Packet Coding for Strong Anonymity in Ad Hoc Networks

Imad Aad	Claude Castelluccia	Jean-Pierre Hubaux
DoCoMo Euro Labs	INRIA	EPFL
Germany	France	Switzerland
aad@docomolab-euro.com	claude.castelluccia@inria.fr	jean-pierre.hubaux@epfl.ch

***Abstract*— Several techniques to improve anonymity have been proposed in the literature. They rely basically on multicast or on onion routing to thwart global attackers or local attackers respectively. None of the techniques provide a combined solution due to the incompatibility between the two components, as we show in this paper. We propose novel packet coding techniques that make the combination possible, thus integrating the advantages in a more complete and robust solution.**

I. Introduction

The need for anonymity in ad hoc networks, typically for military applications, drove several researchers to explore a wide range of techniques that aim to thwarting omni-present attackers and local attackers, given various optimization criteria such as complexity, transmission costs and processing costs [1], [2], [3], [4], [5], [6].

In the context of anonymity, the goal of an attacker is to gather as much information as possible on the network activities, namely: *who is communicating with whom?* The problem is highly relevant in ad hoc networks since the open nature of the wireless channel goes in favor of the attacker: it can eavesdrop "local" communications and gather the information from the packets themselves, or it can get a "global" view of the communications, inferring the information from the traffic patterns.

The starting point of several existing techniques for anonymity in ad hoc networks is the work by Chaum et al. [7]. The basic idea in [7], for the wired Internet, is to use "mix" mail servers that randomly delay mail forwarding, thus reducing the correlation between incoming and outgoing mails and hiding who is communicating with whom. The same principle is used in mix-routes in ad hoc networks [3], where a set of nodes takes care of diversifying the routes in order to confuse attackers. Still, the traffic patterns reveal pronounced shapes that help adversaries deduce the location of strategic nodes (destinations, traffic sinks)[2]. Deng et al. [2] introduced a mechanism that relies on random fake routes to confuse an adversary from tracking a packet as it moves to the sink. Fake routes, typically forming multicast trees, help randomizing the pronounced traffic shapes, therefore minimizing the efficiency of traffic analysis.

On a smaller scale, packets contain all the necessary information to be forwarded along the path from the source to the destination. Intercepting any transmission on the path, or even compromising a forwarding node, gives the adversary clear information about the communicating nodes. Onion routing [8], also inherited from the wired Internet, helps concealing relevant routing information from potential adversaries: using its secret key, each relaying node decrypts (peels) one layer of the packet (onion) and sends it forward, while being unable to read the content of inner layers of the onion, to be decrypted with the secret keys of successor nodes. Onion routing is at the basis of several enhanced techniques used for anonymous communications in ad hoc networks. Ultimately, ANODR [1] introduces a quite efficient approach for untraceable routing based on link pseudonyms. The establishment of those pseudonyms is, of high relevance to our work, onion-based.

The two main pieces of the puzzle are therefore:

- To thwart adversaries that analyze the traffic using *global views* of the traffic patterns, traffic sources must multicast their packets on redundant routes to confuse the attacker and conceal the communications.
- To thwart *local* eavesdroppers, traffic sources must encrypt their packets in an onion-based way.

However, the two pieces of the puzzle *cannot* be put together! As we show later in this paper, onion-based methods cannot be combined with solutions that use multicast routes. Therefore, existing solutions apply pretty well to thwart either of the two different attacker models, but none of the solutions offers a *complete* mechanism for both models applicable in a generic hostile environment.

The contribution of this paper is the introduction of novel packet coding techniques that:

- Combine multicast and onion-based packet encryption to provide both global and local anonymity solutions, putting the two pieces of the complete puzzle together.
- Make the packets, *and their headers*, change at each hop to reduce traceability. This is an inherent property of (unicast) onion routing that cannot be maintained when combined with multicast routing, as we describe in Section III.
- Leverage the wireless/open nature of the radio channel to add supporting components to make those mechanisms even more efficient in hiding network communications (Section V).

We further distinguish between networks deploying tamper-resistant devices from networks with non-tamper-resistant devices, to propose "light" coding methods for the first and very robust ones for the second. Even though we consider pure ad hoc networks throughout the paper, the coding methods also apply to hybrid networks combining a wireless network with a fixed infrastructure.

