PAG: Private and Accountable Gossip

Jérémie Decouchant, Sonia Ben Mokhtar, Albin Petit, Vivien Quéma

PAG: Private and Accountable Gossip

Jérémie Decouchant
SnT, University of Luxembourg
Luxembourg

Sonia Ben Mokhtar
CNRS LIRIS
Lyon, France

Albin Petit
INSA
Lyon, France

Vivien Quéma
Grenoble INP
Grenoble, France

***Abstract*—A large variety of content sharing applications rely, at least partially, on gossip-based dissemination protocols. However, these protocols are subject to various types of faults, among which selfish behaviours performed by nodes that benefit from the system without contributing their fair share to it. Accountability mechanisms (e.g., PeerReview, AVMs, FullReview), which require that nodes log their interactions with others and periodically inspect each others' logs are effective solutions to deter faults. However, these solutions require that nodes disclose the content of their logs, which may leak sensitive information about them. Building on a monitoring infrastructure and on homomorphic cryptographic procedures, we propose in this paper PAG, the first accountable and partially privacy-preserving gossip protocol. We assess PAG theoretically using the ProVerif cryptographic protocol verifier and evaluate it experimentally using both a real deployment on a cluster of 48 machines and simulations. The performance evaluation of PAG, performed using a video live streaming application, shows that it is compatible with the visualisation of live video content on commodity Internet connections. Furthermore, PAG's bandwidth consumption inherits the desirable scalability properties of gossip when the number of users in the system grows.**

I. Introduction

Peer-to-peer content sharing systems account for a large amount of traffic in today's Internet [1]. Examples of such systems that are widely used by thousands of users everyday include P2P content delivery networks (e.g., BitTorrent, eMule), on demand audio and video streaming (e.g., Popcorn-Time, Velocix [2]) and P2P TV (e.g., PPLive [3], LiveSky [4]). These systems often rely (totally or partially) on gossip-based message dissemination schemes [5], [6] where nodes periodically exchange data chunks with randomly chosen nodes.

Gossip-based systems are cost effective, scalable up to millions of users and highly resilient to churn. However, they rely on the willingness of the users to share their resources (e.g., upload bandwidth, CPU cycles, memory) with each other. Consequently, they are vulnerable to selfish behaviours performed by users that modify their software or use a tampered version of it (e.g., BitThief, BitTyrant).

Recent studies performed on real peer-to-peer systems (e.g., [7], [8]) have shown that clients behaving selfishly get better performance than compliant ones (e.g., in a live streaming system, they would download the video stream faster while saving upload bandwidth). Moreover, these studies show that above a given proportion of selfish clients, the compliant clients observe a major degradation in the quality of the video stream they obtain. Dealing with selfish behaviours has thus been the centre of active research in the last decade.

In this context, accountability protocols, which have recently been proposed for fault detection in distributed systems (e.g., [9]–[11]) are promising candidates for effectively dealing with selfish behaviours. In an accountable system, nodes register their interactions with each other in secure logs. These logs are periodically inspected by a set of nodes in the system acting as monitors. In case of fault detection, the monitors generate a proof of misbehaviour and the misbehaving nodes get punished. Thanks to their effectiveness, these protocols have been recently used to build selfish-resilient content dissemination systems (e.g., [12], [13]).

However, these solutions require nodes to share their log with each other, which may leak sensitive information about them. For instance, in a video streaming application a curious monitor inspecting a node's log may learn about the user's *interests*, which may disclose information such as her sex, age, sexual, political or religious preferences. Furthermore, monitors can infer further information by analysing *the interest graphs* made of links between nodes sharing similar interests.

In this context, a challenging objective is to build protocols that enforce accountability while preserving their users' privacy. However, this objective may seem contradictory as there is a clear trade-off between privacy and accountability: the deeper the verifications that can be performed regarding the behaviour of a node, the more faults can be deterred but the more information need to be collected.

Few protocols have been recently proposed to address both accountability and privacy issues in distributed systems. Among them Dissent [14] and RAC [15] allow nodes to exchange messages anonymously while forcing them to stick to the original protocol specification. However, these systems heavily rely on all-to-all communications, which makes their performance unsuitable for live content dissemination systems.

We propose in this paper PAG, an accountable and privacy-preserving gossip protocol, practical for live content sharing applications, where messages exchanged between any two nodes are kept private. PAG is decentralised and relies on nodes acting as monitors to enforce the correct dissemination of content updates. Specifically, through an homomorphic encryption of the disseminated messages, monitors enforce accountability without getting access to the content of the exchanged messages, thus protecting users' *interests*. Finally, through the modification of cryptographic keys at every hop of the dissemination process, monitors can not follow the progress of a given content update in the dissemination graph, thus preventing monitors from building *interest graphs*.

We assess PAG both theoretically and experimentally regarding accountability, privacy and performance. Regarding *accountability*, our analysis shows that PAG is a Nash equilibrium [16], which means that selfish nodes have no interest in deviating from the protocol. Further, regarding *privacy*, we prove the resilience of PAG against a global and active opponent using the ProVerif cryptographic protocol verifier [17]. Finally, regarding *performance*, we show through the implementation of a video live streaming application instantiated using 432 clients deployed in a cluster of 48 machines that: (1) PAG is practical in terms of bandwidth and CPU costs for streaming live video content compared to anonymous communication protocols like RAC and (2) its cryptographic overhead is reasonable for modern architectures. Complementary simulations show that PAG scales logarithmically in terms of bandwidth consumption.

The rest of the paper is structured as follows. Section II provides some background about accountability and privacy in gossip. Section III presents our system model and our objectives. Section IV presents the building blocks of PAG, which consist in a monitoring infrastructure and homomorphic messages hashing. Section V details the message exchanges of nodes running PAG. Section VI presents the privacy guarantees and an accountability analysis of PAG. Section VII presents a detailed performance evaluation. Section VIII reviews the related works. Section IX concludes this paper.

II. Accountable Forwarding and Privacy

In this section, we present the principles of the gossip paradigm and introduce selfish deviations that nodes may execute (part II-A). We further present accountability solutions (part II-B) and explain how accountability solutions threaten the privacy of users (part II-C).

A. Principles of gossip and selfishness

Peer-to-peer gossip protocols aim at reliably distributing a content (e.g., a video stream, membership updates) to a set of interested nodes. To do so, gossip protocols handle two tasks. First, they handle the neighbourhood of nodes by providing them with a selection of partners with which they can interact. This is commonly achieved by relying on a full membership protocol (e.g., [18], [19]), or on a distributed random peer sampling protocol (e.g., [20], [21]). Second, gossip protocols handle message exchanges enabling a content to be disseminated to all the participating nodes with a high probability [6]. As membership management does not handle content, we focus on bringing accountability and privacy to the second task. Using the gossip principle, message exchanges are organized in *rounds* (whose duration is called the *gossip period*). A special node that holds the content to disseminate (also called the *source*), generates and periodically sends chunks of this content (also called *updates*), to a set of nodes chosen uniformly at random. Then, periodically, each node taking part in the dissemination is in charge of sharing the updates it receives with f other randomly selected nodes (f is also called the dissemination *fanout*).

