the maximum temperature in the controlled area, then it is not necessary to forward all individual measurements. Instead, each node only forwards the maximum of all measurements routed through it, keeping the total message length small.

However, aggregating measurements along the way makes securing the transmission difficult. One important question is whether the aggregated data received by the server actually corresponds to the measurements done by the nodes, or whether the aggregated value was modified along the way. Cryptographic standard techniques are often not directly applicable due to key management issues[1]. Thus, a number of advanced protocols have been proposed to provide message authenticity and integrity [8, 9, 1, 2, 6, 11, 5] for networks with in-network aggregation.

[1] One simple solution is the use a network-wide key, but this means that corrupting a single node is equivalent to corrupting the whole network. Another option would be to let nodes share pairwise different keys with all of their neighbouring nodes, which creates problems in the network setup as well as the amount of non-volatile memory consumed by the keys.

In this paper, we concentrate on the End-by-Hop protocol by Farrell and Jensen [4], presented at ESAS 2007. It was claimed that this solution provides a secure integrity service for networks with in-network aggregation, allowing to trade off bandwidth (and thus battery) consumption with server computation time. In the remainder of this paper, we will evaluate these claims.

Organisation of the Paper: We start by giving a description of the End-by-Hop protocol in Section 2. Subsequently, Section 3 presents three different types of attack against the protocol. Finally, Section 4 discusses the resource consumption of the End-by-Hop protocol, compares it to a naive solution and shows that it does not compare well to other protocols discussed in the literature.

2 The End-by-Hop Protocol

The following is a high-level description of the End-by-Hop protocol [4].

2.1 Network Architecture

The sensor network considered is tree-structured, though the tree is not necessarily binary or balanced. The nodes in the tree are labelled $N_1, \dots, N_n$, with N_1 being the root or sink node. This sink node is the only one that has a connection to the server or **querier** Q which has significantly more computational power than the sensor nodes. The querier asks the network for measurements and processes the results upon receiving them.

Since radio transmission is expensive in terms of battery consumption, network traffic shall be minimised as much as possible. Thus, it is desirable that data is aggregated en-route, i.e., the internal nodes of the tree preprocess the output of lower-level nodes before passing the result on to higher-level nodes. This saves significant bandwidth if the server is only interested in e.g. the minimum, maximum, average, sum etc. of the individual measurements. The authors make the assumption that the individual measurements are from a small set of possible values.

2.2 Security Goals

The purpose of the security solution is to provide a special type of message integrity. The final aggregated output of the sensor network (as received by the server) contains the following information:

1. Which query gave rise to the result (type and time of query)?
2. Which nodes have contributed to the result?
3. What is the result?

The protocol goal is that any illegitimate modification to any of these information will be detected by the server.

Note that the protocol does not protect against the removal of individual nodes or even parts of the network. In many typical application situations for a sensor network, it is possible for an adversary to either physically remove nodes or to jam their radio connection.

In this case, their inputs will be missing from the total result. However, an explicit goal is that the link node can identify the slave nodes which did not contribute.

Another problem is that the adversary might be able to take over individual nodes or parts of the network. In this case, he can not only produce fake measurements for these nodes, but also control the aggregation of results (if he controls an inner node). While we can not prevent the adversary from providing wrong measurements for the hijacked nodes, the protocol should prevent him from modifying measurements from lower-level nodes, or from combining those in a wrong way.

2.3 Protocol Description

In order to achieve the above security goals, the End-by-Hop protcol proceeds as follows (note that our notation deviates from the one used in [4]):

Key distribution: For the whole network, public Diffie-Hellman (DH) parameters (a modulus p and a generator g) are fixed upon setup. The server as well as all nodes have a copy of those parameters. In addition, the server has a fixed DH exponent x (which is only known to itself), and each node N_i has a fixed DH exponent y_i (which is known only to itself and to the server).