This paper is organized as follows. Section II introduces the system model and assumptions. Section III shows the motivation and problem statement. Section IV introduces our packet coding techniques with their respective evaluations. Section V shortly introduces a method that complements existing anonymity techniques and enhances their efficiency. Section VI discusses various related issues. Section VII shows the related work and Section VIII concludes the paper.

II. System model

We consider an ad hoc network of n wireless devices communicating with each other (or with a fixed infrastructure) over multi-hop paths.

The nodes: Each node has a publicly known identifier (ID), which is not necessarily an IP address. Nodes have limited battery and processing power. We initially assume that nodes are fixed, then we discuss mobility issues in a later section. In our work we consider two types of network devices: Tamper-resistant and non-tamper-resistant. In the first we assume that attackers can eavesdrop communications, analyze the traffic etc., while not being able to compromise secret keys in the network devices. In the second case, we assume the attacker is able to compromise any number of nodes along with the keys stored within.

Nodes collaborate to forward packets even when ignoring the source and destination IDs. Checking the authenticity of a packet is possible (while still ignoring the source and destination IDs) when using tamper-resistant devices. When we consider networks with non-tamper-resistant devices, packet authenticity becomes irrelevant.

Routing: We assume that global routing information is requested proactively, then packets are routed using source routing. Routes are requested periodically to cope with the dynamics of the network. During the (global) route request phases node IDs are clear to eavesdroppers/attackers. However, using the mechanisms proposed in this paper, IDs of all nodes involved in a specific transmission (source, destination and routing nodes) will be highly concealed.

The packet coding techniques we propose in this paper can be used over insecure route establishment methods (e.g. legacy DSR [9]) or over secure ones (e.g. ARIADNE [10]). The paper deals with issues to help anonymity, while caring of making it independent of other security problems in ad hoc networks, and without imposing any constraints on their corresponding solutions or their performances. For instance, our mechanisms can be used over normal DSR route establishments, or over ARIADNE if someone wants to consider the attacks that ARIADNE thwarts.

The attacker(s): We only consider attackers with the willingness of identifying the source, the destination and the relaying nodes of a given communication. Thwarting attackers who aim to disrupt the network (e.g. jamming, DoS, route-breaking) is out of scope of the paper. Nevertheless, in the proposed anonymity mechanisms we define the necessary functions to avoid packet replays and packet injection by “malicious” attackers (who’s goal is to drain device batteries or to mascarade another node).

Attackers may vary in “strength”, and we model their strength levels by one parameter which is the portion of the network where the attacker can eavesdrop the ongoing communications. For simplicity, and without any loss of generality, we consider the two extreme cases of attacker levels:

- “local attackers” with knowledge limited to their immediate neighborhood.
- “global attackers” able to intercept any communication in the whole network at any time.

Another aspect of an attacker that we consider is whether he is an insider (compromised node) or outsider (intruder). This is closely related to the model of the nodes we consider, tamper-resistant or non-tamper-resistant, as considered previously.

Whether local or global, insider or outsider, attackers have limited processing power but unlimited energy.

Secret keys: The assumptions in terms of secret keys depend on the node model that we consider:

- In the tamper-resistant devices case, Section IV-A introduces “light” coding mechanisms that are encryption-free. A single symmetric shared key is used among nodes to authenticate the packet sender and to hide (using keyed hash functions) packet headers from eavesdroppers. Note that replacing the system-wide key by a pairwise shared key is possible, however this comes at the (high) cost of revealing the identity of the source node to relaying nodes.
- In the non-tamper-resistant devices case, requirements in terms of encryption keys increase. Section IV-B introduces coding methods that require every node to have a pair of public/private keys, and a publicly known prime number. To avoid repetitive use of asymmetric cryptography, each pair of nodes may establish a symmetric shared key. However, again, this comes at the cost of revealing the identity of the source node to the relaying (possibly compromised) nodes.

Note that node IDs are known to all other nodes in the network and possibly to the attackers.