Figure 1 illustrates the gossip-based dissemination of updates, from the point of view of a node X depicted in the centre of the figure. Specifically, node X has a set of f_p predecessors $\{P_1, \ldots, P_{f_p}\}$ and a set of f_s successors $\{S_1, \ldots, S_{f_s}\}$ that have been picked uniformly at random from the nodes participating in the system. In this example, during round R, node X receives a set of data chunks from its predecessors (i.e., $\{u_1\}$ from P_1, ..., $\{u_{f_p}\}$ from P_{f_p} in the figure) and has to forward the received chunks in the following round R+1 to all its successors (i.e., $\{u_1, \ldots, u_{f_p}\}$ to $S_1, \ldots, S_{f_s}$ in the figure).

However, as presented in various studies (e.g., [22], [23]), gossip-based dissemination suffers from nodes behaving selfishly. Selfish behaviour takes place when nodes tamper with their software or use tampered software in order to maximise their benefit (e.g., receiving the disseminated content as fast as possible) while minimising their contribution to the system (e.g., saving bandwidth or computational resources).

B. Accountability solutions

Accountability mechanisms (e.g., PeerReview [9], FullReview [11], AVMs [10]) are effective solutions to deter faults in distributed systems. These mechanisms have already been used as incentives for forcing selfish nodes to participate in gossip-based content sharing protocols (e.g., [12]). Figure 2 shows an accountable gossip protocol in which a node X logs its interactions with its predecessors and successors in a secure log (depicted in the right part of the figure). For example, the first line of this log specifies that node X received $\{u_1\}$ from node P_1 during round R. Secure logs can either rely on cryptography techniques (e.g., recursive hash functions in [9], [10]) or on secure hardware (as in [24]) to make them tamper evident and append only. In these systems, each node X is further assigned a set of monitors (depicted above X in the figure) that periodically audit its log in order to assess whether the logged entries correspond to a correct execution of the gossip protocol. For instance, in the figure each monitor can check that node X has forwarded all the updates it received during round R (i.e., $\{u_1, ..., u_{fp}\}$) to all its successors (i.e., $S_1, ..., S_{fs}$) during round R+1.

C. Privacy of users

A major drawback of accountability mechanisms is that nodes must share their interaction logs with their monitors. In gossip-based applications such as content sharing or live video streaming applications, this allows monitors to learn about nodes *interests* and thus possibly infer sensitive information about them. Indeed, various studies (e.g., [25], [26]) have shown that the consumed media can disclose information about individuals (e.g., gender, sexual, religious or political preferences). Further to learning nodes interests, this allows to learn the *interest graphs* between nodes sharing similar interests, thus possibly inferring private information about them (e.g., learning that a given user is a lesbian because it is interested in similar contents as a person known to be a lesbian).

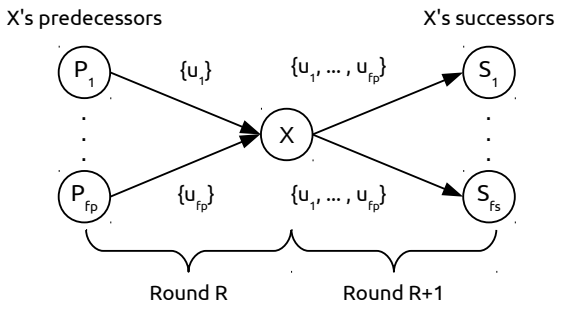

Fig. 1. Forwarding of updates in a gossip-based system

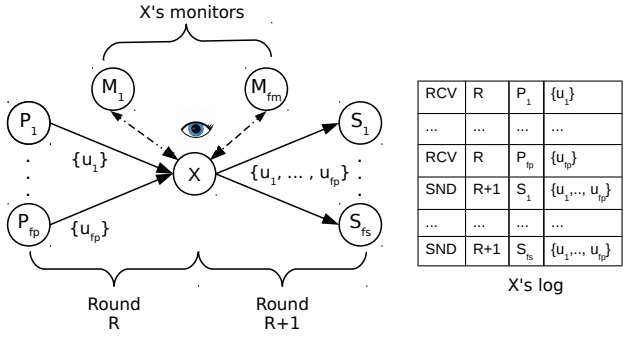

Fig. 2. Accountable gossip

III. System Model and Objectives

In this section, we present the assumptions we make in the rest of this paper and the objectives of PAG.

Communications and cryptographic assumptions. As classically made in gossip-based protocols (e.g., in [19] and [27]) we structure time using rounds. Nodes are roughly synchronized, which allows them to check each others' periodical exchanges based on the specification of the exchange protocol. Nodes are uniquely identified with an integer identifier, for example deterministically computed using their IP addresses, and cannot generate multiple identities. Further, we assume that nodes can generate prime numbers, and have access to secure asymmetric key encryptions and signatures. We will denote $p_k(X)$ the public key of a node X, $\{m\}_X$ the encryption of a message m by node X, and $\langle m \rangle_X$ a message m along with its signature by node X.

Gossip sessions and monitoring infrastructure. We assume that several gossip sessions disseminating different contents can hold simultaneously in the system. Each content is generated and signed by its source. Updates are propagated along with their signature so that they can be verified by the nodes upon reception, which prevents data tampering. Nodes interested in a content have to obtain the public key of its source using an external service. We assume that a membership protocol (e.g., Fireflies [18]) provides nodes with a set of successors and monitors that can be identified, for a given round, by each node in the system (as it has been done in [12], [19], [27]).

Nodes and adversary models. We consider several types of nodes. *Correct nodes* strictly follow the protocols. In particular, the source of each session is assumed to be correct. *Selfish nodes* are self-interested, and deviate from a protocol in any way that would improve their benefit (e.g., reduced bandwidth consumption or CPU overhead). We consider a *global and active opponent*, which is the strongest model of attacker. A global opponent can monitor and record the traffic on network links. Active means that it can control some nodes in the system and make them share information or deviate from the protocol (if possible) in order to reduce the privacy of other nodes. The only limitation of the global and active opponent is that it is not able to invert encryptions.

Objectives. To protect a gossip-based system from selfish deviations, we aim at enforcing the two following properties:

- R_1 *Obligation to receive:* At a given communication round, a node must receive the updates sent by its predecessors that it never received.
- R_2 *Obligation to forward:* A node must forward the updates it received at a given communication round R to all its successors during round R+1.

In addition, we aim at enforcing the following privacy property, which prevents an attacker from building interest graphs:

- P_1 *Unlinkability between updates and nodes:* Suppose that node A sends an update u to node B. Other nodes than A and B should not be able to link A or B with u.

