Lightweight Swarm Authentication

George Teşeleanu[1,2]

[1] Advanced Technologies Institute
10 Dinu Vintilă, Bucharest, Romania
tgeorge@dcti.ro
[2] Simion Stoilow Institute of Mathematics of the Romanian Academy
21 Calea Grivitei, Bucharest, Romania

Abstract. In this paper we describe a provably secure authentication protocol for resource limited devices. The proposed algorithm performs whole-network authentication using very few rounds and in a time logarithmic in the number of nodes. Compared to one-to-one node authentication and previous proposals, our protocol is more efficient: it requires less communication and computation and, in turn, lower energy consumption.

1 Introduction

With the rise in popularity of the Internet of Things paradigm (IoT), low-cost devices with limited resources are used much more frequently by the industry and, implicitly, by us. In an IoT setting, spatially distributed nodes form a network and are able to control or monitor physical or environmental conditions[3], perform computations or store data. Due to the nodes' limited resources, either computational or physical, they normally transmit the acquired data through the network to a gateway which collects information and sends it to a processing unit.

Usually, nodes are deployed in hostile environments and therefore there are a number of serious security concerns that need to be addressed. Unfortunately, the lightweight nature of nodes heavily restricts cryptographic operations and makes any communication costly. Thus, the need for specific cryptographic solutions becomes obvious. The Fiat-Shamir-like distributed authentication protocol presented in [1] (denoted by $QR\text{-}Swarm$) represents such an example. Based on the $QR\text{-}Swarm$ construction, an unified generic zero-knowledge protocol is described in [7] (denoted by $Unif\text{-}Swarm$).

Although the $Unif\text{-}Swarm$ construction offers flexibility when choosing the underlying security assumption, the discrete logarithm instantiation (denoted by $DL\text{-}Swarm$) is one of interest. To ensure a certain security level, the $QR\text{-}Swarm$ protocol needs to be run several times. To reach the same security level it is sufficient to run the $DL\text{-}Swarm$ construction once. Therefore, it requires less communication. Note that the security of $DL\text{-}Swarm$ is based on the discrete logarithm problem.

[3] *e.g.* temperature, pressure, image, sound

In this paper we describe an authentication protocol that is an optimization of the *DL-Swarm* protocol. More precisely, our protocol reduces the number of messages transmitted during the authentication process, while requiring a lower level of processing. This in turn ensures lower power consumption, and hence nodes have a longer battery life. The security of our main proposal is based on the computational Diffie-Hellman assumption, instead of the discrete logarithm problem. Although we base our security on a weaker security notion, in practice, the only known way to attack the computational Diffie-Hellman is to attempt to retrieve one of the secret exponents (*i.e.* to solve the discrete logarithm) [2]. In the appendix we also propose a version of our protocol that is based on the computational bilinear Diffie-Hellman assumption. Even though this version is less efficient, we believe that the bilinear version is of theoretical interest, since it shows that we can implement the protocol in multiple ways.

Structure of the paper. Notations and preliminaries are presented in Section 2. We describe the core of our paper, a lightweight authentication protocol for IoT devices, in Section 3. We conclude in Section 4. A variation of our protocol is proposed in Appendix A.

2 Preliminaries

Notations. Throughout the paper, the notation $\|$ denotes string concatenation. The subset $\{0, \ldots, s\} \in \mathbb{N}$ is denoted by $[0, s]$. The action of selecting a random element x from a sample space X is represented by $x \xleftarrow{\$} X$, while $x \leftarrow y$ indicates the assignment of value y to variable x.

2.1 Hardness Assumptions

Definition 1 (Discrete Logarithm - DL). *Let $\mathbb{G}$ be a cyclic group of order q, g a generator of $\mathbb{G}$ and let A be a probabilistic polynomial-time algorithm (PPT algorithm) that returns an element from $\mathbb{Z}_q^*$. We define the advantage*

$$ADV^{\mathrm{DL}}_{\mathbb{G},g}(A) = Pr[A(g^x) = x | x \xleftarrow{\$} \mathbb{Z}_q^*].$$

If $ADV^{\mathrm{DL}}_{\mathbb{G},g}(A)$ is negligible for any PPT algorithm A, we say that the discrete logarithm problem is hard in $\mathbb{G}$.

