Public Key-Based Lightweight Swarm Authentication

Simon Cogliani, Bao Feng, Houda Ferradi, Rémi Géraud, Diana Maimuţ, David Naccache, Rodrigo Portella do Canto, and Guilin Wang

Abstract We describe a lightweight algorithm performing whole-network authentication in a distributed way. This protocol is more efficient than one-to-one node authentication: it results in less communication, less computation and overall lower energy consumption. The proposed algorithm is provably secure and achieves zero-knowledge authentication of a network in a time logarithmic in the number of nodes.

1 Introduction

A growing market focuses on lightweight devices, whose low cost and easy production allow for creative and pervasive uses. The Internet of Things (IoT) consists in spatially distributed nodes that form a network, able to control or monitor physical or environmental conditions (such as temperature, pressure, image and sound), perform computations or store data. IoT nodes are typically low-cost devices with limited computational resources and limited battery. They transmit the data they acquire through the network to a gateway, also called the transceiver, which collects information and sends it to a processing unit. Nodes are usually deployed in hostile environments and are therefore susceptible to physical attacks, harsh weather conditions and communication interferences.

S. Cogliani (✉) · H. Ferradi · R. Géraud · D. Naccache · R. Portella do Canto
École normale supérieure, Paris, France
e-mail: simon.cogliani@ens.fr; houda.ferradi@ens.fr; remi.geraud@ens.fr; david.naccache@ens.fr; rodrigoportella.docanto@ens.fr

B. Feng · G. Wang
Huawei Technologies Co. Ltd., Shenzhen, China
e-mail: bao.feng@huawei.com; guilin.wang@huawei.com

D. Maimuţ
École normale supérieure, Paris, France

Advanced Technologies Institute, Bucharest, Romania
e-mail: diana.maimut@ens.fr; ati@dcti.ro

Ç. K. Koç (ed.), *Cyber-Physical Systems Security*,
https://doi.org/10.1007/978-3-319-98935-8_12

Due to the open and distributed nature of the IoT, security is key to the entire network's proper operation [14]. However, the lightweight nature of sensor nodes heavily restricts the type of cryptographic operations that they can perform, and the constrained power resources make any communication costly.

This chapter describes an authentication protocol that establishes network integrity and leverages the distributed nature of computing nodes to alleviate individual computational effort. This enables the base station to identify which nodes need replacement or attention.

This is most useful in the context of wireless sensor networks and the IoT, but applies equally well to mesh network authentication and similar situations.

1.1 Related Work

Zero-knowledge (ZK) protocols have been considered for authentication of wireless sensor networks. For instance, Anshul and Roy [1] describe a modified version of the Guillou–Quisquater identification scheme [8], combined with the μTesla protocol [11] for authentication broadcast in constrained environments. We stress that the purpose of the scheme of [1], and similar ones, is to authenticate the base station.

Aggregate signature schemes such as [2, 15] may be used to achieve the goal pursued here—however they are intrinsically noninteractive—and the most efficient aggregate constructions use elliptic curve pairings, which require powerful devices.

Closer to our concerns, [13] describes a ZK network authentication protocol, but it only authenticates two nodes at a time, and the base station acts like a trusted third party. As such it takes a very large number of interactions to authenticate the network as a whole.

What we propose instead is a collective perspective on authentication and not an isolated one.

1.2 Structure of This Chapter

Section 2 recalls the Fiat–Shamir authentication scheme and presents a distributed algorithm for topology-aware networks. We describe the core idea of our chapter, a distributed Fiat–Shamir protocol for IoT authentication, in Sect. 3. We analyse the security of the proposed protocol in Sect. 4. Section 5 provides several improvements and explores trade-offs between security, transmission and storage.

2 Preliminaries

2.1 Fiat–Shamir Authentication

The Fiat–Shamir authentication protocol [5] enables a prover $\mathscr{P}$ to convince a verifier $\mathscr{V}$ that $\mathscr{P}$ possesses a secret key without ever revealing the secret key [4, 7].

The algorithm first runs a one-time setup, whereby a trusted authority publishes an RSA modulus $n = pq$ but keeps the factors p and q private. The prover $\mathscr{P}$ selects a secret $s < n$ such that $\gcd(n, s) = 1$ computes $v = s^2 \bmod n$ and publishes v as its public key.