IV. PAG in a Nutshell

In this section, we present two mechanisms composing PAG that when combined provide both accountability and privacy to dissemination protocols. We first present how nodes monitor each other to enforce accountability in part IV-A. Then, we introduce the intuition of the cryptographic procedures that preserve privacy in part IV-B.

A. Enforcing accountability using a monitoring infrastructure

We use a log-less monitoring infrastructure, because maintaining the consistency of secure logs is costly in terms of exchanges and computations. In addition, logs, which are public, reveal too much information about nodes. The first key idea of this infrastructure is that the monitors of a node A learn which updates A receives from declarations of the node (which are then verified by its predecessors' monitors), and check that A's successors receive these updates (from their monitors). Second, message transmissions between monitors allow them to maintain the same information about the updates a node receives and has to retransmit. Finally, using classical techniques we handle omission failures.

In the following, let M(A), respectively M(B), be the set of monitors of node A, respectively node B. Figure 3 illustrates the situation where a node A that received an update u, forwards it to node B (message 1.) during round R. Upon reception of u, node B sends an acknowledgement to its monitors (message 3.) and to node A (message 2.). If node A does not receive an Ack, it emits an accusation against node B, which consists in sending to nodes in M(B) the update u, and making them forward it to node B and ask

for an acknowledgement. If nodes in M(B) receive an Ack from node B, they transmit it to the nodes in M(A) using the $Confirm(\langle Ack(u, A)\rangle_B)$ message (message 4.), otherwise they send a Nack message to nodes in M(A). Using this message, nodes in M(A) can check that node A (i) contacted all its successors, and (ii) forwarded the right update. In the meanwhile, nodes in M(B) have learnt that node B received the update u, or that node B is unresponsive.

We now briefly explain why the best interest of nodes is to transmit each message represented in Figure 3. We assume that node A received update u, and that its monitors are informed of this reception. In addition, at least one node in each monitoring set is assumed to be correct. First, node A will correctly send u to node B, because sending accusations is more costly than sending a single update. Second, nodes in M(A) expect to receive either a $Confirm(\langle Ack(u, A, B)\rangle_B)$ message (which proves that node B received u), or a Nack message (if node B did not send a signed Ack to A). If nodes in M(A) receive none of these messages, they have to determine if node A did not send update u, or if node B did not send the acknowledgement to its monitors. To reach this goal, they ask node A for the acknowledgement that node B should have sent. If node A cannot exhibit this acknowledgement it is considered guilty because it did not accuse node B, otherwise node B is considered guilty.

To summarise, this monitoring infrastructure forces nodes to interact with their successors as well as to receive and forward updates. However, monitors are aware of all the transmitted messages. Our next goal is to hide this information.

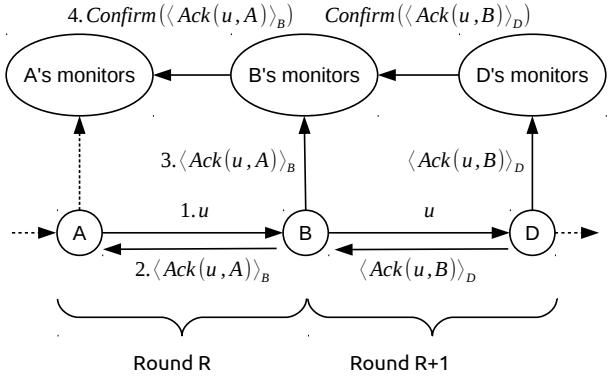

Fig. 3. Monitoring of nodes to ensure the forwarding of messages

B. Enforcing privacy using homomorphic hashes

A straightforward solution that one may think of to enforce privacy is to encrypt updates. However, doing so is not sufficient against a global and active opponent, because nodes which participate in the protocol would know the correspondence between a content and its encryption, and the monitors of a node would know which encrypted updates were received. To deal with this issue, we rely on homomorphic procedures that allow monitors to check that nodes forward updates and prevents them from learning which updates are exchanged.

Specifically, we use a hash function, noted H, based on an unpadded RSA encryption, that exploits two of its multiplicative properties. Its public key consists of a modulus M and an exponent p, then the hash of an update u is given by $H(u)_{(p,M)} = u^p \mod M$. Let u_1 and u_2 be two updates, and p_1 and p_2 be two exponents. The following homomorphic properties can be established:

$$H(u_1)_{(p,M)} \cdot H(u_2)_{(p,M)} = H(u_1 \cdot u_2)_{(p,M)}$$
$$H\left(H(u)_{(p_1,M)}\right)_{(p_2,M)} = H(u)_{(p_1 \cdot p_2,M)}$$

Any hash function verifying these two properties could be used to check the dissemination of messages. We will use a modulo size of 512 bits, as recommended in [28]. Nodes cannot decrypt the hashed updates, as the value of the modulus M is smaller than the size of updates.

Figure 4 illustrates the intuition of the application of this hash to check retransmissions. In this figure, nodes A and F are two predecessors of node B, and node D is a successor of node B. We represent only two of the j predecessors of node B for the sake of simplicity, even though having 3 predecessors is a minimum to ensure privacy. We only focus on the reception and forwarding of the messages that node B receives. The same steps would also apply to the nodes A, F and D to secure each forwarding step.

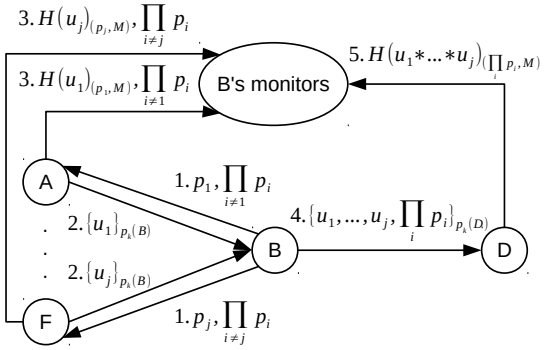

Fig. 4. Privacy preserving verification of a forwarding of a node B

Nodes A and F send updates u_1 and u_j to node B respectively. First, nodes A and F ask node B to send them a prime number. Node B chooses two prime numbers p_1 and p_j and respectively sends $(p_1, \prod_{i\neq 1} p_i)$ and $(p_j, \prod_{i\neq j} p_i)$ (messages 1.) to nodes A and F respectively. Then, nodes A and F send their two updates to node B encrypted with its public key (messages 2.). Nodes A and F declare (messages 3.) to the monitors of node B, that they sent updates to node B whose hashes are respectively equal to $H(u_1)_{(p_1,M)}$ and $H(u_j)_{(p_j,M)}$. Node B then forwards the updates u_1 and u_j to node D, and joins the product $\prod_i p_i$ (message 4.). Node D acknowledges the reception of u_1, ..., u_j using the hash value $H(u_1 * ... * u_j)_{(\prod_i p_i, M)}$ to the monitors of node B that verify that the following equation is verified:

$$\left(H(u_1)_{(p_1,M)}\right)^{\prod_{i\neq 1} p_i} * ... * \left(H(u_j)_{(p_j,M)}\right)^{\prod_{i\neq j} p_i} \mod M = H(u_1 * ... * u_j)_{(\prod_i p_i, M)}$$

This short example shows that the monitors of a node are able to check that it forwards the updates it receives without learning the actual content. To break this privacy, an attacker would have to learn the prime numbers a node has

chosen. With this information it would decrypt the exchanges a node had with its predecessors, or successors. In practice, predecessors and monitors of a node receive the product of prime numbers, and are not able to factorise it efficiently, as it is a notoriously known hard problem.

