Homomorphic Cryptography Based Anonymous Routing

Antoine Guellier

Master research Internship

Master report

Homomorphic Cryptography Based Anonymous Routing

Author:
Antoine Guellier

Supervisor:
Christophe Bidan
Nicolas Prigent
CiDRE

Acknowledgment

I sincerely acknowledge and thank Simon Boche, PhD student at Supélec Rennes, for the insightful suggestions and discussions he provided during my internship. Several components the work I produced are the results of exchanges with him.

Also, I must thank Christophe Bidan and Nicolas Prigent for their guidance and support during the time spent in the CIDRe team. The knowledge they brought me and their invaluable advices were a lighthouse on my journey to awareness.

More generally, I would like to thank the whole CIDRe team for their sympathy and insight on various subjects around information security.

Contents

Abstract

The impressive development of ubiquitous systems and *anywhere, anytime* communications is faces security issues that prevent this industry from growing any further. Because of the properties of these systems, it is especially difficult to prevent users' private information leakage such as the users' identities or the relations between users. In the world of mobile ad hoc networks (MANETs), ad hoc routing protocols should ensure the protection of the nodes' identities, as well as their geographic and relative location. In this paper, we propose a *proactive* routing protocol using fully homomorphic cryptography to guarantee the protection of nodes' private information. This incipient technology allows to use an information without decrypting it, i.e. without having access to it. Using as little assumptions as possible, we show that a privacy-preserving proactive ad hoc routing protocol is possible.

1 Introduction

The impressive development of ubiquitous systems and *anywhere, anytime* communications is security issues that prevent this industry from growing any further. User anonymity and privacy are important aspects of security from the user's point of view. Protection of private information differs from traditional data protection. It supposes preventing information leakage, including the user's identity, his geographic position, the relationships between users, etc.

In the world of mobile ad hoc networks (MANETs), where users (called *nodes*) communicate over a wireless medium without any network infrastructure, performing this level of security is especially arduous. This type of network is composed of mobile nodes communicating over WiFi with a limited range. As there is no router or *central node*, two out-of-range communicating nodes have to rely on their peers to forward their messages. All these characteristics have to be taken in account when trying to design privacy-preserving ad hoc routing protocols.

Secure ad hoc protocols using traditional cryptography techniques as encryption and signature already exist. Used together with malicious node detection and reputation techniques, they provide protection from malicious nodes trying to disrupt the routing. However, security does not entail privacy. Encrypting a message is sufficient for confidentiality, but privacy also considers hiding who's communicating with whom for instance.

Several solutions of anonymous (i.e. privacy preserving) ad hoc routing have been proposed, and most of them use a *reactive* approach. Reactive protocols use *route requests* (RREQ): when a node needs to communicate with a given destination, it floods a request in the whole network. When the destination receives the request, it will answer with a *route reply* (RREP) containing routing information. In this type of routing, nodes keep track of little information about the network. On the contrary, the *proactive* approach maintains at all time complete routing tables concerning all destinations. Therefore, nodes are aware of the complete network topology. Using periodic messages, nodes permanently exchange their knowledge of the network.

Our main contribution is to propose an anonymous *proactive* ad hoc routing protocol, using as less assumptions as possible. At first, the proactive approach may not seem suited when trying to ensure the nodes' privacy, as each node possesses a lot of

information on the network. However, this conjecture can be mitigated. In particular, by using Gentry's fully homomorphic cryptography [14], one can process data without accessing it. Even though this technology is currently quite inefficient from a computational perspective, especially for an application in ad hoc networks, we merely show that an anonymous proactive protocol is possible thanks to it.

The rest of the report is organized as follows. Section 2 presents the characteristics and security issues of ad hoc routing. Section 3 defines the underlying concepts of privacy, and possible adversarial model. Before introducing homomorphic cryptography and describing our protocol in section 5, we present several existing anonymous protocols. Section 6 gives perspectives for future works, and Section 7 concludes.

2 Ad hoc Routing Protocols

This section introduces ad hoc networks and their properties. Then we set out the concepts behind reactive and proactive routing, along with examples of protocols. Finally, we discuss solutions to secure ad hoc routing protocols.

2.1 Ad hoc Networks

In this document, we focus on wireless mobile ad hoc networks (MANETs). The particularity of these networks is the lack of infrastructure to organize the nodes. Unlike a structured network with nodes connected through a WiFi Access Point for example, the nodes are all equal and communicate over wireless links without central control. Indeed, ad hoc networks offer the possibility to communicate with local devices without having to plan the insertion of the node in the network. For instance a collection of mobile phones in a city can create an ad hoc network to exchange information or play multiplayer games. The condition is that nodes use the same routing protocol, a protocol that allows them to self-organise.

Mobile nodes having a limited range, to communicate with an out-of-range node, other nodes have to relay messages and play the role of routers. When a node wants to send a message, its only way is to broadcast it. Thus, every node is listening to all messages it receives, and re-broadcasts them if it is on the path between the source and the destination. This is basically the role of an ad hoc routing protocol: to provide techniques such that out of range nodes may communicate with each other through intermediary nodes. This is the main challenge in ad hoc networks, and that's why ad hoc routing protocols are different and more complex than those for typical structured networks (with a central authority or access point).

There exists different routing protocols trying to answer these requirements in one way or another. We distinguish two types of protocols: the *proactive* and the *reactive* routing protocols.

2.2 Reactive Routing Protocols

In *on demand* (or reactive) routing protocols, a node willing to communicate with a destination initiates a *route discovery* process: it floods a *Route Request* (RREQ) message in the network. When the destination receives the messages, it answers with a *Route Reply* (RREP) message that contains the relevant routing information for the

source and relay nodes. Thus, a source requests a path only when it needs to reach a destination. Then, all actor (i.e. source, destination and relay nodes) keep track of the routing information as long as needed, and if the source wants to communicate with the same destination later, it might start a new route discovery.

The main drawback of this approach is the latency induced by the route discovery process. The time needed for the RREQ and RREP to traverse network may limit the size of the network (in terms of number of nodes). This is particularly true when considering the mobility of the nodes: if by the time the RREP comes back, the source moved from its initial location, the route cannot be constructed and the source must send a new RREQ.

AODV

One of the most popular reactive routing protocol is AODV [33] (*Ad-Hoc On-Demand Distance Vector routing*). In AODV, when a node needs to send a packet to a destination for which it has no route, it starts a *Path Discovery* process by broadcasting a RREQ (route request) packet to its neighbors. A RREQ contains the source and destination addresses, and a broadcast id that identifies it. When a node receives a RREQ it cannot satisfy, it forwards the message (i.e. it re-broadcasts it), and sets up the *reverse path* towards the originator of the RREQ (it does this only the first time it receives this RREQ, identified by the source address and the broadcast id). When the destination finally receives the RREQ, it sends back a RREP (*route reply*) packet which travels back to the source node along the reverse path. A RREP packet does not itself contain routing information, but merely the addresses of the source and destination. Nodes that relayed the RREQ know they must use the reverse path to bring the RREP to the source (they recognize the destination and source addresses). As the RREP goes back along the reverse path, a *forward path* is created: each node sets a forward pointer from which it received the RREP. When the source receives the RREP, it knows a path has been found, and it will use the relay nodes along the forward path to reach the destination. All other reverse paths that were constructed but that are not used eventually timeout. Figure 1 illustrates the reverse and forward path setup.

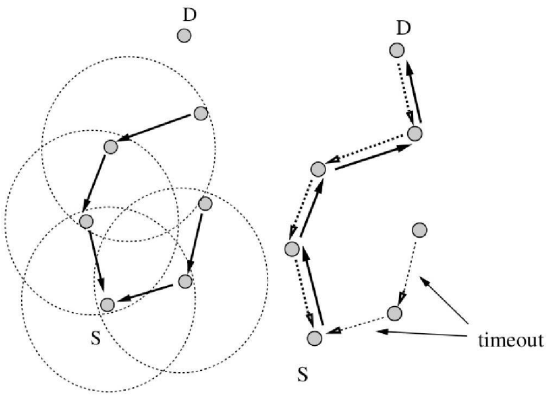

Figure 1: Left: Reverse Path setup. Right: Forward (and reverse) path [33]

On top of this basic routing service, AODV uses several other mechanisms to improve the routing. For instance, thanks to sequence numbers, the source is ensured to have the

most recent routing information. Also, in order to maintain forward paths, neighboring nodes periodically broadcast *hello messages* in order to show their presence (or their absence by a lack of hello message). Finally, in AODV, RREQs can be answered by any node that knows a path to the destination. This optimisation accelerates the acquisition of a route.

2.3 Proactive Routing Protocols

The proactive or *table driven* routing is a different approach. In this case, nodes maintain information about every destination in the network and every node is aware of the complete topology. Therefore, when a node wants to send a message to another node, it already knows the path to it thanks to stored tables. In order to maintain fresh information about destinations, the nodes periodically flood their knowledge of the topology throughout the network *via* control messages.

The price for knowing the complete network is the bandwidth consumed by those periodic messages. Indeed, proactive protocols have a high overhead due to periodic control messages exchanges. In compensation, every destination is accessible at any time and mobility is well tolerated.

OLSR

The OLSR [22] protocol (*Optimized Link State Routing*) is a good example of proactive routing. The first step is the neighbor sensing : each node sends *HELLO* messages to its 1-hop neighbors (nodes that are in its radio range) to advertise its presence. *HELLO* messages also carry the emitter's knowledge of its 1-hop neighborhood. Thanks to these messages, a node can learn about its 2-hop neighbood (nodes that it can reach with 1 relay node) and is able to choose its *Multipoint Relays* (MPR) among its direct neighbors. MPRs of a node are special neighbors that relay its messages. They are chosen so that they can reach all the node's 2-hop neighbors, i.e. the union of the neighbor sets of the node's MPRs contains the entire 2-hop neighbor set. Routing information concerning 3-hop (and more) nodes are extracted from TC (*Topology Control*) messages. These messages are flooded in the entire network by the MPR nodes. Thanks to the TC messages, every node is able to construct, step by step, the entire network graph. As a result, a path from a source to a destination is a sequence of MPR nodes: the MPR of the source relays the source's message (i.e. re-broadcasts it), then the MPR of this MPR relays it, and so on until the destination is reached. The aim of MPRs is to minimize the bandwidth consumption and find the shortest path between any two nodes. Figure 2 shows how the MPR mechanism avoids bandwidth consumption without degrading the network connectivity.

Some *hybrid* routing protocols using both techniques (proactive and reactive) have been proposed. In such protocols, a node keeps information about some destinations while requesting routes on demand for other destinations. Every protocol presented here is *raw*, in the sense that they do not suspect any malicious behavior from network entities. These protocols work only when all nodes are compliant to the protocol, but security measures are required to cope with malicious behaviors.

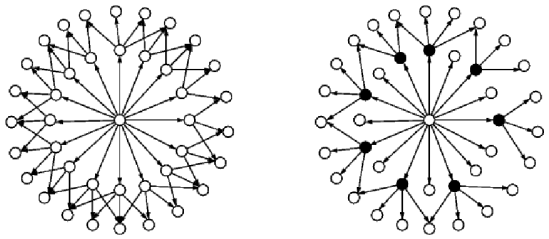

Figure 2: Left: Without MPR. Right: With MPRs (black nodes) [22]

2.4 Security Issues

Even before considering privacy issues, an ad hoc routing protocol should be *secure* and ensure that the routing will be possible even in the presence of attackers in the network. Indeed, ad hoc networks offer a great number of vulnerabilities: without central administration, securing communications becomes challenging. Techniques must be used to ensure integrity and authenticity of control messages. Note that we do not consider security for data (application) messages, as they can be protected with any cryptographic scheme, and we focus on securing control messages (like *HELLO* or TC messages). Attacks on ad hoc routing protocols can be divided in four categories: identity spoofing (impersonation), control message modification (misrelay), false information generation (lying on one's knowledge of the network's topology) and privacy leaking. We briefly present basic techniques to protect against the first 3 categories. Privacy concerns are studied in Sections 3 and 4.

2.4.1 Protection using traditional cryptography

The first two categories can be prevented using traditional cryptography tools like *hash functions*, *encryption* of messages or *signature*. Hash functions ensure integrity by generating an irreversible, unforgeable summary of a message. Upon receiving a message and its hash, one can compute the hash of the received message and compare it to the hash received to detect any modification by an attacker (to the message or the hash received). Encryption simply consists in hiding a message from the entities that do not own the appropriate keying materials. One can encrypt with a symmetric key shared with other communicants, or with a public key from a pair of public/private key (the public key is known from all, the private is secret to the owner). In the latter case, only the associated private key can decrypt the message. On the contrary, to perform a signature, one must use his private key to sign the message. As only the owner has knowledge of this key, the authenticity of the message's sender is proved when the message correctly "decrypts" with the associated public key.

Secure versions of AODV and OLSR, noted SAODV [43] and SOLSR [21], use these tools to secure the routing from the two first categories of attack: identity spoofing and message modification. Basically, every node possesses a pair of public/private keys and uses signature, hash functions and/or encryption to respectively ensure authenticity, integrity and confidentiality of control messages like RREQs or TCs. More precisely, non-mutable fields (i.e. fields that do not change from source to destination) are signed by the source, but mutable fields like the hop count or TTL

values need special protection. For these kind of values, both SAODV and SOLSR use the fact that $TTL_{current} = TTL_{origin} - HopCount$, with TTL the *Time To Live* of an IP packet, decreasing each time the hop count increases. If the source communicates the values $TopHash = hash^{TTL_{origin}}(Seed)$ and $CurrentHash = Seed$ (with $hash^X(\cdot)$ the function $hash(\cdot)$ repeated X times), each forwarding node can check that $TopHash = hash^{TTL_{current}}(CurrentHash)$, increase the hop count, set $CurrentHash = hash(CurrentHash)$, and forward the message. This way, the hop count field is protected from modifications along the way that might try to disturb the routing by lying on the distances. SOLSR also includes a mechanism for the nodes to authenticate their neighbors in order to carefully choose their MPR.

We see that both SAODV and SOLSR suppose nodes are in possession of a pair of keys, thanks to a PKI (*Public Key Infrastructure*) or any other key distribution mechanisms. More precisely, the assumption is made that only *honest* nodes are able to obtain keying materials. But if an attacker successfully compromise a honest node and take control of it, all protections fall. For this kind of threat, traditional cryptography is not sufficient and we need detection and reputation techniques to isolate misbehaving nodes.

2.4.2 Compromised nodes : detection and reputation

In addition to performing identity spoofing attacks and misrelay packets, a compromised node may freely lie on its knowledge of the network and corrupt the routing, as there is no verification whatsoever of the trustworthiness of a node. Two mechanisms are essential to deal with malicious node infiltrated in the network : *detection* and *reputation*.