Definition 2 (Computational Diffie-Hellman - CDH). *Let $\mathbb{G}$ be a cyclic group of order q, g a generator of $\mathbb{G}$ and let A be a probabilistic PPT algorithm that returns an element from $\mathbb{G}$. We define the advantage*

$$ADV^{\mathrm{CDH}}_{\mathbb{G},g}(A) = Pr[A(g^x, g^y) = g^{xy} | x, y \xleftarrow{\$} \mathbb{Z}_q^*].$$

If $ADV^{\mathrm{CDH}}_{\mathbb{G},g}(A)$ is negligible for any PPT algorithm A, we say that the computational Diffie-Hellman problem is hard in $\mathbb{G}$.

2.2 Zero-Knowledge Protocols

In a traditional proof of knowledge a prover, called Peggy, tries to convince a verifier, called Victor, that she knows a piece of knowledge. To achieve that, they engage in an interactive protocol. To make sense, the protocol must be complete and sound. Completeness means that if Peggy is honest she can succeed to convince an honest Victor that her claim is true, while soundness means that a dishonest Peggy cannot convince Victor of a false statement.

Formally, let $Q : \{0,1\}^* \times \{0,1\}^* \rightarrow \{\texttt{true}, \texttt{false}\}$ be a predicate. Given a value z, Peggy will try to convince Victor that she knows a value x such that $Q(z,x) = \texttt{true}$. The following definition [3, 8] captures the notions of completeness and soundness for a proof of knowledge protocol.

Definition 3 (Proof of Knowledge Protocol). *An interactive protocol (P,V) is a proof of knowledge protocol for predicate Q if the following properties hold*

- **Completeness***: V accepts the proof when P has as input a value x with $Q(z,x) = \texttt{true}$;*
- **Soundness***: there exists an efficient program K (called knowledge extractor) such that for any $\bar{P}$ (possibly dishonest) with non-negligible probability of making V accept the proof, K can interact with $\bar{P}$ and output (with overwhelming probability) an x such that $Q(z,x) = \texttt{true}$.*

An interesting property of proofs of knowledge is that they can be performed without leaking any information besides the validity of Peggy's claim. More precisely, the protocol can be implemented such that Victor can simulate it by himself (*i.e.* without requiring Peggy to be part of the protocol). The aforementioned property is called zero-knowledge and it is formally defined next [4, 8].

Definition 4 (Zero Knowledge Protocol). *A protocol (P,V) is zero-knowledge if for every efficient program $\bar{V}$ there exists an efficient program S, the simulator, such that the output of S is indistinguishable from a transcript of the protocol execution between P and $\bar{V}$.*

2.3 A Distributed Unified Protocol

Let us consider an n-node network consisting of $\mathcal{N}_1, ..., \mathcal{N}_n$. The nodes $\mathcal{N}_i$ can be seen as users and the base station $\mathcal{T}$ as a trusted center. To achieve the authentication of the entire network, the authors of [7] propose a unified Fiat-Shamir-like construction. The construction which we detail next is an instantiation of $Unif\text{-}Swarm$ using discrete logarithms.

Before deploying the nodes, we must select the network's security parameters. Thus, we choose a group $\mathbb{G}$ of order p and we select an element g of order q, where q is a large prime such that $q|p$. After selecting the public parameters, each node $\mathcal{N}_i$ randomly selects its private key $x_i \xleftarrow{\$} \mathbb{Z}_q^*$ and computes the corresponding public key $z_i \leftarrow g^{x_i}$.

After the nodes are deployed, the network topology has to converge and a spanning tree needs to be constructed. For example, we can use an algorithm similar with the one presented in [9].

The protocol proposed in [7] can be summarized as follows:

1. First, $\mathcal{T}$ sends an authentication request message to all the $\mathcal{N}_i$ nodes directly connected to it. The request message may contain a commitment *com* to the challenge c (see 3.) to ensure the protocol's zero-knowledge property even against dishonest verifiers.