V. PAG Detailed Description

The monitoring infrastructure and the homomorphic hashes must be carefully combined so that selfish nodes can not deviate from the protocol without being detected. In this section, we detail the PAG protocol. We finally explain how the forwarding mechanism is adapted to build a gossip-based content dissemination protocol.

A. Transmission of updates between monitored nodes

Figure 5 presents the exchanges that occur when node A forwards a set S_A of updates to node B, which owns the set S_B of updates, during round number R. First, node A asks a prime number to node B that it will use to hash the product of the updates in S_A (message 1.). Node B generates one prime number p_i for each of its predecessors. We note $K(R, B) = \prod_i p_i$ the product of the prime numbers that node B chooses during round R to receive updates.

In message 2., node B replies with the primary key p_j in a message signed using its private key, and then encrypted using node A's public key. It also joins the homomorphic hashes $H(u_{i \in S_B})_{(p_j, M)}$ of the messages in S_B using p_j. Upon reception of this message, node A can check if the updates in S_A are not in S_B, and thus avoid to send them, as node B already owns them.

In message 3., node A serves in a message signed using its private key, and then encrypted using node B's public key, the updates in $S_A \setminus S_B$, that node B does not have, and K(R-1,A). The value of K(R-1,A) is the product of the prime numbers node A used to receive the updates in S_A from its predecessors during round R-1. Node B has to use this value to acknowledge the reception of updates using the hash function, in order for the monitors of node A to check its forwarding.

In message 4., node A sends to node B a signed attestation that declares the value of the hash of the product of the messages in S_A using p_j. This message will later be transmitted to the monitors of node B, which will then check the forwarding of node B based on its value.

In message 5., which is signed, node B acknowledges the reception of the messages in S_A using the hash of their product with $K(R-1, A)$. If necessary, node A can later use this message as a proof that it did forward the right set of messages to node B during round R.

B. Transmission of hashes to the monitoring infrastructure

Monitors check that the node they monitor (i) contacts all its successors, (ii) forwards all the messages it received at round R during round R+1. The first verification consists in checking the reception of messages from the monitors of each successor, which correspond to forwarded updates. For the second verification, monitors have to compute the homomorphic hash of the product of all the messages that the node receives during a given round, and check that its successors during the following round acknowledge this hash.

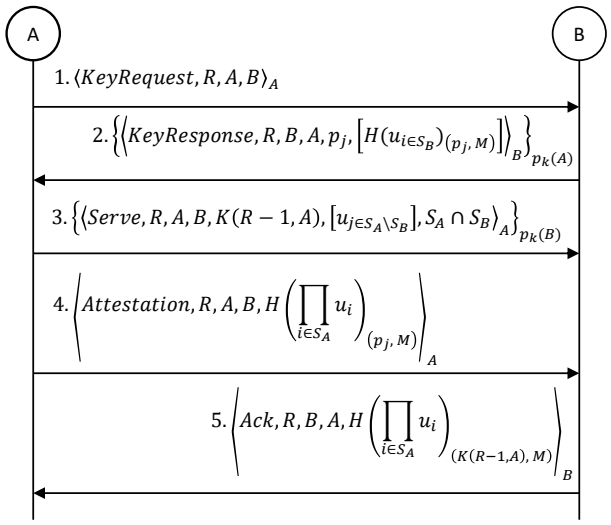

Fig. 5. Exchange of updates between nodes

In practice, at each round, the monitors of a node expect to receive messages from it, and from the monitors of its successors. Figure 6 illustrates the mechanisms that allow the monitors to perform these tasks after node B has received the set S_A of updates from node A. In this figure, the monitors of node B are nodes A, D and G, and node B sends two messages to only one of its own monitors, to prevent monitors from receiving all the products of the prime numbers.

Message 6. is a copy of the acknowledgement that node B sent to node A in (message 5. of Figure 5), and message 7., which is signed, contains the attestation that node A sent in message 4. of Figure 5, and the product of the prime numbers that node B used to receive messages from its other predecessors during round R.

The monitor that receives these two messages, here node D, from node B computes the value

$$\left(H\left(\prod_{i \in S_A} u_i\right)_{(p_j, M)}\right)^{\prod_{k \neq j} p_k} \mod M = H\left(\prod_{i \in S_A} u_i\right)_{\left(\prod_k p_k, M\right)} = H\left(\prod_{i \in S_A} u_i\right)_{(K(R,B), M)}$$

and broadcasts it to the other monitors of node B, along with message 6. To check that monitors correctly compute and forward the hashes of updates, nodes can compute this value and send it to their monitors. Monitors are then able to check each other's correctness.

C. Homomorphic combination of hashes by monitors

During a round, each monitor of node B computes the product of all the hash values forwarded by the other monitors of node B. At the end of the round, the monitors of node B know the homomorphic hash of the updates that node B received, computed using the product of the prime numbers that node B has chosen. This hash must then be acknowledged by the successors of node B during the following round.

Suppose that node B receives the set of messages S_A from

node A, and the set of messages S_F from node F during a given round. Let $\prod_j p_j$ be the product of the prime numbers that node B used to receive these messages. The monitors of node B obtain the hash of the union of S_A and S_F applying the formula

$$H\left(S_A \cup S_F\right)_{\left(\prod_j p_j, M\right)} = H\left(S_A\right)_{\left(\prod_j p_j, M\right)} \times H\left(S_F\right)_{\left(\prod_j p_j, M\right)}$$

To allow the monitors of node A to obtain the right part of this formula, the monitors of node B have to forward them the acknowledgement (message 9.). The monitors of node A can then verify that node B received the correct set of messages from node A.