Concerning detection, solutions exist for AODV [2] and OLSR [1]. In both cases, the protocol is analyzed in details to extract implicit *trust rules*. Based on received and overheard messages, every node can verify the coherence of the information communicated by its peers. For instance in OLSR, if a node X receives from Y the information *Z is one of my neighbors*, and from node Z *Y is not in my neighborhood*, X knows that either Y or Z is lying. The trust-based analysis of OLSR [1] and the counter-measures provided focus on the MPR selection, as it is a crucial point of the protocol. In AODV [2], by overhearing messages in its range, a node is able to approximate its neighbors' routing tables, which is very useful to analyse their behavior. In both protocols, nodes can therefore detect most of message replication or forgery attacks only by maintaining additional information about neighbors. Proposed counter-measures, apart from protocol specific solutions, include isolating the misbehaving nodes when enough nodes judge them malicious.

An alternative to detection and trust management is the reputation system [10]. The idea is that each node maintains ratings about other nodes of interest, and use them to decide whether to trust a certain peer or not, and to include it in the routing process. Reputation of a node is usually represented by two metrics : a trust rating (is the node's telling the truth?) and reputation rating (does the node participate actively in the routing?). These metrics evolve considering *good* or *bad* events performed by the node. Also, nodes permanently exchange their opinion on other nodes. When node N receives information about P from M, it may slightly update its reputation for P. Also, it modifies its trust in M depending on the worthiness and likeliness of the information.

Finally, the last category of attacks in ad hoc networks, privacy leaking, differs from other vulnerabilities. Here we do not only want to protect *the information X is giving to Y*, we want to hide the very fact that *X is communicating to Y*.

3 Privacy and Anonymity

As *privacy* is a rather broad and imprecise term, we clarify relevant notions before describing existing solutions for anonymous (i.e. privacy-preserving) ad hoc routing. Our goals in terms of anonymity are given in Section 5.2, along with the solutions we propose.

3.1 Terminology and Definitions

We refer to Pfitzmann and Köhntopp [34] to define the concepts of *anonymity* and *unlinkability*. Here we take the point of view of an attacker or observer willing to gather as much information as possible on the network.

Definition 1 *Anonymity of a subject (a node) is the state of being not identifiable within a set of subjects (called the* anonymity set*) considering the information available to the observer.*

Definition 2 *Unlinkability of two or more items (e.g. subjects, messages, events) means that it is impossible for the attacker, considering his knowledge of the whole system, to tell whether the two items are related or not. Put another way, the probability that two items are related stays the same before network setup (with the* a priori *knowledge) and after the run of the system (with the* a posteriori *knowledge).*

More precisely, we consider the following properties:

- *Source, Destination and Route Anonymity*: it is impossible for some observer to uncover the source's (or destination's) identity within a set of identities. Route anonymity is the impossibility to find the identity of the relay nodes between a source and a destination;
- *Source/Message Unlinkability*: given a message, it is not possible to find the identity of the source (i.e. sender);
- *Destination/Message Unlinkability*: given a message, it is not possible to find the identity of the destination (i.e. recipient);
- *Source/Destination Unlinkability*: given a sender and a recipient, it is impossible to say whether they communicate together or not (includes source/message *and* destination/message unlinkability);
- *Message/Message Unlinkability*: it is impossible to relate two messages from the same communication;
- *Non-Localization*: at any time, uncovering the geographical or relative location of a node is impossible.

Note that source anonymity is actually equivalent to source/message unlinkability: source, destination and relay nodes anonymity can be expressed in terms of unlinkability. The *non-localization* property is specific to our context where nodes are mobile, and can also be expressed as *node/location unlinkability*. Of course the geographical, absolute location of a node must not be disclosed, but neither should the relative position. For instance, the information *at time τ, X is 3 hops away from Y in* that *direction* should stay secret.

Pseudonymity [34] is another notion defined by Pfitzmann and Köhntopp. It is merely the use of *pseudonyms* to hide identities. Several anonymous routing solutions, including our protocol, use pseudonyms instead of identifiers. Depending on the number of pseudonyms employed by a node and its pseudonym switching frequency, its anonymity is more or less insured: the more pseudonyms a node uses, the better the anonymity is. The simplest version of pseudonymity consists in assigning one pseudonym to each subject (which is equivalent to a unique identifier), and the extreme opposite is using one pseudonym for each *transaction* (i.e. "one-time" pseudonyms). Whatever the choice, every pseudonym in the system must be unique and distinct from all others. A possible way to generate those pseudonyms is to use a pseudo-random function like the keyed hash function HMAC [24] which produces a 160 bits output, with random bytes as keys and public keys of the nodes as values for instance. Furthermore, one can also chose to use public pseudonyms (it is publicly known who owns that pseudonym), verifiable pseudonyms (only a central, trusted authority can link pseudonyms and identities), or unlinkable pseudonyms (pseudonyms and identities are unlinkable). Of course anonymity is stronger with unlinkable pseudonyms, but it may be preferable that some authority keep a trace of who owns which pseudonym, for instance when tracing misbehaviors or abuses is necessary.

Finally, one could imagine even stronger anonymity by realizing *unobservability* [34]. Unobservability is stronger than anonymity: in this case, items (e.g. message, identity, pseudonym, ...) can not even be distinguished from any item at all. One way to implement unobservability in MANETs is for the nodes to constantly emit random noise, which is particularly costly in terms of bandwidth and energy. We argue that the stated properties are already sufficient for most privacy-critical applications.

3.2 Adversarial Models

The level of anonymity (and security) of a protocol always depends on the considered adversarial model. Before analysing performances in terms of security or anonymity, one should define the type of attackers envisaged. Indeed, vulnerabilities appear as the chosen adversarial model gains in capabilities. Several models can be imagined and combined:

- *Active/Passive*: injects or modifies packets/stays silent and tries to gather information on the network by listening to communications;
- *Internal/External*: possesses/does not possess the network's secret cryptographic material (e.g. a secret key shared by all nodes or a certificate emitted by a trusted authority for its public key);
- *Local/Global*: eavesdrops WiFi communication in a limited geographical area/in the whole network.

For example, an active internal global attacker will be able to listen and analyse all communications and traffic, but also inject, replay, modify packets and impersonate nodes everywhere in the network. This is actually the most powerful combination considering the above list. Basically, an active attacker has more capacities than a passive one (but passive attackers do not show their presence and are harder to detect); an internal attacker is by default believed trustworthy and possibly owns cryptographic material an external attack does not; and a global attacker is omnipresent while a local one has a limited view of the network. Note that an internal attacker can be an external one that successfully compromised a legitimate node. Also, between the local and global attacker models is the *collusion* of local attackers: several attackers collude and share their knowledge in order to improve their attack capacity. A collusion of internal nodes on a path might be very dangerous when trying to ensure nodes' privacy, as they can track a message from end-to-end. Note that a collusion of nodes can comprise from 2 nodes to (almost) all nodes, thus resulting in a global unique internal adversary.

Generally an attacker is supposed to have unlimited storage capacity and great computational power, but not infinite (i.e. he can not break cryptographic schemes). Miscellaneous other models have been used in specific cases. For instance, the *honest but curious* model [17] refers to an attacker that will not deviate from the routing protocol, but will try to gather as much knowledge on the system as it can.

4 Anonymous Routing Protocols

Anonymity in traditional networks begins with Chaum's MIX-nets [11]. This solutions relies on specific MIX routers that modify packet's appearances by encrypting and decrypting them when they cross the router, and that reorders incoming traffic. As a result the paths are untraceable and the identity of the source is concealed. Onion routing [18] is another famous solution where the source of a message builds a path through *onion routers* by encrypting its message with several encryption layers. Each layer is meant to be decrypted by a router on the route and contains routing information about the next hop.

In both approaches, a publicly known infrastructure is supposed, along with trusted routers. These assumptions do not hold in MANETs and solutions from traditional networks ought to be adapted to this untrusted, decentralized environment. This section presents several reactive or proactive anonymous ad hoc protocols, more or less efficient according to their given purpose. Each presentation begins by a explanation of the concepts used in the protocol, then the protocol itself is described, and finally we propose a short anonymity analysis.

4.1 Anonymous Reactive Routing

Solutions for anonymous ad hoc routing using the reactive approach are numerous. We have selected some typical and interesting (from an anonymous or technological point of view) solutions: ANODR [23], ODAR [41] and MASK [44].

4.1.1 ANODR

ANODR (*ANonymous On Demand Routing*) [23], created by Kong et al., is among the first propositions for an anonymous routing. It borrows the concept of onion routing [18], where a message M travelling through routers (or nodes) $X_1, X_2, \ldots, X_n$ respectively in possessions of keys $K_1, K_2, \ldots, K_n$ is encrypted by the source to form the following structure: $K_1(K_2(\ldots(K_n(M))\ldots)$. Upon receiving this value, router X_1 decrypts the first layer, forwards the result to the next hop, and so on. Usually information about the relevant next hop is inserted by the source in each layer. The keys K_i can be public or symmetric, but the latter is preferred because of the great computational cost of asymmetric encryption/decryption.

ANODR also uses *trapdoors* to anonymously address the destination of a message. A trapdoor is a one-way function easily computable but difficult to reverse, except for users in possession of a secret parameter. Thus, if only the destination knows the secret parameter, only it can open the trapdoor. Typically, a trapdoor consists in encrypting a message with the public key of the destination: anyone trying to decrypt it with a secret key other than the corresponding one will fail.

Description of the protocol

As a reactive routing protocol, ANODR proceeds by route requests (RREQ) and route replies (RREP) to acquire routes. When a source S wants to communicate with a destination D for whom it does not have a route table entry, it locally broadcasts the following packet:

$$\langle\ RREQ,\ seq\_num,\ tr_D,\ onion\ \rangle$$

Where *seq_num* is a sequence number useful to identify the RREQ (nodes process each RREQ only once and keep track of processed sequence numbers), tr_D a trapdoor designed for D, and *onion* is the onion structure. It is formed as depicted in Figure 3: the source creates it by encrypting the flag *src*, then each node receiving the RREQ appends a nonce N_i, re-encrypts the result with a random symmetric key K_i and rebroadcast it. All nodes receiving the RREQ try to open the trapdoor tr_D, and when one succeeds, it knows it is the destination and sends back a RREP of the form:

$$\langle\ RREP,\ N'_i,\ pr_D,\ onion\ \rangle$$

With N'_i the route pseudonym of node i (in Figure 3, $i \in \{B, C, D, E\}$), pr_D the proof of trapdoor opening authenticating the destination as the one intended by the source (e.g. the plaintext that was encrypted with the destination's public key). The *onion* value is sent back by the destination exactly as it received it. The previous node $i-1$ on the path from the source to the destination, when receiving the RREP message given above, will decrypt the first layer of *onion*, the one it itself constructed. Then it notes in its routing table the correspondence between N'_i and its older nonce N_{i-1} it inserted in the RREQ's onion, generates a new locally unique nonce N'_{i-1}, and forwards the message with N'_i replaced by N'_{i-1}. Other relay nodes proceed in the same way. When the source receives then RREP (with the original onion it sent), it knows a route has been found. By checking the proof of trapdoor opening, it can make sure the destination is the one intended. Figure 3 shows the evolution of the onion (named *Trapdoor Boomerang Onion* by the authors) when A requests a route to E.

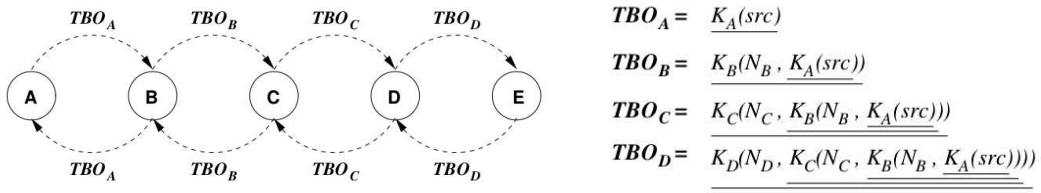

Figure 3: Trapdoor Boomerang Onion during path discovery from A to E in ANODR [23]

Once the source has received $\langle\ RREP,\ N'_1,\ pr_D, K_0(src)\ \rangle$ it can then use the route pseudonym N'_1 to send data to the destination. The first relay node will recognize its nonce N'_1 and forward the message, replacing N'_1 by N'_2, etc. Route maintenance (i.e. management of broken links due to mobility) simply consists in sending a RERR (*Route Error*) packet to the previous hop on the route when a next hop disappears (detectable with link layer acknowledgments for instance).

Anonymity Properties

ANODR ensures that the identities of all nodes are concealed: they use a trapdoor for the destination, and the source and relay nodes' identities never appear (consequently source- and destination-message unlinkability are respected). However plenty of other vulnerabilities are inherent to the protocol. In particular, message/message unlinkability is not ensured as a route is for 1 source and 1 destination only, relay nodes can link two messages from the same communication. Nodes' location are not protected. Due to the fact that RREQs and RREPs contain constant fields, it is very easy to trace control and data messages in the network and localise the source, relay and destination nodes. Moreover, route pseudonyms N'_i being constant, is is very easy to link messages from the same communication, and even find the whole route. To avoid this, ANODR proposes several measures to modify packets' and route pseudonyms' appearance.

However, an active attacker can still replay a RREQ multiple times in order to approximate the destination's location and the number of hops of a route: RREQs still have constant fields and their *onion* structure is growing in size at each hop. This attack is even easier for a collusion of internal attackers. The authors evaluate the probability of a collusion of internal attackers tracing a route correctly: it mostly depend on the length of the route, the number of attackers and their relative position (i.e. if they are consecutive en route, probability of success is higher). Also, traffic analysis attacks are possible, and in particular, *timing analysis* is a side-channel an attacker can use to infer knowledge. The timing attack is based on the assumption that packets sent by a source arrive in the same order at the destination (in average). In order to prevent traffic analysis, ANODR proposes that each node (i) re-order its packets as a MIX router [11], and (ii) injects dummy traffic. These solutions have a great cost in terms of performances. More generally, anonymity and performances are two components hard to reconcile. Note that the above mentioned traffic analysis attack are hard to address, and most protocols are vulnerable to some or all of them (timing, flooding, predecessor, message size, message coding attacks).

Several other ad hoc routing protocols chose to use an onion structure [40, 7, 38], in a more or less elaborate way or for slightly different purposes. In general, their flaws and strengths are similar to ANODR: identities are concealed (except in SDAR [7]) but the onion structure and possibly constant fields in packets help tracing messages and inferring information such as nodes' locations.

4.1.2 ODAR

ODAR [41] proposes a completely different approach using Bloom Filters. Described for the first time in [3], this data structure is a very light and efficient (in terms of space and time). Although a Bloom filter has a fixed size, it contains elements that can be added dynamically and which presence can be easily tested. This structure is represented by a m-bits array and k hash functions such that $h_{i\in 1..k} : \{0,1\}^* \rightarrow [0..m-1]$ (m and k fixed at its creation). To add an element e, one should compute $v_i = h_i(e)$ for all $i \in 1..k$ and put each v_i^{th} bit of the Bloom filter to 1: $\forall\, i \in 1..k,\ BF[h_i(e)] \leftarrow 1$. To test the presence of an element e, compute $h_i(e)$ for all $i \in 1..k$ and test if *all* corresponding bits are 1: $\vee_{i\in 1..k}(BF[h_i(e)]) \stackrel{?}{=} 1$. This structure is efficient and dynamic, but, by construction, very sensible to false positives. Indeed, when too many elements are inserted, an element can be detected as present while never actually added (at some point, all bits are eventually set to 1). Consequently, a Bloom filter is also characterized by a *false positive rate* P_{fp} and a maximum number of elements n it can contain without exceeding the threshold. Hence, one should carefully chose the m, k and n parameters. A study [6] evaluates P_{fp} as a function of m, k and n and prove their formula[1].