2. After receiving an authentication request message:
   - Each $\mathcal{N}_i$ generates a private $k_i \xleftarrow{\$} \mathbb{Z}_q^*$ and computes $t_i \leftarrow g^{k_i}$;
   - The $\mathcal{N}_i$ nodes send authentication messages to all their (existing) children;
   - After the children respond, nodes $\mathcal{N}_i$ compute $t_i \leftarrow t_i \cdot \left(\prod_j t_j\right)$ and send the result up to their parents. Note that the t_j values are sent by the nodes' children.

   Such a construction permits the network to compute the product of all the t_i values and send the result t_c to the top of the tree in d steps, where d represents the degree of the spanning tree. We refer the reader to Figure 1 for a toy example of this step.

3. $\mathcal{T}$ sends a random $c = (c_1, \ldots, c_n) \in [0, q-1]^n$ as an authentication challenge to the $\mathcal{N}_i$ nodes directly connected to it.

4. After receiving an authentication challenge c:
   - Each $\mathcal{N}_i$ computes $r_i \leftarrow k_i + c_i x_i \bmod q$;
   - The $\mathcal{N}_i$ nodes then send the authentication challenge c to all their (existing) children;
   - After the children respond, the nodes $\mathcal{N}_i$ compute $r_i \leftarrow r_i + \left(\sum_j r_j\right) \bmod q$ and send the result to their parents. Note that the r_j values are sent by the nodes' children.

   The network therefore computes collectively the sum of all the r_i values and transmits the result r_c to $\mathcal{T}$. Again, we refer the reader to Figure 1 for a toy example of this step.

5. After receiving the response r_c, $\mathcal{T}$ authenticates the whole network if and only if $g^{r_c} = t_c \cdot (\prod_{i=1}^{n} z_i^{c_i})$ holds.

In [7], the authors investigate the relation between the *DL-Swarm* protocol and the discrete logarithm assumption. Their result is summarized in Theorem 1.

Theorem 1. *The DL-Swarm protocol is a proof of knowledge if and only if the* DL *assumption holds. Moreover, the protocol is zero knowledge.*

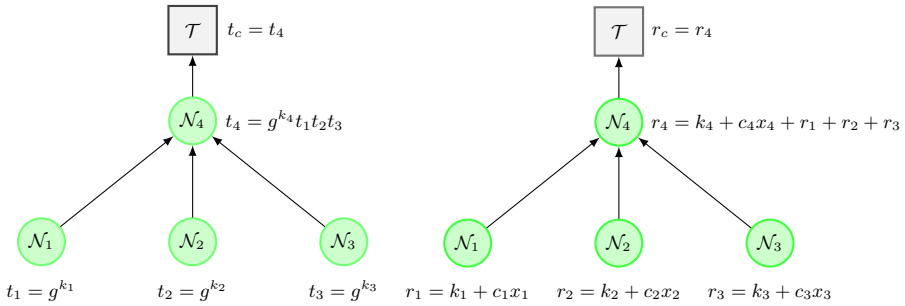

Fig. 1: The *DL-Swarm* algorithm running on a network consisting of 4 nodes: computation of t_c (left) and of r_c (right).

3 Computational Diffie-Hellman Swarm Protocol

3.1 Description

Let us consider again an n-node network consisting of the nodes $\mathcal{N}_1, ..., \mathcal{N}_n$ and a base station $\mathcal{T}$. The core idea of our proposal is that the base station does a Diffie-Hellman type key exchange with all its children and then compares the resulting network key with its own key.

We further describe our proposed distributed protocol (further denoted by *CDH-Swarm*).

1. After the network is set, $\mathcal{T}$ sends an authentication request message to all the $\mathcal{N}_i$ nodes directly connected to it. The request message contains a challenge $c \leftarrow g^k$, where $k \xleftarrow{\$} \mathbb{Z}_q^*$.
2. After receiving an authentication request message:
   - Each $\mathcal{N}_i$ computes $t_i \leftarrow c^{x_i}$;
   - The $\mathcal{N}_i$ nodes send authentication messages to all their (existing) children;
   - After the children respond, $\mathcal{N}_i$ nodes compute $t_i \leftarrow t_i \cdot \left(\prod_j t_j\right)$ and send the result up to their parents. Note that the t_j values are sent by the nodes' children.