When a verifier $\mathscr{V}$ wishes to identify $\mathscr{P}$, he uses the protocol of Protocol 1. $\mathscr{V}$ may run this protocol several times until $\mathscr{V}$ is convinced that $\mathscr{P}$ indeed knows the square root s of v modulo n.

Protocol 1: Fiat–Shamir authentication

$$
\begin{array}{lcl}
\textbf{Prover} & & \textbf{Verifier} \\
r \overset{\$}{\leftarrow} [1, n-1] & & \\
x \leftarrow r^2 \bmod n & & \\
 & \xrightarrow{x} & \\
 & & \{e_1, \dots, e_k\} \overset{\$}{\leftarrow} \{0,1\}^k \\
 & \xleftarrow{e_1,\dots,e_k} & \\
y \leftarrow r \prod_{i=1}^{k} s_i^{e_i} \bmod n & & \\
 & \xrightarrow{y} & \\
 & & \text{Check } y^2 = x \prod_{i=1}^{k} v_i^{e_i} \bmod n
\end{array}
$$

Protocol 1 describes the original Fiat–Shamir authentication protocol [5], which is *honest verifier* zero knowledge[1] and whose security is proven assuming the hardness of computing arbitrary square roots modulo a composite n, which is equivalent to factoring n.

As pointed out by [5], instead of sending x, $\mathscr{P}$ can hash it and send the first bits of $H(x)$ to $\mathscr{V}$, for instance, the first 128 bits. With that variant, the last step of the protocol is replaced by the computation of $H(y^2 \prod_{i=1}^{k} v_i^{a_i} \bmod n)$, truncated to the first 128 bits and compared to the value sent by $\mathscr{P}$. Using this "short commitment" version reduces somewhat the number of communicated bits. However, it comes at the expense of a reduced security level. A refined analysis of this technique is given in [6].

[1] This can be fixed by requiring $\mathscr{V}$ to commit on the a_i before $\mathscr{P}$ has sent anything, but this modification will not be necessary for our purpose.

2.2 Topology-Aware Distributed Spanning Trees

Due to the unreliable nature of sensors, their small size and wireless communication system, the overall network topology is subject to change. Since sensors send data through the network, a sudden disruption of the usual route may result in the whole network shutting down.

2.2.1 Topology-Aware Networks

A *topology-aware* network detects changes in the connectivity of neighbours, so that each node has an accurate description of its position within the network. This information is used to determine a good route for sending sensor data to the base station. This could be implemented in many ways, for instance, by sending discovery messages (to detect additions) and detecting unacknowledged packets (for deletions). Note that the precise implementation strategy does not impact the algorithm.

Given any graph $G = (V, E)$ with a distinguished vertex B (the base station), the optimal route for any vertex v is the shortest path from v to B on the minimum degree spanning tree $S = (V, E')$ of G. Unfortunately, the problem of finding such a spanning tree is **NP**-hard [12], even though there exist optimal approximation algorithms [9, 12]. Any spanning tree would work for the proposed algorithm; however the performance of the algorithm gets better as the spanning tree degree gets smaller.

2.2.2 Mooij–Goga–Wesselink's Algorithm

The network's topology is described by a spanning tree W constructed in a distributed fashion by the Mooij–Goga–Wesselink algorithm [10]. We assume that nodes can locally detect whether a neighbour has appeared or disappeared, i.e. graph edge deletion and additions.

W is constructed by aggregating smaller subtrees together. Each node in W is attributed a "parent" node, which already belongs to a subtree. The complete tree structure of W is characterized by the parenthood relationship, which the Mooij–Goga–Wesselink algorithm computes. Finally, by topological reordering, the base station $\mathscr{T}$ can be put as the root of W.

Each node in W has three local variables $\{\mathsf{parent}, \mathsf{root}, \mathsf{dist}\}$ that are initially set to a null value $\perp$. Nodes construct distributively a spanning tree by exchanging "M-messages" containing a root information, distance information and a type. The algorithm has two parts:

- *Basic*: maintains a spanning tree as long as no edge is removed (it is a variant of the union-find algorithm [3]). When a new neighbour w is detected, a discovery M-message $(\mathsf{root}, \mathsf{dist})$ is sent to it. If no topology change is detected for w, and an M-message is received from it, it is processed by Algorithm 1. Note that a node only becomes active upon an event such as the arriving of an M-message or a topology change.