The MAC layer: To preserve anonymity, packets are locally (1-hop) broadcast and not ACKed at the MAC layer. The application layer is therefore responsible for the communication reliability. Nodes use omnidirectional antennas, therefore any communication can be eavesdropped by any attacker in a node’s neighborhood.

III. Multicasting onions: the puzzle

The goal of our work is to help increasing anonymity in ad hoc networks, and hide the roles of communicating nodes from traffic analysis. In other words, we aim at:

- Concealing the source of a given packet
- Concealing the destination of a given packet
- Concealing the fact that two given nodes are communicating

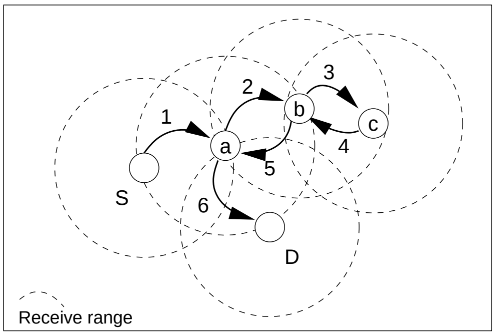

Fig. 1. Reaching an anonymity set using "unicast" routes

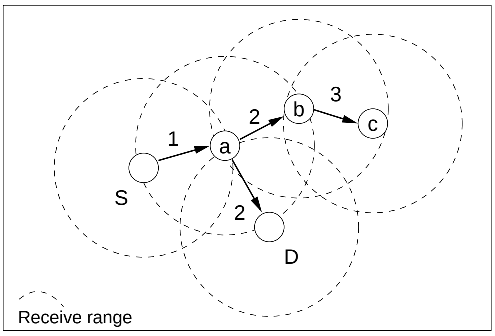

Fig. 2. Reaching an anonymity set using multicast

To reach these goals, we will incrementally describe the building blocks (multicast and onions) of the complete mechanism.

Before getting into the description of anonymity mechanisms, let us first explain what we mean by "anonymity set", that we use extensively throughout the paper: We define an "anonymity set" of the destination, for example, as the set of possible destination nodes. In other words, when a packet is sent to a given anonymity set of nodes, the confusion of an attacker increases as the size of the anonymity set gets larger.

In a DSR-like packet header, IDs of routing nodes are inserted in clear (unencrypted), and in order (from the source to the destination). A local attacker that eavesdrops the packet, anywhere along its path, can easily read where the packet is coming from and where it goes to. To thwart local eavesdroppers, the source node can shuffle the positions of routing nodes' IDs in the packet header. Any receiving node that detects its own identity in the routing list, regardless of the position in the list, will *broadcast* the packet to its first-hop neighbors, ignoring where the packet came from, where it goes to, where the final destination is and where the originating node was. Packet coding will be discussed in Section IV. To a *local* attacker, this increases the anonymity set of the destination (and of the source as well), from 1 to the size of the routing list. In other words, the eavesdropper would be confused which node in the list is the actual destination. However, a *global* attacker is still capable of identifying the source and destination nodes, at both edges of the packet route.

For a *global* attacker, the "picture" is still clear regardless of the packet header format, shuffled or ordered, encrypted or clear. This motivates the use of enlarged anonymity sets as introduced in the following subsections.

A. Extend the routing list beyond the destination

The first basic technique to thwart global attackers is to make the traffic source extend the route *beyond* the (real) intended destination of a packet. This will increase the size of the anonymity set linearly, at the cost of additional transmissions and receptions, of course. For instance, let l_0 be the distance (in hops) from the source to the intended destination. Extending the path by l_e will increase the size of the anonymity set to $l_0 + l_e$, at the cost of l_e additional transmissions and l_e receptions. Note that the packet size will increase because of the enlarged routing list in the header. This issue will be solved and thoroughly analyzed in Section IV. Extending flow paths can be viewed as a variant of sending redundant packets over the network. However, extending a flow route beyond the specific destination helps better hiding the latter, which is hard to achieve by sending redundancy elsewhere in the network.

B. Extend the routing list beyond the source

The previous techniques help hiding the traffic destination, without hiding in which *direction* the data is flowing. Therefore, the routing list should also go beyond the source, giving similar advantages and drawback as going beyond the destination, with the advantage of hiding the traffic *direction* as well.