   Such a construction permits the network to compute the product of all the t_i values and send the result t_c to the top of the tree in d steps, where d represents the degree of the spanning tree. We refer the reader to Figure 2 for a toy example of this step.
3. After receiving the response t_c, $\mathcal{T}$ authenticates the whole network if and only if $t_c = \left(\prod_{i=1}^{n} z_i\right)^k$ holds.

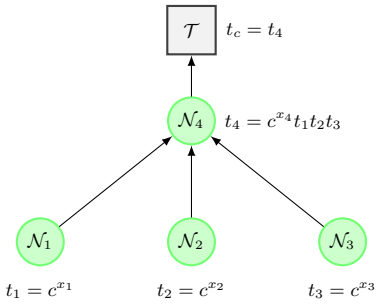

Fig. 2: The *CDH-Swarm* algorithm running on a network consisting of 4 nodes: computation of t_c.

Remark 1. The *CDH-Swarm* protocol either authenticates the whole network or none of the nodes. More precisely, a single defective node suffices for authentication to fail. In certain cases this is not acceptable and more information is needed. For instance, one could wish to know which node is responsible for the authentication failure. A simple back-up strategy consists in performing the protocol with each of the nodes that still respond and thus identify where the problem lies. Since all nodes already have the hardware and software required to perform the protocol, the nodes can use the same keys to perform the one-to-one protocol with the base station. Hence, this back-up solution adds no implementation overhead. Note that, as long as the network is healthy, using our distributed algorithm instead is more efficient and consumes less bandwidth and less energy.

3.2 Security Analysis

Before stating the security proof, we first want to point out that in the case of our protocol Peggy is given as input (g^{x_i}, g^k) and she will try to convince Victor that she knows an element v such that the predicate "Is $v = g^{x_i k}$?" is true.

Theorem 2. *The CDH-Swarm protocol is a proof of knowledge if and only if the* CDH *assumption holds. Moreover, the protocol is zero knowledge.*

Proof. If $\mathcal{T}$ receives a genuine t_c, then we have

$$t_c = \prod_{i=1}^{n} t_i = \prod_{i=1}^{n} c^{x_i} = \prod_{i=1}^{n} (g^k)^{x_i} = \prod_{i=1}^{n} (g^{x_i})^k = \left(\prod_{i=1}^{n} z_i \right)^k .$$

Hence, $\mathcal{T}$ will always accept honest t_c values, and thus the completeness property is satisfied.

Let $\tilde{P}$ be a PPT algorithm that takes as input $z_1, \ldots, z_n$ and makes $\mathcal{T}$ accept the proof with non-negligible probability $Pr(\tilde{P})$. Then we are able to construct a PPT algorithm K (described in Algorithm 1) that interacts with $\tilde{P}$ and that

has a non-negligible advantage $ADV^{\text{CDH}}_{\langle g\rangle,g}(K) = Pr(\tilde{P})$. Therefore, given $y_0 \leftarrow g^u$ and $y_1 \leftarrow g^v$ algorithm K can compute g^{uv} with non-negligible probability.

More precisely, on input y_0 and y_1, algorithm K first assigns as the public key of node $\mathcal{N}_i$ the value y_0, randomly selects the secret keys for the remaining $n-1$ nodes and computes their corresponding public keys. Then, he runs the protocol with $\tilde{P}$, where K plays the role of the base station. Note that the challenge K send to $\tilde{P}$ is the value y_1. Once the protocol is finished, K can recover the correct value of g^{uv} only if $\tilde{P}$ is successful into authenticate himself. Hence, the success probability of K is the same as the one of $\tilde{P}$, and thus the soundness property is satisfied.

Algorithm 1. Algorithm K.

Input: Two elements $y_0 \leftarrow g^u$ and $y_1 \leftarrow g^v$

1 Select $i \xleftarrow{\$} [1,n]$ and $x_j \xleftarrow{\$} \mathbb{Z}_q^*$, where $j \in [1,n] \setminus \{i\}$
2 Compute $z_j \leftarrow g^{x_j}$ and set $z_i \leftarrow y_0$
3 Send $z_1, \ldots, z_n$ to $\tilde{P}$
4 Send y_1 to $\tilde{P}$
5 Receive t_c from $\tilde{P}$
6 Compute $w \leftarrow \prod_{j \neq i} y_1^{-x_j}$
7 **return** $t_c \cdot w$

The last part of our proof consists in constructing a simulator S such that its output is indistinguishable from a genuine transcript between the nodes and the base station. Such a simulator is described in Algorithm 2.