- *Removal*: intervenes after the edge deletion so that the basic algorithm can be run again and give correct results.

Algorithm 1: Mooij–Goga–Wesselink algorithm (Basic part)

Receive: An M-message (r, d) coming from a neighbour w.

```
(parent, root, dist) ← $(\perp, \perp, \perp)$
if $(r, d + 1) < (\text{root}, \text{dist})$
    parent ← $w$
    root ← $r$
    dist ← $d + 1$
    send the $M$-message (root, dist) to all neighbours except $w$
```

Algorithm 1 has converged once all topology change events have been processed. At that point we have a spanning tree [10].

For our purposes, we may assume that the network was set up and that Algorithm 1 is running on it, so that at all times the nodes of the network have access to their parent node. Note that this incurs very little overhead as long as topology changes are rare.

3 Distributed Fiat–Shamir Authentication

3.1 The Approach

Given a k-node network $\mathscr{N}_1, \ldots, \mathscr{N}_k$, we may consider the nodes $\mathscr{N}_i$ as users and the base station as a trusted centre $\mathscr{T}$. In this context, each node will be given only an[2] s_i. To achieve collective authentication, we propose the following Fiat–Shamir-based algorithm:

- *Step 0:* Wait until the network topology has converged and a spanning tree W is constructed with Algorithm 1 presented in Sect. 2.2. When that happens, $\mathscr{T}$ sends an authentication request message (AR-message) to all the $\mathscr{N}_i$ directly connected to it. The AR-message may contain a commitment to e (*cf.* Step 2) to guarantee the protocol's zero-knowledge property even against dishonest verifiers.

[2] This is for clarity. It is straightforward to give each node several private keys and adapt the algorithm accordingly.

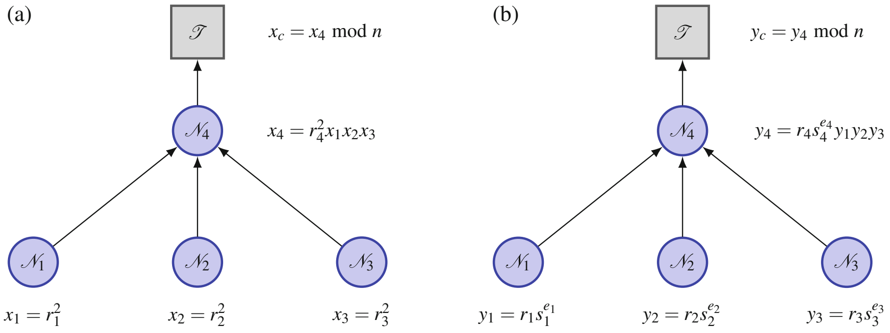

Fig. 1 The proposed algorithm running on a network: computation of x_c (left) and of y_c (right). Each parent node aggregates the values computed by its children and adds its own information before transmitting the result upwards to the base station

- *Step 1:* Upon receiving an AR-message, each $\mathcal{N}_i$ generates a private r_i and computes $x_i \leftarrow r_i^2 \bmod n$. $\mathcal{N}_i$ then sends an A-message to all its children, if any. When they respond, $\mathcal{N}_i$ multiplies all the x_j sent by its children together, and with its own x_i, and sends the result up to its own parent. This recursive construction enables the network to compute the product of all the x_is and send the result x_c to the top of the tree in d steps (where $d = \deg W$). This is illustrated for a simple network including four nodes and a base station in Fig. 1.
- *Step 2:* $\mathcal{T}$ sends a random e as an authentication challenge (AC-message) to the $\mathcal{N}_i$ directly connected to it.
- *Step 3:* Upon receiving an AC-message e, each $\mathcal{N}_i$ computes $y_i \leftarrow r_i s_i^{e_i}$. $\mathcal{N}_i$ then sends the AC-message to all its children, if any. When they respond, $\mathcal{N}_i$ multiplies the y_j values received from all its children together, and with its own y_i, and sends the result to its own parent. The network therefore computes collectively the product of all the y_i's and transmits the result y_c to $\mathcal{T}$. This is illustrated in Fig. 1.
- *Step 4:* Upon receiving y_c, $\mathcal{T}$ checks that $y_c^2 = x_c \prod v_i^{e_i}$, where $v_1, \ldots, v_k$ are the public keys corresponding to $s_1, \ldots, s_k$, respectively.