Query phase: The server sends a query q which is forwarded through the network to all relevant nodes. This query consists of a request type (e.g., the code for "Send max. temperature") and a nonce. Along with this query, the server also sends his DH value $g^x \bmod p$.[2]

Response phase: Each node N_i that contributes to the result does the following:

1. Derive a shared key $k_{1,i} = \mathrm{KDF}_1(g^{xy_i} \bmod p)$, where KDF_1 is a suitable key derivation function.[3] Use this key to compute a **Query Integrity Field** (QuIF) as follows:
$$\mathrm{QuIF}_i = \left(q^{k_{1,i}} \cdot \prod_{j \in C_i} \mathrm{QuIF}_j \right) \bmod p,$$
where C_i is the set of N_i's children (possibly empty, in which case $\mathrm{QuIF}_i = q^{k_{1,i}}$).
2. Derive a shared key $k_{2,i} = \mathrm{KDF}_2(g^{xy_i} \bmod p)$, where KDF_2 is a suitable key derivation function. Use this key to compute a **Response Integrity Field** (called MEASMAC) as follows:
$$\mathrm{MEASMAC}_i = \mathrm{HMAC}_{k_{2,i}}(d_i) \oplus \bigoplus_{j \in C_i} \mathrm{MEASMAC}_j,$$

[2] The assumption seems to be that the nodes don't have the non-volatile memory to permanently store this value.

[3] Note that KDF details in [4] are rather fishy. When computing g^x and g^{xy_i}, the authors omit the reduction modulo p in their notation. While this is sometimes done for simplicity, it is understood that the operation is nonetheless present. Thus, we assume that the result of g^{xy_i} is in the range $\{1, \ldots, p-1\}$, being non-uniformly distributed [12]. Now the authors apply two key derivation functions which are not specified (and which we compose into KDF_1 for simplicity), stating that the final result is uniformly distributed over $\{1, \ldots, p-1\}$. This, however, is obviously not possible, since KDF_1 must be a permutation and can thus not change the probability distribution. The problem could be solved, though, if the output of KDF_1 was in a group of order significantly less than p.

where d_i is the actual data provided by N_i and C_i is the set of its children (possibly empty, in which case $\mathrm{MEASMAC}_i = \mathrm{HMAC}_{k_{2,i}}(d_i)$).

3. Send QuIF_i and $\mathrm{MEASMAC}_i$ on to the parent node. In addition, attach the aggregated result for the subtree for which N_i is a root. Note that d_i itself is not sent.

Response Evaluation: In order to verify the response, the server proceeds as follows:

1. Given the value QuIF_1, the server tries (using brute force) to find a subset S of all nodes such that the integer
$$s = \left(\sum_{N_i \in S} k_{1,i} \right) \bmod (p-1)$$
fulfills the equality
$$\mathrm{QuIF}_1 = q^s \bmod p.$$
If the server finds such a set S, then it assumes that the authentication was sent by the nodes in S. Otherwise, it assumes that something went wrong.
2. Given the value $\mathrm{MEASMAC}_1$, the server tries (using brute force) to find a combination of inputs d_i (for $N_i \in S$) such that
$$\mathrm{MEASMAC}_1 = \bigoplus_{N_i \in S} \mathrm{HMAC}_{k_{2,i}}(d_i).$$
If the server can find such a combination, then it assumes that the message d_i was sent by the node N_i. Otherwise, it assumes that something went wrong.
3. Finally, given all contributing messages d_i, the server checks whether they indeed produce the aggregate result received.

3 Security Analysis

From a security perspective, there are a number of problems with this protocol. It is overly complicated for its purpose, uses cryptographic primitives in an unfamiliar fashion, and gives no real indication for why it should be secure. In the following, we give some examples of attacks that become possible due to these shortcomings.

3.1 Deletion Attack

It is easy for an adversary to remove a node's contribution from the reply. If he can observe the input and output for a node N_i, then he can easily compute the values $q^{k_{1,i}} \bmod p$ and $\mathrm{HMAC}_{k_{2,i}}(d_i)$. He can then use this knowledge to later remove N_i's contribution at any point in the network tree by dividing / xoring it out. Note that this attack requires next to no computational resources and can thus be executed on a very small computational device (e.g., a rogue node).