Algorithm 2. Simulator S.

Input: The nodes' public keys $z_1, \ldots, z_n$

1 Choose $k \xleftarrow{\$} \mathbb{Z}_q^*$
2 For each node, compute $t_i \leftarrow z_i^k \cdot \left(\prod_j t_j\right)$, where t_j are the values computed by the current node's children
3 **return** $g^k, t_1, \ldots, t_n$

□

Remark 2. Note if an adversary is simulating only n' out of n nodes of the network, then he still has to face an *CDH-Swarm* protocol with n' nodes.

Small subgroup attack. The small subgroup attack [5,6] demonstrates that validating ephemeral keys is a prudent and, in some cases, essential measure in Diffie-Hellman type protocols. We further illustrate what happens in the case

of the *CDH-Swarm* protocol. For simplicity, we further assume that $n = 1$. The exact details of the *CDH-Swarm* protocol with $n = 1$ are illustrated in Figure 3.

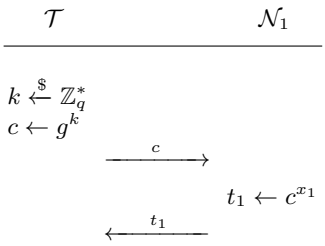

Fig. 3: The *CDH-Swarm* protocol.

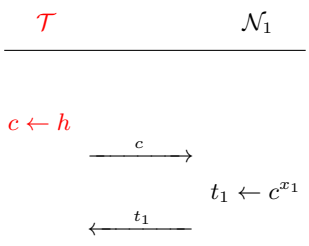

Fig. 4: The small subgroup attack.

The small subgroup attack works only if the order of $\mathbb{G}$ is not prime. More precisely, if $p = ms$ where $s > 1$ is small. Using this assumption, the attacker forces t_1 to be from the subgroup of order s. Therefore, he is able to obtain some information about the node's secret key.

Let h be an element of order m. The exact details of the attack are presented in Figure 4. Note that t_1 now lies in the subgroup of order s, and thus the attacker can learn using brute force the value x_1 mod s. By iterating this attack for each small prime factor of p, an attacker can learn μ bits of x_1, where μ is the bit length of the small factors of p. Hence, the small subgroup attack lowers the security margin by μ. For more details, we refer the reader to [6].

As long as p is chosen such that $\log_2 q - \mu$ is large[4], the small subgroup attack does not affect future authentications of $\mathcal{N}_1$, since the attacker still needs to find the remaining bits of x_1 in order to impersonate $\mathcal{N}_1$.

3.3 Complexity Analysis

The number of operations[5] necessary to authenticate the network depends on the topology at hand. Note that each node performs in average only a few operations (a constant number). Precise complexity evaluations are given in Tables 1 and 2. The motivation of considering per node metrics is to show that our protocol reduces the number of operation, and hence minimizes the risk of one node running out of batteries. We can clearly see that compared to *DL-Swarm* *CDH-Swarm* reduces the overall complexity of authenticating the network.

[4] *e.g.* when $\mathbb{G} = \mathbb{Z}_p$, if prime p is chosen such that $q = (p-1)/2$ is also prime, then we only lower the security margin by 1 bit.

[5] Random number generations are denoted by RNG.

Operation	Number of computations	
	DL-Swarm [7]	*CDH-Swarm*
Exponentiation	$2n+1$	$n+2$
Multiplications	$\leq 3n$	$\leq 2n$
Additions	$\leq 2n$	0
RNG	$2n$	1

Table 1: The computational complexity of authenticating the network.

Operation	Number of computations	
	DL-Swarm [7]	*CDH-Swarm*
Exponentiation	1	1
Multiplications	$\leq n$	$\leq n-1$
Additions	$\leq n$	0
RNG	1	0

Table 2: The computational complexity per node.

