ODIN: Obfuscation-based privacy preserving consensus algorithm for Decentralized Information fusion in smart device Networks

MORENO AMBROSIN, University of Padua, Italy
PAOLO BRACA, NATO STO CMRE, La Spezia, Italy
MAURO CONTI, University of Padua, Italy
RICCARDO LAZZARETTI, University of Padua, Italy

The large spread of sensors and smart devices in urban infrastructures are motivating research in the area of Internet of Thing (IoT), to develop new services and improve citizens' quality of life. Sensors and smart devices generate large amount of measurement data from sensing the environment, which is used to enable services, such as control power consumption or traffic density. To deal with such a large amount of information, and provide accurate measurements, service providers can adopt *information fusion*, which, given the decentralized nature of urban deployments, can be performed by means of *consensus algorithms*. These algorithms allow distributed agents to (iteratively) compute linear functions on the exchanged data, and take decisions based on the outcome, without the need for the support of a central entity. However, the use of consensus algorithms raises several security concerns, especially when private or security critical information are involved in the computation.

This paper proposes ODIN, a novel algorithm that allows information fusion over encrypted data. ODIN is a privacy-preserving extension of the popular consensus gossip algorithm, that prevents distributed agents have direct access to the data while they iteratively reach consensus; agents cannot access even the final consensus value, but can only retrieve partial information, e.g., a binary decision. ODIN uses efficient additive obfuscation and proxy re-encryption during the update steps, and Garbled Circuits to take final decisions on the obfuscated consensus. We discuss the security of our proposal, and show its practicability and efficiency on real-world resource constrained devices, developing a prototype implementation for Raspberry Pi devices.

CCS Concepts: • **Security and privacy → Privacy-preserving protocols**; **Distributed systems security**; *Network security*; • **Theory of computation → Cryptographic protocols**; • **Computing methodologies → Distributed computing methodologies**; • **Networks** → *Network performance analysis*;

Additional Key Words and Phrases: Consensus Algorithms, Information Fusion, Internet of Things, Privacy Preserving Applications, Proxy Re-Encryption, Secure Multi-Party Computation

ACM Reference format:
Moreno Ambrosin, Paolo Braca, Mauro Conti, and Riccardo Lazzaretti. 2017. ODIN: Obfuscation-based privacy preserving consensus algorithm for Decentralized Information fusion in smart device Networks. 1, 1, Article 1 (March 2017), 21 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

Mauro Conti is supported by a Marie Curie Fellowship funded by the European Commission under the agreement n. PCIG11-GA-2012-321980. This work has been partially supported by the TENACE PRIN Project 20103P34XC funded by the Italian MIUR. Author's addresses: M. Ambrosin, M. Conti and R. Lazzeretti (contact author: riccardo.lazzeretti@math.unipd.it), Department of Mathematics, University of Padua, Italy; P. Braca, NATO STO CMRE, La Spezia, Italy.

DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Urban infrastructures are rich of sensors placed in devices, vehicles and buildings connected to the Internet of Thing (IoT). With smart and forward-looking leadership, IoT has the potential to create a revolution in city planning and management. By embracing the potential of IoT, governments can improve service delivery, increase sustainability, and make their cities safer and more livable places for all residents. As an example, sensors distributed in urban areas can be used to monitor air and water pollution, or the energy consumption in city buildings and light infrastructures. Two examples of deployment of urban sensor networks are Chicago's Array of Things[1] and Dublin's City Watch[2], which use sensors to monitor environmental information, make predictions about vehicle and pedestrian congestion, and manage incidents. While single sensor measures have limited interest and can be affected by sensor noise, data collection and fusion from many sources (a.k.a. *sensor fusion*) has the potential to improve information accuracy, and enable more meaningful statistics on the resulting data [29].

Sensor fusion can be performed either in centralized or distributed environments. In the former case, a central server collects and elaborates the data provided by sensor, while in latter case, agents (i.e., sensors) take full responsibility for fusing the data. In decentralized sensor fusion protocols, every agent can be viewed as an intelligent asset with some degree of autonomy in taking decisions [64]. A possible application example could be vehicle-to-vehicle communication[3], which is tested by the US Department of Transportation, and aims at enabling cars to avoid crashes, ease traffic congestion and improve the environment. Other examples include object or people tracking, smart meter data fusion, etc. In modern paradigms of decentralized information fusion, agents (sensors, or other smart devices in distributed IoT deployments) are usually interconnected and communicate, via wireless [1]. These scenarios are characterized by a dynamic network topology, and intermittent connectivity among devices. Therefore, distributed signal processing algorithms performing data fusion must be robust to any changes in network topology. The consensus paradigm [8, 14, 16–18, 23, 36, 51] fits well with the decentralized and intermittent nature of such networks, and allow distributed units to corroborate local observations (e.g., measurements), with observations made by neighboring agents. In a consensus algorithm, agents exchange data and update the locally computed statistics, to (asymptotically) reach the agreement about a common value shared by all the agents, which represents the final statistic. In practice, during the average consensus protocol, agents update their measures by computing *the average* between their measures and the one provided by adjacent nodes. After several iterations, each node obtains a new measures, "close" to the average of all network nodes' measures. Despite its limited capability of average estimator, consensus protocol is a building block that has demonstrated its utility in several urban environment scenarios, such as coordination of groups of mobile agents [34], vehicle formation [28, 55], tracking and data fusion [58], flocking [50].

Unfortunately, even if many information are of public domain and the communication protocols in the network can be considered secure (i.e., using transport layer security techniques), connected IoT technologies could potentially open up private data to nefarious entities, such as hackers or cyber criminals. For this reason, private information protected by privacy laws may not be shared in their plain form to other agents involved in the computation, and sometimes neither in their aggregated form. Furthermore, there are cases in which, even if the data itself does not need to be protected, sensor owners can be interested to not reveal the original measures to protect some characteristic of the sensor, such as its accuracy. To solve these problems, some privacy preserving

[1]https://arrayofthings.github.io/
[2]http://citywatch.ie/
[3]https://www.nhtsa.gov/technology-innovation/vehicle-vehicle-communications

strategies must be applied to perform analysis, so that interaction is achieved by exchanging *encrypted* or *masked* information between agents.

This paper considers a distributed IoT scenario in which agents cooperate, and run a distributed consensus protocol, but at the same time do not want to reveal each other their own information, for security reasons and privacy preservation (e.g., classified information, or protection of sensor characteristics). Similarly to other privacy preserving multi-party applications, such as data mining [47], biometric matching [6, 11, 25], recommendation systems [24], and biomedical analysis [7], the consensus protocol can be implemented in the encrypted domain [40] by using secure protocols.

In this way, the consensus on a common value is reached while each agent has access *only* to its inputs, and to the final decision, obtained by evaluating the protocol on encrypted statistics. To simply clarify this situation, assume that a pair of agents, say i and j, want to make a binary decision $\{\mathcal{H}_0, \mathcal{H}_1\}$ based on some functionality $\mathcal{L}()$ evaluated on their measures x_i and x_j, respectively. Decision can be taken on the average of the statistics. However, both agents i and j do not want to reveal each other their plain local data, and instead exchange them only in encrypted form $[\![\mathcal{L}(x_{i,j})]\!]$ (see Fig. 1). The extension of the above simple example to large distributed networks can be implemented by relying on privacy preserving consensus algorithms.

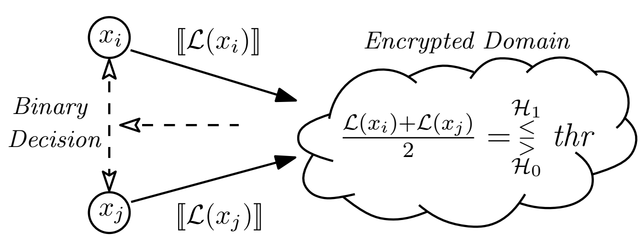

Fig. 1. Conceptual scheme of cooperation between agents i and j to solve a binary decision testing problem in a privacy preserving setting. Agents can communicate with each other and exchange only encrypted data, for instance an encrypted version of $[\![\mathcal{L}(x_{i,j})]\!]$, however at the same time they want to have access to the binary decision based on both the observations x_i and x_j.

Contribution. In this paper, we present ODIN, a novel and efficient solution that: (i) allows a network of devices (agents) to achieve a consensus in a privacy-preserving way; and (ii) works also in dynamic networks. Our solution involves additive blinding (sometimes referred as obfuscation or masking) and proxy re-encryption for the iterative update steps, and garbled circuits for the final decision step.

The core idea behind ODIN is that, during the computation, sensitive values provided to an agent are always masked by random values chosen by another agent. At the end of each update step, each of the two agents involved, knowing the random value, sends it to the other agent in an encrypted form. In this way, by using proxy re-encryption protocols [5, 33], the latter can re-encrypt it (but not access it) into a ciphertext that another node involved in the next consensus step can decrypt. To the best of our knowledge, ODIN is the first protocol that uses obfuscation and proxy re-encryption for the implementation of a secure multi party computation protocol.

ODIN is secure in the semi-honest model with non-colluding nodes. Every node in the network can be interested in observing other agents information, but does not deviate from the protocol for this purpose. Despite its simplicity, designing and evaluating the performance of protocols in the semi- honest model is a *first stepping stone* towards protocols with stronger security guarantees for IoT device coordination in a distributed and decentralized setting. Moreover we uderline that the semi-honest model applies to existing relevant IoT use cases, such as in privacy-preserving techniques for smart metering systems [22, 26].

We finally underline that despite operating on blinded values, ODIN algorithm is able to reach the same final binary decision of the equivalent plain gossip algorithm, without errors and without disclosing the final consensus to the agents involved. Previous solutions, such as [15], are only able to reach an approximated consensus or provide a public consensus from obfuscated inputs under specific assumptions. The only error introduced by the protocol is due to the quantization necessary to represent inputs as a ratio, that can be made as little as desired.

Related works. In the realm of distributed consensus, some privacy-preserving approaches have been recently proposed. Many of them [15, 30, 49] propose solutions that protect the measures in consensus networks by introducing in the first step a random noise that decreases during the protocol so that a consensus close to the correct one is reached. Other works in privacy preserving data fusion were addressed in [56, 57]. In these works, the authors use additive blinding or secret sharing to estimate the position of one or more targets, by computing the average of the measurements of multiple sensors. The final result is not affected by noise, but the aforementioned works require that agents are connected through a well defined path starting and finishing at each agent, and passing through all the agents, as for instance in a ring network.