C. The general technique: use multicast trees

The previous mechanisms, though usable separately, were described to provide incremental understanding of the drawbacks and advantages of each component of this general one. For a packet transmission to reach a given anonymity set of nodes, by leveraging the broadcast nature of the wireless channel, we can use multicast trees. This considerably reduces the overall number of transmissions with respect to unicast routing, to reach an equal number of nodes. Consider for instance the topology in Fig. 1 where the source S intends to send a packet to destination D.[1]

To provide a high anonymity level for destination D, the source S may use mix routes to increase the anonymity set to $\{a, b, c, D\}$ rather than $\{a, D\}$. Moreover, mix routes can be combined with onion routing, either using shared key between the source and each intermediate node (hence the source is known), or using the public keys of the intermediate nodes (hence the source can remain anonymous).

Regardless of the use of onion routing, we can see that the use of additional intermediate nodes will increase the number of transmissions (6 in this example), thus the battery consumption. One can overcome these drawbacks by making use of the open nature of the radio channel, namely using multicast, as shown in Fig. 2.

[1] Note that traditional multicast problems such as tree establishment, address format and joining/leaving the multicast group, are irrelevant here: the source, based on the information gathered during the initial route request phase, sends its packet to redundant (and cooperative) destinations and routers.

We can see that the packet to destination D reached the same anonymity set as in Fig. 1 with much less transmissions (3 in this example) and a shorter delay (2 "slots"). The delays should not be taken into consideration here, since the scenario is not generic enough.[2] The evaluation of the gain in the general case is shown in the next paragraph.

Performance evaluation: Benefit: Using multicast to reach a given anonymity set is highly efficient (especially if combined with the "eavesdropping destination" introduced in Section V). Large anonymity set sizes make it hard for the attacker, local or global, internal or external, to identify the source and the destination. Using multicast to reach those large anonymity sets makes routing easier and more efficient. Moreover, multicast becomes even more efficient when the source needs to send a packet to multiple destinations. Claiming the method to be highly efficient may look contradictory to the fact that we multicast the packet to several redundant nodes on various redundant paths. However, our goal is not to flood the network with redundant transmissions, but rather to reach a *required anonymity set size* with the least number of transmissions possible. In fact, consider a multicast tree that is h hops deep (the destination must be within h hops), where every transmission is relayed by r neighbors. The resulting number of transmissions is

$$\sum_{i=1}^{h} r^i$$

and the resulting number of receiving nodes (i.e. anonymity set size) is

$$\sum_{i=1}^{h} r^{i+1}$$

Therefore we increase the anonymity set by r at every transmission, compared to an increase of 1 at every transmission with techniques using unicast. Another interesting performance aspect of using multicast is how it helps eliminating the pronounced data traffic patterns that may reveal strategic node locations. This was introduced and well analyzed in [2].

Overhead: As in the previous mechanisms, packet header sizes will increase, and so does the number of transmissions and receptions, proportionally to the multicast group size. We will tackle the packet-level issues in Section IV.

D. Hiding the source, using multicast "forests"

Note that so far, we were dealing with hiding (increasing the anonymity set size of) the destination only. The source, even though hidden for local attackers, can be still easily identified by global ones. Existing techniques such as stop-and-go, transmission of redundant flows etc. can be used in combination with our techniques to provide source anonymity as well. In this case, the flow patterns would look like a multicast "forest" (several trees), or fragmented multicast trees, highly concealing the destination(s) and the source(s) as well.

E. The onion

The multicast mechanisms described in the previous subsections were motivated by the need to thwart "global attackers" that have a global view of the communications in an ad hoc network regardless of whether the packets are encrypted or not. Another piece of the puzzle is thwarting "local" eavesdroppers or compromised forwarders that intercept transmitted packets and read the relevant information therein (source, routing nodes, destination). One basic technique is to use disordered routing lists in packet headers. Furthermore, one can use *onion routing* [8] where each routing nodes peels a layer (decrypting the packet with its secret key) and forwards it further. One property of onion routing is that packets "change" as they get forwarded, therefore becoming untraceable, a highly relevant property that we aim to keep in our final (complete) solution.