Description of the Protocol

With this tool, ODAR realizes *source routing*: the source specifies the route when sending its messages, and relay nodes do not need to look in their routing tables, they just check if they are specified as forwarder in the message. Basically, when a node S wants to communicate with a destination D, it broadcasts a RREQ with an empty Bloom filter. Nodes receiving it add a keyed hash of their identities in the Bloom filter, with a constant secret random number as key. They process each RREQ only once, i.e. they process a RREQ when they are not already in its Bloom filter. Destination recognizes itself thanks to a trapdoor, sends back the RREP with the filled Bloom filter that contains the actual path from S to D (nodes on the path relay the RREP back to S). S records the Bloom filter in its routing table. Afterwards, when S sends a data message to D, it attaches the Bloom filter so that relay nodes know they must re-broadcast it.

To authenticate each other and at the same time create a session key (used to protect communication and address each others), S and D use the *Diffie-Hellman* key exchange. The source includes a temporary public key $Y_s = g^{X_s} \, mod \, q$ (g and q are public, X_s is S's temporary private key). Thanks to a trusted *Key Server* delivering certified keys, S gets D's long term public key $Y_D = g^{X_D} \, mod \, q$ (X_D being D's private key) and generates the session key $K = Y_D^{X_s} = g^{X_s X_D} \, mod \, q$. D can also compute

[1] $P_{fp}(m,k,n) = \frac{1}{m^{k(n+1)}} \sum_{i=1}^{m} \left(i^k \left(\begin{array}{c} m \\ i \end{array} \right) \sum_{j=0}^{i} (-1)^j \left(\begin{array}{c} i \\ j \end{array} \right) j^{kn} \right)$, according to [6]

$K = Y_s^{X_D} \, mod \, q$. The trapdoor is constructed with K, that only S and D are able to compute. The *Diffie-Hellman* exchange is believed secure: given g, q, $g^x \, mod \, q$ and $g^y \, mod \, q$, it is hard to find $g^{xy} \, mod \, q$.

Anonymity Properties

ODAR intends to provide nodes (source, destination, relay) anonymity and non-localization. The first point is almost achieved: relays' identities are securely hashed with secret numbers, the source's identity is never divulged (and Y_s is not linkable to ID_S). However, ODAR leaves the destination's identity is in clear when the source requests D key Y_D to the Key Server. Plus, although it is not among the goals, we note that message/message unlinkability is not respected: the Bloom filter is constant during the communication between S and D, allowing any observer to relate two messages: the Bloom filter actually identifies the route. More serious, by replaying RREQs, any attacker can approximately locate a destination (similar strategy works with RREPs for the source), and a node receiving a RREQ with an empty Bloom filter knows the source is 1 hop away: location anonymity is not achieved

Remark that, even if identities of relay nodes are not divulged, the fact that a node N uses a constant secret number $Rand_N$ to securely hash its identity entails that the value $keyed\_hash(Rand_N, ID_N)$ it inserts in the Bloom filters uniquely identifies N in the network. To counter this, ODAR merely recommends the usage of *multiple* secret numbers $Rand_N$ so that a given node possesses several keyed hash identities.

The Diffie-Hellman exchange ODAR uses is very famous but is vulnerable to the *man-in-the-middle* attack: an attacker could place himself between S and D and perform a Diffie-Hellman exchange with S and another with D. He can then decrypt every message passing through with his shared key with S and re-encrypt them with the one shared with D: that would be transparent for the two parties, and the attacker could read all exchanged data. Here however, the $KeyServer$ is trusted and Y_D can be certified (with a *Certification Authority*) as D's key, meaning D is the only one knowing X_D, the only one capable on computing $K_{sD} = Y_s^{X_D} \, mod \, q$, and thus the only one capable of opening the trapdoor.

4.1.3 MASK

As a last example of reactive routing, we will present MASK [44], which has the particularity to incorporate an anonymous proactive neighborhood discovery and authentication, a mechanism we will borrow in our proposition. To achieve *anonymity* and *authentication*, MASK uses *pairing* [31]. Let $\mathbb{G}_1$ be an additive group and $\mathbb{G}_2$ a multiplicative group, both of prime order q. Let $P, Q \in \mathbb{G}_1$. A pairing is a map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ satisfying the following property: $\forall a \in \mathbb{Z}_q^*$, $\hat{e}(aP, Q) = \hat{e}(P, Q)^a = \hat{e}(P, aQ)$. The application $\hat{e}$ also has to be efficiently computable, for practical purposes. The Weil pairing, based on elliptic curves, is an example of such maps. The security of bilinear maps relies on the *Bilinear Diffie-Hellman Problem* (BDHP): computing $\hat{e}(P, P)^{abc}$ when $P, aP, bP, cP \in \mathbb{G}_1$ are given is known to be hard. The reader might refer to [31] for more details.

Description of the protocol

To authenticate, two neighbors N and M do not need to reveal their identities and use pseudonyms instead. To discover its neighborhood, N broadcasts $\langle\, PS_N,\, Nonce_N \,\rangle$. His neighbor M answers with $\langle\, PS_M,\, Nonce_M,\, hash(K0_{MN}||Nonce_N||Nonce_M||0) \,\rangle$. To finalize, N sends back $\langle\, hash(K0_{MN}||Nonce_N||Nonce_M||1) \,\rangle$. The two hash values are used as *proof* to the other that we were able compute the key $K0_{MN}$. This key is computed thanks to a (publicly known) bilinear map: $K0_{MN} = \hat{e}(gPS_M, PS_N) = \hat{e}(PS_M, gPS_N) = \hat{e}(PS_M, PS_N)^g$. Values $g \times PS$ are *secret points* given to the nodes at the same time they are given a set of collision-resistant pseudonyms PS: before network setup, a server generates multiple sets of pseudonyms, picks a secret number $g \in \mathbb{Z}_q^*$, multiply pseudonyms by g and distribute pseudonyms and secret points to every node. Thus pseudonyms are *verifiable* by the server, in the way described in Section 3.1. After authentication, M and N generate sets of *link identifiers* $\{L^i_{MN} = hash(K0_{MN}||Nonce_N||Nonce_M\ ||2 \times i)\}$ and symmetric keys $\{K^i_{MN} = hash(K0_{MN}||\ Nonce_N||Nonce_M||2 \times i + 1)\}$ $(i > 1)$ that enable them to address each other (link identifiers become their somewhat shared addresses) and to secure their 1-hop communications. Each i^{th} link id and key are used together and only once (after 1 utilization, they are discarded). For simplicity we note L_{MN} and K_{MN} without giving the rank i.

When nodes are authenticated, they use their keying material to discover routes and to communicate. RREQs contain: a *RREQ_id* very similar to the *seq_num* field in ANODR, the destination identity in clear, and the source's current pseudonym. Each relay node notes the pseudonym specified in the RREQ it received, replaces it by its own pseudonym and re-broadcasts the RREQ. The destination recognizes its identity in the RREQ and answers with a RREP of the form $\langle\, LinkID,\, K(ID_D) \,\rangle$: the link identifiers ($LinkID$) and shared keys ($K$) generated after authentication are used at each hop to protect the message. Each relay node knows where to send back the RREP as they noted the pseudonym of their predecessor upon receiving the RREQ. While the RREP travels back, relay nodes insert in their *Forwarding Route Table* (FRT) the entry $\langle\, ID_D,\, preLinkID,\, nextLinkID \,\rangle$, with $nextLinkID$ its *next-hop* on the route (i.e. the node from which it received the RREP) and $preLinkID$ its *pre-hop* (i.e. its predecessor, from whom it received the RREQ). D marks that the link id shared with his predecessor as the end of a route towards itself, so as to recognize incoming messages from this link id as addressed to itself.

After a route discovery process, a source often possesses several routes to a destination, for example if there was several responses to its RREQ (any node in knowledge of a route to D can answer the RREQ). Relay nodes also have multiple routes, because they might have contacted D in the past or because they were relay nodes towards D for another source. Anyhow, when the route has been set up, to communicate with D, the source chooses one of its $LinkID$ associated to ID_D in its FRT and sends data using the link id and key shared with its *next-hop* on the constructed path to D: $\langle\, L_{nexthop},\, K_{nexthop}(payload) \,\rangle$. Relay nodes forward packets according to their FRT until D is reached: if the value of the $L_{nexthop}$ field in a received message is equal to a $preLinkID$ in their FRT, they must relay the message. For this, they randomly choose a $nextLinkID$ among those pointing toward D (i.e. those in an entry with ID_D). As a result, the forwarding is probabilistic and several paths between S and D are possible (see Figure 4, where two packets take two different paths). Route maintenance merely consists for a node in sending a RERR packet to all its predecessors when it loses all its $nextLinkID$ (a link id can be lost if neighboring nodes move away from each other).

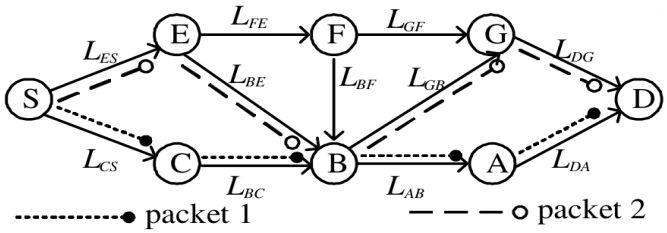

Figure 4: MASK anonymous packet forwarding from S to D [44]

Anonymity Properties

MASK considers two types of attackers: external active global or internal active local (possibly collusive) attackers. First of all, pseudonyms efficiently hide identities of the nodes, and they are not linkable to nodes' identities except by the server who generated them. But it is trusted, and we prefer *verifiable* pseudonyms rather than totally unlinkable ones (see Section 3.1 on pseudonyms). The neighbor authentication is thus anonymous and leads to the generation of very useful keying materials.

Replay attacks in MASK are difficult to perform: replay a RREQ is useless, as *every node* in the networks replays it and replay a RREP is impossible for an external adversary. Indeed, the one-time link ids prevent any replay of RREP and data messages from an external attacker. For the same reason, tracing a message in the network is impossible to an external attacker: link ids do not mean anything to them, and the appearance of the messages change at every hop thanks to per-hop encryption. Moreover, because different paths are taken between a source and a destination (even during the same communication) and because the forwarding is probabilistic, the task become very uneasy for an external attacker. All the stated properties, if we add that nodes re-order packets, lead to a MIX-net [11] where nodes change packets appearance and are able to optimize the routing as they know several paths to a destination (when on break, they use another). MASK indeed propose that nodes wait a random delay before forwarding messages in order to thwart timing attacks. However, the protocol could be vulnerable to *message size* attacks (trace a message thanks to its size), but the authors add a random padding changing at every hop that avoids external attackers to perform it (at the cost of more bandwidth consumption).

Most of the above stated protections are effective, but only against external attackers. Internal attackers colluding can replay RREPs, locate the source and trace data messages, even if the *multipath* routing property mitigates this possibility. More generally, by replaying multiple times and following data messages, internal attackers can probabilistically find the source and destination nodes' location. Indeed, S and D *always* receive these messages: in average, they receive them more than any other node.

Although internal attackers can find nodes' location, the identities of the source and relay nodes are protected against all attackers thanks to pseudonyms. Only the destination anonymity is not achieved: ID_D is clearly mentioned in the RREQ. The authors [44] argue that, although it is an infringement to the destination's privacy, the benefits of this choice are not negligible, and the cost of a traditional trapdoor-based solution is too great for the route discovery process. Indeed, a trapdoor often involves asymmetric cryptography operations that *every* node in the network must perform (to test the trapdoor), while latency should be minimal in the route discovery process.

Plus, inserting the identity of the destination in the FRT of the nodes allows MIXing, route repairing and routing optimization.

In spite of relay nodes knowing the destination, the message/message unlinkability is ensured, even against internal colluding attackers. For instance in Figure 4, when B forwards a message received from C on L_{BC}, it can not know if which of S or C is the source. Consequently, message/source and source/destination unlinkability are provided, but not the destination/message unlinkability.

4.1.4 Comparison

We have presented here 3 representative reactive routing protocols, one using onion routing, one using Bloom filters, and another atypical one providing a way to anonymously secure 1-hop communications. Table 1 summarizes the anonymity properties provided by each protocols. A "X" in a cell denotes a full respect of the property by the protocol, and "/" a partial protection (e.g. not against internal attackers).

Protocol	ANODR	ODAR	MASK
Source anonymity	X	X	X
Destination anonymity	X	X	
Route anonymity	X	X	X
Source/Message unlink.	X	X	X
Destination/Message unlink.	X	X	/
Source/Destination unlink.	X	X	X
Message/Message unlink.			X
Source non-localization			/
Destination non-localization			/
Relay nodes non-localization			X

Table 1: Comparison of privacy in ANODR, ODAR and MASK

Another analysis and comparison of protocols ANODR [23], ASR [45], ARM [38], MASK [44] and SDAR [38] are available in [4].

4.2 Anonymous Proactive Routing

In the same format as the last section, we present 2 anonymous proactive routing protocols, one from Nezhad *et al.* [27], and another by Robert and Bidan [37].

4.2.1 V-Routing

The protocol from Nezhad *et al.*, V-Routing [27], bears this name because of the *triangular* virtual path it builds to ensure nodes' anonymity. Indeed, it is an overlay protocol that runs over a standard proactive protocol as OLSR [22]. It intends to provide non-localization and anonymity only to the end nodes of a communication (i.e. the source and destination nodes of a *current* communication). The V-Routing protocol proposes a *weak privacy* version where destinations trust at least one node in the network and a *strong* one where this assumption is suppressed. We first present the weak version, as the strong version is a simple extension.

Description of the protocol

In the weak version, the network model supposes the existence of routers (or *access points*) which do not need to hide their identities. They are willing to fully participate in the routing process, and are going to generate and relay topology updates of the underlying protocol. Other nodes, called *ordinary nodes* or *clients*, do not participate in the underlying routing protocol. Each client names its *access router* (denoted AD for node D, AN for N, ...), a neighboring router which it trusts (it shares a secret key with it), and it identifies itself to it. They communicate together through a secure link layer channel. All routers are known in the network, even by clients, whereas clients are silent and no one even know they are in the network except their access router. All nodes are possibly moving in the network.

In a first phase, each destination node D (i.e. all nodes) establishes a virtual path to itself: it builds the path allowing other nodes to contact it, without anyone knowing its location. For this purpose, it picks a router as his *Rendezvous Point* (RP), depending on characteristics left to the node's discretion (distance, trust, ...). All messages intended to D will first travel through RP_D, hence the triangular path S-RP_D-D, with S-RP_D the *first leg* and RP_D-D the *second leg* of the path. This RP can change over time and a node may pick several RP at a time. In any case, D ought to know the certified public key of its RP to avoid id spoofing (possible with a public key infrastructure delivering keys and guaranteeing the identity of the owner).