Note that the protocol may be interrupted at any step. In the version of the algorithm that we have just described, this results in a failed authentication.

3.2 Backup Authentication

Network authentication may fail for many reasons described and analysed in detail in Sect. 4.3.3. As a consequence of the algorithm's distributed nature that we have just described, a single defective node suffices for authentication to fail.

This is the intended behaviour; however there are contexts in which such a brutal answer is not enough, and more information is needed. For instance, one could wish to know *which* node is responsible for the authentication failure.

A simple backup strategy consists in performing *usual* Fiat–Shamir authentication with all the nodes that still respond, to try and identify where the problem lies. Note that, as long as the network is healthy, using our distributed algorithm instead is more efficient and consumes less bandwidth and less energy.

Since all nodes already embark the hardware and software required for Fiat–Shamir computations, and can use the same keys, there is no real additional burden in implementing this solution.

4 Security Proofs

In this section we wish to discuss the security properties relevant to our construction. The first and foremost fact is that algorithm given in Sect. 3 is *correct*: a legitimate network will always succeed in proving its authenticity, provided that packets are correctly transmitted to the base station $\mathscr{T}$ (possibly hopping from node to node) and that nodes perform correct computations.

The interesting part, therefore, is to understand what happens when such hypotheses do *not* hold.

4.1 Soundness

Lemma 1 (Soundness) *If the authentication protocol succeeds, then with overwhelming probability the network nodes are genuine.*

Proof Assume that an adversary $\mathscr{A}$ simulates the whole network, but does not know the s_i, and cannot compute in polynomial time the square roots of the public keys v_i. Then, as for the original Fiat–Shamir protocol [5], the base station will accept $\mathscr{A}$'s identification with probability bounded by 2^k where k is the number of nodes.

4.2 Zero Knowledge

Lemma 2 (Zero Knowledge) *The distributed authentication protocol of Sect. 3.1 achieves statistical zero knowledge.*

Proof Let $\mathscr{P}$ be a prover and $\mathscr{A}$ be a (possibly cheating) verifier, who can use any adaptive strategy and bias the choice of the challenges to try and obtain information about the secret keys.

Consider the following simulator $\mathscr{S}$:

Step 1. Choose $\overline{e} \overset{\$}{\leftarrow} \{0, 1\}^k$ and $\overline{y} \overset{\$}{\leftarrow} [0, n-1]$ using any random tape ω'.
Step 2. Compute $\overline{x} \leftarrow \overline{y}^2 \prod v_i^{\overline{e}_i}$ and output $(\overline{x}, \overline{e}, \overline{y})$.

The simulator $\mathscr{S}$ runs in polynomial time and outputs triples that are indistinguishable from the output of a prover that knows the corresponding private key.

If we assume the protocol is run N times, and that $\mathscr{A}$ has learnt information which we denote η, then $\mathscr{A}$ chooses adaptively a challenge using all information available to it $e(x, \eta, \omega)$ (where ω is a random tape). The proof still holds if we modify $\mathscr{S}$ in the following way:

Step 1. Choose $\overline{e} \overset{\$}{\leftarrow} \{0, 1\}^k$ and $\overline{y} \overset{\$}{\leftarrow} [0, n-1]$ using any random tape ω'.
Step 2. Compute $\overline{x} \leftarrow \overline{y}^2 \prod v_i^{\overline{e}_i}$.
Step 3. If $e(\overline{x}, \eta, \omega) = \overline{e}$, then go to Step 1; else output $(\overline{x}, \overline{e}, \overline{y})$.

Note that the protocol is also "locally" ZK, in the sense that an adversary simulating ℓ out of k nodes of the network still has to face the original Fiat–Shamir protocol.

4.3 Security Analysis

4.3.1 Choice of Parameters

Let λ be a security parameter. To ensure this security level, the following constraints should be enforced on parameters:

- The identification protocol should be run $t \geq \lceil \lambda / k \rceil$ times (according to Lemma 1), which is reasonably close to one as soon as the network is large enough.
- The modulus n should take more than $2^{\lambda t}$ operations to factor.
- Private and public keys are of size comparable to n.