F. Combining multicast and onions

In the previous subsections we cited the various advantages of using multicast and onion routing respectively. A complete solution that aims to thwart local *and* global attackers simultaneously should apply both, multicast and onion. However, combining the two is not a straightforward procedure: If we rule out the problem of how to place the various identifiers in the onion, the onion *payload* must take one of the two forms shown in Fig. 3.

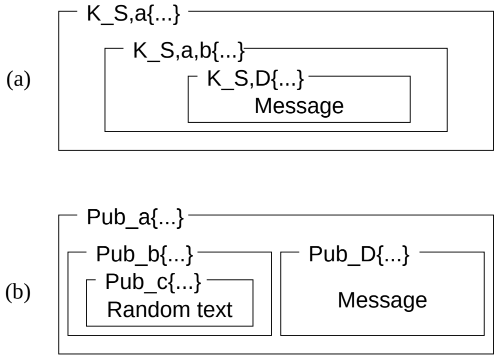

Fig. 3. The problem of combining multicast with onion routing. $K_{i,j,...,k}\{...\}$ is the encryption using a shared symmetric key between nodes i,j,..., and k. $Pub_i\{...\}$ is the encryption using public key of node i.

We have two choices for the encryption/decryption:

(a) Using symmetric keys shared between the source and routing nodes makes the combination possible. However, it reveals the source to *all* routing nodes, and the destination to some of them. Furthermore, it limits the propagation of the packet (ex. beyond node c in Fig. 2), unless the payload is split in 2 sub-onions, as in Fig. 3(b).

(b) On the other hand, using public keys to encrypt the various onion layers becomes cumbersome when the packet goes on 2 (or more) different paths. The approach becomes impossible to use if the packet is to be routed on diverse paths, requiring recursive sub-onions, as shown in Fig. 3(b).

Therefore, the first step towards our complete solution is to consider perfect separation between the routing header and the packet payload (possibly onion-encrypted).

[2] Using unicast, the flow could have passed through D right after a, before reaching an extension list, resulting in a similar delay as multicast, similar anonymity set size, but still with more transmissions.

Note that as mentioned before, we use source routing. Since the goal is to hide the destination, this last cannot be revealed to intermediate nodes, therefore routing protocols like AODV [11] cannot be used when *destination anonymity* comes into the picture.

IV. Packet coding

Now that we motivated the separation between packet headers and payloads, in this section we describe the actual coding techniques of the two parts. For packet headers, we distinguish between tamper-resistant devices and non-tamper-resistant devices and we devise the appropriate coding techniques for each (summarized in Table I). In the first case, "light" coding is used, basically relying on hash functions. For the second case, more encryption-demanding techniques show to be inevitable to tackle with the problem. As for payload coding, common techniques apply for both tamper-resistant and non-tamper-resistant cases.

A. Routing headers for tamper-resistant devices

In these networks we assume that devices can be compromised: The attacker can use the compromised devices to generate malicious/erroneous packets. However he cannot retrieve encryption keys shared between the network nodes neither the decrypted clear-text. We will relax this tamper-resistance assumption in the next section (IV-B).

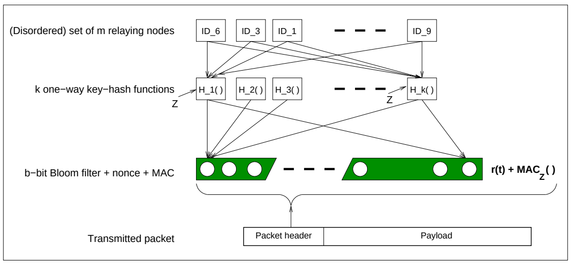

Fig. 4. Using Bloom filters to compress the source routing list

Some of the mechanisms described above have a common drawback: the packet header size increases with the size of the anonymity set. However, the extensive list of routing entities contains much more information than a forwarder needs. In fact, the *only* information a receiving node needs is a binary: to forward or not. In other words it is a binary test: does the receiving node belong to the list or not?