Then, D has to set up the 2nd leg path *manually*: he sends notifications to several routers of his choice, noted HD for D, which will be intermediaries forming a virtual path to bring messages from RP_D to D. These routers do not need to be 1-hop away from each other, they can find their next hop because routers are known in the network thank to the underlying protocol. The aim of this manipulation is to thwart traffic analysis attacks by making the packets roam in the network though an unknown number of relay routers. It also avoids RP_D to know D's access router AD. When an RP is chosen, and if it accepts its role, it floods a I_AM_RP message in the network to advertise that to contact D, one should first contact RP_D (this message contains RP_D and D's identities in clear).

Afterwards, any node S can contact D by addressing to RP_D a message of the form: $\langle\, ID_{RP_D},\, PK_{RP_D}(D),\, PK_D(S, payload) \,\rangle$ (with $PK_X(\cdot)$ a encryption with X's public key). RP_D receives this message thanks to the underlying protocol, and sends $\langle\, ID_{HD_1},\, PK_{HD_1}(D),\, PK_D(S, payload)) \,\rangle$ to HD_1. Each HD_i then forwards this message to HD_{i+1}, and eventually, AD gives the message to D (at the link layer). Figure 5 shows how V-Routing builds a triangular virtual path on top of a standard routing protocol.

This weak version, which supposes nodes willing to reveal their identities and help others, is adapted to an infrastructure like Mobile IPv6. In the strong version, nodes do not trust anyone, and each node is a router. Each node uses an *alias* instead of its real identity. It uses it to exchange topology information in the underlying protocol with all other nodes. The rest is similar except there is no more access routers: the last HD on the second leg forwards messages to D's alias in the network, and every node can be a RP or a HD.

Anonymity Properties

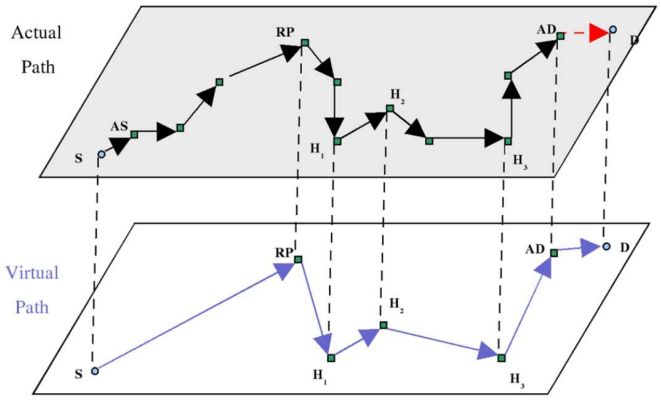

Figure 5: V-Routing Virtual Triangular Path [27]

V-Routing only aims at preserving communicating nodes anonymity and non - localization. This means that nodes do not need to be actually anonymous. Indeed, every node D reveals its identity in clear to its HDs, and RP_D even broadcasts ID_D. Then, not considering AS, the identity of source S is concealed because it is encrypted in all its messages all along the path to D. However, the identity of a message's destination is known of many internal actors as RP_D, HDs and AD (but not from external eavesdropper). Locations of source and destination nodes are hidden because they either do not flood topological information in the network in the weak version, or they use an alias instead of their identities in the strong version. From the point of view of the HDs, D's location stays unknown because they never know whether they are the last hop to AD in the weak version or to D in the strong version, they can only suspect it.

Packet tracing is prevented thanks to per-hop encryption at the link layer (the IEEE 802.11 standard on the wireless physical layer allows it) and to HDs that act as MIXes when they work for several destinations. But internal attackers acting as routers can still perform this kind of attack, especially virtual hop routers and RPs. The authors [27] discuss the chances of locating D. Basically, an attacker can intercept all *forward requests* sent by D and compromise all corresponding routers in order to locate AD (and D subsequently), but in practice this attack is hard to perform: the global eavesdropper can easily miss a forward request, compromising a node or obtaining its keying material is not that easy, etc. The probability to locate AD and D can be evaluated to $(pq)^n$ with p the probability of capturing all forward requests, q the probability of deciphering these messages to uncover HDs' identities, and n the number of HDs. This overall probability decreases when n increases, but performances degrade in the same time.

V-Routing is an example of the specific case where destination/message unlinkability and source/message unlinkability properties are broken (respectively by AD, RP_D and HDs, and by AS), but destination/source unlinkability is respected: no one has both the knowledge of the source *and* the destination of a message. However message/message unlinkability is not respected at all, RP_D can break it for example. Generally speaking, in V-Routing RPs, access routers and virtual hop routers have very privileged positions, to launch attacks by replaying packets for example. In the strong version for instance, when D selects a RP_D and gives it its real identity, by colluding with internal attackers the RP may link D's alias with its identity (easy because 1 node has only 1 pseudonym

which uniquely identifies it). However, performing a successful attack requires a very strong attacker model: all HDs must collude, and a global eavesdropper is needed.

To sump up, V-Routing assumes a lot about the network (shared secrets between D and AD, certified keys, enough routers in the network so that everyone can select an access router, RPs supposed honest, ...), but as a result, their privacy goals are met.

Nezhad *et al.* also designed an anonymous network topology discovery protocol for *Wireless Sensor Networks* (WSN) [26]. This kind of network, though close to ad hoc networks, differs on several points: devices are usually smaller (in size, memory and battery), and the network includes a special node called *sink* or *base station* that collects data from the sensors. The goals of the authors is to allow the sink to know the complete topology of the network, without allowing an attacker to know *who* or *where* is the sink. This problem is different from ours and we will not further describe this solution.

4.2.2 OLSR-based Anonymous Proactive Routing

Another work [37] proposes to extend the OLSR protocol in order to hide the identities and locations of communicants. More precisely, the paper makes the assumption that all nodes are willing to reveals their identities, but intends to hide the source and destination of a communication. The objectives are given in the form of 3 *problems* to solve: (i) conceal the source, (ii) conceal the destination and (iii) prevent an attacker from linking two flows of the same communication (this last objective is close to message/message unlinkability).

Description of the protocol

As mentioned before, the presented approach uses OLSR as an underlying protocol: all nodes flood their knowledge of the network, and each participant knows the complete network graph. In OLSR, the forwarding mechanism is based on MPRs: each node chooses a few neighbors to be their MPRs, i.e. to relay their messages. As a result, a path from a source to a destination is a sequence of MPRs. This last point is also true in the solution of Robert *et al.*, but, unlike OLSR which use next-hop addresses in packet, here it is the source that builds a path: they use *source routing*. However, in order to prevent an attacker to learn the relay nodes of a flow and to address problem (iii) while allowing the relevant relay nodes to be aware of their role, the authors use *ephemeral forwarding tags* and *ephemeral private identifiers*.

Ephemeral forwarding tags are Bloom filters containing the ephemeral private identifiers of the chosen relay nodes (that have to be MPRs to respect the description of OLSR). Ephemeral private identifiers are constructed by the source in such a way that only the source and the addressed relay node R can compute it and know R is supposed to forward. Thus, when receiving a packet, a relay node looks into the Bloom filter for its ephemeral private identifier, and relays the message only if it finds it. The source constructs the private identifiers with a Diffie-Hellman key agreement. Indeed, each node i owns a private key d_i and a public key $PK_i = d_iQ$ (Q publicly known) certified by an authority that the source is supposed to know. Also, the source issue, for each packets it sends an *ephemeral flow identifier* $Flow_{id} = sQ$ (with s a random

secret number chosen by the source). Then, the source computes the ephemeral private identifiers of the chosen MPR nodes on the path: for a node i the private identifier is $sP_i = s \times d_i \times Q = d_i Flow_{id}$, with sP_i computable by the source and $d_i Flow_{id}$ computable by the relay node. As there is no other node that can compute this value, no one except the addressed relay node know it must relay. Thus, the problem (iii) is solved with this mechanism.

Then, to address a destination while concealing it, as required by problem (ii), the source also inserts the destination in the ephemeral forwarding tag. More exactly, the source does not insert sPK_D in the bloom filter, but $sP\bar{K}_D$, with $P\bar{K}_D$ the bit-wise negation of D's public key. Thus, the destination knows when a packet is intended to it. The author, like those of MASK, deprecate the use of a trapdoor to address the destination because of its great cost. However, problem (ii) is not completely solved by inserting the destination in the ephemeral forwarding tag: as is, the forwarding stops at a MPR of the destination, leaving eavesdropper a good indication on where and who is the destination. In order to thwart this, a source can add 0, 1 or 2 hops after the destination, i.e. the source inserts 0, 1 or 2 more MPRs in the forwarding tag. This way, an eavesdropper can not be sure of who was the *real* last MPR.

Finally, to address problem (i), the authors argue that concealing the source from a global eavesdropper nearly impossible (except at the cost of sending periodic dummy messages) and focus on the case of the local observer. Their local attacker model supposes that the attacker observes one MPR, and tries to find the source by listing all paths that lead to this MPR. In order to increase the number of potential sources, the authors propose to drop the shortest path feature of OLSR. Indeed, by constructing a non deterministic path longer than the optimal path, local attackers can not use the assumption that the source chose the most direct path, and have to consider more nodes as potential sources. However, note that this defense is inefficient if the source is not an MPR and if the local observer is located just aside of it: the local observer will overhear a non-MPR node emit a packet, meaning it has to be a source.

Anonymity Properties

In the paper, the authors evaluate how well the goals stated through problems (i), (ii) and (iii) are fulfilled. They consider two models of attacker: a external global attacker that is aware of all communications (it knows who sends what to who for all 1-hop communications thanks to the identities in clear in OLSR packets), and an external local observer that has the same capacities, but only in the delimited area of its neighborhood.

In order to evaluate the relative leaked information for the omniscient attacker and the local observers, the authors used a MATLAB program simulating their solutions. The program runs a simulation of the protocol with 128 nodes and outputs metrics on the leaked information, i.e. metrics that quantify the probability for an attacker to infer the source, the destination or the path taken by a packet. To evaluate the leaked information, the authors use Shannon's entropy: the leaked information after an experiment is $H(\mathcal{A}) - H(\mathcal{U})$, where $H(\cdot)$ is the Shannon entropy function, $\mathcal{U}$ is the set of all nodes and $\mathcal{A} \subset \mathcal{U}$ the anonymity set containing the potential actors of the experiment as far a the attacker knows (e.g. the potential sources or destinations). The authors are interested in measuring the *relative leaked information*: $\frac{H(\mathcal{A})-H(\mathcal{U})}{H(\mathcal{U})}$ which

gives the importance of the leaked information compared to the *a priori* knowledge of the attacker. To compute the Shanon entropy of a set of identities, one needs to find the probability distribution of the identities. To be more precise, if we consider an attacker willing to discover the destination of a message for instance, the entropy is

$$H(\mathcal{A}) = -\sum_{i \in \mathcal{A}} p_i . log\, p_i$$

with p_i the probability for the node i to be the destination (with respect to the attacker's knowledge). If $H(\mathcal{A}) = 0$, it means that the destination is identified with a 100 percent probability.

Thus, authors always evaluate the anonymity set and its probability distribution for each proposed measure to improve the privacy of the nodes. Their results show that adding 0, 1 or 2 MPRs after the destination reduces the relative leaked information by 20% (compared to the basic OLSR protocol) for an *omniscient* observer, resulting in anonymity sets comprising between 11 and 18 nodes. For local observers, anonymity sets vary between one third and one half of the total number of nodes. The measures aiming at concealing the source of a communication make the relative leaked information drop by 70% for dense networks.

These figures show that the proposed measures are (very) effective and that the stated goals are met. However, the model could be more realistic, and several hypothesis are needed to achieve anonymity. Furthermore, the source is not concealed against omniscient observers (except if the source is a MPR) and local observers can uncover the source if it is close to it. Also, traffic analysis attacks are easy to perform in order to locate the source or destination. The paper do not consider internal attackers that could replay packets and see where the flows concentrate in order to locate the destination of a communication. At last, we argue that the anonymity goals themselves are not strong enough (e.g. all identities are known and omniscient observers know every message exchanged by all neighbors). As the title of the paper suggests, the solution only provides *some* privacy. The solution is thus a trade-off between efficiency and privacy, with a preference for efficiency.

4.3 Conclusion

In this section we have presented several approaches achieving anonymous routing in MANETs. Each solution has pros and cons, and depending on the aim of the authors, it is more or less efficient to preserve nodes' privacy. Anyhow, we note that much less efforts have been focused on designing anonymous proactive routing protocols: mostly *reactive* solutions exist. Furthermore, the proactive solution from Nezhad et al. has to make many assumptions in order to achieve anonymity, and both proactive approaches only hide the communicants, the identities of the network members being publicly known.

At first it may seem somewhat hazardous to create an anonymous *proactive* routing protocol, as every node has a complete knowledge of the network, identities and routing information should be harder to keep hidden from nodes. However, this assumption can be mitigated. In particular, with the advent of (fully) homomorphic cryptography, one can now process encrypted data without accessing it. This property is very useful, and

the APART protocol make use of it to preserve users' privacy, with as less assumptions as possible.

5 The APART Protocol

In this section we present our contribution: the APART (Anonymous Proactive Ad hoc RouTing) protocol. First we introduce the concept of fully homomorphic cryptography and we present our anonymity goals along with our network and adversarial models. After which, an anonymity analysis of the protocol is proposed, and we detail the (still on-going) simulation of the protocol and the tools used.

5.1 Introduction to (Fully) Homomorphic Cryptography

The idea of processing data without actually accessing it emanated shortly after the creation of RSA. The basic version of the cryptosystem being multiplicatively homomorphic, one can compute a ciphertext encrypting the product of two original plaintexts, given the encryption of these two plaintexts. In other words, given a RSA public key (N, e) and two ciphertexts $c_1 = m_1^e \ mod \ N$ and $c_2 = m_2^e \ mod \ N$, the product $C = c_1 \times c_2 \ mod \ N$ gives the encryption of $(m_1^e \times m_2^e) \ mod \ N = (m_1 \times m_2)^e \ mod \ N = C$. After noticing this property, Rivest, Adleman and Dertouzous suggested that a *fully* homomorphic scheme was possible, i.e. a scheme capable of computing *any function* on encrypted data.

The first homomorphic schemes were either additive or multiplicative: only one of the operations could be performed on encrypted messages. Pailler's cryptosystem [30], for instance, can only compute the addition of two encrypted messages. Then, advances in this field allowed to compute both operations in the same cryptosystem. As any complex operation is a composition of these two basic operation, we call these kind of schemes *fully homomorphic*. One of the first fully homomorphic systems was invented by Boneh et al. [5]. It supported an unlimited number of additions but only one multiplication. The main limitation restraining the number of operations came from the *noise expansion* of the ciphertexts. Indeed, because cryptographic systems are more secure when *probabilistic*[2], randomness is incorporated in homomorphic ciphertexts. Thus, upon adding and/or multiplying two ciphertexts, the random parts are added or multiplied along with the plaintexts. Thus, the *noise* of ciphertexts grows, and when it becomes too important, the decryption operation fails (e.g. because the noise exceeds a modulo). This is the main challenge Gentry overcame in 2009 with his scheme: he solved the noise expansion problem, allowing any function to be computed.