A first step towards a privacy preserving implementation of the consensus algorithm has been proposed in [44], where authors approach the sensor fusion problem by using the popular iterative gossip consensus protocol [14, 23] in the encrypted domain. In each step, measures from two adjacent agents are updated by relying on an expensive homomorphic encryption protocol [52] that, after any update step, outputs the state of each node encrypted with the public key of all the adjacent nodes, permitting a node to continue the computation with any other neighbour. Such solution presents two main concerns. First of all the computation and communication complexity linearly depends on the number of adjacent nodes of the agents i and j involved in the computation. This can be acceptable only in sparse scenarios, where each sensor has few neighbor nodes. Analogously, the complexity of the protocol could be very high in dense networks, such as in urban environments. Secondly the protocol can be applied only to static networks (application to dynamic networks implies some changes that make the complexity for each update linear with the total number of nodes, i.e., impractical in large urban networks).

2 PRELIMINARIES

This section presents the main cryptographic tools that are used in ODIN. We first introduce additive blinding, a simple cryptographic protocol that is used in the update step. Then we present proxy re-encryption, used to interface two following update steps where different agents are involved. Finally, we present Garbled Circuit, that we use in the final decision step.

2.1 Additive Blinding and Data Representation

The simplest way to protect data provided by a party to another one is through blinding (sometimes referred to as obfuscation or masking). We say that a blinding $y = ax + b$ preserves the meaning of a functionality $f(x)$ if a corresponding operation $g(y)$ exists such that $f(x) = \alpha(a, b)g(y) + \beta(a, b)x + \gamma(a, b)$, where $\alpha,\ \beta,\ \gamma$ are arbitrary functions of $a,\ b$. The idea is that a party can evaluate a function f' on a blinded value, so that the party that blinded the output can remove blinding from the output of g.

In simple additive blinding, a user i masks his value x by adding a random value b and transmits $x + b$ to another party. The receiver is not able to obtain x, but whoever knows b can retrieve x. To make this scheme really secure some assumptions must be made on the representation of x and

b. If the input x is a floating point number, b cannot be generated uniformly in the set of all the possible floating point numbers, because their sum can cause the loss of many significant digits of x, when $b \gg x$. For this reason, it is preferable to represent each input value as an integer number obtained by quantization, i.e., given an amplification factor K (usually a power of 2), x is mapped in the integer value $x' = \lfloor K \cdot x \rfloor$. At this point additive blinding is performed by using integer numbers. We underline that it is also possible to approximate a value to a close rational number and then represent each value X as a ratio num/den between an integer numerator num and an integer denominator den, which can be both represented in $\mathbb{Z}_n$. In this paper we both amplify input values and represent them as a ratio where at the beginning $den = 1$.

To achieve perfect secrecy, additive blinding must be performed by using modular arithmetic, as in a one-time pad. Assuming that $x' \in \mathbb{Z}_n$, additive blinding is secure if b is uniformly chosen in $\mathbb{Z}_n$. However, for efficiency reasons, given the bitlength ℓ necessary to represent any possible input x' to the protocol, b is often chosen in $\mathbb{Z}_{2^{\ell+t}}$ according to a uniform distribution [10], where t is a number of bits sufficiently large to statistically guarantee low information leakage (usually $t = 80$).

Additive blinding is commonly used in hybrid protocols, as described in [38], to permit efficient evaluation of complex functions for which solutions based on a single cryptographic tool would be inefficient (or even impossible). Being addition efficient in both secure multi-party computation protocols and homomorphic protocols, the interface between different cryptographic protocols is performed by using additive blinding. Random values are added by a cryptographic protocol, the obfuscated value is then disclosed, and used as input to the following cryptographic protocol that will remove the obfuscation. Several hybrid protocols working on homomorphic encryption and garbled circuits have been proposed for privacy preserving biometric authentication [6, 11], biomedical applications [7, 43], etc. Similarly, the implementation of a secure multi-party consensus gossip algorithm in [44] relies on homomorphic encryption and garbled circuit.

2.2 Proxy Re-encryption

Proxy re-encryption allows a semi-trusted proxy to convert a ciphertext, computed under the public key of a party, into a ciphertext that can be opened by using the secret key of another party, without seeing the underlying plaintext. Proxy re-encryption has many applications (secure network file storage [5, 69], email forwarding [4], Digital Right Management [61] or secure mailing lists [37]. In this paper we use proxy re-encryption to allow a node of the consensus network (the recipient) to decrypt values encrypted under the public key of another node (the sender) so that: (i) the node in the middle (the proxy) cannot decrypt the message; and (ii) the sender does not know who is the recipient, which is chosen by the proxy.

A proxy re-encryption scheme is a tuple of (possibly probabilistic) polynomial time algorithms (KeyGen, Enc, Dec, ReEncGen, ReEnc), where KeyGen, Enc, Dec are standard key generation, encryption, and decryption algorithms for the underlying cryptosystem, ReEncGen is the algorithm for the generation of the re-encryption keys, and ReEnc converts a ciphertext for a party into a ciphertext for another party.

Among many interesting proxy re-encryption protocols such as [19, 20, 46] (and many others), we focus on the one proposed in [5], because: (1) it guarantees *indistinguishability under chosen ciphertext attacks* (CPA), which presumes that an attacker can obtain the ciphertexts for arbitrary plaintexts without gaining any advantage (guaranteed by the probabilistic component of the encryption scheme); (2) is *unidirectional*, i.e., the delegation of a user A to another user B, does not allow re-encryption from B to A; (3) is *non-transitive*, the proxy cannot construct a re-encryption key $\pi_{A\to C}$ from the two keys $\pi_{A\to B}$ and $\pi_{B\to C}$; (4) is *non-interactive*, i.e., a user A cannot construct a re-encryption key $\pi_{A\to B}$ without the participation of B or of the Private Key Generator; and (5) is *space optimal*, i.e., additional communication costs are not needed in order to support re-encryption (the scheme does not cause ciphertext expansion upon re-encryption and the size of B's secret storage remain constant, regardless of how many delegations he accepts).

In order to be self-contained in the description of our protocol, we now briefly recall the construction in [5]. The scheme operates over two groups G_1, G_2 of prime order q with a bilinear map $e : G_1 \times G_1 \rightarrow G_2$ [13, 35]. The system parameters are random generators $g \in G_1$ and $Z = e(g, g) \in G_2$. The scheme is defined as follows:

Key Generation (KeyGen). The algorithm outputs a key pair (pk_A, sk_A) for a user A of the form: $pk_A = (Z^{a_1}, g^{a_2})$ and $sk_A = (a_1, a_2)$;

Re-encryption key genaration (ReEncGen). The algorithm permits user A to generate a re-encryption key $\pi_{A\rightarrow B}$ for a user B, as $\pi_{A\rightarrow B} \leftarrow \mathsf{ReEncGen}(pk_A, pk_B) = (g^{b_2})^{a_1} = g^{a_1 b_2} \in G_1$;

Encryption (Enc_1, Enc_2). To encrypt a message $m \in G_2$ under pk_A in such a way that only the holder of sk_A can decrypt it, the algorithm outputs $[\![m]\!]_A = \mathsf{Enc}_1(m, pk_A) = (Z^{a_1 k}, mZ^k)$ (*First Level Encryption*), where k is a random value; to encrypt a message $m \in G_2$ under pk_A in such a way it can be decrypted by A and her delegatees (after having performed proxy re-encryption), the algorithm outputs $[m]_A = \mathsf{Enc}_1(m, pk_A) = (g^k, mZ^{a_1 k})$ (*Second Level Encryption*);

Re-encryption (ReEnc). Anyone can change a second-level ciphertext for A into a first-level ciphertext for B by evaluating $[\![m]\!]_B = \mathsf{ReEnc}([m]_A, \pi_{A\rightarrow B}) = (Z^{b_2 k'}, mZ^{k'})$, where $Z^{b_2 k'} = Z^{b_2 a_1 k} = e(g^k, \pi_{A\rightarrow B})$ and $mZ^{k'} = mZ^{a_1 k}$ (the second part of $[m]_a$);

Decryption (Dec). A first level ciphertext $[\![m]\!]_A = (\alpha, \beta)$ can be decrypted with the secret key $a_i \in sk_A$ by computing $m = \mathsf{Dec}_1([\![m]\!]_A, sk_A) = \beta/\alpha^{1/a_i}$, where $i = 1$ if the ciphertext is obtained by first level encryption, while $i = 2$ if the ciphertext is obtained by re-encryption; a second level ciphertext $[m]_A = (\alpha, \beta)$ is decrypted with the secret key $a_1 \in sk_A$, by computing $m = \mathsf{Dec}_2([m]_A, sk_A) = \beta/e(\alpha, g)^{a_1}$.

2.3 Garbled Circuit

First proposed in the seminal work of Yao [66, 67], Garbled Circuit (GC) protocols allow two parties to jointly evaluate any boolean circuit, on their respective inputs, while protecting them from each other. Communication and computational overhead of such protocols depend on input bit length and circuit size. As outlined in [38, 42], a GC protocol comprises three sub-routines: circuit garbling, data exchange, and evaluation (see Fig. 2).

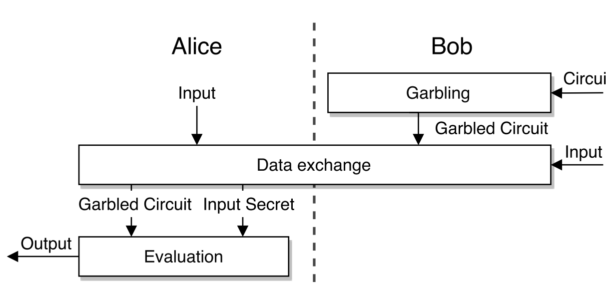

Fig. 2. Garbled Circuits scheme.

At a high level, a GC protocol works as follows. First, a party, say Bob, creates a boolean circuit, which represents the final function to be (securely) computed. Then, Bob "garbles" gates and wires composing the circuit, and transmits the garbled circuit, together with the secrets relative to his inputs, to another party, say Alice. The latter obtains secrets associated to her inputs from Bob through Oblivious Transfer (OT) protocol (OT) [27] and evaluates the garbled circuit to obtain the final result.