One common technique to use in list checking problems is *Bloom filters*. Bloom filters [12] have been widely used in databases, in peer-to-peer applications and are enjoying growing popularity in computer communications nowadays [13], [14]. They offer high compression rates, low false positives and no false negatives. In our case, the compression rate will help reducing header sizes considerably, while (low) false positives have no considerable impact since they slightly increase the anonymity set size. We will analyze the impact of false positives later in this section.

The use of Bloom filters, in combination with the above techniques, is described in Figure 4. The source establishes the list of m forwarder/destination IDs. Each entry is passed to k keyed-hash functions H_i with a shared key Z. Each of the H_is defines the position of a single bit to set in the Bloom filter (initially all bits are 0s). The resulting string of bits, i.e. the Bloom filter, replaces the routing list in the packet header.

Each node applies the k hash functions to its own ID once for all packets. Upon receiving the transmitted packet every node checks whether the k positions are set in the Bloom filter sent in the packet header. If it is the case, the node knows it belongs to the routing list, it therefore forwards the packet.

This is the routing aspect of using Bloom filters. However, additional functions must be added to preserve anonymity, while avoiding flooding or packet replays:

- Since the node IDs are possibly transmitted in clear during the route request phase, and since the hash functions are not secret, it is possible for an attacker to check whether any of the IDs is included in the Bloom filter or not. To avoid this vulnerability, we use keyed hash functions that use a secret key (Z) shared among all nodes. Using pairwise shared keys may look as a more robust alternative. However, it has a major drawback: how does a receiving node know which key to use to check whether it is included in the filter? and if it does, that means the source ID is revealed to this node. Therefore, since the nodes are assumed to be tamper-resistant, we adopt the system-wide shared key solution. For non-tamper-resistant devices we adopt the solution described in Section IV-B.
- To avoid that an attacker forges a Bloom filter to flood the network with redundant packets, therefore consuming node batteries, a message authentication code (MAC) is used with the shared key (Z) then concatenated to the filter.
- To avoid an attacker from reusing an authentic filter (i.e. compressed routing list), the source concatenates a nonce $r(t)$ to the filter before computing the MAC. $r(t)$ can be the value of a loosely synchronized clock.

Due to the finite filter size, some nodes may find all their hash functions pointing to positions that are set in the Bloom filter, while in fact they do not belong to the routing list. This happens when a node's bits positions in the filter are a combination of other nodes' bits positions. This occurs with probability $(1 - e^{km/b})^k$.

These false positives can be reduced by increasing the Bloom filter size. Hence, one should consider one of the two options:

- Keep the filter size low and, with low probability, few unintended nodes will retransmit the packet (consuming their energy for transmission and their neighbors' for reception) or,
- Increase the filter size and therefore the number of bits to be transmitted by every forwarding node. This also increases the number of bits received by all neighboring nodes along the path(s).

Note that the small energy consumed by the first option performing unintentional transmissions goes in favor of increasing the anonymity set, making it advantageous with respect to the second option.

In brief, the main properties we derive from the use of Bloom filters as described are:

- It hides the routing list from local eavesdroppers.
- It does not mandate the use of encryption.
- It compresses the routing list considerably.
- It perfectly fits multicast lists (to obfuscate global attackers while reducing the number of transmissions, as described above).

Yet, the proposed mechanisms rely on the strong assumption that nodes are tamper-resistant. i.e. attackers cannot access the shared key used for the filter hash functions or in the header MAC computation. When this assumption does not hold, an attacker can identify entries in the routing list/filter, can replay packets, or it can even flood the network with redundant packets that will be easily forwarded by (cooperating) nodes. In the next section (IV-B) we define a new technique that is better adapted to networks with vulnerable devices.

Performance evaluation: Consider g to be the multicast group size and l the length (in bits) of each entry/ID. If we consider the Bloom filter to be b bits long, then the compression ratio it offers is $g \times l/b$. For practical values of g, l and b, the ratio is drastically high, considerably reducing the header size with respect to normal source routing, e.g. DSR packet headers.

B. Routing headers for non-tamper-resistant devices

In a network with vulnerable (non-tamper-resistant) devices, attackers have access to all keys in the compromised nodes, therefore no authentication can be performed. In this case compromised nodes can be used to flood the network in order to drain the battery energies of all cooperative nodes, and the ones hearing their transmissions. To deal with this problem one must consider rate limiters at each forwarding node, a solution that is well studied in the literature [15] and somehow out of scope of this paper.