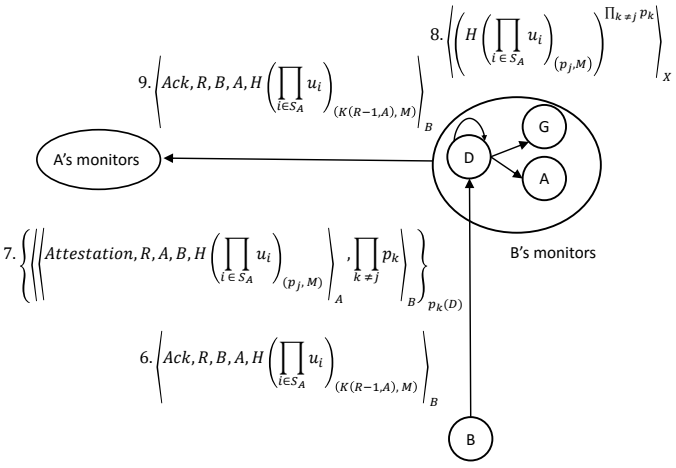

Fig. 6. Monitoring part of an interaction between two nodes

D. Application to a content-dissemination system.

In this section we present the important details or optimisations that allow the protocol to become practical when applied to a gossip-based dissemination protocol.

Buffermap transmissions. A node sends to its predecessors the hashes of a proportion of the messages it owns, in order to avoid multiple receptions. Determining how many hashes to send is dependent on the applications, and more particularly on the sizes of updates and of hashes. In our scenario, updates were bigger than their hashes, and the best results in terms of bandwidth consumptions were obtained when the updates of the last 4 rounds were hashed and transmitted.

Multiple receptions. While PAG avoids some multiple receptions of a same update, they can still occur when a node simultaneously receive updates from different predecessors. However, to limit the bandwidth consumption of nodes it is necessary to make them forward these updates only once. To do so, when a node sends an update it also joins to it an integer which describes the number of times it was received by the sending node during the previous round. This enables the receiving node to accurately compute the hash of the set of received updates, and the monitors to match the hashes of received updates with the ones of forwarded messages.

Expiration of updates. In the context of live-streaming, updates have an expiration date after which nodes should not continue to forward them. Determining this expiration delay is up to the system designer. To allow updates to stop being propagated, when a node forwards them to another node, it separates the updates in two lists: the first one contains updates that will expire in the next round, and that should not be forwarded, while the other one contains updates that must be forwarded. A small modification of the messages and monitoring exchanges allow updates to expire. The monitors of a node acknowledge the reception of the first list and check the propagation of the second list.

VI. Privacy and Accountability Analysis

In this section, we present the results of the security analysis we made using the cryptographic protocol verifier ProVerif (part VI-A). This proof shows that property P_1 holds against a global and active attacker if it controls less than f nodes, where f is the number of successors per node. We then briefly explain why the implementation of the forwarding mechanism provides accountability (part VI-B).

A. Privacy Guarantees

ProVerif [17] is an automatic cryptographic protocol verifier that uses Horn clauses to detect possible attacks. Using ProVerif, we modelled the cryptographic mechanisms of PAG[1]. This model shows that there is no attack on the privacy property P_1 that involves less than f nodes, where f is the number of predecessors, successors and monitors per node.

We consider the representative situation where a node B, assumed to be correct, receives updates from three predecessors A_1, A_2 and A_3, and has to forward them to one of its successors C. For each node, we instantiated a set of monitors. The case where f=3 is the simplest where the protocol can be proved secure. Increasing the value of f reinforces the security of the protocol, as the necessary number of colluding nodes sharing information in order to break the privacy also increases. The aim of an attacker is to obtain the value of a prime number that node B chooses in order to obtain the detail of the exchange between node B and this node. For attacks to be feasible, we assume that the attacker has access to the list of updates that node B may have received from its predecessor. In order to find the updates that B received, the attacker would have to hash any possible combination of updates using the prime number and see if it is equal to the observation. This attack is not really practical because the number of subsets of a set of size N is equal to 2^N.

We modelled several attack scenarios to assess the privacy property P_1. These scenarios can be grouped under two cases:

- **Case (1).** The attacker listens all communications on the network, and tries to break the privacy of exchanges between nodes A_1 and B. The attacker can replay, or inject messages in the network.
- **Case (2).** In addition to the assumptions of case 1., we consider that at most (f-1) nodes among the monitors or predecessors of a node are part of a coalition. This case can be instantiated with several configurations (e.g., (f-2)

[1]The code is available at https://github.com/jdecouchant/PAG

monitors and 1 predecessor, (f-3) monitors and 2 predecessors, etc.) that were all tested in our configuration.

In case (1), ProVerif proves that no attack exists on the cryptographic procedures of PAG. The experiments in case (2) confirm that no attacks exist if the opponent controls less than f nodes. An attack is possible if f nodes collude among the monitors or predecessors of a node, and ProVerif found it. In this case, the opponent is able to obtain the prime numbers that B generated and thus learn the updates node B received.

B. Accountability Analysis

We present in this section a sketch of the incentives that enforce properties R_1 and R_2.

Let us consider the exchanges depicted in Figure 5. In the following, we briefly explain the incentives that force a selfish node, say node A, to follow the steps depicted in this figure. Remember that nodes register the messages they send or receive, and can use them to prove their correctness or that another node deviated.

Node A computes the set of updates that its successor does not have (which enforces R_1) and send them, along with the identifiers of the updates its successor already have (message 3). If a node does not send the right set of updates to its successors then the verification its monitors run will fail. Eventually, as its successors receive signed messages that they can exhibit, it will be proved guilty. The attestation (message 4) that node A sends can be verified by node B, thus a selfish node will correctly compute its value. In return, the acknowledgement (message 5) that node B sends can also be verified by node A. This acknowledgement forces node B to inform its monitors about the updates it received from node A (messages 6 and 7 of figure 6). Finally, if the verifications of node A pass then it means that it forwarded the right set of updates to the right nodes. After having received these updates, node B is engaged towards its own monitors to continue the forwarding of updates, which enforces property R_2.

VII. Performance Evaluation

In this section, we present the performance evaluation of PAG. We start by introducing our methodology and the values of the protocol's parameters (part VII-A). We then evaluate the overhead of PAG in terms of bandwidth consumption using both simulations and real code deployments compared to state-of-the art competitors while varying the size of the content being disseminated (parts VII-B). Further, we evaluate the cryptographic costs of PAG (part VII-C) as well as its scalability with respect to the number of users (part VII-D). We finally evaluate the proportion of exchanges that an active and global attacker may discover if it controls more than f nodes in the system (part VII-E).

Overall, our evaluation shows that PAG is more costly than existing accountable gossip protocols which do not preserve privacy. Yet, contrary to accountable anonymous communication protocols, its performance is compatible with streaming live video content on commodity Internet connections. Furthermore, PAG's cryptographic overhead can be handled by modern architectures and thanks to its inherited gossip properties, its bandwidth overhead scales logarithmically with the number of nodes in the system. Finally, PAG improves the resilience to active and global opponents compared to state of the art protocols.