3 SYSTEM MODEL AND ASSUMPTIONS

We consider a network of devices, which we formally describe as an undirected graph $\mathcal{G}$, whose vertices are the agents (or nodes), and edges are the available communication links, hence each node i in $\mathcal{G}$ can communicate only with nodes in the set of his neighbours $\mathcal{N}_i \subseteq [1, 2, \ldots, N]$. Furthermore, we denote the adjacency matrix associated to the network graph as $\mathbf{A}$, with $\{A\}_{ij} = 1$ if $j \in \mathcal{N}_i$, and $\{A\}_{ij} = 0$ otherwise. We assume that $j \in \mathcal{N}_i$ if and only if $i \in \mathcal{N}_j$; as a result, A is symmetric, i.e., $\mathbf{A} = \mathbf{A}^T$. Note that, in dynamic networks, the set of neighbours $\mathcal{N}_i(\tau)$ of a generic agent i may change over time, and therefore also the adjacency matrix $\mathbf{A}(\tau)$. In order to simplify the exposition, in what follows we consider a static network setting. However the protocol described in Section 4 can be applied to dynamic networks.

Using the adjacency matrix, we define a random averaging consensus matrix $\mathbf{W}(\tau)$, at time τ; agents update their state $\mathbf{y}(\tau)$, based on their previous state $\mathbf{y}(\tau-1)$, and on $\mathbf{W}(\tau)$, according to the iterative rule: $\mathbf{y}(\tau) = \mathbf{W}(\tau)\mathbf{y}(\tau-1)$, where the initial state is given by the local measures $\mathbf{y}(0) = [\mathcal{L}(x_1), \mathcal{L}(x_2), \ldots, \mathcal{L}(x_N)]^T$.

The consensus procedure has interesting properties. In particular, under mild conditions (e.g., low connectivity of the network graph), the convergence is guaranteed to the average of the initial values, i.e. $\lim_{\tau\to\inf} y_i(\tau) = \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(x_i)\ \forall i = 1, \ldots, N$ (see details in [14, 51]). After a given number of steps T, each agent is interested in computing the binary statistical decision $\mathcal{D}_i(T) \in \{\mathcal{H}_0, \mathcal{H}_1\}$ given by the test $y_i(T) \gtrless^{\mathcal{H}_1}_{\mathcal{H}_0} thr_i$, where thr_i is a threshold chosen by agent i.

For simplicity this work focuses on the *randomized gossip algorithm* [14, 23] where at each consensus step τ a pair of adjacent nodes, say agents i and j, is randomly selected according to the network graph to perform an update. Fig. 3 shows a possible sequence of update steps in a consensus networks. While this solution is unpractical because it needs a third party supervising the choice of communicating agents, it can be used to simply model real scenarios where each agent, after having finished an update step, wait for a period and then contact an adjacent agent for the next update, as shown in Fig. 4. If two updates between two couples of distinct agents are run in parallel, they can be seen as performed in sequence in the model. In order for a real implementation to succeed, two agents must not start an update if at least one of the two is still involved in another update. Moreover, after that agents i and j have performed an update, we avoid that they start another useless update together until one of them has updated his status with a third agent. Real strategies related to the waiting time between two updates of the same agent and the choice of the node to communicate with are not in the scope of this paper.

The agents involved in step τ, exchange their information and update their states by the averaging rule [14]

$$y_i(\tau) = y_j(\tau) = \frac{y_i(\tau-1) + y_j(\tau-1)}{2}, \tag{1}$$

while the other agents hold their previous value $y_l(\tau) = y_l(\tau-1), \forall l \neq i, j$.

Note that, as outlined in Section 2.1, we work with integer numbers. We represent each agent state y_i as a ratio between a numerator n_i and a denominator d_i. In this way, we avoid division, which would cause a loss of information. Hence, given $y_i(\tau-1) = n_i(\tau-1)/d_i(\tau-1)$ and $y_j(\tau-1) = n_j(\tau-1)/d_j(\tau-1)$ and computed the least common multiplier $lcm(\tau-1)$ between $d_i(\tau-1)$ and

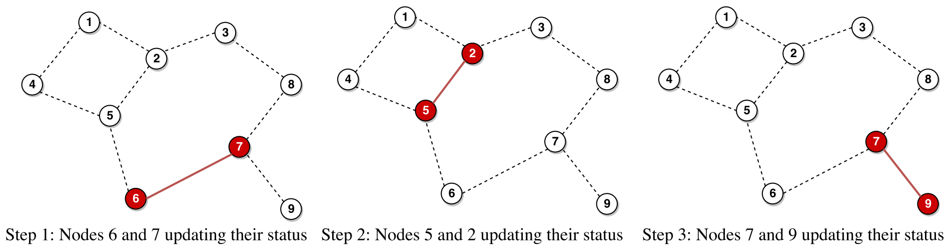

Fig. 3. Example of consensus network and possible first update steps.

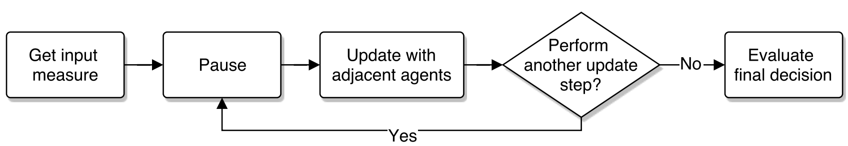

Fig. 4. Sequence of operations performed by an agent to implement the consensus and get the final decision.

$d_j(\tau-1)$, agents update their state by computing:

$$\begin{aligned} n_i(\tau) &= n_j(\tau) = n_i(\tau-1)\frac{lcm(\tau-1)}{d_i(\tau-1)} + n_j(\tau-1)\frac{lcm(\tau-1)}{d_j(\tau-1)}, \\ d_i(\tau) &= d_j(\tau) = 2\,lcm(\tau-1). \end{aligned} \tag{2}$$

We underline that the numerator carries information related to the inputs, but the denominator only depends on the number of steps performed; therefore it is not necessary to keep it secret. Moreover, since $d_i(0) = 1\ \forall i$, then $lcm(0) = 1$ and $d_i(1) = 2$. During the computation, one can easily infer that $d_i(\tau-1)$ are powers of $2\ \forall i, \tau$[4], and then as a consequence each least common multiplier can be computed as

$$lcm(\tau-1) = \max\{d_i(\tau-1), d_j(\tau-1)\}. \tag{3}$$

In the final step T, each agent i evaluates the comparison with his threshold thr_i, with the help of an adjacent agent. The choice of when performing the final step (after a given interval from the protocol starts, a given number of updates, etc.) is out of the scope of this paper.

Security Model. Throughout the paper, we consider non-colluding agents operating in the semi-honest security model. In practice, all the parties involved follow the protocol without deviating from it, but try to infer as much as possible from their observations, without interacting with other agents, except for the operations described in the protocol. We consider protection against external (network) adversaries out of the scope of this work. We do not consider attacks such as message manipulation, or fake message injection, which target message integrity and authenticity. However, each device could make use of shared or pairwise keys and apply signatures or message authentication codes (e.g., HMAC), in order to protect against such attacks.

4 ODIN: OUR PRIVACY-PRESERVING CONSENSUS PROTOCOL

In what follows, we present the details of ODIN, our novel solution for privacy-preserving decentralized consensus. Before proceeding with the description of our proposal, we highlight that

[4]The protocol can be optimized by using the logarithm of the denominators, but to make the paper more readable, we avoid this optimization.

ODIN presents a completely innovative algorithm w.r.t. [44], which shares with ODIN only system model and protocol goals. In fact ODIN replaces homomorphic encryption with simple additive blinding, reducing the complexity of the protocol operations, but still guaranteeing that nodes do not have access to any plain numerators (except their own inputs at $\tau = 0$). Moreover, by using proxy re-encryption, agents are able to operate in further steps with other nodes with constant communication and computation complexity, as we show in Section 5.2.

ODIN comprises three main phases: a *setup phase* (Section 4.1), which has the purpose of generating all the parameters to configure the consensus network; an *update phase* (Section 4.2), which involves the computation of multiple status update steps between pair of agents in the network; and a *decision phase* (Section 4.3), which allows each agent in the network to reach a decision, in a privacy preserving way.

4.1 Setup phase

We assume each node i in the network owns a proxy re-encryption key pair pk_i, sk_i, public and secret, respectively. Each agent propagates its public key, and other public keys it received, together with some additional information identifying the owner node, to all the adjacent nodes, until each node gets the public keys of all the nodes in the network. Once obtained all the public keys, each node i generates all the re-encryption keys $\pi_{i \to j}$ $\forall j \in N, j \neq i$ and distributes to any adjacent node j the re-encryption keys $\pi_{i \to k}$ $\forall k \in \mathcal{N}_j, k \neq i$. In a dynamic network, agent i provides re-encryption keys to any other node in the network to each neighbour, and shares re-encryption keys through the network to non adjacent nodes, encrypted with the public keys of the recipients. Whether a new agent joins the network, the whole networks needs to be updated, by sharing his public key and generating and distributing all the re-encryption keys to and from him.

The procedure can be simplified if a semi-honest third party participates to the setup phase, taking care to the distribution of public and re-encryption keys. In the case a trusted party is available, he can generates all the public, secret and re-encryption keys and then distribute them to the network.

4.2 Update phase

We now describe the protocol implementing a generic update phase, shown in Fig. 5, focusing on a step τ involving two agents i and j. We assume that at time $\tau - 1$ i and j performed their previous status updates with nodes k and l respectively.

At the beginning of step τ, agent i owns $n_i(\tau - 1) + s_k(\tau - 1)$, i.e. his numerator obfuscated by a random value[5] generated by node k, the denominator $d_i(\tau - 1)$ and the encryption $[s_k(\tau - 1)]_k$ of the random value chosen by k under the public key pk_k. Similarly agent j owns $n_j(\tau - 1) + s_l(\tau - 1)$, $[s_l(\tau - 1)]_l$ and $d_j(\tau - 1)$. At the end of the step agent i obtains $n_i(\tau) + s_j(\tau)$, $[s_j(\tau)]_j$ and $d_i(\tau)$, while agent j obtains $n_j(\tau) + s_i(\tau)$, $[s_i(\tau)]_i$ and $d_j(\tau)$. Any other agent h $\forall h \in N,\ h \neq i, j$ simply sets the new status equal to the previous one. Note that, in the case agent i (or similarly j) has not yet updated his status, he inputs $n_i(\tau - 1) = y_i(0)$ and $d_i(\tau - 1) = 1$. The following protocol can be simply adapted by considering the masking value $s_k(\tau - 1) = 0$. We describe the activities carried out by agent i (agent j follows the same protocol in parallel).