4.3.2 Algorithmic Complexity

The number of operations required to authenticate the network depends on the exact topology at hand, but can safely be bounded above:

- Number of modular squarings: $2kt$
- Number of modular multiplications $\leq 3kt$

In average, each $\mathscr{N}_i$ performs only a constant (a small) number of operations. Finally, only $O(d)$ messages are sent, where d is the degree of the minimum spanning tree of the network. Pathological cases aside, $d = O(\log k)$, so that only a logarithmic number of messages are sent during authentication.

All in all, for $\lambda = 256$, $k = 1024$ nodes and $t = 1$, we have $n \geq 2^{1024}$ and up to 5 modular operations per node.

4.3.3 Root Causes of Authentication Failure

Authentication may fail for several reasons. This may be caused by network disruption, so that no response is received from the network—at which point not much can be done.

However, more interestingly, $\mathscr{T}$ may have received an invalid value of y_c. The possible causes are easy to spot:

1. A topology change occurred during the protocol:
    - If all the nodes are still active and responding, the topology will eventually converge and the algorithm will get back to Step 0.
    - If, however, the topology change is due to nodes being added or removed, the network's integrity has been altered.
2. A message was not transmitted: this is equivalent to a change in topology.
3. A node sent a wrong result. This may stem from low battery failure or when errors appear within the algorithm the node has to perform (fault injection, malfunctioning, etc). In that case authentication is expected to fail.

4.3.4 Effect of Network Noise

Individual nodes may occasionally receive incorrect (ill-formed, or well-formed but containing wrong information) messages, be it during topology reconstruction (M-messages) or distributed authentication (A-messages). Upon receiving incorrect A- or M-messages, nodes may dismiss them or try and acknowledge them, which may result in a temporary failure to authenticate. An important parameter which has to be taken into account in such an authentication context is the number of children of a node. When a node with many children starts failing, all its children are disconnected from the network and cannot be contacted or authenticated anymore. While a dysfunction at the leaf level might be benign, the failure of a fertile node is catastrophic.

4.3.5 Man-in-the-Middle Attacks

An adversary could instal itself between nodes, or between nodes and the base station, and try to intercept or modify communications. Lemma 2 proves that a passive adversary cannot learn anything valuable, and Lemma 1 shows that an active adversary cannot fool authentication.

It is still possible that the adversary *relays* information, but any attempt to intercept or send messages over the network would be detected.

5 Variants and Implementation Trade-Offs

The protocol may be adapted to better fit operational constraints: in the context of IoT, for instance, communication is a very costly operation. We describe variants that aim at reducing the amount of information sent by individual nodes while maintaining security.

5.1 *Shorter Challenge Variant*

In the protocol, the *long* (say, 128-bit) challenge e is sent throughout the network to all individual nodes. One way to reduce the length of e without compromising security is the following:

- A *short* (say, 80-bit) value e is sent to the nodes.
- Each node i computes $e_i \leftarrow H(e\|i)$ and uses e_i as a challenge.
- The base station also computes e_i the same way and uses this challenge to check authentication.

This variant does not impact security, assuming an ideal hash function H, and it can be used in conjunction with the other improvements described below.

5.2 *Multiple-Secret Variant*

Instead of keeping one secret value s_i, each node could have multiple-secret values $s_{i,1}, \ldots, s_{i,\ell}$. Note that these additional secrets need not be stored: they can be derived from a secret seed.

The multiple-secret variant is described here for a single node, for the sake of clarity. Upon receiving a challenge e_i (assuming, for instance, that e_i was generated by the above procedure), each node computes a response:

$$y_i \leftarrow r_i s_{i,1}^{e_{i,1}} s_{i,2}^{e_{i,2}} \cdots s_{i,\ell}^{e_{i,\ell}} \bmod n.$$

This can be checked by the verifier by checking whether:

$$y_i^2 \overset{?}{=} x_i v_{i,1}^{e_{i,1}} v_{i,2}^{e_{i,2}} \cdots v_{i,\ell}^{e_{i,\ell}} \bmod n.$$

To do swarm authentication, it suffices to perform aggregation as described in the protocol of Sect. 3 at intermediate nodes.