The key is to *recrypt* (refresh) ciphertexts periodically by encrypting the noisy ciphertext under a new secret key, and then *homomorphically decrypt* it. This idea, called *bootstrapping*, is the key for a fully homomorphic scheme: using the decryption function homomorphically in order to reset ciphertexts' noise. However this solution implies that the *Decrypt* function is accepted as an evaluable function. To fulfill this constraint, *Decrypt* needs to be *squashed*, i.e. simplified to lower its complexity in terms of number of operations. Bootstrapping leads to a fully homomorphic encryption: as the noise is

[2] A probabilistic scheme ensures that two encryptions of the same message do not look alike, in such a way that no one can say whether two ciphertexts encrypt the same message of two different messages.

no longer a problem, any function of any complexity can be homomorphically computed. Very recent work [9] propose a fully homomorphic scheme without bootstrapping (using *modulus switching*), allowing reasonable complexity for evaluated function. Detailed information about Gentry's scheme(s) can be found in Gentry's thesis [13], however for a simpler and more intuitive approach, article [14] makes a metaphor with a jewelery store owner *Alice* that would like to allow its employees to work on precious materials while those are locked into a box.

Technical concerns such as semantic security or correctness (evaluation of a function on encrypted plaintexts outputs the right result) are not presented here. These are discussed in papers [14, 42, 9]. Although it is not our main concern, complexity and performances (and thus execution time) of these implementations must be considered. Lauter et al. [25] have implemented Brakerski and Vaikuntanathan's scheme[3] [8] (without the bootstrapping part) in order to test its performances. For a security of roughly 200 bits, meaning that 2^{200} atomic operations are necessary to an attacker to break the scheme (which is more than enough, 80 bits being the norm), the size of the public key is 29KB, the secret key is 14KB and the ciphertext of one bit (the scheme encrypts one bit at a time) is between 29 and 44KB. On a laptop with Intel Core 2 Duo processor running at 2.1 GHz, with 1GB of memory, key generation takes 250ms, encryption of one bit takes 24ms, decryption is between 15 and 26ms, addition is less than 1ms, and multiplication (more complex) takes 40ms. We note that the sizes of the keys and ciphertexts are very important, but on the other hand, the measured times are quite reasonable, compared to other optimized schemes. Another experiment [15], by Gentry et al. in 2012, tried to homomorphically evaluate the AES circuit. According to their results, one AES encryption with a homomorphic AES circuit takes between 34 and more than 60 hours, which is unacceptable. We see that in *real-life* examples, homomorphic cryptography is still inefficient, even though memory is not a problem anymore. In our context, where performances are crucial so as to minimize latencies in communications, these computation times are way too high. However, fully homomorphic cryptography is still an incipient technology, and considering the blazing fast evolution on this field, we can hope to have reasonable time and memory costs in a few years.

Applications

The main and general application for fully homomorphic encryption (FHE) is to delegate heavy computations to computational resourceful structures like clouds, without giving them access to the data. For instance, electronic voting can be implemented with FHE. Some works also investigated the possibility of performing machine learning on encrypted data [19], always with the idea of delegating heavy computations without disclosing the contents.

5.2 Models & Anonymity Goals

Before describing in details our protocol, this section states the network and adversarial models, along with our anonymity goals.

[3]This scheme was designed in 2011, and more efficient solutions have been proposed since then.

5.2.1 Network Model

We focus on wireless mobile ad hoc networks (MANETs), where nodes communicate over a WiFi medium with a limited range. We only consider bidirectional links between neighbors (i.e. if a node X hears a node Y, then Y hears X). Each node has a unique identifier. Our protocol runs over layer 2 or layer 3 protocols, and we do not propose anonymity measures for these layers, although the whole protocol stack should protect the nodes' identities. In particular, we assume that the nodes are capable of functioning in *promiscuous mode* in order to listen to all incoming traffic, and that their MAC address can be set to the broadcast address (i.e. `FF:FF:FF:FF:FF:FF`).

At last, we suppose that prior to the network setup, nodes willing to communicate exchanged their respective *real* identifiers and long term public keys. Indeed, each node X has several types of public keys: one unique long term public key PK_X^{perm}, and several public keys PK_X (as many as necessary) that it uses successively. The latter keys are not linked to the node's identity. Also, each node can generate pairs of keys for one-time usages, noted PK_X^{tmp} and SK_X^{tmp}.

5.2.2 Anonymity Goals

Following the definitions in Section 3.1, the proposed protocol intends to provide the following properties: *anonymity for all nodes*, *source/message unlinkability*, *destination/message unlinkability*, *source/destination unlinkability*, and *non-localization*.

APART also makes use of pseudonyms, similar to those in MASK, as we borrow the neighbor authentication mechanism from this protocol. These pseudonyms are *verifiable* by the trusted authority that distributed them. We will not use *transaction* pseudonyms (one pseudonym per message), which is the pseudonym management providing the better anonymity but requiring a great number of pseudonyms per node. Instead we use together *role pseudonyms* and *relationship pseudonyms* [34]: a node always uses a different pseudonym for each role (e.g. sender, recipient, relay, ...), and for some specific roles, it also uses one pseudonym per communication partner. In addition to this pseudonym management, we periodically change some specific pseudonyms of the node, for example when one has been used for too long. Section 5.4 analyses the anonymity properties of the protocol and considers these goals.

5.2.3 Chosen Adversarial Model

In our proposition, we consider protecting against external (local or global, passive or active) attackers as described in section 3.2. As for internal attackers, much more difficult to contain, we will use the *honest but curious* model [17]. Our primary intention is to protect user privacy. This goal is different from traditional security related concerns and *denial of service* attacks, although the exposed solution might be resistant against them.

5.3 System Design

The APART protocol splits the routing process in two components, separating the routing activity of the nodes in two. The first component consists in route initialization, the second is the *blind* forwarding. This forwarding is blind in the sense that nodes relay

ID_A	identity of node A	$E_{PK_A}(M)$	M's *homomorphic* encryption w/ PK_A
PK_A^{perm}	A's permanent key	L_{AB}	link identifier between A and B
PK_A	one of A's public keys	K_{AB}	secret key associated to L_{AB}
SK_A	one of A's secret keys	g_A	A's secret number, member of $\mathbb{Z}_q^*$
PK_A^{tmp}	a temporary public key of A	$LocalID_A^D$	A's identifier for the destination D
PS_A	one of A's pseudonyms	$HopCount$	length of a route (in number of hops)
$Nonce_A$	nonce generated by A	$\mathbb{G}_1$	additive group of prime order q
DRT_A	A's Destination Routing Table	$\mathbb{G}_2$	multiplicative group of prime order q
FRT_A	A's Forwarding Routing Table	H_A	A's hash function: $\{0,1\}^* \rightarrow \mathbb{G}_1$
$K(M)$	encryption of M with key K	$\hat{e}$	bilinear map $\mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$

Table 2: Terminology & Notations

a message without knowing the destination's identity. Broadly speaking, nodes' routing tables do not give explicit information on the destinations available in the network. This is the reason why route initialization requires a specific procedure (see Section 5.3.3).

In the following, we rely on MASK's neighbor authentication system as described in Section 4.1.3. For more simplicity, we use the notation L_{NM} to refer to the link identifier shared between nodes N and M. Even though N and M do share *several* link identifiers, we disregard this and consider that each i^{th} link L_{NM}^i is used only once and replaced by its successor L_{NM}^{i+1}. The same goes for shared symmetric keys K_{NM}. Pseudonyms of a node N are also designated with a unique notation PS_N. Notations used in this paper are given in Table 2.

5.3.1 Routing Tables

Every node maintains two routing tables: a *Destination Routing Table* (DRT) and a *Forwarding Routing Table* (FRT). The first one lists all the destination and routes known by the node, and the other is used to relay messages. An entry of node P's DRT has the following format:

$$\langle\ LocalID_{Dest},\ Dest,\ RouteLength,\ DestPublicKey\ \rangle$$

In particular, a DRT entry concerning destination D would be $\langle\ LocalID_D^P,\ E_{PK_D}(ID_D),\ E_{PK_D}(HopCount),\ PK_D\ \rangle$, where $LocalID_D^P$ is a value generated jointly by P and D, used by P to anonymously and uniquely identify destination D. PK_D is *one* of D's public keys and it is not linked to its *real* identity (unlike its long term public key PK_D^{perm}). The FRT of node P consists of entries of format:

$$\langle\ Link_{prehop},\ LocalID_{Dest},\ Link_{nexthop}\ \rangle$$

An entry regarding a route to D on which P is a forwarding node is $\langle\ L_{XP},\ LocalID_D^P,\ L_{PY}\ \rangle$, with X the predecessor of P on the route towards D (P's *pre hop*), and Y its successor (P's *next hop*).

5.3.2 Topology Discovery

Route Proposition. The DRT and FRT tables are filled during the topology discovery process. Our proposal is diametrically opposed to reactive protocols using route requests. Here, the topology is flooded by route *propositions*: when a node knows a

route, it offers to forward messages towards the destination of this route for its direct neighbors. All the routes known by a node are in its DRT. At the early life of the network, this table only contains the node itself. Every node then proposes routes toward itself to its neighbors, who will themselves propose this new route to their own neighbors, etc. Hence, routes grow in length as the proposition heads towards the edge of the network.

When a node N proposes a route towards destination D, it locally broadcasts the following message:

$$\langle\ PS_N,\ Nonce_N,\ E_{PK_D}(ID_D),\ E_{PK_D}(HopCount),\ PK_D\ \rangle \tag{1}$$

With PS_N one of N's pseudonyms (one never used so far) and $Nonce_N$ a nonce generated by N, two parameters useful for the authentication and the generation of link identifiers and keys. D's identity ID_D is homomorphically encrypted with one of D's public key, by D itself. At first, this value (along with PK_D) is given by D to its neighbors upon proposing the route to itself, and it then spreads in the network. As a consequence, no one except D itself can access the value ID_D. Likewise, the value $HopCount$ is kept secret, except that it will be homomorphically incremented by every node accepting the route proposition.

Route Acceptance/Refusal. Neighbors receiving N's route offer use PK_D and $E_{PK_D}(HopCount)$ to decide whether they accept the route or not. This decision is partly *unconscious* for it is automatically taken by applying a decisional (deterministic) function on the $HopCount$ ciphertext. Thanks to the properties of fully homomorphic cryptography, processing $E_{PK_D}(HopCount)$ actually transforms the $HopCount$ value. The decision process for a node P receiving a route proposition is as follows:

1. If this is the first time P encounters the key PK_D, it immediately accepts the route because it might lead to a destination still unknown to P;

2. If P has enough routes toward D (according to a certain threshold to be defined), it immediately refuses the route. To count the number of routes toward a destination D, a node uses its DRT;

3. If none of the two conditions above are fulfilled, P homomorphically processes route length to compute $(E_{PK_D}(HopCount) \leq E_{PK_D}(HopCount_{MAX}))$ (routes that are too long are discarded). To perform this operation, P must compute $E_{PK_D}(HopCount) - E_{PK_D}(HopCount_{MAX})$ and takes the most significant bit (the sign bit). The result will be $E_{PK_D}(1)$ if $HopCount - HopCount_{MAX} < 0$.

This decision function outputs $E_{PK_D}(b)$ with $b = 1$ if P accepts the route, and $b = 0$ otherwise. Of course, a node refuses a route if it is toward itself. It can realize it by checking if PK_D is one of its keys or not. Also, a node accepts a given proposition only once (a route proposition can be identified by (PS_N, PK_D))

Route Proposition Answer. At this point, P still doesn't know if it accepted the route or not. To obtain the result, $E_{PK_D}(b)$ must be decrypted. Yet, only D can do that. P has to send the ciphertext to N, who will pass it down to D through its route

towards D. Before that, P processes the ciphertext of ID_D (given in message (1)): it applies an arbitrary deterministic *one-way* function H_P mapping arbitrary strings to points in a multiplicative group $\mathbb{G}_1$ (e.g. a hash function). The output of $H_P(ID_D)$ is noted $PreLocalID_D^P \in \mathbb{G}_1$. P sends the two ciphertexts to N in a message of the following format (with PS_P and $Nonce_P$ used in the authentication between N and P, and PK_P^{tmp} a one-time public key generated by P that will be used to encrypt a part of D's answer):

$$\langle\ PS_P,\ Nonce_P,\ E_{PK_D}(b),\ E_{PK_D}(PreLocalID_D^P),\ PK_P^{tmp}\ \rangle \tag{2}$$

Node N then transmits $E_{PK_D}(b)$ and $E_{PK_D}(PreLocalID_D^P)$ to D through nodes $X_1, X_2, \ldots, X_n$ who forward them hop by hop using link identifiers and symmetric keys generated during the construction of the route (e.g. N and X_1 authenticated just like N and P did with messages (1) and (2)). The message from N to D is formatted as follows:

$$\begin{aligned}\langle\ L_{X_kX_{k+1}},\ K_{X_kX_{k+1}}(Nonce'_N,\ PK_N^{tmp},\ E_{PK_D}(PreLocalID_D^P),\\ E_{PK_D}(b),\ PK_P^{tmp})\ \rangle\end{aligned} \tag{3}$$

With $k \in [0, n]$, $X_0 = N$ and $X_n = D$. Nonce'_N is a fresh random number generated by N that will be useful to recognize the answer from D when it comes back, and PK_N^{tmp} a one-time public key from N that is used to encrypt encrypt a part of D's answer. We note that thanks to per-hop encryption/decryption with the keys $K_{X_kX_{k+1}}$, the message in not traceable by an external eavesdropper (but it is by colluding internal attackers).

When D gets the two ciphertexts $E_{PK_D}(b)$ and $E_{PK_D}(PreLocalID_D^P)$, it decrypts them and sends them back to N. Before, it processes $PreLocalID_D^P$ by multiplying it by a secret number $g_D \in \mathbb{Z}_q^*$ (q prime) it chose at the network setup. The result is $LocalID_D^P = g_D \times PreLocalID_D^P$. This last value will be used during route initialization. Values $LocalID_D^P$ and b are respectively encrypted with PK_P^{tmp} and PK_N^{tmp} so that only P and N can access them. After that, they are bounced back to N along the *reverse* route (the route being usually unidirectional from N to D, relay nodes will have marked their predecessor on the route for this occasion). The return message is:

$$\langle\ L_{X_{k+1}X_k},\ K_{X_{k+1}X_k}(Nonce'_N,\ PK_N^{tmp}(b),\ PK_P^{tmp}(LocalID_D^P))\ \rangle \tag{4}$$

N will recognize the message thanks to the nonce $Nonce'_N$ it embeds. If N receives $b = 0$, it knows the route have been refused. If $b = 1$, it means P accepted the proposition. N informs P of this decision and communicates the value $LocalID_D^P$ (encrypted with PK_P^{tmp}). P then creates a new entry in its DRT with this value for $LocalID_{Dest}$ plus the key PK_D and the ciphertexts of ID_D and $HopCount$ it already knows. N creates a new entry in its FRT with $Link_{prehop} = L_{PN}$, $Link_{nexthop} = L_{NX_1}$ and $LocalID_{Dest} = LocalID_D^N$ (N finds this last value in the entry of its DRT it is actually proposing to P). Figure 6 sums up the messages exchanged during a route proposition.