One might argue that the adversary could achieve the same result by simply destroying the node N_i, or by jamming the signal sent by this node. However, such physical attacks require the ability to actively manipulate precisely the node whose signal the adversary wants to suppress. The cryptographic attack described above, on the other hand, requires only the ability to observe input and output for node N_i, and to modify the signal of any one node in the network. Timing of signals plays a crucial role for this type of attack, since the adversary has to transmit input and output of the target node to the corrupted node before the end of the response time. Note that the corrupted node can be anywhere in the network - even a leaf node in another (slower) branch of the tree can introduce the necessary changes. Thus, the practical applicability of this attack depends on the network's response delay: If some nodes are allowed to respond later than others, then the deletion attack becomes possible.

3.2 Modifying the Query

The security goal for the QuIF value is to allow the server to verify that his query was unaltered on the way to the nodes, i.e. that the nodes are answering the question that was actually asked. As it turns out, this security goal is not met, since a discrete power $q^k \bmod p$ is not a secure authentication of a message q.[4] A simple standard attack against such an "authentication scheme" is to query two authenticators $a_1 = (q_1)^k$ and $a_2 = (q_2)^k$ and to construct a third one by computing $a_3 = a_1 \cdot a_2 = (q_1 \cdot q_2)^k$, which is the correct authenticator for the message $q_3 = q_1 \cdot q_2$.

Details of such an attack depend on the protocol specification, but it is easy to give an example. Consider an implementation of the End-by-Hop protocol with the following specification:

- Any l-bit value is a valid nonce.
- The code for "Give avg. temperature" is 0.
  The code for "Give min. temperature" is 1.
  The code for "Give max. temperature" is 2.
- A query consists of a concatenation of the query code c and the nonce N. For the purpose of computing the authenticator, such a query is interpreted as an integer $q = c \cdot 2^l + N$.

The adversary can now proceed as follows:

1. As preparation, he sends a fake query, using query code 0 and nonce 2. This means that the integer representation of the query is 2, and the node will answer by sending an authenticator 2^k.
2. When the server sends a query with code 2 (maximum temperature), the corresponding integer is $2 \cdot 2^l + N$ (for some arbitrary nonce N). Now the adversary divides this query by two before forwarding it to the node, thus sending $1 \cdot 2^l + N/2$ to the node.[5] The node believes that the minimum temperature was requested and answers accordingly.
3. The node also sends the authenticator $(1 \cdot 2^l + N/2)^k$. The adversary multiplies this with the value 2^k, which he knows from step 1, obtaining $(2 \cdot 2^l + N)^k$. This forged authenticator is sent to the server who identifies it as a correct answer to its query.

As a result, the server believes that the node sent a response to a "maximum temperature" query, even though the node responded to a "minimum temperature" query.

[4] In this subsection, we write k instead of $k_{1,i}$ in order to keep notation simple. We also assume that all arithmetics is modulo p, without explictly saying so every time.

[5] For simplicity, we assume that N was even, which happens for every second query.

The above example can be prevented by ad-hoc countermeasures such as padding. However, this does not solve the underlying problem of the QuIF solution: A value of the form $q^k \bmod p$ is not a cryptographically secure authenticator. It protects the key k, but does not give any guarantees considering the message q. Thus, in the End-by-Hop protocol, the adversary can trick the nodes into answering the wrong kind of query, without this being noticed by the server.

3.3 Forging the Aggregate Result

As opposed to most wide-spread solutions in cryptography, the values QuIF and MEASMAC mix both message and integrity check into one authenticator[6]. Thus, their security level is in fact less than the authenticator length indicates.

As an example, consider MEASMAC. Denote the output size of HMAC by b (in bits) and the number of contributing nodes by ν. In addition, we denote the number of possible messages by d and the number of possible aggregation results by d'.