Using this approach, one can adjust the memory-communication trade-off, as the security level is $\lambda = t\ell$ (single-node compromission). Therefore, if $\ell = 80$, for instance, it suffices to authenticate once to get the same security as $t = 80$ authentications with $\ell = 1$ (which is the protocol of Sect. 3). This drastically cuts bandwidth usage, a scarce resource for IoT devices.

Furthermore, computational effort can be reduced by using batch exponentiation techniques to compute y_i.

5.3 Precomputed Alphabet Variant

A way to further reduce computational cost is the following: each node chooses an alphabet of m words $w_0, \ldots, w_{m-1}$ (a word is a 32-bit value) and computes once and for all the table of all pairwise products $p_{i,j} = m_i m_j$. Note that each $p_{i,j}$ entry is 64 bits long.

The values s_i are generated by randomly sampling from this alphabet. Put differently, s_i is built by concatenating u words (bit patterns) taken from the alphabet only. We thus see that the s_i, which are mu-bit integers, can take m^u possible values.

Example 1 For instance, if $m = u = 32$, then s_i is a 1024-bit number chosen amongst $32^{32} = 2^{160}$ possible values. Thanks to the lookup table, most multiplications need not be performed, which provides a substantial speed-up over the naive approach.

The size of the lookup table is moderate, for the example given; all we need to store is $32 \times 31/2 + 32 = 528$ values. This can be further reduced by noting that the first lines in the table can be removed: 32 values are zeros, 31 values are the results of multiplications by 1, 30 values are left shifts by 1 of the previous line, 29 values are the sum of the previous 2 and 28 values are left shifts by 2. Hence all in all the table can be compressed into $528 - 32 - 31 - 29 - 28 = 408$ entries. Because each entry is a word, this boils down to 1632 bytes only.

5.4 Precomputed Combination Variant

The idea is that computational cost can be cut down if we precompute and store some products, only to assemble them online during Fiat–Shamir authentication: the values of $s_{i,1,2} \leftarrow s_{i,1}s_{i,2}$, $s_{i,2,3} \leftarrow s_{i,2}s_{i,3}, \ldots$ are stored in a lookup table.

The use of combined values $s_{i,a,b}$ in the evaluation of y results in three possible scenarios for each:

1. $s_a s_b$ appears in y—the probability of this occurring is 1/4—in which case one additional multiplication must be performed.
2. $s_a s_b$ does not appear in y—the probability of this occurring is 1/4—in which case no action is performed.
3. s_a or s_b appears, but not both—this happens with probability 1/2—in which case one single multiplication is required.

As a result the expected number of multiplications is reduced by 25%, to wit $\frac{3}{4} \times 2^{m-1}$, where m is the size of e.

The method can be extended to work in a window of size $\kappa \geq 2$; for instance, with $\kappa = 3$, we would precompute:

$$\begin{aligned} s_{i,3n,3n+1} &\leftarrow s_{i,3n} s_{i,3n+1} \\ s_{i,3n+1,3n+2} &\leftarrow s_{i,3n+1} s_{i,3n+2} \\ s_{i,3n,3n+2} &\leftarrow s_{i,3n} s_{i,3n+2} \\ s_{i,3n,3n+1,3n+2} &\leftarrow s_{i,3n} s_{i,3n+1} s_{i,3n+2} \end{aligned}$$

Following the same analysis as above, the expected number of multiplications during the challenge-response phase is $\frac{7}{8} \times \frac{2^m}{3}$. The price to pay is that larger values of κ claim more precomputing and memory. More precisely, we have the following trade-offs, writing $\mu = 2^m \bmod \kappa$:

$$\begin{array}{ll} \text{Multiplications(expected)} & = 2^m \left(\frac{2^\kappa - 1}{2^\kappa} \left(\left\lfloor \frac{2^m}{\kappa} - 1 \right\rfloor \right) - \frac{2^\mu - 1}{2^\mu} \right) \\ \text{Premultiplications} & = \ell - 1 + \left((2^\kappa - \kappa - 1) \left\lfloor \frac{2^m}{\kappa} \right\rfloor \right) + (2^\mu - \mu - 1) \\ \text{Storedvalues} & = (2^\kappa - 1) \left\lfloor \frac{2^m}{\kappa} \right\rfloor + (2^\mu - 1) \end{array}$$

where ℓ is the number of components of s_i.