LocalID$_{\text{Dest}}$ Field. With the value $LocalID_D^P$, P has of an identifier it can not link to any identity but that allows to count the number of routes it knows toward a given destination D (by counting the number of entries of its DRT having the same value $LocalID_D^P$). Moreover, the way this data is built allows nodes to initialize routes.

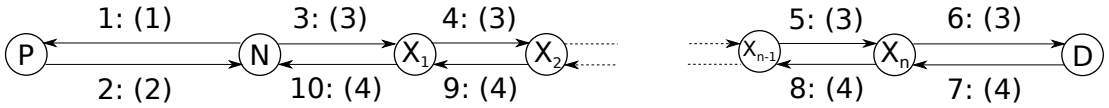

Figure 6: Packets exchanges during a route proposition (first numbers give the ordering, those in brackets refer to messages (1), (2), (3) et (4))

Indeed, DRT and FRT tables do no give any explicit indication on the destination of a route, and a communication can't be initialized trivially.

As for D, which also knows the value $LocalID_D^P$ (and the value $PreLocalID_D^P$), it doesn't know which node is using it. All it can infer when it receives the value $PreLocaID_D^P$ is that *someone* accepted (or refused) a route towards it. Furthermore, since $LocalID_D^P$ and $PreLocalID_D^P$ do not change over time for a given node P, D can count how many routes this *someone* has accepted. Anyhow, this is not an infringement to P or anybody's privacy.

Furthermore, as the discrete logarithm problem is supposed hard in $\mathbb{G}_1$, it is not possible to uncover g_D given $LocalID_D^P$ and $PreLocalID_D^P$. Finally, D's identity ID_D can't be deduced from $PreLocalID_D^P$ as H_P is a *one-way* function.

5.3.3 Routing

Once the DRT and FRT tables filled, nodes can begin to communicate. The routing process is divided in two steps: route initialization and forwarding to the destination. The first step consists in finding a valid first link identifier towards the destination. However, given that the tables do not give information about the destination of the entries, a mechanism is needed to find a valid first hop.

Route Initialization. A source S willing to start a communication with a destination D needs help from a (direct) neighbor V. By cooperating with S, V can find the value of $LocalID^V$ that S needs, without ever knowing who is the actual destination intended. S obtains through V a valid route towards D, but doesn't know which of its $LocalID^S$ is associated to D. If it was the case, then when S relays a message destined to D for another source, it would know that this message is intended to D, which is a violation of the destination/message unlinkability property. This is why tables should not (and do not) clearly give the identity of the destination.

We assume that, beforehand, S knows D's *real* identity. For this, nodes that wish to communicate can exchange their *real* identities and long term public keys prior to network setup (we can also assume S and D know each other). In order to find a neighbor, S locally broadcasts a route initialization assistance request. The first neighbor V that answers it will assist S. We suppose that S and V both have a route to D (without knowing it actually points towards D), as they should have full knowledge of the network.

At first, S and V authenticate with new pseudonyms and use the link identifiers and shared keys from this authentication to secure all subsequent communications. The source S first gives V a set of homomorphic ciphertexts encrypted with a one-time public key PK_S^{tmp} it generated for this purpose (which it also gives to V): $E_{PK_S^{tmp}}(ID_D)$,

$E_{PK_S^{tmp}}(PreLocalID_D^S)$ and for *every destination* X of its DRT, $E_{PK_S^{tmp}}(LocalID_X^S)$ (actually, S encrypt every different values of $LocalID^S$ it knows). With this, V computes $E_{PK_S^{tmp}}(PreLocalID_D^V)$ from $E_{PK_S^{tmp}}(ID_D)$ and encrypts, just like S, all its different values of $LocalID_Y^V$ for all destinations Y.

Now, remind that $PreLocalID \in \mathbb{G}_1$ and thus $LocalID \in \mathbb{G}_1$ since $g \in \mathbb{Z}_q^*$. We define a bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ (we suppose the reader familiar with the concept of *pairing* in cryptography [31]) such that:

$$\begin{aligned}\hat{e}(PreLocalID_D^S, LocalID_D^V) &= \hat{e}(PreLocalID_D^S, g_D \times PreLocalID_D^V) \\ &= \hat{e}(PreLocalID_D^S, PreLocalID_D^V)^{g_D} \\ &= \hat{e}(g_D \times PreLocalID_D^S, PreLocalID_D^V) \\ &= \hat{e}(LocalID_D^S, PreLocalID_D^V)\end{aligned}$$

Now, V is in possession of enough elements to perform an exhaustive search on $LocalID_X^S$ and $LocalID_Y^V$ until it finds (X, Y) such that the test

$$\begin{gathered}\hat{e}(E_{PK_S^{tmp}}(LocalID_X^S), E_{PK_S^{tmp}}(PreLocalID_D^V)) \\ = \\ \hat{e}(E_{PK_S^{tmp}}(PreLocalID_D^S), E_{PK_S^{tmp}}(LocalID_Y^V))\end{gathered}$$

outputs *true*. That is to say, V must iterate on X and Y until X=D and Y=D. V has to test all possibilities, since neither V nor S know (and must know) which one of their respective $LocalID_{Dest}$ is linked to D. More precisely, V performs the test homomorphically, but it has to give the resulting bit of the test to S for decryption with its (one-time) private key.

Through this process, V can thus find $LocalID_D^V$ and relay S's messages to D without even knowing who is the destination. Note that a source should regularly change its assistant neighbor V. This avoids V being S's privileged relay to D. This position should not be authorized for long periods as it allows to link two messages from the same communication (the communication between S and D), which is an infringement to the message/message unlinkability property.

Forwarding. Once the route is initialized, S uses the link identifier shared with V as its first hop towards D. V uses one of its links associated to $LocalID_D^V$ (found at the route initialization) in its FRT. The subsequent nodes on the path that V chose merely forward the packet thanks to their FRT. A data message has the format $\langle\ L_{nexthop},\ K_{nexthop}(MESSAGE)\ \rangle$, with $MESSAGE$ encrypted by the source with the destination's long term public key. When a node receives this kind of packet, it examines the value $L_{nexthop}$ and looks up in its FRT if one of its entries has a value of the field $Link_{prehop}$ equal to $L_{nexthop}$. If not, it discards the packet (the node is not on the path). Else, it means the node should relay the packet. For this purpose, it marks the $LocaID_{Dest}$ matching the $Link_{prehop}$ previously found and randomly chooses a link identifier $Link_{nexthop}$ among those related to this value of $LocaID_{Dest}$. It replaces the value $L_{nexthop}$ of the message by its own $Link_{nexthop}$ and locally re-broadcasts the message. Besides, every message is encrypted with the shared key associated to the link identifier $L_{nexthop}$ of the message. A relay node should decrypt it with this key $K_{nexthop}$,

and then re-encrypt it with the key associated to the new link identifier. Hence, hop by hop, the message is securely forwarded in such a way that no external attacker can trace it. Furthermore, the path taken by a message is not predetermined, and the routes have the nice property to be *partially disjoint*. Note that, during a route proposal, the described process is exactly how nodes $X_1, X_2, \ldots, X_n$ forward message (3).

When the destination receives the message, it knows it is intended to it, as the link identifier on which it receives the message corresponds to a link identifier for a route toward itself (i.e. a *route end*). Finally, in APART, when a destination receives a data message, it also re-broadcasts it in order to confuse external eavesdroppers. It randomly chooses a $Link_{nexthop}$ and sends a dummy message (of the same size as the received message) with probability $\mathbb{P}_{dum\_fwd} \leq 1/2$ that subsequent nodes will recognize and forward with the same probability.

5.3.4 Miscellaneous Components

The APART protocol includes several other components and optimizations. We briefly enumerate them here.

Mobility and Route Maintenance. Because nodes are moving in the network, topology is dynamic. Put another way, links appear and disappear constantly. Given that nodes often have several routes to a destination, they should most of the time have at least one $Link_{nexthop}$ to any destination. Moreover, the $LocalID_{Dest}$ field in a node's FRT enables it to act as a MIX [11]: nodes encrypt/decrypt messages, and the forwarding is probabilistic as node randomly choose their next hops. Furthermore, nodes are able to repair routes: if a next hop link breaks, the node can still use another link related to the same $LocalID_{Dest}$ value. In the event that a node loses all its routes to a given destination, it informs all its predecessors (its $Link_{prehop}$) for this destination with a LINK_ERR message.

Loop Management. Loop management is crucial: many loops can appear when proposing routes, and we should prevent their formation. For this, we use Bloom filters [6]. Each node N maintains one filter per destination (i.e. per $LocalID_{Dest}$) it knows. This filter contains the keyed hash of all nodes on the path from N to D. More precisely, as a route may split in several routes at each node on the path, the filter maintained by node N must contain *all* nodes of *all* routes departing from N to the destination. Thus, when a node proposes a route, it includes the Bloom filter for this route in message (1), and if a neighbor finds itself in the filter, it immediately refuses the route, as an acceptation would create a loop. This mechanism is sufficient to efficiently prevent routing loops. However, we must make sure Bloom filters contains the most up-to-date information about nodes along the paths. For this, Bloom filters must be updated frequently, and not just upon accepting a route proposition. Indeed, when a node accepts a proposition, it inserts itself in the received filter and computes the union of this filter and the one it already had for this destination (if it had one). To compute the union of two Bloom filters, a simple bit-wise OR is required. But then, a node must continue to update its Bloom filter by eavesdropping the route proposition it previously accepted, in case a modification occurs on the paths of this route proposition. For instance, in the case of Figure 7 (where forwarding links are constructed in the order

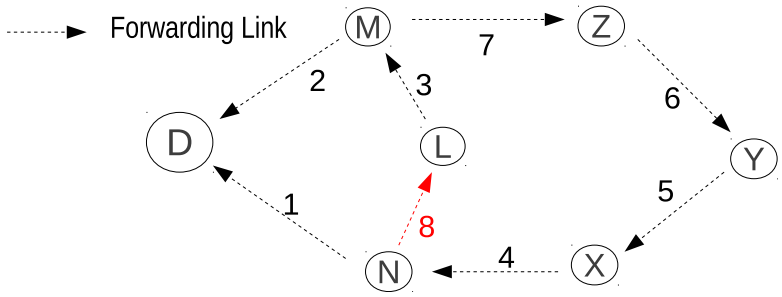

Figure 7: Formation of a loop N-L-M-Z-Y-X-N on a route to D

given by the numbers associated to the links), just before step 8, if L does not update its Bloom filter for destination D, it will propose a route towards D to N with an obsolete Bloom filter, and N will accept it and create the link number 8. This results in the creation of a routing loop N-L-M-Z-Y-X-N. We would like N to see itself in the proposition message of L, so that it knows it must refuse the proposition. This is why L should eavesdrop the proposition of M for the route towards D even after accepting M's proposition, and update its Bloom filter so that L (and thus N) can know *all* current nodes on *all* current paths from L to D.

Pseudonym Management and Anonymity. This implicit component appearing in the description of the protocol is extremely important: a trade-off must be performed between *anonymity* (changing pseudonyms frequently) and *routes and links loss* (changing pseudonym entails breaking a link identifier, so breaking a route). Basically, a node uses a different pseudonym for each route it proposes, and regularly changes the pseudonym it uses to propose routes to itself to its neighbors. When changing this kind of pseudonym, a node must immediately re-propose itself with a new pseudonym so that its neighbors do not lose their routes. Else, the consequence would be the node completely disappearing from the network for a certain time. Also, a node regularly (but rarely) changes its secret number g in order to change the $LocalID$ value other nodes use to identify it. Indeed, there should never be a constant value that uniquely identifies a node in the network. For this same reason, every node should have a different appearance for the ID_D's ciphertext in the route proposition message (1). For this, nodes accepting proposals multiply by one the ciphertext before proposing in their turn: $E_{PK_D}(ID_D)' = E_{PK_D}(ID_D) \times E_{PK_D}(1)$. Thanks to the probabilistic nature of Gentry's scheme, $E_{PK_D}(ID_D)'$ appears different to $E_{PK_D}(ID_D)$. Note that PK_D also uniquely identifies D, so message (1) should be split in two message, the second encapsulating the key in some way so that external eavesdropper do not trace the value in the network.

Fisheye. Finally, in order to reduce the number of periodic proposition messages, APART includes a *fisheye* mechanism inspired by FSR [32]. The idea is to propose routes more or less frequently depending on their length. The longer the route, the less frequent the propositions. As we can not give the length of the routes in clear (it is an indication on the relative location of the destination), *fisheye levels* need to be be homomorphically computed by a node when it receives a route proposition. As

the fisheye levels depend on the length of the route, the homomorphic computation is performed on $E_{PK_D}(HopCount)$. The result is sent along with $E_{PK_D}(PreLocalID_D^P)$ and follows the same path. In order to blur the relative location of the destination, each fisheye level should represent several route length values, and two consecutive levels should overlap (see why in the destination non-localization analysis of Section 5.4). We set the first level to represent lengths of 0, 1 or 2 hops, level 2 for 2, 3 and 4 hops, and level 3 for 5 hops or more. In order to precisely calibrate the route proposition frequency of each levels, tests are needed. Qualitatively, we can say that level 1 routes are proposed very frequently, level 2 route quite frequently, and level 3 are rarely proposed.

5.4 Anonymity Analysis

Here we analyse how well the objectives listed in Section 5.2.2 are met. We assume 3 types of adversaries: external (local or global, passive or active) adversaries (type I), internal, possibly collusive, *honest but curious* adversaries (type II), and internal collusive (active or passive), non-honest adversaries (type III). We will see that we fulfill the anonymity goals of Section 5.2.2 against the attackers I and II stated in Section 5.2.3 but not against type III. We will proceed by enumerating all possible anonymity properties listed in Section 3 and discuss them.

In all the following paragraphs, we make the assumption that traffic is dense enough so that eavesdroppers can not distinguish a source from a relay node (i.e. the source of a packet flow can be any node, with the same probability). However, they can use traffic analysis methods, and in particular they use the fact that when a message is received by a node, and another comes out of it few milliseconds later, the two messages are related (e.g. the node forwarded a message). We thwart this attack by using MIXing [11] techniques: nodes not only decrypt/encrypt and probabilistically forward packets, they also add random delays and reorder them. Note that thanks to MIXing, type I attackers do not have many possibilities to attack nodes' privacy: they can not perform traffic analysis attacks, and they can not replay data messages. Indeed, link identifiers look like random numbers to them (i.e. they do not know who sends or receives messages), and per-hop encryption change the appearance of messages. Actually, eavesdropper only see emissions of WiFi signals, and can not relate them to specific nodes. But they still have few possibilities, described in the subsequent paragraphs.