A. Methodology and Parameter Settings

To assess the performance of PAG, we implemented it in Java and used it as a video live streaming application. When it is not specified, PAG is configured with the same numbers of successors and monitors per node (e.g., 3 when the system contains 1000 nodes). In this context, a source node diffuses a video stream at a fixed rate and sends each update to random successors. Updates are then disseminated using PAG or one of the protocols we compare PAG with. Among these protocols are an accountable gossip protocol, and two anonymous communications protocols. **AcTinG** [12] is an accountable gossip protocol that does not preserve the privacy of nodes as nodes maintain a secure log, and audit each other. **RAC** [15] is an anonymous communication system that forces nodes to relay the messages that other nodes send. Using RAC, a source could send a content to all nodes anonymously while enforcing accountability. We do not study Dissent [14] at it was shown to be even more costly than RAC in [15].

Real deployment settings. We deployed PAG on 48 machines of the Grid5000 cluster, interconnected using a 1Gb/s network, and using 9 instances per machine, thus totalling 432 nodes. Each machine contains a 4-cores Intel Xeon L5420 processor clocked at 2.5Ghz with 32GB of RAM. A source groups packets in windows of 40 packets. The duration of one round is set to one second, and updates of 938B are released 10 seconds before being consumed by the nodes' media player. Signatures are generated using RSA-2048. The sizes of the generated prime numbers is set to 512 bits. The modulus used in the homomorphic hashes is 512 bits long.

Simulations settings. We used the OMNeT++ [29] simulator, and ran a C++ implementation of PAG using the same parameters value we used in our deployment. We also computed the scalability of the protocol when the number of nodes was too high to be simulated.

B. Comparisons with existing protocols

We first compare the bandwidth consumption of PAG to the one of AcTinG [12]. Figure 7 presents the cumulative distribution functions of the bandwidth consumptions of nodes during a 300Kbps streaming session. In average, nodes running AcTinG consume 460 Kbps, while they consume 1050Kbps using PAG. This additional cost comes from the forwarding policy: nodes must receive, at least through hashes, the updates of their predecessors. Hence, a given node may have to forward several times a given update to its successors. AcTinG is less costly because nodes can refuse updates, and it is then controlled using their log during audits. Increasing the number of monitors does not significantly increase the bandwidth cost of the protocol, because the messages transmitted between

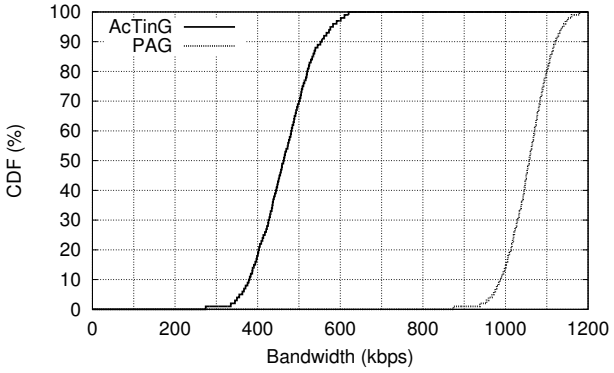

Fig. 7. Bandwidth consumption with a 300 kbps stream and 3 monitors

and to monitors are small, and allows a better resilience to collective deviations between nodes.

Relying on anonymous communication systems to run a gossip protocol would enforce privacy. However, these protocols are costly and can not scale like PAG and AcTinG. We thus designed a second set of experiments. The first two lines of Table I present the video qualities we considered and the associated payload size. Table II details the results we obtained with 1000 nodes. For each network capacity, ranging from 1.5Mbps to 10Gbps, we study the maximum video quality that each protocol can provide, and the amount of bandwidth that is used. For example, with 1.5Mbps network links AcTinG provides a 480p video using 1.4Mbps. PAG is more costly than AcTinG which is also accountable but not privacy preserving. Using 10Mbps network links, PAG provides at most a 480p video, consuming 6.9Mbps of bandwidth. In comparison, AcTinG is able to send a 1080p video using 6Mbps of bandwidth.

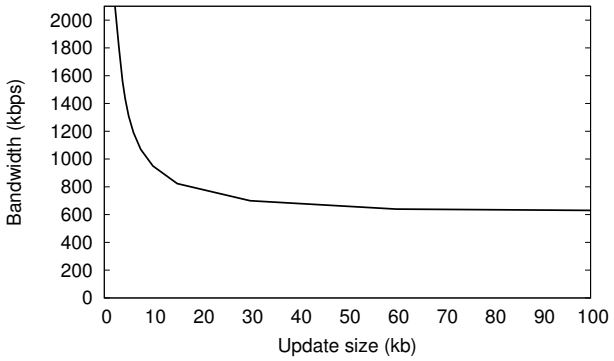

Fig. 8. Bandwidth consumption with 1000 nodes and a 300Kbps stream in function of the size of updates [sim]

However, using anonymous communication systems would be much more costly. The maximum payload that RAC is able to provide using 10Gbps network links is equal to 63kpbs, which is far from the minimum of 300Kbps that a basic streaming session would require.

Impact of updates size. Although we used 938B updates in the previous experiments, as was done in [19], [30], Figure 8 shows that using bigger updates can further decrease the bandwidth consumption of PAG. This is due to the fact that more content can be represented under each hash. For example, nodes propagating 10Kb updates needed to perform 370 homomorphic hashes per second, while propagating 100Kb updates decreased this number to 52 hashes per second. In addition, using larger updates also enables the CPU overhead to decrease, as hashes are computed modulo M.

Video quality	144p	240p	360p	480p	720p	1080p
Payload size (Kbps)	80	300	750	1000	2500	4500
RSA signatures	33	33	33	33	33	33
Hashes	133	475	1170	1560	3934	7200

TABLE I
NUMBER OF RSA SIGNATURES AND HOMOMORPHIC HASHES PER SECOND IN A SYSTEM OF 1000 NODES [SIM]

C. Cryptographic costs

PAG relies on cryptographic mechanisms, which dominate its CPU cost. To evaluate this cost, we measured the number of generated RSA encryptions and homomorphic hashes per second rather than the CPU load, which depends on the hardware used. We measured these numbers depending on the video quality, and depicted the results in Table I. The number of RSA signatures is always equal to 33, as it depends on the number of messages generated by the protocol, while the number of homomorphic hashes performed depends on the video quality, and more precisely on the number of 938B updates. Using openssl, we measured that each core of the machines we used is able to perform 4800 hashes per second with a 512-bits modulus. Thus using a single core to compute homomorphic hashes is enough to obtain a video quality up to 720p using a 512 bits modulus, which would generate 3924 hashes per second. Using more cores would provide enough cryptographic power to support better video qualities. In addition, using a 256 bits modulus can also be considered secure enough in many situations, and it would significantly reduce the bandwidth overhead of the protocol. Overall, we believe that PAG can be used by a wide range of commodity hardware.