Part 1. Agent i generates two new random i.i.d. values $r_i(\tau)$ and $s_i(\tau)$ in $\mathbb{Z}_n$ where n must be equal or lower than the order q of the re-encryption scheme. The value $r_i(\tau)$ is used to add an additional mask to the numerator, obtaining $n_i(\tau - 1) + s_k(\tau - 1) + r_i(\tau) \mod n$. The mask added by agent k is re-encrypted as: $[\![s_k(\tau - 1)]\!]_j \leftarrow \mathsf{ReEnc}([s_k(\tau - 1)]_k, \pi_{k \to j})$, so that agent j can be able to decrypt it.

[5] We assume that all the random values are independent and identically distributed (i.i.d.).

At this point agent i transmits $n_i(\tau-1)+s_k(\tau-1)+r_i(\tau)$, $[\![s_k(\tau-1)]\!]_j$, $d_i(\tau-1)$ to agent j and receives $n_j(\tau-1)+s_l(\tau-1)+r_j(\tau)$, $[\![s_l(\tau-1)]\!]_i$, $d_j(\tau-1)$ from j.

Part 2. Agent i decrypts the first-level ciphertext $[\![s_l(\tau-1)]\!]_i$ by using his secret key and removes the value by the numerator received, obtaining $n_j(\tau-1)+r_j(\tau) \mod n$, yet obfuscated by the random value chosen by agent j. Then it computes the least common multiplier $lcm(\tau-1)$ between his denominator and the one received by agent j. By reminding that denominators are always powers of 2, the least common denominator is computed as in Eq. (3) and is used to update the numerator and denominator. Considering that $n_i(\tau)=n_j(\tau)$ and that agent i cannot remove the mask currently applied to the numerator, he computes the numerator for agent j, masked by a term $obf_j(\tau)$ that agent j can remove. Moreover agent i adds the random value $s_i(\tau)$, so that at the end

	Agent i	**Agent j**
	Input: $n_i(\tau-1)+s_k(\tau-1)$, $[s_k(\tau-1)]_k$, $d_i(\tau-1)$	*Input:* $n_j(\tau-1)+s_l(\tau-1)$, $[s_l(\tau-1)]_l$, $d_j(\tau-1)$
1	$r_i(\tau), s_i(\tau) \leftarrow_R \mathbb{Z}_N$;	$r_j(\tau), s_j(\tau) \leftarrow_R \mathbb{Z}_N$;
	Compute $n_i(\tau-1)+s_k(\tau-1)+r_i(\tau)$;	Compute $n_j(\tau-1)+s_l(\tau-1)+r_j(\tau)$;
	$[\![\ s_k(\tau-1)\ ]\!]_j \leftarrow \texttt{ReEnc}([s_k(\tau-1)]_k, \pi_{k\to j})$;	$[\![\ s_l(\tau-1)\ ]\!]_i \leftarrow \texttt{ReEnc}([s_l(\tau-1)]_l, \pi_{l\to i})$;
	$\xrightarrow{n_i(\tau-1)+s_k(\tau-1)+r_i(\tau),\ [\![\ s_k(\tau-1)\ ]\!]_j,\ d_i(\tau-1)}$	
		$\xleftarrow{n_j(\tau-1)+s_l(\tau-1)+r_j(\tau),\ [\![\ s_l(\tau-1)\ ]\!]_i,\ d_j(\tau-1)}$
2	$s_l(\tau-1) \leftarrow \texttt{Dec}([\![\ s_l(\tau-1)\ ]\!]_i, ppk_i)$;	$s_k(\tau-1) \leftarrow \texttt{Dec}([\![\ s_k(\tau-1)\ ]\!]_j, ppk_j)$;
	Compute $n_j(\tau-1)+r_j(\tau)$;	Compute $n_i(\tau-1)+r_i(\tau)$;
	$lcm(\tau-1) \leftarrow \max(d_i(\tau-1), d_j(\tau-1))$;	$lcm(\tau-1) \leftarrow \max(d_i(\tau-1), d_j(\tau-1))$;
	$d_i(\tau) \leftarrow 2lcm(\tau-1)$;	$d_j(\tau) \leftarrow 2lcm(\tau-1)$;
	$obf_j(\tau) = s_k(\tau-1)\frac{lcm(\tau-1)}{d_i(\tau-1)} + r_j(\tau)\frac{lcm(\tau-1)}{d_j(\tau-1)}$;	$obf_i(\tau) = s_l(\tau-1)\frac{lcm(\tau-1)}{d_j(\tau-1)} + r_i(\tau)\frac{lcm(\tau-1)}{d_i(\tau-1)}$;
	Compute $n_j(\tau)+obf_j(\tau)+s_i(\tau)$; // as in Eq. (5)	Compute $n_i(\tau)+obf_i(\tau)+s_j(\tau)$; // as in Eq. (5)
	$[s_i(\tau)]_i \leftarrow \texttt{Enc}_2(s_i(\tau), ppk_i)$;	$[s_j(\tau)]_j \leftarrow \texttt{Enc}_2(s_j(\tau), ppk_j)$;
	$\xrightarrow{n_j(\tau)+obf_j(\tau)+s_i(\tau),\ [s_i(\tau)]_i}$	
		$\xleftarrow{n_i(\tau)+obf_i(\tau)+s_j(\tau),\ [s_j(\tau)]_j}$
3	Compute $obf_i(\tau)$;	Compute $obf_j(\tau)$;
	Compute $n_i(\tau)+s_j(\tau)$;	Compute $n_j(\tau)+s_i(\tau)$;
	Output: $n_i(\tau)+s_j(\tau)$, $[s_j(\tau)]_j$, $d_i(\tau)$;	*Output:* $n_j(\tau)+s_i(\tau-1)$, $[s_i(\tau)]_i$, $d_j(\tau)$;

Fig. 5. Step τ of ODIN's update phase involving agents i and j. We assume i and j previously updated their status with two different agents k and l, respectively. All the operations are performed in $\mathbb{Z}_N$. Outputs of other agents at step τ are equal to their own inputs.

agent j is not able to obtain the plain value of $n_j(\tau)$:

$$\begin{aligned}
n_j(\tau) \quad &+ \quad obf_j(\tau) + s_i(\tau) = \\
&= \quad (n_i(\tau-1) + s_k(\tau-1))\frac{lcm(\tau-1)}{d_i(\tau-1)} + (n_j(\tau-1) + r_j(\tau))\frac{lcm(\tau-1)}{d_j(\tau-1)} + s_i(\tau) \mod n \\
&= \quad \underbrace{n_i(\tau-1)\frac{lcm(\tau-1)}{d_i(\tau-1)} + n_j(\tau-1)\frac{lcm(\tau-1)}{d_j(\tau-1)}}_{n_j(\tau)} \\
&\quad + \underbrace{s_k(\tau-1)\frac{lcm(\tau-1)}{d_i(\tau-1)} + r_j(\tau)\frac{lcm(\tau-1)}{d_j(\tau-1)}}_{obf_j(\tau)} + s_i(\tau) \mod n,
\end{aligned} \tag{4}$$

where we can easily observe that $\frac{lcm(\tau-1)}{d_i(\tau-1)}$ and $\frac{lcm(\tau-1)}{d_i(\tau-1)}$ are integer numbers (powers of 2), while $obf_j(\tau)$ is composed by terms that agents j knows. At this point agent i computes the denominator $d_i(\tau)$ as in Eq. (2) and computes the second level encryption of $s_i(\tau)$ with his public key, as $[s_i(\tau)]_i = \mathsf{Enc}_2(s_i(\tau), pk_i)$, allowing future re-encryption.

The obfuscated numerator $n_j(\tau) + obf_j(\tau) + s_i(\tau)$ and the encrypted random value $[s_i(\tau)]_i$ are transmitted to agent j, while agent i receives $n_i(\tau) + obf_i(\tau) + s_j(\tau)$ and $[s_j(\tau)]_j$ from j.

Part 3. Agent i is able to compute $obf_i(\tau) = s_l(\tau-1)\frac{lcm(\tau-1)}{d_j(\tau-1)} + r_i(\tau)\frac{lcm(\tau-1)}{d_i(\tau-1)} \mod n$ and remove it from the received obfuscated numerator, obtaining $n_i(\tau) + s_j(\tau) \mod n$.

Variant with data size reduction. Being the measures $y_i(0)$ mapped in the couple $n_i(0), d_i(0)$, where the first one is represented with ℓ bits and $d_i(0) = 1$, and considering that after each step the ratio $n_i(\tau)/d_i(\tau)$ represents an estimation of the average of all the sensors' measures, then $n_i(\tau)$ can be represented by using a number of bits equal to ℓ plus $\log_2 d_i(\tau) = \log_2 lcm(\tau-1) + 1 \leq \tau$. If the protocol runs for a big number of steps, there is the risk that $n_i(\tau)$ exceeds the modulus n. Therefore, it is necessary to reduce the numerator bitsize by dividing it (and also the denominator) by a given factor 2^k when the bitlength exceeds a previous established $\ell + \ell_1$ (a discussion is provided forward in the section). This can be easily performed by modifying protocol of step τ.

After computing $d_i(\tau) = d_j(\tau)$, if $\log_2 d_i(\tau) = \ell_1$, agents i and j decide to perform the data size reduction. For simplicity we describe only the operations performed by agent i, agent j follows the same protocol in parallel. Given the upperbound of the numerator $\max(n_i(\tau)) = \max(n_j(\tau)) = 2^{\ell+\ell_1}$ and given the statistical security parameter t (usually $t = 80$) used to guarantee statistic security, agents i randomly selects $s_i(\tau)$ in $\mathbb{Z}_{2^{\ell+\ell_1+t-k}}$, instead that in $\mathbb{Z}_n$, and right appends k zero bits to it, obtaining $s'_i(\tau) \leftarrow s_i(\tau)2^k \in \mathbb{Z}_{2^{\ell+\ell_1+t}}$. At this point agent i computes $n_j(\tau) + obf_j(\tau) + s'_i(\tau)$ and sends it to agent j together with the encryption of the random value $[s_i(\tau)]_i$, while receives $n_i(\tau) + obf_i(\tau) + s'_j(\tau)$ and $[s_j(\tau)]_j$ from agent j. Agents i computes $obf_i(\tau)$ as in Eq. (4), removes it from the numerator and divides it by 2^k (discards the k least significant bits) obtaining

$$\lfloor \frac{n_i(\tau) + s'_j(\tau)}{2^k} \rfloor = \lfloor \frac{n_i(\tau)}{2^k} \rfloor + s_j(\tau), \tag{5}$$

i.e., the updated numerator, blinded exactly by the value $s_j(\tau)$, received encrypted.