6 Conclusion

In this work we describe a distributed Fiat–Shamir authentication protocol that enables network authentication using very few communication rounds, thereby alleviating the burden of resource-limited devices such as wireless sensors and other IoT nodes. Instead of performing one-on-one authentication to check the network's integrity, our protocol gives a proof of integrity for the whole network at once.

References

1. D. Anshul, S. Roy, A ZKP-based identification scheme for base nodes in wireless sensor networks, in *Proceedings of the 2005 ACM Symposium on Applied Computing, SAC '05* (ACM, New York, 2005), pp. 319–323
2. D. Boneh, C. Gentry, B. Lynn, H. Shacham, Aggregate and verifiably encrypted signatures from bilinear maps, in *Advances in Cryptology – EUROCRYPT 2003*, ed. by E. Biham. Lecture Notes in Computer Science, vol. 2656 (Springer, Heidelberg, 2003), pp. 416–432
3. T.H. Cormen, C. Stein, R.L. Rivest, C.E. Leiserson, *Introduction to Algorithms*, 2nd edn. (McGraw-Hill Higher Education, New York, 2001)
4. U. Feige, A. Fiat, A. Shamir, Zero-knowledge proofs of identity. J. Cryptol. **1**(2), 77–94 (1988)
5. A. Fiat, A. Shamir, How to prove yourself: practical solutions to identification and signature problems, in *Advances in Cryptology – CRYPTO'86*, ed. by A.M. Odlyzko. Lecture Notes in Computer Science, vol. 263 (Springer, Heidelberg, 1987), pp. 186–194
6. M. Girault, J. Stern, On the length of cryptographic hash-values used in identification schemes, in *Advances in Cryptology – CRYPTO'94*. ed. by Y. Desmedt. Lecture Notes in Computer Science, vol. 839 (Springer, Heidelberg, 1994), pp. 202–215
7. S. Goldwasser, S. Micali, C. Rackoff, The knowledge complexity of interactive proof-systems (extended abstract), in *Proceedings of the 17th Annual ACM Symposium on Theory of Computing, May 6–8, 1985, Providence, Rhode Island, USA*, ed. by R. Sedgewick (ACM, New York, 1985), pp. 291–304. http://doi.acm.org/10.1145/22145.22178
8. L.C. Guillou, J.J. Quisquater, A practical zero-knowledge protocol fitted to security microprocessor minimizing both trasmission and memory, in *Advances in Cryptology – EUROCRYPT'88*, ed. by C.G. Günther. Lecture Notes in Computer Science, vol. 330 (Springer, Heidelberg, 1988), pp. 123–128
9. C. Lavault, M. Valencia-Pabon, A distributed approximation algorithm for the minimum degree minimum weight spanning trees. J. Parallel Distrib. Comput. **68**(2), 200–208 (2008)
10. A.J. Mooij, N. Goga, J.W. Wesselink, A distributed spanning tree algorithm for topology-aware networks. Technische Universiteit Eindhoven, Department of Mathematics and Computer Science (2003)
11. A. Perrig, R. Szewczyk, J.D. Tygar, V. Wen, D.E. Culler, Spins: security protocols for sensor networks. Wirel. Netw. **8**(5), 521–534 (2002)
12. M. Singh, L.C. Lau, Approximating minimum bounded degree spanning trees to within one of optimal, in *Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing* (ACM, New York, 2007), pp. 661–670
13. S.K. Udgata, A. Mubeen, S.L. Sabat, Wireless sensor network security model using zero knowledge protocol, in *2011 IEEE International Conference on Communications* (IEEE, Piscataway, 2011), pp. 1–5
14. M. Hashim, G. Santhosh Kumar, A. Sreekumar, Authentication in wireless sensor networks using zero knowledge protocol, in *Computer Networks and Intelligent Computing*. vol. 157 (Springer, Berlin, 2011), pp. 416–421
15. L. Zhang, B. Qin, Q. Wu, F. Zhang, Efficient many-to-one authentication with certificateless aggregate signatures. Comput. Netw. **54**(14), 2482–2491 (2010)