Now the adversary picks the aggregate result of his choice and then simply guesses a final authenticator MEASMAC$_1$. The server verifies this authenticator as follows:

1. He first tries to find a combination of measurements that produce the correct MAC. Since there are d^ν such combinations, he succeeds with probability $\approx d^\nu/2^b$.
2. He then checks whether the supposed measurements from step 1 lead to the aggregate result that he received. This happens in 1 out of d' cases if all d' results are equally likely, and with even higher probability otherwise.

Summing up, the adversary's success probability for a simple guessing attack is $\approx d^\nu/(2^b \cdot d')$. If $d' \approx d$ (as is the case for many functions like e.g. SUM, AVG, MIN, MAX etc.), then we have a success probability of $d^{\nu-1}/2^b$, as opposed to $1/2^b$ for a standard MAC.

Put another way, the security provided by the MAC has been reduced by $(\nu-1)\cdot\log_2(d)$ bit. Even for extremely small values for d and ν, this is a significant loss. Using the toy example from [4], where $d = 8$ and $\nu = 5$, the MAC strength is reduced by $4 \cdot \log_2(8) = 12$ bit. For real-world examples with $\nu \geq 100$ and $d \geq 20$, a standard-sized MAC (e.g. $b = 160$ for HMAC-SHA-1) would provide no security at all.

In general, mixing the message and the authenticator into one value is not recommendable[7]. It provides neither extra security nor extra bandwidth, but makes analysis of both parts more difficult. Thus, we recommend to clearly distinguish between a message part and an authenticator part. The resulting solution may be less fancy, but it allows us to build upon existing cryptographic techniques and to give thorough analysis and security proofs.

4 Resource Consumption

An obvious question is how the End-by-Hop protocol can be repaired to become secure. However, even if the protocol was secure, it would be too inefficient for its purpose, meaning that other solutions proposed in the literature provide a better starting point for improved protocols. In the following, we will illustrate our point by comparing the End-by-Hop protocol to a naive solution.

[6] This is related to digital signatures with message recovery, a technique that is hardly used in practice.

[7] This also holds for the QuIF value, although analysis is more complicated there.

4.1 A Naive Solution

Consider the following naive solution *without* in-network aggregation. Assume that each node N_i shares a secret key k_i with the server. Then the protocol has the following phases:

Query Phase: The server sends a query, consisting of a query type code c and a nonce N.

Response Phase: In the message field, the nodes concatenate their measurements into a long string, without aggregation. In addition, each node contributes an indicator that is has contributed – either by appending its node ID to the message or by setting a flag in a bit array. For integrity, each node computes a contribution $a_i = \mathrm{MAC}_{k_i}(c||N||d_i)$. All of these a_i are xored to obtain the final authenticator. For a formal security treatment of this scheme, see Katz and Lindell [10]. For our purposes, it suffices to know that this MAC is secure and does not suffer from the shortcomings of the End-by-Hop protocol.

Response Evaluation: Given the individual measurements and the list of contributing nodes, the server computes the MAC and compares it to the one received. If they match, he accepts the measurements, otherwise he rejects.

4.2 Bandwidth Consumption

Battery consumption is a major bottleneck for a sensor network, and the radio transmitter drains the battery faster than the processor. Thus, as little data as possible should be sent over the network. In the following, we analyse the required bandwidth of both protocols.

To simplify analysis, we assume in the following that all n nodes contribute to the aggregated result, and that all measurements occur with equal probability. We point out, though, that our observations also hold for a smaller number of nodes and for more complicated measurement distributions, even though the resulting formulas would be more complicated.

End-by-Hop Protocol: We start by making two observations:

1. The QuIF part is very expensive in terms of bandwidth. The most recent ECRYPT report recommends a key length of at least 1024 bit [7] for DH-based algorithms, meaning that the length of QuIF alone would be in the order of 128 byte[8]. This seems to be wasteful since the Diffie-Hellman key is in fact used here as a symmetric key. Note that the value $g^{y_i} \bmod p$ is neither used as a public key (i.e. known to every node) nor as a private key (i.e. known only to node N_i), but as a shared key between N_i and S. Thus, the QuIF part could be implemented using a standard symmetric solution similar to the one used for MEASMAC, significantly reducing the bandwidth consumption without reducing the security.
2. The End-by-Hop protocols sends sufficient information to reconstruct the full list of contributing nodes and the full list of measurements. We know from information theory that there is no way of doing this more efficiently than by sending a perfect compression of these values, meaning that the QuIF and MEASMAC fields have to be at least as long as the entropy of the contributors list and measurements list, respectively.