The protocol works correctly whether $n_i(\tau) + s'_j(\tau) < n$ (it does not exceed the modulus n), so that the protocol performs an integer division, not a modulus division. Given that $\max(n_i(\tau)) = 2^{\ell+\ell_1}$ and $s'_i(\tau) \in \mathbb{Z}_{2^{\ell+\ell_1+t}}$, their sum representation needs $\ell + \ell_1 + t + 1$ bits and ℓ_1 must be chosen so that $\ell_1 < \lfloor \log_2 n \rfloor - \ell - t - 1$.

As stated above, the modified update step of ODIN relies on statistical secrecy, where one party can deduce the value $n_i(\tau)$ with probability 2^{-t}. Moreover the k least significant bits (discarded by the division) are revealed to agents. If higher security is desired, it is possible to use the less efficient protocols described in [41, 63], involving homomorphic encryption or GCs.

4.3 Decision phase

The final goal of an agent i is to discover whether the value obtained by the consensus protocol is greater than a given threshold thr_i. Let suppose that at step T agent i has just updated his state together with agent j and is interested to evaluate $n_i(T)/d_i(T) \gtreqless thr_i$. For simplicity, in what follows we consider $<$ (note that, this does not affect the overall protocol complexity). Since agent i knows $d_i(T)$ and $n_i(T) + s_j(T)$, he can evaluate together with agent j a GC implementing $\big((n_i(T)+s_j(T))-s_j(T) \mod n\big) < thr_i * d_i(T)$ where agent i inputs $n_i(T)+s_j(T)$ and $thr_i * d_i(T)$, while agent j inputs $s_j(T)$. The circuit first removes the obfuscation $s_j(T)$, then compares the numerator with $thr_i * d_i(T)$. If also agent j is interested to evaluate the comparison with his threshold thr_j, it is not necessary that a second circuit is evaluated, but the above GC can be modified so that it also evaluates $n_i(T) < thr_j * d_j(T)$, where $thr_j * d_j(T)$ is input by agent j.

Despite its simplicity, the above circuit have some associated complexity, because the evaluation of the modular difference with GCs is expensive. Its implementation requires a subtractor that can return a negative result, to which n is added if the difference is negative. The circuit is hence composed by a subtractor implementing the first operation, an adder that adds n and finally a multiplexer that selects among their results, according the carry bit of the first operation. Considering that all the operations involve values represented with $\log_2 n$ bits, and that adders, subtractors and multiplexers require a non-xor gate for each input bit [39], we can easily observe that we would need to evaluate $3 \log_2 n$ non-XOR gates. However, to make the circuit efficient, it is sufficient that during the previous update step, agent j generates $s_j(T)$ in $\mathbb{Z}_{2^{\ell+\log_2 d_i(T)+t}}$ instead of in $\mathbb{Z}_n$, relying on statistical security. In such a way $n_i(T) + s_j(T) < n$ and GC needs only an integer subtractor that removes the obfuscation with only $\ell + \log_2 d_i(T)$ non-XOR gates. The final comparison is computed among two values, each of them represented with $\ell + \log_2 d_i(T)$ bits.

5 ANALYSIS

We now briefly discuss convergence (Section 5.1), complexity (Section 5.2), and security (Section 5.3) of ODIN.

5.1 Convergence

A commonly accepted choice of T can be based on the concept of ϵ-averaging time [14], i.e. the earliest gossip time in which the state vector $\mathbf{y}(\tau)$ is ϵ away from the normalized true average with probability greater than $1 - \epsilon$. A sufficiently small ϵ, which guarantees that all agents take the same decision with high probability, requires an average time $T(\epsilon) \leq \frac{3 \log \epsilon^{-1}}{\log \lambda_2(\mathbb{E}\mathbf{W})^{-1}}$, in terms of update steps, where $\mathbb{E}[\mathbf{W}]$ is the expected value operator of randomly selected averaging matrices $W(t)$ and $\lambda_2(\mathbb{E}\mathbf{W})$ is its second largest eigenvalue.

As demonstrated in [23], the topology of the network influences the consensus convergence, indeed the matrix $\mathbb{E}[\mathbf{W}]$ is completely specified by the network topology and the consensus protocol. Given that the average time needed for the convergence does not depend on the starting values provided by nodes, it is possible to estimate when the nodes have reached the consensus even without observing the exchanged messages as in ODIN algorithm. We further stress that ODIN only adds a privacy layer on top of the gossip protocol, and does not interfere with the properties of the consensus algorithm. For efficiency reasons, a wide part of the consensus literature is focused on how the consensus protocol (choice of consensus matrices) and the network topology can speedup the convergence. For instance, in [14] $\lambda_2(\mathbb{E}\mathbf{W})$ is minimized subject to the topology and the pairwise nature of the consensus protocol.

5.2 Complexity analysis

In what follows, we discuss computational and communication complexity of the update step of ODIN we introduced in Section 4.2, focusing on a generic step τ, and of the final decision phase introduced in Section 4.3.

Step τ complexity. Following step τ description in Section 4, we observe that agent i performs a modular addition to add $r_i(\tau)$ to the numerator; performs a proxy re-encryption on the value $[s_k(\tau-1)]_k$; transmits the masked numerator ($\lceil \log_2 n \rceil$ bits), the denominator ($\log_2 d_i(\tau-1) < \lceil \log_2 n \rceil$ bits) and 1 ciphertext to j, while receives masked ciphertexts, denominator and 1 ciphertext from him, with similar communication complexity; then he performs first-level decryption of the received ciphertext by using PrK_i; removes the obtained masked value with a modular subtraction and then evaluates eq. (4) that requires a modular product (one or both $\frac{lcm(\tau-1)}{d_i(\tau-1)}$ and $\frac{lcm(\tau-1)}{d_j(\tau-1)}$ are equal to 1) and 2 modular additions; encrypts $s_i(\tau)$ with its public key PuK_i; transmits 1 ciphertexts and the masked numerator ($\lceil \log_2 n \rceil$ bits) to j while receive 1 ciphertexts and the masked numerator from j; he computes $obf_i(\tau)$ with a modular product and a modular addition; and finally removes it from the masked numerator with another modular addition.

In total the agent i (and also agent j) performs 2 modular products, 6 modular additions, 1 re-encryption, 1 first level encryption and 1 second level encryption. The complexity of second level encryption and first level decryption in [5] mainly depends on 2 modular exponentiations, and 1 modular exponentiation, respectively, while re-encryption requires a pairing operation. From a communication point of view, the two agents involved in the computation in step k transmit 4 ciphertexts, 4 modular numbers and 2 integer numbers with variable size in 2 communication rounds. Ciphertexts are composed by 2 messages of prime order q; practical implementations of bilinear maps use elliptic curves for G_1, and elements in $\mathbb{Z}_{q^2}$ for G_2, hence ciphertext representation needs $4\lceil \log_2 q \rceil$ bits.

The complexity of step τ with data size reduction is really similar, because it needs only 2 additional integer divisions for each agent. However they have negligible complexity, since performed discarding the least significant k bits. Step τ complexities are summarized in Table 1, where modular addition (having negligible complexity respect the other operation) is overlooked.

Decision step complexity. The complexity of ODIN's final decision step, mainly depends on the use of a GC, which in turn depends on the number of its non-XOR gates composing the circuit. Each non-XOR gate has an associated garbled table, whose garbling and evaluation are performed by using 3 and 1 Hash functions, respectively (4 for each non-XOR gate in total) [38]. Garbled tables have size $3t$ bits each, where t is a security parameter (usually $t = 80$ bits) and are transmitted from the garbler to the evaluator. XOR gates have negligible computational and communication complexity. Secrets associated to the garbler's input bits (t bits each) are transmitted from the garbler to the evaluator, after having associated them to the input bits. Evaluator secret transmission involves Oblivious Transfer that associates the input bits to secrets chosen by the circuit garbler. Considering that OT can be precomputed [9], many OT's can be evaluated off-line on random values (regardless of the actual values used during the circuit evaluation) and resulting in a lower on-line communication complexity, only $\sim 2t$ bits for each input bit. Offline OT can be performed before the protocol starts or while two adjacent nodes waits to perform next updat step. Therefore the complexity of this "offline" calculation is not considered here.

Table 1. Complexity of each step for all the nodes involved.

Step	Computational complexity				Communication complexity	
	Modular Expo	Modular Prod	Bilinear Map	Hash	Bits	Rounds
τ	6	4	2	0	$< 4 \times 4\lceil \log_2 q \rceil + 6\lceil \log_2 n \rceil$	2
Final	0	0	0	$12(\ell + \lceil \log_2 d(T) \rceil)$	$15(\ell + \lceil \log_2 d(T) \rceil)t$	2

We assume that in the final step, agents i and j are evaluating together the GC and that both of them are interested to obtain the result of the comparison between the numerator and their respective thresholds. Both of two inputs values are represented with $\ell + \lceil \log_2 d(T) \rceil < \log_2 n$ bits (only the least $\ell + \lceil \log_2 d(T) \rceil$ bits of $n_i(T) + s_j(T)$ and $s_j(T)$ are necessary to remove the obfuscation, in the worst case $\lceil \log_2 n \rceil$ bits). Hence the association of the evaluator (let suppose agent i) inputs to secret values through OT requires the transmission of $2(\ell + \lceil \log_2 d(T) \rceil)(2t)$ bits, while the garbler (let suppose agent j) transmits the secrets associated to its input, i.e. $2(\ell + \lceil \log_2 d(T) \rceil)t$ bits. The circuit is composed by a subtracter and two comparison circuit, both of them having $\ell + \lceil \log_2 d(T) \rceil$ non-XOR gates, hence $3(\ell + \lceil \log_2 d(T) \rceil)3t$ bits are transmitted for the circuit and $4 \times 3(\ell + \lceil \log_2 d(T) \rceil)$ hash functions are evaluated in total. Complexities of the ODIN's final step are summarized in Table 1.

Note that, the overall complexity of ODIN is significantly less than the one in [44], which requires $2(|\mathcal{N}_i| + |\mathcal{N}_j|) + 8$ modular exponentiations, and $(|\mathcal{N}_i| + |\mathcal{N}_j| + 2)$ homomorphic ciphertexts to be transmitted, where $\mathcal{N}_i$ (resp. $\mathcal{N}_j$) is the number of nodes adjacent to agent i (resp. j). This makes ODIN really efficient, especially because its computational complexity *is independent from the number of adjacent nodes* (a real bottleneck in dense networks), and *is not affected by the dynamicity of the network*. On the other hand, however, the space complexity of ODIN is slightly bigger than the one in [44], due to the need for each agent to store not only the N public keys of all the nodes, but also the re-encryption keys between other nodes, i.e. $(N-1)(N-2)$ re-encryption keys. However, memory space can be provided at low cost, and this does not affect the power consumption. Firthermore, space complexity can be reduced in static networks, by storing each agent only the re-encryption keys among adjacent nodes.