Let $d = O(\log n)$ be the degree of the minimum spanning tree of the network. Then, only $O(d)$ messages are sent. Hence, throughout the authentication process only a logarithmic number of messages is sent.

Remark that in the case of the *DL-Swarm* protocol we have two rounds of messages from the base station toward the leafs and two from the leafs toward the base station. In our proposed protocol we reduce the protocol to one round from the base station toward the leafs and one from the leafs toward the base station. So, we reduce by half the number of messages transmitted through the network. The exact bandwith requirements per node are presented in Table 3.

Messages	Number of bits	
	DL-Swarm [7]	*CDH-Swarm*
Sent	$\leq (n+2)\lceil \log q \rceil + \lceil \log com \rceil$	$\leq 2\lceil \log q \rceil$
Received	$\leq 3n\lceil \log q \rceil + \lceil \log com \rceil$	$\leq (n+1)\lceil \log q \rceil$

Table 3: The bandwidth requirements per node.

3.4 Hash Based Variant

We further describe a *CDH-Swarm* variant that aims at reducing the amount of information sent by individual nodes. Note that for this protocol to work, the base station must know the network's topology beforehand.

According to the birthday paradox the probability of obtaining two identical public keys is $p \simeq 1 - e^{-n(n-1)/2q}$. Since q is significantly larger than n, then we

can safely assume that $p \simeq 0$. Hence, we can use a node's public key z_i as an unique identification number. Also, the z_i values can be used to induce a total order for the nodes on a given level using the usual $<$ operation.

Let $h : \{0,1\}^* \to \{0,1\}^\ell$ be a hash function. Using the previous remark, instead of transmitting an element t_i in Step 2, we can transmit a digest $t_i \leftarrow h(z_i \| t_i \| (\|_j t_j))$, where $t_{j_1} < t_{j_2}$ if and only if $z_{j_1} < z_{j_2}$. In parallel, $\mathcal{T}$ computes the correct response t_r by using the network's topology and the k value. After receiving the network's response, the base station can check if $t_c = t_r$.

From an efficiency point of view, compared to *DL-Swarm*, multiplications become hash computations and instead of transmitting $\lceil \log q \rceil$ bits, each node transmits ℓ bits. Note that this variant does not impact security, assuming that h is an ideal hash function.

4 Conclusions

In this paper we described a distributed authentication protocol that enables network authentication using fewer rounds per authentication than previous proposed solutions. Thereby making it more suitable for resource-limited devices such as wireless sensors and other IoT devices. To conserve energy and bandwidth, our proposal gives a proof of integrity for the whole network at once, instead of authenticating each individual node.

References

1. Cogliani, S., Feng, B., Ferradi, H., Géraud, R., Maimuţ, D., Naccache, D., do Canto, R.P., Wang, G.: Public Key-Based Lightweight Swarm Authentication. In: Cyber-Physical Systems Security, pp. 255–267. Springer (2018)
2. Crandall, R., Pomerance, C.: Prime Numbers: A Computational Perspective. Springer (2005)
3. Feige, U., Fiat, A., Shamir, A.: Zero-Knowledge Proofs of Identity. Journal of Cryptology **1**(2), 77–94 (1988)
4. Goldwasser, S., Micali, S., Rackoff, C.: The Knowledge Complexity of Interactive Proof Systems. SIAM J. Comput. **18**(1), 186–208 (1989)
5. Law, L., Menezes, A., Qu, M., Solinas, J., Vanstone, S.: An Efficient Protocol for Authenticated Key Agreement. Des. Codes Cryptogr. **28**(2), 119–134 (2003)
6. Lim, C.H., Lee, P.J.: A Key Recovery Attack on Discrete Log-based Schemes Using a Prime Order Subgroupp. In: CRYPTO 1997. Lecture Notes in Computer Science, vol. 1294, pp. 249–263. Springer (1997)
7. Maimuţ, D., Teşeleanu, G.: A Generic View on the Unified Zero-Knowledge Protocol and its Applications. In: WISTP 2019. Lecture Notes in Computer Science, vol. 12024, pp. 32–46. Springer (2019)
8. Maurer, U.: Unifying Zero-Knowledge Proofs of Knowledge. In: AFRICACRYPT 2009. Lecture Notes in Computer Science, vol. 5580, pp. 272–286. Springer (2009)
9. Mooij, A.J., Goga, N., Wesselink, J.W.: A Distributed Spanning Tree Algorithm for Topology-Aware Networks. Technische Universiteit Eindhoven, Department of Mathematics and Computer Science (2003)

A Computational Bilinear Diffie-Hellman Swarm Protocol

In this section we provide the reader with a swarm protocol based on a different security assumption. Namely, the computational bilinear Diffie-Hellman assumption.