[8] Elliptic-curve based variants could reduce this length to 20-40 byte, but the way the protocol is described in [4] indicates that the idea was to use a multiplicative group modulo a prime.

Looking at the concrete bandwidth consumption for the End-by-Hop solution, we see that at least the following information needs to be sent:

- In the QuIF part, sufficient information to reconstruct all contributing nodes, plus security information. Since it must be possible to reconstruct the subset of contributing nodes from the QuIF part, its length has to be at least n bit. In addition, it has to provide an additional number of bits for security, which we denote by s_1.
- In the MEASMAC part, sufficient information to reconstruct all contributing measurements, plus security information. Since all n nodes could send any of the d measurements, we need $n \cdot \log(d)$ bit for the MEASMAC part. If we also want the forgery probability to be at most 2^{-s_2}, then the MEASMAC length goes up to at least $n \cdot \log(d) + s_2$ bit.
- In the message part, the aggregated result. Thus, here we have to spent another $\log(d')$ bit, where d' is the length of the aggregation.

Thus, the total length of a End-by-Hop response is $n + n \cdot \log(d) + \log(d') + s_1 + s_2$. This is much larger than assumed in the original paper, which implicitly states a bandwidth consumption of only $\log(d') + s_1 + s_2$ bit.

Naive Protocol: Now consider the bandwidth consumption of the naive solution. It has to send the concatenated measurements ($n \cdot \log(d)$ bit) and the message IDs (n bit), plus a MAC of length s_2 bits. Thus, the total bandwidth consumption is $n + n \cdot \log(d) + s_2$, doing away with the aggregated result and the expensive Diffie-Hellman parameter. We see that even though the naive solution provides a higher security level, it consumes much less bandwidth.

4.3 Other Resources

In the remainder of this section, we discuss the use of other resources by the two protocols.

Computation Time (Nodes): For the resource-restricted nodes, computations cost both time and battery power and should thus be kept to a minimum. By using Diffie-Hellman multiplication, the End-by-Hop protocol places a heavy computational workload on the nodes. As an example, the benchmarks for the Crypto++ v5.5 library [3] state that a DH key agreement for 1024-bit keys takes about 2.1 million cycles on a Pentium 4 CPU, i.e. a 64-bit processor. For a light-weight processor as used for sensor node purposes, such as the Mica mote (8-bit processor at 4 MHz), one such computation can be expected to take between several seconds and more than a minute and thus to be unrealistic. But even for larger nodes, using significant computation time just for the security part is typically not in the best interest of the system designer.

The naive protocol does not have this expensive DH multiplication. Here, the main workload is generated by computing the MAC which maps an input of $n + n \cdot \log(d)$ bits to an output of s_2 bits. Already for moderate sizes of n and d, this is more efficient than the MEASMAC, which has to map $n \cdot \log(d)$ bits onto $n \cdot \log(d) + s_2$ bits. Thus, the naive protocol clearly uses significantly less computation time than the End-by-Hop protocol.

Memory (Nodes): Memory (both volatile and non-volatile) is also an important limiting factor. For the End-by-Hop protocol, the Diffie-Hellman key dominate the memory consumption. Note that the node has to store the generator g, the prime p, and the secret exponent y_i, all of which have a length of about 128 byte. Thus, in order to store the keys alone, 384 byte of memory are consumed.

On the other hand, the naive solution only needs to store a symmetric key that is shared with the server. This key would typically be between 10 and 16 byte long, offering significant savings compared to the End-by-Hop protocol.