5.3 Security discussion

Before discussing the security of ODIN, we briefly recall the security of its building blocks:

Proxy re-encryption: The security of the proxy re-encryption scheme in [5] relies on an extension of the Decisional Bilinear Diffie-Hellman (DBDH) assumption [13].

Garbled Circuit: Standard GC construction and execution using a secure OT protocol [9], are secure in the semi-honest model, as demonstrated in the multiple existing constructions and proofs in the literature (e.g., [48].

Additive blinding: Additive blinding is secure in an information-theoretical sense [10]. The masked message $y = x + r$ (i.e., the ciphertext) would provide no information about the original message x to a cryptanalyst with infinite computational power, when the mutual information [21] between ciphertext and plaintext is $I(x; y) = 0$ [59]. When our protocol performs modular operations in $\mathbb{Z}_n$ and any r is i.i.d. in $\mathbb{Z}_n$, additive blinding implements the Vernam system (also named one-time pad) [62], which guarantees perfect secrecy. Knowing that $x \in [0, M_x]$, if additive blinding is performed choosing $\mathbf{r}$ independently and uniformly distributed in the interval $[0, M_r]$, with $M_x + M_r < n$, a statistical blinding $y = x + r$, where has mutual information $I(X; Y) = \frac{M_x}{2M_r} \log e + o(\frac{1}{M_r})$ [10]. Hence if $M_x = 2^\ell$ and $M_r = 2^{\ell+t}$, $I(X; Y) \sim 2^{-t}$.

In ODIN scenario, a (non-colluding) p.p.t. adversary $\mathcal{A}$, in the semi-honest model, has the goal of disclosing his input and output values, and the ones of other nodes he interacts with. We formalize this goal as a security experiment $\mathbf{Exp}_{\mathcal{A}}$ between the adversary agent $\mathcal{A}$, the current (honest) agent j interacting with $\mathcal{A}$, and the previous agent k who interacted with $\mathcal{A}$. In this experiment, $\mathcal{A}$ interacts with j usign ODIN (Section 4), and, after a polinomial number of steps, outputs one or more of the values $< \hat{s}_k(\tau-1), \hat{s}_j(\tau), \hat{r}_j(\tau) >$, that can use to infer some input/output of the protocols.

We define the notion of security for a privacy-preserving consensus algorithm as:

Definition 5.1 (Security of a privacy-preserving consensus algorithm). A privacy-preserving consensus algorithm is said to be secure in the semi-honest model, and in presence of a non colluding p.p.t adversary, if $P[\hat{s}_k(\tau-1) = s_k(\tau-1) \mid \mathbf{Exp}_{\mathcal{A}}(1^\ell) = s'_k(\tau-1)]$, $P[\hat{s}_j(\tau) = s_j(\tau) \mid \mathbf{Exp}_{\mathcal{A}}(1^\ell) = \hat{s}_j(\tau)]$ and $P[\hat{r}_j(\tau) = r_j(\tau) \mid \mathbf{Exp}_{\mathcal{A}}(1^\ell) = \hat{r}_j(\tau)]$, are negligible in $\ell = f(\ell_q, \ell_n, \ell_t)$, where f is polinomial in ℓ_q, ℓ_n, and ℓ_t.

Theorem 5.2. *The privacy-preserving consensus algorithm construction of the protocol in Section 4 is secure according to Definition 5.1, if the adopted proxy re-encryption, garbled circuit, and additive blinding building blocks are secure.*

(Sketch) of Theorem 5.2. We start our proof sketch starting from the update phase of ODIN (Section 4.2), focusing on one update step. In this case, the goal of a (non colluding) p.p.t. adversary agent $\mathcal{A} = i$ is to disclose the values $n_i(\tau-1) = n_k(\tau-1)$, $n_j(\tau-1)$ or $n_i(\tau) = n_j(\tau)$. In order to do so, an adversary i that interacts with another (honest) agent j, must either obtain $s_k(\tau-1)$, $r_j(\tau)$ or $s_j(\tau)$. To obtain $s_k(\tau-1)$ (resp. $s_j(\tau)$), i can only try to: (1) decrypt the encrypted input value $[s_k(\tau-1)]_k$ (resp. $[s_k(\tau)]_j$); (2) re-encrypt $[s_k(\tau-1)]_k$ (resp. $[s_k(\tau)]_j$) so that he can decrypt it using sk_i; or (3) infer $s_k(\tau-1)$ (resp. $s_j(\tau)$ or $r_j(\tau)$) from the observed messages.

To be able to achieve (1), excluding the possibility for i to obtain the secret key sk_k of agent k (resp. sk_j), given the non-colluding nodes assumption, i could only attack the proxy re-encryption scheme as follows: i selects several values x in the set of values admissible values for $n_i(\tau-1)$ (resp. $n_i(\tau)$), and computes the encryption of $(n_i(\tau-1) + s_k(\tau-1)) - x$ (resp. $(n_i(\tau) + s_j(\tau)) - x$); then, i checks whether the result is equal to the given input value $[s_k(\tau-1)]_k$ (resp. $[s_j(\tau))]_j$). However, being proxy re-encryption scheme in [5] CPA-secure (thanks to its probabilistic properties), this turns out to be computationally unfeasible for any p.p.t. adversary, i.e., the probability of success is negligible in ℓ_q. Similarly, goal (2) is proven to be computationally unfeasible for a p.p.t. adversary, using the scheme in [5]. In fact if setup has been correctly run, i neither possess $\pi_{k\to i}$ or $\pi_{j\to i}$, nor he can generate them, due to the non-transitive and non-interactive properties of the proxy re-encryption scheme in [5]. Finally, a p.p.t. adversary i cannot achieve goal (3) since both $s_k(\tau-1)$, $s_j(\tau)$ and $r_j(\tau)$ are always added to a numerator (i.e., his final objective). Adversary i could try to remove numerators by performing some linear combination of the observed messages. However we can easily see that any linear combination of messages contains the sum of at least 2 values unknown to i. We can easily infer that a p.p.t. adversary i can try to decrypt numerators by picking random values $< \hat{s}_k(\tau-1), \hat{s}_j(\tau), \hat{r}_j(\tau) >$. However they will be equal to $< s_k(\tau-1), s_j(\tau), r_j(\tau) >$ with negligible probability and the attacker is not able to understand if this happens.

Note that, a similar analysis can be carried out to assert the security of step τ with data size reduction. In this case, we use statistical security, but having $t = 80$ guarantees a low mutual information between numerators and their relative ciphertexts.

Finally, the decision phase of ODIN (Section 4.3) can be considered secure, since it relies on both a GC protocol and additive blinding, which both have been prooven secure against a semi-honest non-colluding p.p.t. adversary. □

(a) Raspberry Pi 1 Mod B (b) Raspberry Pi 3 (c) USB Power Monitor V2

Fig. 6. Setting used for evaluation.

6 PROTOTYPE IMPLEMENTATION AND EVALUATION

In this section, we briefly present an evaluation of a proof-of-concept implementation of ODIN on commodity IoT devices, in order to show its practicability. We run our implementation on a Raspberry Pi 1 Mod B, equipped with a 700 MHz ARM CPU, and 512 MB RAM, and a more recent Raspberry Pi 3, equipped with a 1.2 GHz quad-core ARM Cortex-A53 CPU, and 1 GB RAM (see Fig. 6); both devices run Raspbian Jessie Lite OS, with kernel v4.4. These devices represent a typical example of low cost (the latest model, Raspberry Pi mod 3, can be found at < 35$[6]) and wide spread IoT boards.

Our code relies on a proxy re-encryption library that implements the scheme in [5][7], which is in turn based on the MIRACL Cryptographic SDK[8]. We used the MIRACL library to implement also the simple modular arithmetic operations performed in our protocol. We consider runtime performance, approximate energy consumption (obtained as the average power, which we measured with a USB Power Monitor V2 device shown in Fig. 6, multiplied by the execution time), and communication overhead, of the update step of ODIN (Section 4.2) and of its variant with data size reduction. Furthermore, we measured runtime and approximate energy consumption of the decision phase of ODIN (Section 4.3).

Runtime and energy consumption. In order to provide an estimate of the overall runtime and energy consumption of ODIN, we benchmarked the operations involving cryptographic primitives, which dominate the overall performance. Results are summarized in Table 2. Apart from data transmissions, we estimate the runtime of one update step of ODIN, and of its variant with data size reduction, based on our complexity analysis in Section 5.2, and on the measurements in Table 2. The overall runtime of the update step of ODIN is 111.66 ms, with an energy consumption of 241.19 mJ, on a Raspberry Pi 1 Mod B; similarly, the runtime on a Raspberry Pi 3 can be approximated as 41.70 ms, with an energy consumption of 88.40 mJ. We further run ODIN on both devices and benchmarked the Update Phase. The results of this evaluation are shown in Table 3, divided into three parts. The variant with data size reduction in Section 4 shows a similar complexity, adding only two divisions. Note that, as previously mentioned in Section 5.2, our divisions involves only powers of 2, i.e., are performed by discarding less significant bits; this introduces a negligible complexity on the device, and therefore, we did not consider it in our evaluation. Similarly, as computing *lcm* is merely a comparison between two big integers, we did not include it in our evaluation.

[6]https://www.raspberrypi.org/blog/raspberry-pi-3-on-sale/
[7]https://isi.jhu.edu/~mgreen/prl/index.html
[8]https://github.com/miracl/MIRACL

The final decision step of ODIN involves the evaluation of a GC, which, according to our complexity analysis in Section 5.2, in the worst case involves $3 \times 4 \times \log_2 q$ operations (as $\ell + \lceil \log_2 d(T) \rceil < \log_2 n \leq \log_2 q$). Using a 256 bit representation for q, and considering SHA-1 as a cryptographic primitive, we can estimate the overall runtime as 25.80 ms, and the associated energy consumption as 55.73 mJ on a Raspberry Pi 1 Mod B, and 19.05 ms runtime and 40.39 mJ energy consumption on a Raspberry Pi 3.