When the system's shared key and the nodes' IDs get disclosed to an attacker, all routing lists in packet headers become clear to him and the techniques described above do not apply anymore. To avoid this problem, we adopt asymmetric cryptography as described below. We assume that each node i has:

- an identifier ID_i, known to all other nodes, and possibly to the attacker(s).
- a public key Pub_i known to all other nodes, and possibly to the attacker(s).
- a private key Prv_i known to node i only, possibly revealed to the attacker that compromises the node.

1) Encrypt each entry with public key: The source node establishes the routing list (set of IDs), then it encrypts each entry ID_i[3] with the entry's public key Pub_i (Fig. 5). To avoid replays by attackers, the source should concatenate the entry's ID with a random short string (or nonce) before encrypting it. This method ensures that for a given forwarder/destination ID_i, *only* ID_i is capable of checking whether it belongs to the forwarding list, by using its private key Prv_i to decrypt the ciphered entry, and checking its integrity. This makes it impossible to attackers, whether insiders or outsiders, to read the entries of the routing list.

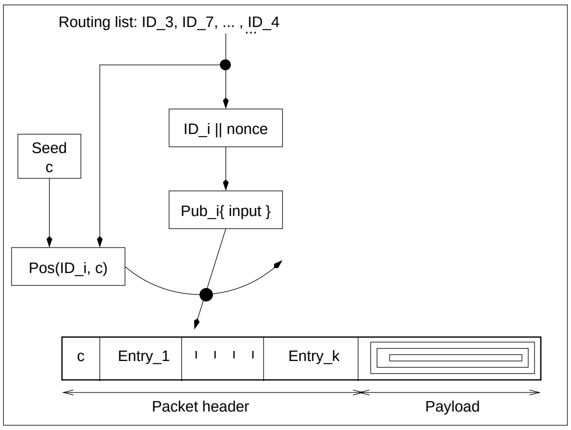

Fig. 5. Encrypting the routing list at the source. $Pub_i\{M\}$ stands for the encryption of M using the public key of node i

Upon receiving a packet, nodes should check whether they are included in the ciphered list. One straightforward solution for a node ID_j is to decrypt every entry in the list using its private key Prv_j and check whether it retrieves its ID_j. However, decrypting every entry in the (potentially large) routing list is a time and power consuming task. To cope with this, the source should give every forwarder a hint on which position in the list to decrypt, reducing the decryption load, in a way such that the hint cannot be used by attackers to identify the forwarder in the list. This is done as follows (cf. Fig. 5):

The source uses a function $Pos(id, c)$ which first input is the ID of the node to be inserted in the list, and second input is a seed number c. The output of the function is a number pointing to the position to fill in the routing list. c is added to the routing list, in clear. If two or more IDs happen to refer to the same position in the list, the source changes c and computes the positions again until no collisions occur. Empty positions are filled with random text to make it impossible for attackers to reduce the anonymity set by distinguishing empty and filled positions in the routing list.

Upon receiving a packet, a node ID_i reads c from the packet header and computes $Pos(ID_i, c)$. It then decrypts the entry at position $Pos(ID_i, c)$ in the list using its private key Prv_i. If it reads its ID inside, ID_i, then it knows it belongs to the list, and it forwards the packet. Else, the field must be containing a different ID_j encrypted with Pub_j, or it is just random text. In both cases ID_i is not in the list, and the node discards the packet.

For the intended destination ID_d to know whether it should read the payload, the source may include some indicators along with ID_d in the packet header, before encrypting them with Pub_d.

As for the payload, we totally separated it from the routing list encryption in order to keep it possible to use multicast lists, as described before. Section IV-C discusses the various

[3] Or any conventional short text.

operations that we can apply to the packet payload.

The big advantage of this technique is that compromising one or several nodes (and their keys or ID databases) does not compromise the whole system. It limits the attacker to the knowledge of whether the compromised node belongs to the routing list, without compromising any other information.