D. Scalability

In this experiment we increase the number of nodes in the system and measure their bandwidth consumption. The bandwidth scalability of PAG comes from its gossip nature, as in a system of N nodes, each user has $log(N)$ successors. Figure 9 presents the bandwidth consumption of AcTinG and PAG depending on the system size when a 300Kbps video stream is disseminated. With a million nodes PAG consumes 2.5Mbps, while AcTinG needs 840Kbps. In these conditions, PAG is able to provide nodes with the full 300Kbps stream, while consuming less bandwidth than anonymous communication systems, which are too costly to be used.

E. Probabilistic study of the impact of coalitions on privacy

As proved using ProVerif, a coalition of at least f nodes can obtain the prime numbers used in some nodes' interactions. We now evaluate the privacy leakage performed by a global

	Privacy	Accountability	1.5Mbps ADSL Lite	10Mbps Ethernet	100Mbps Fast Ethernet	1 Gbps Gigabit Ethernet	10Gbps 10 Gigabit Ethernet
PAG	✓	✓	144p (660 Kbps)	480p (6.9 Mbps)	1080p (31 Mbps)	1080p (31 Mbps)	1080p (31 Mbps)
AcTinG	×	✓	480p (1.4 Mbps)	1080p (6 Mbps)	1080p (6 Mbps)	1080p (6 Mbps)	1080p (6 Mbps)
RAC	✓	✓	∅	∅	∅	∅	∅

TABLE II
MAXIMUM VIDEO QUALITY SUSTAINABLE IN FUNCTION OF THE NETWORK LINKS CAPACITY, AND THE ASSOCIATED BANDWIDTH CONSUMPTION, IN A SYSTEM WITH 1000 NODES

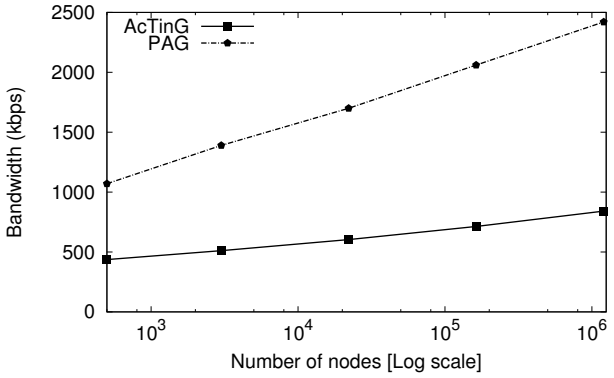

Fig. 9. Scalability of PAG and AcTinG with a 300Kbps content [sim]

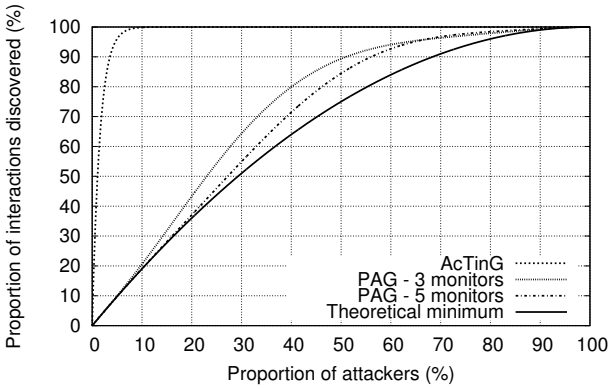

Fig. 10. Resiliency against a global and active attacker

and active attacker that would control more than f nodes in PAG. We also perform this attack on an existing state of the art protocol, i.e., AcTinG [12]. In AcTinG, monitors probabilistically audits nodes' secure logs, which contain the detail of past interactions. Differently, in PAG, it is possible to discover the details of the interactions of a node if all its predecessors except at most two and at least one of the monitors of this node collude. This essentially means that collecting the prime numbers a node used, and observing all its encrypted interactions is enough to understand them. The success of attacks is evaluated in terms of probabilities as nodes are randomly affected predecessors, successors, and monitors. We evaluate the probability that an exchange between two nodes is discovered in function of the size of the coalition.

In Figure 10, we evaluate the proportion of exchanges that an attacker controlling a variable proportion of the membership can discover. The lowest possible proportion in this case is represented in plain black, and expresses the probability that at least one of the two nodes that interact is corrupted. As this figure shows, increasing the number of monitors, and the number of nodes in the system, makes the privacy guarantees of PAG close to ideal, while all interactions are discovered when an attacker controls 10% of nodes in AcTinG.

VIII. RELATED WORKS

In this section we describe how existing solutions relate to the problem we solve in this paper.

Selfish resilient gossip protocols. BAR Gossip [19] and FlightPath [27] are streaming protocols that handle both selfish and Byzantine deviations through deterministic interactions. However, peers are not forced to initiate exchanges. Differently, LiFTinG [30] uses audits to control that nodes forward the updates they receive. However, it relies on statistical properties that may produce up to 12% of false-positives, and false-negatives, and reveal the detail of the interactions of nodes.

Anonymous communications. The first anonymous communication protocols, DC-Net [31] and Onion Routing [32], focused on enabling the strongest possible anonymity level for the former, and on providing practical performance for the latter. However, they take the participation of nodes for granted. More recently, Dissent [14] and RAC [15] force nodes to execute their role of relay. However, using these systems can be seen as a performance overkill. For example, for each message sent anonymously, Dissent [14] uses trusted nodes which receive anonymous communication requests and runs a protocol involving all-to-all communications.

Zero-Knowledge proofs. Classical zero-knowledge proofs can not always be applied because they are designed to verify functions that has a fixed number of inputs, whereas in many distributed systems both the size and the number of a node's "inputs" (the messages it has received from other nodes) are not known. In particular, in gossip, the quantity of messages a node receives during a time interval is not predictable.

Accountability approaches. Software-based accountability approaches, which include PeerReview [9], and A2M [33], make nodes maintain a secure log, and audit each other. Audits transfer logs, and reveal detailed information about nodes. Similarly, hardware-based accountability approaches [24], [33] rely on a trusted hardware to maintain secure logs. For example, PeerReview associates each node with a set of monitors, which verify it using the secure log. In our context, these applications would allow an attacker to track the propagation of an update among a membership, even if its encrypted.

Privacy-preserving accountability. In [34], nodes maintain a Merkle Hash Tree in which the leaves represent all the possible states of a node, and regularly publish its root hash

value. Nodes that interacted with a given node are then able to collectively, but anonymously, check its state. This approach has been applied to a BGP routing system [35]. However elegant, we believe that this approach cannot be applied to gossip protocols, as it is not possible to concisely represent all the possible states of a node.

Virtual currency. In [36], a virtual currency approach is shown to provide accountability without compromising privacy in a peer-to-peer system. However, contrary to PAG, which is fully-decentralised, this solution requires two trusted entities that have access to nodes' information: i) a bank, which maintain an account for each user and knows about all transactions in the system; and ii) the arbiter, which ensures the fair exchange of e-cash for data.