Communication overhead. During one update step of ODIN (Section 4.2), each node generates messages of different size. In our prototype implementation, we performed message "serialization" leveraging the serialization routines provided by the MIRACL library, as well as from the proxy re-encryption library in use. The result is that in one update step of out protocol, nodes generate (and receive) messages of size 140 B, and 72 B. These messages are quite small, and can be efficiently exchanged even over channels with low data rate. Note that, the actual impact of such messages on the total transmission overhead highly depends on nodes deployment, protocol in use, and the physical antenna used for wireless communication. As an example, we consider the use of 6lowPAN protocol, which provides an adaptation layer to allow the use of UDP and IPv6 on top of the 802.15.4 protocol, which is widely used in the IoT domain [31]. In the simplest case, i.e., where two devices have local link addresses as in our case, in a 127 B frame we can use up to 108 B of payload [32]. Therefore, sending the first message in the update step translates into sending two 127 B frames, while the other messages simply fit into a single link layer frame. These results confirm the low impact our approach has on the overall transmission cost, which makes it particularly suitable for low power devices.

Simulation. For a better understanding of the feasibility of ODIN on large networks of IoT devices, similarly to [2, 3], we performed a set of simulations using Omnet++[9]. In our simulations, we generated random networks of nodes of different size and density; nodes are placed randomly in an area of 100 m^2, and connected to neighbors within a range of 10 m through links simulating the IEEE 802.14.5 protocol (according to the parameters in [60]). We simulated the execution of ODIN on Rasperry Pi Mod 1 devices using delays, i.e., using the ones in Table 2 and Table 3. We considered different values for ϵ, from 0.05 to 0.01 with steps of 0.01, and networks of size 400, 500,

[9] https://omnetpp.org/

Table 2. Runtime and energy consumption of the operations performed in ODIN; measurements taken from a Raspberry Pi 1 Mod B and a Raspberry Pi 3

Operation	**Raspberry Pi 1 Mod B**		**Raspberry Pi 3**	
	Runtime (ms)	**Energy (mJ)**	**Runtime (ms)**	**Energy (mJ)**
Sum in $\mathbb{Z}_n$	0.0069	0.0149	0.0037	0.0078
Sub in $\mathbb{Z}_n$	0.0054	0.0117	0.0033	0.0070
Mul in $\mathbb{Z}_n$	0.0208	0.0449	0.0138	0.0293
SHA-1	0.0084	0.0181	0.0062	0.0131
Enc_2	29.7899	64.3462	11.4394	24.2515
ReEnc	68.5674	148.1056	25.1504	53.3188
Dec_1	13.2256	28.5673	5.0657	10.7393

Table 3. Runtime and energy consumption of the Update phase of ODIN (Section 4.2); measurements taken from a Raspberry Pi 1 Mod B and a Raspberry Pi 3

Part	Raspberry Pi 1 Mod B		Raspberry Pi 3	
	Runtime (ms)	**Energy (mJ)**	**Runtime (ms)**	**Energy (mJ)**
Part 1	77.3010	166.9702	25.9983	55.1164
Part 2	47.6750	102.9780	16.9280	35.8874
Part 3	0.7400	1.5984	0.3147	0.6672
TOT:	125.72	271.56	43.24	91.67

and 600. For each generated random topology, we computed the necessary number of iterations T to reach a consensus according to the results in [14] (see Section 5.1). Results are reported in Fig. 7. Each reported value is the average of 100 executions. We can observe that ODIN execution needs no more than 18 s in the networks generated. Our simulations show encouraging results, suggesting that a consensus can be reached even in large networks of hundreds of nodes, in a small amount of time.

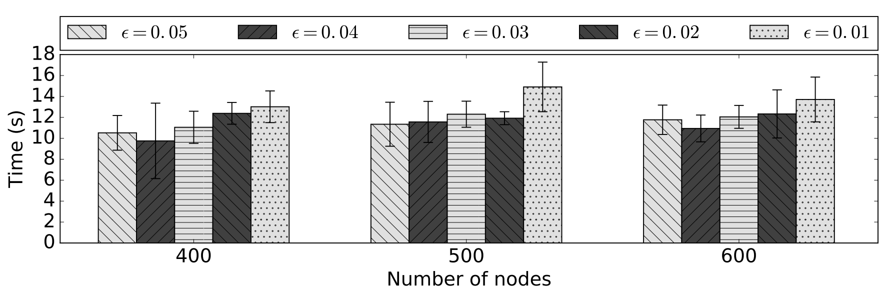

Fig. 7. ODIN runtime varying number of nodes (in an area of 100 m^2) and ϵ.

7 CONCLUSIONS

This paper proposes ODIN, an innovative mechanism to fuse private information through a secure extension of the consensus algorithm. ODIN is based on the randomized gossip algorithm, in which a pair of agents participate in data exchange in each time frame, and is secure against non-colluding semi-honest nodes. We believe our construction represents an important step forward towards the implementation of efficient and useful algorithms, able to take decisions based on the average consensus of private inputs. As demonstrated through practical tests, ODIN is efficient also on low power devices, and IoT networks can reach the consensus in reasonable time, even without having access to plain values. This opens the way to its application in distributed and dynamic urban networks, and to the IoT in general. Tests have been performed in static networks to validate protocol performances, but can be easily extended to dynamic networks. The duality between plain and privacy-preserving consensus networks guarantees that the secure implementation can reach the consensus as the plain implementation does, with a small delay. On the other side variation of the protocol should be studied to provide a privacy preserving consensus also in consensus networks with time varying state, where sensor observations change during the protocol evaluation.

Despite its simple security model, ODIN can already be applied to some IoT scenarios, such as smart metering systems. Future work has the goal of relaxing the "non-colluding" assumption, to allow the use of privacy-preserving consensus protocols in a larger set of scenarios. Guaranteeing security and correctness against misbehaving nodes will be one of the most difficult steps in future research. We highlight that malicious nodes can modify the messages exchanged during step τ, so that the network reach a wrong consensus. This is a problem of difficult solution also in the plain domain where some defense strategies have been proposed [45, 53, 54, 65, 68]. For this reason the problem could have no efficient solution in a privacy preserving consensus networks, where exchanged messages cannot be distinguished from random values. However the possibility to extend the protocol to malicious users deserves to be exploited. A possible solution can rely on proxy re-Zero-Knowledge Proof of knowledge (proxy re-ZKP) schemes [12].

Future work also includes the application to practical urban scenarios, such as smart meter data fusion, object tracking or vehicle coordination, as well as the analysis of the impact of network topology to the performances of the secure consensus algorithm. We finally underline that the protocol could represent the basis for privacy-preserving protocols in other relevant (non-IoT) domains. For example it can be used to detect replicas in Big Data storage while protecting the privacy of users, reduce contents excess in networks that use Information-Centric Networking protocols, without disclosing the content of single caches, or evaluate the quality of contents shared in peer-to-peer networks while protecting the single user evaluations.

REFERENCES

[1] Ian F Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. 2002. A Survey On Sensor Networks. *IEEE Commun. Mag.* 40, 8 (2002), 102–114.

[2] Moreno Ambrosin, Mauro Conti, Ahmad Ibrahim, Gregory Neven, Ahmad-Reza Sadeghi, and Matthias Schunter. 2016. SANA: Secure and Scalable Aggregate Network Attestation. In *CCS '16*. ACM, 731–742.

[3] N Asokan, Ferdinand Brasser, Ahmad Ibrahim, Ahmad-Reza Sadeghi, Matthias Schunter, Gene Tsudik, and Christian Wachsmann. 2015. SEDA: Scalable Embedded Device Attestation. In *CCS '15*. ACM, 964–975.

[4] Giuseppe Ateniese, Karyn Benson, and Susan Hohenberger. 2009. Key-private proxy re-encryption. In *CT-RSA '09*. Springer, 279–294.

[5] Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. 2006. Improved proxy re-encryption schemes with applications to secure distributed storage. *Transaction Inf. Syst. Secur.* 9, 1 (2006), 1–30.

[6] Mauro Barni, Giulia Droandi, and Riccardo Lazzeretti. 2015. Privacy Protection in Biometric-Based Recognition Systems: A Marriage Between Cryptography and Signal Processing. *IEEE Signal Process. Mag.* 32, 5 (2015), 66–76.

[7] Mauro Barni, Pierluigi Failla, Riccardo Lazzeretti, Ahmad-Reza Sadeghi, and Thomas Schneider. 2011. Privacy-preserving ECG Classification with Branching Programs and Neural Networks. *IEEE Trans. Inf. Forensics Security* 6, 2 (2011), 452–468.

[8] Giorgio Battistelli, Luigi Chisci, Claudio Fantacci, Alfonso Farina, and Antonio Graziano. 2013. Consensus CPHD Filter for Distributed Multitarget Tracking. *IEEE J. Sel. Topics Signal Process.* 7, 3 (2013), 508–520.

[9] Donald Beaver. 1995. Precomputing Oblivious Transfer. In *CRYPTO '95*. Springer, 97–109.

[10] Tiziano Bianchi, Alessandro Piva, and Mauro Barni. 2011. Analysis of the Security of Linear Blinding Techniques From an Information Theoretical Point of View. In *ICASSP '11*. IEEE, 5852–5855.

[11] Marina Blanton and Paolo Gasti. 2011. Secure and Efficient Protocols For Iris and Fingerprint Identification. In *ESORICS '11*. Springer, 190–209.

[12] Matt Blaze, Gerrit Bleumer, and Martin Strauss. 1998. Divertible protocols and atomic proxy cryptography. In *EUROCRYPT '98*. Springer, 127–144.

[13] Dan Boneh and Matt Franklin. 2001. Identity-based encryption from the Weil pairing. In *CRYPTO '01*. Springer, 213–229.

[14] Stephen Boyd, Arpita Ghosh, Balaji Prabhakar, and Devavrat Shah. 2006. Randomized Gossip Algorithms. *IEEE Trans. Inf. Theory* 52, 6 (2006), 2508–2530.

[15] Paolo Braca, Riccardo Lazzeretti, Stefano Marano, and Vincenzo Matta. 2016. Learning With Privacy in Consensus + Obfuscation. *IEEE Signal Processing Letters* 23, 9 (2016), 1174–1178.