Definition 5 (Computational Bilinear Diffie-Hellman - CBDH). *Let $\mathbb{G}$ be a cyclic group of order q, P a generator of $\mathbb{G}$ and $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ a cryptographic bilinear map, where G_T is a cyclic group of order q. We will use the convention of writing $\mathbb{G}$ additively and $\mathbb{G}_T$ multiplicatively. Let A be a probabilistic PPT algorithm that returns an element from $\mathbb{G}_T$. We define the advantage*

$$ADV^{\text{CBDH}}_{\mathbb{G},g,e}(A) = Pr[A(xP, yP, zP) = e(P, P)^{xyz} | x, y, z \xleftarrow{\$} \mathbb{Z}^*_q].$$

If $ADV^{\text{CBDH}}_{\mathbb{G},g,e}(A)$ is negligible for any PPT algorithm A, we say that the computational bilinear Diffie-Hellman problem is hard in $\mathbb{G}$.

We further assume that the group $\mathbb{G}$ admits a computationally efficient bilinear map $e(\cdot, \cdot)$ such that CBDH is hard in $\mathbb{G}$. Using the same setup[6] as in the case of $CDH\text{-}Swarm$, we present below the full details of the bilinear version of the swarm protocol (denoted by $CBDH\text{-}Swarm$):

1. Let $x_i, y_i \xleftarrow{\$} \mathbb{Z}^*_q$ be the private keys given to node $\mathcal{N}_i$ and $z_i \leftarrow x_i P$, $w_i \leftarrow y_i P$ the node's public keys. After the network is set, $\mathcal{T}$ sends an authentication request message to all the $\mathcal{N}_i$ nodes directly connected to it. The request message contains a challenge $c \leftarrow kP$, where $k \xleftarrow{\$} \mathbb{Z}^*_q$.
2. After receiving an authentication request message:
   - Each $\mathcal{N}_i$ computes $t_i \leftarrow e(c, P)^{x_i y_i}$;
   - The $\mathcal{N}_i$ nodes send authentication messages to all their (existing) children;
   - After the children respond, $\mathcal{N}_i$ nodes compute $t_i \leftarrow t_i \cdot \left(\prod_j t_j\right)$ and send the result up to their parents. Note that the t_j values are sent by the nodes' children.

   Such a construction permits the network to compute the product of all the t_i values and send the result t_c to the top of the tree in d steps, where d represents the degree of the spanning tree.
3. After receiving the response t_c, $\mathcal{T}$ authenticates the whole network if and only if $t_c = \left(\prod_{i=1}^{n} e(z_i, w_i)\right)^k$ holds.

Remark 3. Note that the hash based variant of the $CDH\text{-}Swarm$ protocol can also be easily adapted to the $CBDH\text{-}Swarm$ version.

[6] Traditionally, in the case of CBDH, the generator is denoted by P instead of g.

We further link the security of the *CBDH-Swarm* protocol to the CBDH assumption.

Theorem 3. *The CBDH-Swarm protocol is a proof of knowledge if and only if the* CBDH *assumption holds. Moreover, the protocol is zero knowledge.*

Proof (sketch). We will only prove that the scheme is sound, since the remaining security requirements are proven similarly to Theorem 2. Hence, we have

$$\begin{aligned} t_c &= \prod_{i=1}^{n} t_i = \prod_{i=1}^{n} e(c, P)^{x_i y_i} = \prod_{i=1}^{n} e(P, P)^{x_i y_i k} \\ &= \prod_{i=1}^{n} e(x_i P, y_i P)^k = \left(\prod_{i=1}^{n} e(z_i, w_i) \right)^k . \end{aligned}$$

□