IX. Conclusion

A number of gossip-based content dissemination protocols tolerating selfish behaviours have been proposed in the past. However, they do not preserve the privacy of users. On the other side of the spectrum, accountable anonymous communication protocols are too costly to be used to disseminate live multimedia content. In this paper, we have presented the first protocol that enforces accountability through a monitoring infrastructure and still preserves the privacy of users thanks to homomorphic cryptographic procedures. Performance evaluation of PAG combining both a real deployment and simulations has demonstrated that its bandwidth consumption is compatible with streaming live content and that the partial privacy of nodes is close to optimal, even in presence of a global and active attacker. We have also shown that the reasonable cryptographic overhead of PAG makes it accessible to modern architectures, and that it exhibits very desirable scalability properties with a logarithmic growth of bandwidth consumption, comparable to standard gossip-based protocols. The privacy of nodes could be further enhanced if even the direct neighbors of nodes could not determine the media content they are interested in. Doing so in a peer-to-peer system is challenging, and future works include the design of a dissemination protocol that would improve on the obfuscation approach, which hide the interests of nodes by making them receive several contents at the same time.

Acknowledgment

This research was partially supported by the SnT and by the National Research Fund Luxembourg (FNR), through PEARL grant FNR/P14/8149128, and by the LABEX IMU (ANR-10-LABX-0088) of Université de Lyon, within the program "Investissements d'Avenir" (ANR-11-IDEX-0007) operated by the French National Research Agency (ANR).

Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see https://www.grid5000.fr).

References

[1] N. Basher, A. Mahanti *et al.*, "A comparative analysis of web and peer-to-peer traffic," in *WWW*, 2008.

[2] "Velocix," http://www.velocix.com, accessed: 2015-07-26.

[3] L. Vu, I. Gupta *et al.*, "Understanding overlay characteristics of a large-scale peer-to-peer iptv system," *TOMCCAP*, 2010.

[4] H. Yin, X. Liu *et al.*, "Design and deployment of a hybrid cdn-p2p system for live video streaming: experiences with livesky," in *ACM Multimedia*, 2009.

[5] T. Bonald, L. Massoulié *et al.*, "Epidemic live streaming: optimal performance trade-offs," in *ACM SIGMETRICS*, 2008.

[6] A.-M. Kermarrec, L. Massoulié *et al.*, "Probabilistic reliable dissemination in large-scale systems," *Parallel and Distributed Systems*, 2003.

[7] R. Eidenbenz, T. Locher *et al.*, "Hidden communication in p2p networks steganographic handshake and broadcast," in *INFOCOM*, 2011.

[8] I. Cunha, E. C. Miguel *et al.*, "Can peer-to-peer live streaming systems coexist with free riders?" in *P2P*, 2013.

[9] A. Haeberlen, P. Kouznetsov *et al.*, "Peerreview: Practical accountability for distributed systems," in *ACM SIGOPS OSR*, vol. 41, no. 6, 2007, pp. 175–188.

[10] A. Haeberlen, P. Aditya *et al.*, "Accountable virtual machines." in *OSDI*, 2010.

[11] A. Diarra, S. B. Mokhtar *et al.*, "Fullreview: Practical accountability in presence of selfish nodes," in *SRDS*, 2014.

[12] S. B. Mokhtar, J. Decouchant *et al.*, "Acting: Accurate freerider tracking in gossip," in *SRDS*, 2014.

[13] P. Aditya, M. Zhao *et al.*, "Reliable client accounting for hybrid content-distribution networks," in *NSDI*, 2012.

[14] D. I. Wolinsky, H. Corrigan-Gibbs *et al.*, "Dissent in numbers: Making strong anonymity scale." in *OSDI*, 2012.

[15] S. Ben Mokhtar, G. Berthou *et al.*, "Rac: a freerider-resilient, scalable, anonymous communication protocol," in *ICDCS*, 2013.

[16] J. Nash, "Non-Cooperative Games," *The Annals of Mathematics*, vol. 54, no. 2, 1951.

[17] B. Blanchet, "An efficient cryptographic protocol verifier based on prolog rules," in *csfw*, 2001.

[18] H. Johansen, A. Allavena *et al.*, "Fireflies: scalable support for intrusion-tolerant network overlays," in *ACM SIGOPS OSR*, 2006.

[19] H. C. Li, A. Clement *et al.*, "Bar gossip," in *OSDI*, 2006.

[20] A. J. Ganesh, A.-M. Kermarrec *et al.*, "Scamp: Peer-to-peer lightweight membership service for large-scale group communication," in *Networked Group Communication*, 2001.

[21] M. Jelasity, S. Voulgaris *et al.*, "Gossip-based peer sampling," *TOCS*, 2007.

[22] E. Adar and B. A. Huberman, "Free riding on gnutella," *First Monday*, 2000.

[23] R. Krishnan, M. D. Smith *et al.*, "The impact of free-riding on peer-to-peer networks," in *System Sciences*, 2004.

[24] D. Levin, J. R. Douceur *et al.*, "Trinc: Small trusted hardware for large distributed systems." in *NSDI*, 2009.

[25] E. Zheleva and L. Getoor, "To join or not to join: the illusion of privacy in social networks with mixed public and private user profiles," in *WWW*, 2009.

[26] H. Hu, G.-J. Ahn *et al.*, "Detecting and resolving privacy conflicts for collaborative data sharing in online social networks," in *ACSAC*, 2011.

[27] H. C. Li, A. Clement *et al.*, "Flightpath: Obedience vs. choice in cooperative services." in *OSDI*, 2008.

[28] Enisa, "Algorithms, key size and parameters report," 2014.

[29] O. D. E. Simulator, https://omnetpp.org/.

[30] R. Guerraoui, K. Huguenin *et al.*, "Lifting: lightweight freerider-tracking in gossip," in *Middleware*, 2010.

[31] D. Chaum, "The dining cryptographers problem: Unconditional sender and recipient untraceability," *Journal of cryptology*, 1988.

[32] D. Goldschlag, M. Reed *et al.*, "Onion routing," *Communications of the ACM*, 1999.

[33] B.-G. Chun, P. Maniatis *et al.*, "Attested append-only memory: Making adversaries stick to their word," in *ACM SIGOPS*, 2007.

[34] A. Papadimitriou, M. Zhao *et al.*, "Towards privacy-preserving fault detection," in *Hot Topics in Dependable Systems*, 2013.

[35] M. Zhao, W. Zhou *et al.*, "Private and verifiable interdomain routing decisions," in *SIGCOMM*, 2012.

[36] M. Belenkiy, M. Chase *et al.*, "Making p2p accountable without losing privacy," in *ACM workshop on Privacy in electronic society*, 2007.