Node anonymity. All nodes stay anonymous in the network, as no identity is ever divulged, at least in clear. If we suppose the homomorphic encryption secure, there is no way for any attacker to obtain any identity by listening to communications and even by participating in the routing. Furthermore, the public keys flooded in the network are not linkable to identities. If they were, like certified public keys are, it would be very easy to identify the destination of a route proposition message (1) for instance. The only users who know the identity of a node X is X itself, and the users to whom X gave its identity. Also, the server distributing the (*verifiable*) pseudonyms can link any pseudonym to the true identity of the owner. However, this server is a trusted authority, and it reveals an identity only in case of abuse (e.g. a node hiding behind this anonymity to perform Denial of Service (DoS) attacks).

Source/Message unlinkability. In the general case, this property is ensured against all attackers: the source of a data or control message is not given in the message. Plus, due to the fact that routes are *partially disjoint*, a relay node can never know, when it receives a message from one of its authenticated neighbors, if this neighbor is the source or not. Because the FRT is used as well for forwarding as for route initialization, this neighbor might be the source or a simple relay node. We note however that the assistant neighbor V of a source S immediately knows that that S is the source. But this is not an infringement to this property, as V never learns the identity of S.

Destination/Message unlinkability. This property is ensured thanks to the routing tables that do not give explicit information on the destinations. This is actually why we need a special procedure to initialize a route. Indeed, let's imagine that the entries of the DRT also include the *real* identities of the destinations ID_D in some way. Then, for a given node X, the link between the values ID_D and $LocalID_D^X$ is trivial: X knows that $LocalID_D^X$ actually designates D. Consequently, when X forwards messages towards D for other nodes, it uses its value $LocalID_D^X$ in its FRT to find a relevant *next-hop* and immediately breaks the destination/message unlinkability (i.e. X knows that the message it is forwarding is intended to D). The described situation must be avoided and this is why routing tables do not give any explicit indication on the destination's identities. Thus, this anonymity property is respected regarding types I and II attackers. However, there is one case where 2 type III attackers can break the property. Let's suppose a malicious source S knowing ID_D and a malicious neighbor V initialize a route towards D. Normally, when S and V are honest, V obtains $LocalID_D^V$ but does not know that it points to D. But if S decides to tell V who was its target by giving V the value ID_D, V now knows that its value $LocalID_D^V$ points to D and we are the case discussed above where V is in position to break the destination/message unlinkability property. However, if we suppose that D gives its identity only to trusted persons, a malicious source should not know ID_D.

Source/Destination unlinkability. To break this property, there must be a node capable of breaking source/message *and* destination/message unlinkability. In APART, there is no such position for a node, considering type I and II attackers, as we have seen they can not break neither source/message nor destination/message unlinkability. Type III attackers can find the destination, but not the source of a message, thus they cannot break this property.

Message/Message unlinkability. To link two messages from a same communication, a node do not need to know the identities of the communicants, it must only know that the messages come from the same source and are intended to the same destination. In APART, it is possible for the route initialization assistant V to relate two messages: S always uses V as its first hop to talk to D. V, even if it is *honest*, is thus in position to know that incoming messages on the link shared with S always belong to the same communication. Hence, type II and III attackers can relate two messages. This is not true for type I attackers: because link identifiers are used once and then discarded, and because two successive link identifiers are unlinkable in appearance, an external observer can not know that S sends a message to its assistant or to another node in its

neighborhood.

Source non-localization. If we suppose eavesdroppers incapable of distinguishing a source from a relay node, type I attackers can not locate any source. They can locate the emitter of messages (1) and (2) as those are different from data messages, thus easily identifiable. But here we focus on the non-localization of the source of a communication. Type II attackers in a specific position can know the relative position of a source. More exactly, the assistant node V knows S is one hop away, but this is inherent to ad hoc networks, where nodes have to reveal their presence when emitting messages. We can also suppose S choses V according to a trust criterion. Type III attackers playing the role of V can locate S even more accurately by moving around its supposed location while assisting it. Indeed, S and V exchange as many messages as the number of destinations in the network, which leaves time for a malicious V to precisely locate S.

Destination non-localization. Maybe the most vulnerable property we try to protect, the destination's location can be approximated in several ways. First of all, colluding type III attackers on the path from a source to a destination can, more or less accurately, locate destination D. To quantify the accuracy of the attack, we borrow the formula given in the anonymity analysis of ANODR [23]: depending on the position of the attackers, and on whether they are consecutively arranged on the route or not, we define the traceable ratio R as:

$$R = \frac{\sum_{i=1}^{K} F_i \cdot W_i}{L} = \frac{\sum_{i=1}^{K} F_i^2}{L^2}$$

With L the total length of the route from S to D in number of hops, K the number of compromised *segments* of the route, and F_i the length of the i^{th} compromised segment. W_i is a weight factor set by default to $\frac{F_i}{L}$ so that $R = 1$ when all nodes on the path are compromised, and $R = 0$ when no node is compromised. This metric gives an indication of the probability with which the collusion of attackers can locate the destination. However, if $R = 1$ (i.e. all relay nodes are attackers), this probability is not 1, because attackers do not know that the route ends with their last attacker. Actually, they can suspect D to be the destination, but they have no way to verify this assumption. To thwart this attack even more, a destination receiving a message intended to it also broadcasts a message so that eavesdroppers can not see that D received a message and did not re-emit one. The subsequent nodes that receive D's dummy message relay it with a probability $\mathbb{P}_{dum\_fwd}$. Thus, here again, type II or III attackers can not know if they received the dummy message from the destination itself or if there was other relay nodes in between.

Also, replay attacks can give indications on were a destination is. Replay can only be done by type III attackers (type I can not, as they need link identifiers and shared keys, type II are supposed honest), and consists in sending multiple messages to a destination and eavesdropping the communications to see where the flow is concentrated at the end. However, it is very limited thanks to the MIXing properties of the nodes, and sending message to a destination implies knowing its identity ID_D (meaning that D gave it to

the attackers). Attackers could still send messages to a random destination, but then they would locate some node without knowing who it is, which is quite acceptable.

Attackers can use other ways to approximate D's location. The fisheye level gives bounds about the destination's location to type II and III attackers (type I attackers never get to know any fisheye level and route propositions do not give information about the destination[4]). If a node P accepts a route from node N and gets a fisheye level of 2, it knows the destination is at least 2 hops away, and at most 4 hops away. Plus, by measuring the frequency of N's propositions, it can know if N had a fisheye level of 1 or 2. If N had level 1, then P knows the destination is 2 or 3 hops aways, else it is 3 or 4 hops away. Because fisheye levels overlap, type II attacker do not know the exact length of their routes, and can only say the destination is within a certain ring. If consecutive fisheye levels did not overlap, say level 1 was for routes lengths 0, 1 and 2 and level 2 for 3, 4 and 5, then if P receives a level 2 fisheye for a given route and measures that the proposer N had a fisheye level of 1 for the same route, P is certain that the destination is exactly 3 hops away. Now, if we consider colluding type III attackers this defense is not enough. Indeed, they could intersect their rings and locate the destination in a small zone. We argue that the fisheye mechanism drastically reduces the control overhead, while a large enough collusion of type III attackers allowing to locate D is not that easy to put in place and represents a small vulnerability. Moreover, as APART do not construct route according to a *shortest path* policy or any other deterministic paradigm, a route with a fisheye level equals to 3 might actually lead to a destination 1-hop away.

Another, less trivial, attack is possible with the Bloom filters used to prevent routing loops. Indeed, in a general terms, the longer the route is, the more bits are set to 1 in the Bloom filter. The main vulnerability of our usage of Bloom filters is not recognizing relay nodes in the bloom filter: nodes use random numbers they insert in the filter, for instance one different per route, so relay nodes can not be identified or recognized. The main knowledge an attacker (type I, II or III) can infer from the Bloom filter of a route is the distance to the destination: the number of bits set to 1 gives, probabilistically, the number of hops to D. For instance, when D proposes a route towards itself, if it sends an empty Bloom filter, everyone immediately knows it is the destination of the proposition. However, because for a given node N, its Bloom filter for destination D contains *all* nodes of *all* routes departing from N, the number of nodes in the filter do no represent the number of hop from N to D. One could imagine the number of nodes in the filter as the result of the number of routes to D departing from N multiplied by the average number of nodes on a route from N to D. Of course, the conjecture *the more 1 in the filter, the further the destination* is still valid, but guessing the distance to the destination with it results in a very approximate measure. As when D proposes routes to itself, it should insert (in addition to itself) between 5 and 15 nodes for instance, in order to hide the fact that D is proposing a route towards itself. Finally, note that type I attackers inferring knowledge on Bloom filters only allows them to locate a *potential* destination, not an actual, current destination of a communication. And as all nodes are potential destinations, this is not a attack to the destination/message unlinkability.

[4]In message (1), all fields change depending on the proposer, except PK_D, but we can imagine that this key is given encrypted with K_{NP} to P *after* authentication between N and P.

Relay nodes non-localization. Because they act as MIXes, relay nodes can not be localized by type I adversaries. However, because of the multi-hop nature of ad hoc routing, a type II or III attacker acting as a relay node can locate at least one of the relay nodes, in the sense that it knows *some relay node* is 1-hop away, which is a really thin indication inherent to ad hoc routing. However, the attack by a type III attacker presented in the source non-localization can be performed. This time, S is malicious and V is not, and during route initialization, S can locate V precisely.

Remarks.

- The proposition messages for a given proposer can be recognized thanks to the pseudonym field, and any attacker can follow a proposing node in the network. This is acceptable, as every node makes propositions, and there is no communication involved: the proposer is neither a source, a destination or a relay node. Plus, pseudonyms changing regularly, a node can not be followed very long.
- Identity spoofing and impersonation is possible for a node N by inserting a false identity in route proposition messages towards itself. However, APART thwarts this attack by never divulging any identity in the network: attackers never learn the identities in the network. Anyhow, a simple solution is to link node's identities to their long term public key (e.g. a node's identity is a secure hash of its public key), resulting in an impossibility for attackers that successfully impersonated a node N to decrypt data messages intended to N, as only N knows the secret key.

This analysis could be a lot more formal, for instance by using information theory, often based on Shanon's entropy to measure the *a priori* and *a posteriori* knowledge of an attacker. The paper [16] shows some shortcomings of the approach based on Shannon's entropy and propose new techniques and models. However, to use them, several assumptions have to be made. In particular, the forwarding between a source and a destination is seen as a MIXing black box, which is not true when considering type II or III attackers. Also, all these theories do no consider node mobility and offer no metrics to measure the probability of locating a node. Still, a formal analysis would be useful to prove our intuitions, or to invalidate them. This task belongs to future work.

5.5 Simulation

When designing a routing protocol, it is necessary to validate that it actually routes and that communication on top of it are possible. To do so, the first step is to make a *simulation* of this protocol. By coding the basic functionalities, we can already make sure that the protocol works. This section presents the conception choices along with the tools used to develop a simulation for APART. As the internship is coming closer to its end, the simulation might not be finalized and results may not be available. Last part of this section gives the progression at the moment this report is being written.

5.5.1 Conception Choices

First of all, a simulation is not an implementation. Routing protocols go through a series of steps before they are implemented as a kernel module (for Unix). The most basic validity verification that can be done for ad hoc routing protocols, is to *simulate* a network with moving nodes in a delimited area, and to make them send each other messages, disregarding medium, physical or link models. In other words, a simulation is a distributed program where nodes are represented by processes producing events and sending messages. The goal is to check if, by following the protocol, given that it is well modeled and programmed, the nodes can find each other and communicate. After that, one should try to add wireless transmission models, with collisions and delays. But for now, we focus on the simulation.

The question is what do we model, and what do we disregard? We chose to leave aside all wireless transmission and physical models, as well as the link layer collision models. The only relevant elements, as we see it, are the neighborhood simulation (only nodes within WiFi range can hear a message) and the mobility of the nodes.

Another important modeling question is how do we simulate the tools used in the protocol? In our case, how do we simulate homomorphic cryptography, bilinear map and Bloom filters? We looked for Java implementations of Bloom filters and pairing, but we made the choice not to use an implementation of fully homomorphic encryption. Indeed, we already know the homomorphic operations would take a lot of time, and we simply develop a void implementation: we simulate homomorphic ciphertexts and operations, but the ciphertexts are simply objects encapsulating plaintexts, and addition or multiplication operations are done directly on the plaintexts.

In the end, the simulation should provide results concerning the delivery ratio (number of delivered packets against the number of sent packets), the control overhead (number of control messages against the total number of messages, including data messages), the end-to-end latency (time taken by a message to travel from a source to a destination), the size of routing tables, and maybe some *anonymity metrics*. For this last element, a formalism is needed in order to measure what an attacker learns from events.

5.5.2 Tools and Simulators

For a simulation, the only necessary tool is a *discrete event simulator*, which is merely a software that creates multiple identical processes that issue events, scheduled by a *supervisor process*. Thus, a node sending messages will be a process issuing events. In the world of networks, a tremendous number of discrete event simulator exist: NS2, NS3 (Network Simulation 2 & 3) written in C/C++, GloMoSim in C, ReactiveML, a whole functional programming language, SimPy in python, OMNeT++ in C++, JiST/SWANS in Java, ... Chapter 14 of the book *Recent Advances in Wireless Communications and Networks* [12] propose a list of network simulatorsThe list contains more than 60 simulators, a number we can easily double if we add the simulation tools that did not successfully make their way to the public community.

Most simulators handle many complicated models we do not need. Using them would require time to understand their architecture and the development would be longer. For instance, NS3 is a powerful tool providing numerous models and allowing realistic tests,

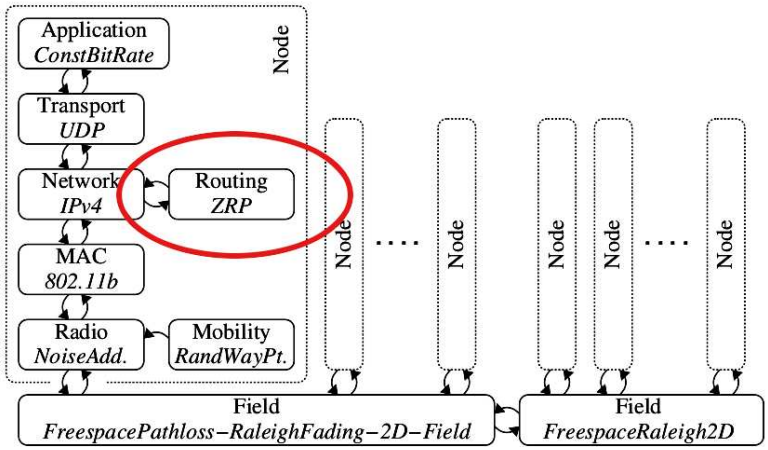

Figure 8: JiST/SWANS simulator architecture (The routing module is circled)

but it is actually *too* powerful for our needs. These kind of simulators would be adapted in the next step, to test the protocol in a more *realistic* environment. On the contrary, SimPy is a very basic simulator, as it is a pure discrete event simulator, very easy to use. However, it is too raw: it does not provide tools to simulate neighborhood and mobility. Writing them would take time and making sure these mechanisms are well modeled is not trivial.