[16] Paolo Braca, Stefano Marano, and Vincenzo Matta. 2008. Enforcing Consensus while Monitoring the Environment in Wireless Sensor Networks. *IEEE Trans. Signal Process.* 56, 7 (2008), 3375–3380.
[17] Paolo Braca, Stefano Marano, Vincent Matta, and Peter Willett. 2010. Asymptotic Optimality of Running Consensus in Testing Statistical Hypotheses. *IEEE Trans. Signal Process.* 58, 2 (2010), 814–825.
[18] Paolo Braca, Stefano Marano, Vincenzo Matta, and Peter Willett. 2011. Consensus-based Page's test in sensor networks. *Signal Processing* 91, 4 (2011), 919–930.
[19] Ran Canetti and Susan Hohenberger. 2007. Chosen-ciphertext Secure Proxy Re-encryption. In *CCS '14*. ACM, 185–194.
[20] Sherman SM Chow, Jian Weng, Yanjiang Yang, and Robert H Deng. 2010. Efficient Unidirectional Proxy Re-encryption. In *AFRICACRYPT '10*. Springer, 316–332.
[21] Thomas M Cover and Joy A Thomas. 2012. *Elements of information theory*. John Wiley & Sons.
[22] George Danezis, Cédric Fournet, Markulf Kohlweiss, and Santiago Zanella-Béguelin. 2013. Smart Meter Aggregation via Secret-Sharing. In *SEGS '13*. ACM, 75–80.
[23] Alexandros G Dimakis, Soummya Kar, José MF Moura, Michael G Rabbat, and Anna Scaglione. 2010. Gossip Algorithms for Distributed Signal Processing. *Proc. IEEE* 98, 11 (2010), 1847–1864.
[24] Zekeriya Erkin, Michael Beye, Thijs Veugen, and Reginald L Lagendijk. 2011. Efficiently computing private recommendations. In *ICASSP '11*. IEEE, 5864–5867.
[25] Zekeriya Erkin, Martin Franz, Jorge Guajardo, Stefan Katzenbeisser, Reginald L. Lagendijk, and Tomas Toft. 2009. Privacy-preserving face recognition. In *Privacy Enhancing Technologies*. Springer, 235–253.
[26] Zekeriya Erkin and Gene Tsudik. 2012. Private Computation of Spatial and Temporal Power Consumption with Smart Meters. In *ACNS '12*. Springer, 561–577.
[27] Shimon Even, Oded Goldreich, and Abraham Lempel. 1985. A Randomized Protocol for Signing Contracts. *Commun. ACM* 28, 6 (1985), 637–647.
[28] J Alexander Fax and Richard M Murray. 2004. Information flow and cooperative control of vehicle formations. *Transactions on Automatic Control* 49, 9 (2004), 1465–1476.
[29] David L. Hall and James Llinas. 1997. An Introduction to Multisensor Data Fusion. *Proc. IEEE* 85, 1 (1997), 6–23.
[30] Zhenqi Huang, Sayan Mitra, and Geir Dullerud. 2012. Differentially Private Iterative Synchronous Consensus. In *WPES '12*. ACM, 81–90.
[31] Jonathan W. Hui and Pascal Thubert. 2011. Compression format for IPv6 datagrams in low power and lossy networks (6LoWPAN). *RFC-6282* (2011).
[32] Isam Ishaq, David Carels, Girum K Teklemariam, Jeroen Hoebeke, Floris Van den Abeele, Eli De Poorter, Ingrid Moerman, and Piet Demeester. 2013. IETF standardization in the field of the internet of things (IoT): a survey. *Journal of Sensor and Actuator Networks* 2, 2 (2013), 235–287.
[33] Anca-Andreea Ivan and Yevgeniy Dodis. 2003. Proxy Cryptography Revisited. In *NDSS '03*. Internet Society.
[34] Ali Jadbabaie, Jie Lin, and A Stephen Morse. 2003. Coordination of groups of mobile autonomous agents using nearest neighbor rules. *Transactions on Automatic Control* 48, 6 (2003), 988–1001.
[35] Antoine Joux. 2000. A one round protocol for tripartite Diffie–Hellman. In *Algorithmic number theory*. Vol. 1838. Springer, 385–393.
[36] Soummya Kar and Jose M.F. Moura. 2013. Consensus + innovations distributed inference over networks: cooperation and sensing in networked systems. *IEEE Signal Process. Mag.* 30, 3 (2013), 99–109.
[37] Himanshu Khurana, Jin Heo, and Meenal Pant. 2006. From Proxy Encryption Primitives to a Deployable Secure-Mailing-List Solution. In *ICICS '06*. Vol. 4307. Springer, 260–281.
[38] Vladimir Kolesnikov, Ahmad-Reza Sadeghi, and Thomas Schneider. 2013. A systematic approach to practically efficient general two-party secure function evaluation protocols and their modular design. *Journal of Computer Security* 21, 2 (2013), 283–315.
[39] Vladimir Kolesnikov and Thomas Schneider. 2008. Improved garbled circuit: Free XOR gates and applications. In *International Colloquium on Automata, Languages, and Programming*. Springer, 486–498.
[40] Reginald L. Lagendijk, Zekeriya Erkin, and Mauro Barni. 2013. Encrypted Signal Processing for Privacy Protection: Conveying the Utility of Homomorphic Encryption and Multiparty Computation. *IEEE Signal Process. Mag.* 30, 1 (2013), 82–105.
[41] Riccardo Lazzeretti and Mauro Barni. 2011. Division Between Encrypted Integers by Means of Garbled Circuits. In *WIFS '11*. IEEE, 1–6.
[42] Riccardo Lazzeretti and Mauro Barni. 2013. Private Computing with Garbled Circuits. *IEEE Signal Process. Mag.* 30, 2 (2013), 123–127.
[43] Riccardo Lazzeretti, Jorge Guajardo, and Mauro Barni. 2012. Privacy preserving ECG Quality Evaluation. In *MM&Sec '12*. ACM, 165–174.

[44] Riccardo Lazzeretti, Sean Horn, Paolo Braca, and Peter Willett. 2014. Secure Multi-party Consensus Gossip Algorithms. In *ICASSP '14*. IEEE, 7406–7410.

[45] Heath J LeBlanc, Haotian Zhang, Xenofon Koutsoukos, and Suresh Sundaram. 2013. Resilient asymptotic consensus in robust networks. *IEEE J. Sel. Areas Commun.* 31, 4 (2013), 766–781.

[46] Benoît Libert and Damien Vergnaud. 2008. Unidirectional Chosen-Ciphertext Secure Proxy Re-encryption. In *PKC '08*. Springer, 360–379.

[47] Yehuda Lindell and Benny Pinkas. 2002. Privacy Preserving Data Mining. *Journal of Cryptology* 15, 3 (2002), 177–206.

[48] Yehuda Lindell and Benny Pinkas. 2009. A proof of security of YaoâĂŹs protocol for two-party computation. *Journal of Cryptology* 22, 2 (2009), 161–188.

[49] Nicolaos E Manitara and Christoforos N Hadjicostis. 2013. Privacy-preserving Asymptotic Average Consensus. In *ECC '13*. IEEE, 760–765.

[50] Reza Olfati-Saber. 2006. Flocking for multi-agent dynamic systems: Algorithms and theory. *Transactions on Automatic Control* 51, 3 (2006), 401–420.

[51] Reza Olfati-Saber, Alex Fax, and Richard M. Murray. 2007. Consensus and Cooperation in Networked Multi-Agent Systems. *Proc. IEEE* 95, 1 (Jan. 2007), 215–233.

[52] Pascal Paillier. 1999. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT '99*. 223–238.

[53] Fabio Pasqualetti, Antonio Bicchi, and Francesco Bullo. 2007. Distributed Intrusion Detection for Secure Consensus Computations. In *CDC '07*. IEEE, 5594–5599.

[54] Fabio Pasqualetti, Antonio Bicchi, and Francesco Bullo. 2012. Consensus Computation in Unreliable Networks: A System Theoretic Approach. *IEEE Trans. Autom. Control* 57, 1 (2012), 90–104.

[55] Wei Ren. 2007. Multi-vehicle consensus with a time-varying reference state. *Systems & Control Letters* 56, 7 (2007), 474–483.

[56] Matthew Roughan and Jon Arnold. 2007. Data Fusion Without Data Fusion: Localization and Tracking Without Sharing Sensitive Information. In *CDC '07*. IEEE, 136–141.

[57] Matthew Roughan and Jon Arnold. 2007. Multiple Target Localisation in Sensor Networks with Location Privacy. In *ESAS '07*. Springer, 116–128.

[58] Venkatesh Saligrama and David A Castanón. 2008. Reliable distributed estimation with intermittent communications. In *Networked Sensing Information and Control*. Springer, 245–265.

[59] Claude E Shannon. 1949. Communication theory of secrecy systems. *Bell system technical journal* 28, 4 (1949), 656–715.

[60] George Spanogiannopoulos, Natalija Vlajic, and Dusan Stevanovic. 2009. A simulation-based performance analysis of various multipath routing techniques in ZigBee sensor networks. In *ADHOCNETS '09*. Springer, 300–315.

[61] Gelareh Taban, Alvaro A Cárdenas, and Virgil D Gligor. 2006. Towards a secure and interoperable DRM architecture. In *ACM workshop on Digital rights management (DRM '06)*. ACM, 69–78.

[62] Gilbert S Vernam. 1926. Cipher printing telegraph systems: For secret wire and radio telegraphic communications. *Journal of the AIEE* 45, 2 (1926), 109–115.

[63] Thijs Veugen. 2010. Encrypted Integer Division. In *WIFS '10*. IEEE, 1–6.

[64] Ning Xiong and Per Svensson. 2002. Multi-sensor Management for Information Fusion: Issues and Approaches. *Information fusion* 3, 2 (2002), 163–186.

[65] Qiben Yan, Ming Li, Tingting Jiang, Wenjing Lou, and Y Thomas Hou. 2012. Vulnerability and Protection for Distributed Consensus-based Spectrum Sensing in Cognitive Radio Networks. In *INFOCOM '12*. IEEE, 900–908.

[66] Andrew Yao. 1982. Protocols for Secure Computations. In *FOCS '82*. IEEE.

[67] Andrew Yao. 1986. How to Generate and Exchange Secrets. In *FOCS '86*. IEEE.

[68] F Richard Yu, Helen Tang, Minyi Huang, Zhiqiang Li, and Peter C Mason. 2009. Defense against spectrum sensing data falsification attacks in mobile ad hoc networks with cognitive radios. In *MILCOM '09*. IEEE, 1–7.

[69] Shucheng Yu, Cong Wang, Kui Ren, and Wenjing Lou. 2010. Achieving Secure, Scalable, and Fine-grained Data Access Control in Cloud Computing. In *INFOCOM '10*. IEEE, 1–9.