Computation Time (Server): Normally, the computational resources of the server are not critical when communicating with light-weight nodes. However, the brute-force operations used by the End-by-Hop protocol in order to de-aggregate the list of contributors and the list of measurements change this picture. Here, the server has to run up to 2^n brute-force steps to reconstruct the list of participants, and d^ν brute-force steps to reconstruct the measurements (where $\nu \leq n$ is the number of contributing nodes).

Again, the naive solution does not need a brute-force search, i.e. the computation time of the server is dominated by computing the MAC, which corresponds to one single brute-force step in the End-by-Hop protocol.

4.4 Comparison to State of the Art

We have shown now that the naive protocol is superior to the End-by-Hop protocol with respect to both security and efficiency. On the other hand, several protocols have been proposed in the literature that are superior to the naive solution in most (if not all) aspects [8, 9, 1, 2, 6, 11, 5]. Thus, by transitivity, we can conclude that they also are superior to the End-by-Hop protocol.

5 Conclusions

We have analysed the End-by-Hop protocol from ESAS 2007 [4], showing that it does not compare well with the state of the art in secure in-network aggregation. As it turns out, the protocol achieves neither its security goals nor the claimed resource savings. It is difficult to see how the protocol could be repaired to be more secure and efficient than competing solutions currently discussed in the literature.

References

1. E.-O. Blass, O. Wilke, M. Zitterbart. “A Security-Energy Trade-Off for Authentic Aggregation in Sensor Networks”. *Proc. 2nd IEEE Workshop on Wireless Mesh Networks (WiMesh 2006).* pp 135–137, IEEE, 2006.
2. H. Chan, A. Perrig, D. Song. “Secure Hierarchical In-Network Aggregation for Sensor Networks. *Proc. 13th ACM Conference on Computer and Communications Security.* pp. 278-287, ACM, 2006.
3. W. Dai. “Crypto++ 5.5 Benchmarks”. Version from May 5, 2007. Available from: http://www.cryptopp.com/benchmarks.html.
4. S. Farrell, C. Jensen. “End-by-Hop Data Integrity”. *Proc. 4th European Workshop on Security and Privacy in Ad hoc and Sensor Networks (ESAS 2007)*, Springer LNCS 4572, pp. 142–155, 2007.

5. K. Frikken, J. Dougherty. “An Efficient Integrity-Preserving Scheme for Hierarchical Sensor Aggregation”. *Proc. First ACM Conference on Wireless Network Security (WiSec 2008).* pp. 68–76, ACM, 2008.
6. M. Garofalakis, J. Hellerstein, P. Maniatis. “Proof Sketches: Verifiable In-Network Aggregation”. *Proc. 23rd IEEE Int. Conference on Data Engineering (ICDE 2007).* pp. 996–1005, IEEE, 2007.
7. C. Gehrmann, M. Näslund. “ECRYPT Yearly Report on Algorithms and Keysizes (2006)”. Revision 1.1, January 2007. Available from:
   http://www.ecrypt.eu.org/documents/D.SPA.21-1.1.pdf.
8. L. Hu, D. Evans. “Secure Aggregation for Wireless Networks”. *Proc. 2003 Symposium on Applications and the Internet Workshops (SAINT'03)*, pp. 384–394, IEEE Computer Society, 2003.
9. P. Jadia, A. Mathuria. “Efficient Secure Aggregation in Sensor Networks”. *Proc. 11th Int. Conference on High Performance Computing (HiPC 2004)*, Springer LNCS 3296, pp. 40–49, 2004.
10. J. Katz, Y. Lindell. “Aggregate Message Authentication Codes”. *Topics in Cryptology – CT-RSA 2008*, Springer LNCS 4964, pp. 155–169, 2008.
11. M. Manulis, J. Schwenk. “Provably Secure Framework for Information Aggregation in Sensor Networks”. *Proc. Int. Conference on Computational Science and its Applications (ICCSA 2007).* Springer LNCS 4705, Part I, pp. 603–621, 2007.
12. C. Waldvogel, J. Massey. “The Probability Distribution of the Diffie-Hellman Key”. *Proc. Asiacrypt '92*, Springer LNCS 718, pp. 492–504, 1992.