JiST/SWANS [36] is network simulator written in Java: JiST is a pure discrete event simulator, and SWANS is a set of *add-ons* using JiST to model mobile ad hoc networks. Although it runs with an old version of Java (JDK 5) and even though the code is no longer maintained as it is an old project, the architecture of JiST/SWANS is quite accessible and SWANS includes several physical and link layers models, including the basic flawless models which we need. Moreover, adding a module is easy (in our case, a routing module). The only task is to program a Java class implementing the right interfaces, and use it instead of the default routing module. For instance, in Figure 8 showing the architecture of JiST/SWANS, one should replace the *Routing ZRP* module[5] by its own protocol. The only shortcoming we see to this simulator is that it does not scale, i.e. it can not handle too many nodes. However, we argue that for a simple simulation involving roughly a hundred nodes as we intend to model, it is sufficient.

Also, we need implementations for Bloom filters and bilinear maps. Java implementations of Bloom filters are easily found on the internet, as efficiency is not our first criteria for now. We decided to use the implementation of Magnus Skjegstad [39] that provides basic $add(\cdot)$ and $contains(\cdot)$ operations on Bloom filters, which is sufficient for our simulation. Concerning pairing, we chose *jPBC* [29], a Java implementation written by Angelo De Caro from University of Salerno (Italy). Both implementations run with Java 5 (or at least older versions of these implementations do), which is necessary, as JiST/SWANS runs with this Java version. To simulate fully homomorphic encryptions, we write dummy Java classes that actually directly manipulate plaintexts. In future, more complete simulations, we might look into implementations of Gentry's

[5]ZRP, *Zone Routing Protocol*, is an ad hoc routing protocol.

schemes, written in C/C++ [20, 28] or in Java [35] by various volunteers from the world of cryptography.

5.5.3 State of Progress

At the moment these lines are written, the simulation is still in development. As discussed above, finding a MANET simulator is not that simple. We have tested several possibilities before deciding, each simulator also requiring time to understand.

Thus, the development is still pending, and results are not available yet. As is, only the messages and routing tables structures are completed. The core routing functions, *receive* (when receiving APART control messages from the link layer), *send* (when sending APART control messages on the link layer) and the *findRoute* hook for the IP layer are still in development.

The next steps are to develop a first very simple version of these 3 functions and run a few minimalist simulations in order to test the implementation. Then we should progressively add the features of the protocol. In the end, we may even consider implementing the homomorphic cryptographic operations, only to evaluate the efficiency of the technology. Most likely, results from simulations including *real* homomorphic cryptography operations should give indications on how inefficient is the first version of the protocol, but more important, on how to improve it.

6 Future Work

The time devoted to designing, analyzing and testing the proposed protocol was unfortunately not enough to completely go around all issues that might appear in the design. First, some concepts as the loops or pseudonym management need to be clarified. More exactly, they theoretically work, but some unsuspected shortcomings might occur when pushing the analysis and testing further (e.g. the number of pseudonyms a node needs might be too important, resulting in several GBs of memory necessary). The simulation is a first step towards a better understanding of the protocol.

Also, a more thorough analysis of the protocol in terms of anonymity is still possible. Some formal models exist to quantify the anonymity provided by a protocol [16]. Even if the models often make approximations on the system and do not take mobility into account, the metrics they allow to compute give a good idea (i.e. an approximation) of how anonymous the protocol is. The first formal studies on APART should, to our mind, focus on what and how much information an (internal) attacker can infer using fisheye levels, Bloom filters, or during the route initialization.

If time was not an issue, the protocol would go through several steps, beginning by the simulation. When results come out of this experiment, some adjustments will (probably) be necessary. Then other simulations with these modifications should give new results, etc. Then the protocol could be tested in more realistic environments, using NS3 and then test beds: implement the protocol in smartphones for instance, and test the protocol within a real life example. In our case, homomorphic cryptography being way too inefficient, all these testings would require homomorphic cryptographic operations to be simulated, not actually implemented.

To sum up, there can always be more to do: optimize performances, consider protecting against internal adversaries, improve anonymity, etc. Trying to design the pro-

tocol with *leveled* fully homomorphic cryptography [9] (or maybe even simpler, more efficient schemes as Pailler's [30]) is an interesting lead to explore, as it is much more efficient. Indeed, Gentry's *bootstrapping* leads to a total fully homomorphic cryptographic scheme, but without bootstrapping, his scheme is said *leveled* fully homomorphic: only functions with complexity under a certain *level* can be evaluated on ciphertexts. To figure out if this is possible, one should first of all take a close look the homomorphic operations involved in APART in order to approximate their complexity.

7 Conclusion

Protecting users privacy in ad hoc networks is still a problem hard to address, especially as trade-offs between efficiency, anonymity and security are necessary.

In this report, we took the side of the anonymity and we favored this component. We proved that a privacy-preserving proactive routing protocol was theoretically possible thanks to (fully) homomorphic cryptography. This technology is the base on which our protocol rests, and with it, we provide strong privacy with reasonable assumptions. Of course, the practical inefficiency of Gentry's scheme is not adapted to the requirements in terms of performance of ad hoc networks. However, we note that since its creation in 2009, performances of fully homomorphic cryptography dramatically progressed, and this technology is still extensively studied in the world of cryptography.

This internship in the team CIDRe at Supélec allowed the INSA engineering student that I am to discover the world of research. During the first weeks, the idea was to explore the research works around the internship topic. This part is exposed in Sections 2, 3 and 4. Then the goal was to answer the internship by imagining a new anonymous ad hoc routing protocol (possibly mixing concepts from existing protocols). After some unsuccessful leads, the idea of APART began to appear. Several weeks were then needed to define it more precisely, while always analyzing it to detect any privacy leak.

This work led to 2 scientific papers, one for SARSSI, one for CANS, plus another small one for APVP not leading to any official proceedings. As I intend to continue with a thesis, this was a great opportunity to practice paper writing. So far, no answer have been given, as the author notifications are given later in the year. Also, the opportunity to attend seminaries and conferences gave me a good insight of what it is to be a PhD student or a researcher. The contents of these conferences were also very rewarding.

Generally speaking, the topic of the internship was very interesting, although ad hoc networks are now somewhat obsolete (especially with the advent of the 4G technology). It was the opportunity to learn more about networks and (homomorhpic) cryptography. In particular, designing a simulation compels to look deep into the functioning of a computer's protocol stack, at least at the routing layer.

This internships should lead to a thesis in the same team. The topic should be in the continuation of the internship, except wider and around homomorphic cryptography for anonymity.

References

[1] A. Adnane, C. Bidan, and R.T. Sousa. Trust-based countermeasures for securing olsr protocol. In *Proc. 12th IEEE International Conference on Computational Science and Engineering (CSE'09)*, volume 2, pages 745–752. IEEE Computer Society, August 2009.

[2] M.A. Ayachi, C. Bidan, T. Abbes, and A. Bouhoula. Misbehavior detection using implicit trust relations in the aodv routing protocol. In *Proc. 12th IEEE International Conference on Computational Science and Engineering (CSE'09)*, pages 802–808. IEEE Computer Society, August 2009.

[3] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, jul 1970.

[4] Simon Boche, Christophe Bidan, Sébastien Gambs, and Nicolas Prigent. Protection de la vie privée dans les réseaux mobiles ubiquitaires. Master's thesis, Université de Rennes 1, Supélec (CIDRE), 21/06/2012.

[5] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-dnf formulas on ciphertexts. In *Proceedings of the Second international conference on Theory of Cryptography*, TCC'05, pages 325–341, Berlin, Heidelberg, 2005. Springer-Verlag.

[6] Prosenjit Bose, Hua Guo, Evangelos Kranakis, Anil Maheshwari, Pat Morin, Jason Morrison, Michiel Smid, and Yihui Tang. On the false-positive rate of bloom filters. *Inf. Process. Lett.*, 108(4):210–213, oct 2008.

[7] A. Boukerche, K. El-Khatib, Li Xu, and L. Korba. Sdar: a secure distributed anonymous routing protocol for wireless and mobile ad hoc networks. In *Local Computer Networks, 2004. 29th Annual IEEE International Conference on*, pages 618 – 624, nov. 2004.

[8] Z. Brakerski and V. Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, pages 97–106, 2011.

[9] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *ITCS 2012*, pages 309–325, Cambridge, Massachusetts, 2012. ACM.

[10] Sonja Buchegger and Jean-Yves Le Boudec. A robust reputation system for mobile ad-hoc networks. Technical report, Proceedings of P2PEcon, 2003.

[11] David Chaum, R. Rivest, and David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24:84–88, 1981.

[12] A.K. Dwivedi and O.P. Vyas. *Recent Advances in Wireless Communications and Networks*, chapter Wireless Sensor Network: At a Glance. InTech, 2011.

[13] Craig Gentry. *A Fully Homomorphic Encryption Scheme*. PhD thesis, Stanford University, 2009.

[14] Craig Gentry. Computing arbitrary functions of encrypted data. *Communications of the ACM*, 53(3):97–105, March 2010.

[15] Craig Gentry, Shai Halevi, and NigelP. Smart. Homomorphic evaluation of the aes circuit. In *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2012.

[16] Benedikt Gierlichs, Carmela Troncoso, Claudia Diaz, Bart Preneel, and Ingrid Verbauwhede. Revisiting a combinatorial approach toward measuring anonymity. In *Proceedings of the 7th ACM workshop on Privacy in the electronic society*, WPES '08, pages 111–116, New York, NY, USA, 2008. ACM.

[17] Oded Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications*, volume 2. Cambridge University Press, 2004. Chapter 7.

[18] David Goldschlag, Michael Reed, and Paul Syverson. Onion routing. *Commun. ACM*, 42(2):39–41, February 1999.

[19] Thore Graepel, Kristin Lauter, and Michael Naehrig. Ml confidential: Machine learning on encrypted data. Cryptology ePrint Archive, Report 2012/323, 2012.

[20] Henning Perl (Leibniz Universität Hannover). Fully homomorphic encryption implementation of the hcrypt project. https://hcrypt.com/scarab-library/.

[21] Fan Hong, Liang Hong, and Cai Fu. Secure olsr. In *Advanced Information Networking and Applications, 2005. AINA 2005. 19th International Conference on*, volume 1, pages 713–718 vol.1, March.

[22] P. Jacquet, P. Muhlethaler, T. Clausen, A. Laouiti, A. Qayyum, and L. Viennot. Optimized link state routing protocol for ad hoc networks. In *Multi Topic Conference, 2001. IEEE INMIC 2001. Technology for the 21st Century. Proceedings. IEEE International*, pages 62–68, 2001.

[23] Jiejun Kong and Xiaoyan Hong. Anodr: anonymous on demand routing with untraceable routes for mobile ad-hoc networks. In *Proceedings of the 4th ACM international symposium on Mobile ad hoc networking and computing*, MobiHoc '03, pages 291–302, New York, NY, USA, 2003. ACM.

[24] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-Hashing for Message Authentication. RFC 2104 (Informational), February 1997. Updated by RFC 6151.

[25] Michael Naehrig, Kristin Lauter, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? In *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, CCSW '11, pages 113–124, New York, NY, USA, 2011. ACM.

[26] Alireza A. Nezhad, Dimitris Makrakis, and Ali Miri. Anonymous topology discovery for multihop wireless sensor networks. In *Proceedings of the 3rd ACM workshop on QoS and security for wireless and mobile networks*, Q2SWinet '07, pages 78–85, New York, NY, USA, 2007. ACM.

[27] AlirezaA. Nezhad, Ali Miri, Dimitris Makrakis, and LuisO. Barbosa. Privacy within pervasive communications. *Telecommunication Systems*, 40:101–116, 2009.

[28] Stephen Crane (University of California). Simple homomorphic encryption implementation. https://github.com/rinon/Simple-Homomorphic-Encryption.

[29] Angelo De Caro (University of Salerno). Pairing java implementation. http://gas.dia.unisa.it/projects/jpbc/index.html.

[30] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Proceedings of the 17th international conference on Theory and application of cryptographic techniques*, EUROCRYPT'99, pages 223–238, Berlin, Heidelberg, 1999. Springer-Verlag.

[31] K.G. Paterson. Cryptography from pairings: a snapshot of current research. *Information Security Technical Report*, 7(3):41–54, 2002.

[32] Guangyu Pei, Mario Gerla, and Tsu-Wei Chen. Fisheye state routing in mobile ad hoc networks. In *ICDCS Workshop on Wireless Networks and Mobile Computing*, pages D71–D78, 2000.

[33] C.E. Perkins and E.M. Royer. Ad-hoc on-demand distance vector routing. In *Proc. 2nd IEEE Workshop on Mobile Computing Systems and Applications, 1999. Proceedings (WMCSA'99)*, volume 2, pages 90–100. IEEE Computer Society, February 1999.

[34] Andreas Pfitzmann and Marit Köhntopp. Anonymity, unobservability, and pseudonymity — a proposal for terminology. In Hannes Federrath, editor, *Designing Privacy Enhancing Technologies*, volume 2009 of *Lecture Notes in Computer Science*, pages 1–9. Springer Berlin Heidelberg, 2001.

[35] Coen Ramaekers. *Fully homomorphic encryption in JCrypTool*. PhD thesis, Eindhoven : Technische Universiteit Eindhoven, 2011.

[36] Robbert van Renesse Rimon Barr, Zygmunt J. Haas. Jist/swans. http://jist.ece.cornell.edu/.

[37] Jean-Marc Robert and Christophe Bidan. A proactive routing protocol for wireless ad hoc networks assuring some privacy. In *Proceedings of the 2nd ACM workshop on Hot topics on wireless network security and privacy*, HotWiSec '13, pages 25–30, New York, NY, USA, 2013. ACM.

[38] Stefaan Seys and Bart Preneel. Arm: anonymous routing protocol for mobile ad hoc networks. *International Journal of Wireless and Mobile Computing*, 3:145–155, 2009.

[39] Magnus Skjegstad. Bloom filter java implementation. https://github.com/magnuss/java-bloomfilter.

[40] Ronggong Song, Larry Korba, and George Yee. Anondsr: Efficient anonymous dynamic source routing for mobile ad-hoc networks. In *Proceedings of the 2005 ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN 2005)*, 2005.

[41] D. Sy, R. Chen, and L. Bao. Odar: On-demand anonymous routing in ad hoc networks. In *Mobile Adhoc and Sensor Systems (MASS), 2006 IEEE International Conference on*, pages 267–276. IEEE, 2006.

[42] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, gilbert, henri edition, 2010.

[43] Manel Guerrero Zapata and N. Asokan. Securing ad hoc routing protocols. In *WiSe: Proceedings of the ACM Workshop on Wireless Security*, 2002.

[44] Yanchao Zhang, Wei Liu, Wenjing Lou, and Yuguang Fang. Mask: anonymous on-demand routing in mobile ad hoc networks. *Wireless Communications, IEEE Transactions on*, 5(9):2376–2385, 2006.

[45] Bo Zhu, Zhiguo Wan, M.S. Kankanhalli, Feng Bao, and R.-H. Deng. Anonymous secure routing in mobile ad-hoc networks. In *Local Computer Networks, 2004. 29th Annual IEEE International Conference on*, pages 102–108, 2004.