Since the routing list is encrypted using asymmetric keys, attackers have no way to read the list entries but the ones of compromised nodes. However, if the routing list remains ordered, local attackers still can get valuable information such as how far they are from the source, from the destination, or in which direction the packet is being routed. Therefore disordering the routing list remains useful even after asymmetric encryption of the list entries.

This method, though practical and clear, lacks three functionalities:

- It does not prevent routing loops (though it is easy to introduce)
- Packet headers cannot be easily changed over the route by intermediate nodes (to avoid tracking packets)

Those functionalities are the two additional building blocks of the techniques presented in Section IV-E.

2) Chinese remainder theorem (CRT) for list aggregation: The traffic source cannot use Bloom filters to compress the routing list since encryption/decryption are asymmetric. We therefore adopt the mechanism presented in [16], based on the Chinese remainder theorem, which in the context of ad hoc networks can be described as follows:

Let each node i be assigned a public number p_i, where all p_is are co-prime,along with its public key Pub_i and private key Prv_i.

When a given source wants to send a set of values x_i to a set of nodes ID_i, it computes the single value X that satisfies $0 < X \leq \prod_{i=1}^{n} p_i$ according to the CRT. Upon reception of X (i.e. the aggregated list of values), each node i recovers x_i from X by computing $x_i = X \mod p_i$. To send a private binary information to a node, i.e. whether it belongs to the list or not, the CRT can be used as follows:

- For each forwarding node i, the source node chooses a random value R_i and sets $x_i = Pub_i\{R_i || hash(R_i)\}$
- The source node computes X using the Euclidean algorithm
- X is sent in the packet header as an aggregated list of forwarding nodes
- When node i receives X, it computes $\hat{x_i} = X \mod p_i$
- Node i decrypts $\hat{x_i}$ using its private key Prv_i
- If the integrity check (using $hash(R_i)$) of x_i succeeds, i infers that it belongs to the routing list. Else, i knows it does not belong to the list and discards the packet.

Discussion: To make the encryption lighter, the source may establish symmetric keys with every forwarding node using public/private keys at an initial phase. This makes encryption/decryption much lighter at subsequent phases (i.e. during message forwarding), using symmetric cryptography only. However, this comes at a high cost: Using the shared key, the source ID will be revealed to any attacker that compromises any of the forwarding nodes. Moreover, the source will have to give routing nodes a hint on its ID for them to use the proper symmetric key to extract their entries. This makes the symmetric solution inconvenient for non-tamper-resistant devices.

In Section V we introduce a technique that assumes the packet destination can be an eavesdropping node close to the packet route. If the packet payload is encrypted, and if the destination D is not included in the aggregated routing list X, D will have to decrypt every packet it eavesdrops, a very costly solution. Therefore, the source must include D in X, with an additional hint concatenated to its ID, that it is the intended destination. Still, the attacker will have no clue of D's presence in the list, unless the attacker compromises D itself, and the eavesdropping destination benefits (increase anonymity set size) still apply, while reducing the decryption costs at D. Note that X does not compress the routing list but rather aggregates it. We are considering the compression of X in future extensions of this work.

C. Packet payload

Due to the separation between the packet header and payload, different operations can be applied to the payload:

- If it is not necessary to hide the content of the message, the payload can be kept in clear.
- At the source node, the payload can be encrypted using the intended destination's public key Pub_D. At each forwarding node the payload remains unchanged. At the destination D, the payload is decrypted using D's private key Prv_D. Since the payload remains unchanged along all the route this technique allows recognizing and tracking packets by global attackers.
- The source encrypts the payload, as in onion-routing (however without including the IDs inside the onion) using the public keys of the forwarding nodes until the intended destination (Fig. 6 and 7). Upon detecting its ID in the forwarding list, decrypting it with its private key, a node peels a layer of the payload onion and transmits the packet. If the transmitting node is on the "right" route (e.g., in Fig. 6, nodes 8,1,6,5) from the source (S) to the intended destination (D'), the message will arrive properly peeled using the right sequence of public keys. On redundant routes (e.g., 8,3,0), the payload will be peeled using wrong private keys, or wrong private key sequence, arriving to the route ends with no meaningful message. To an attacker, whether internal or external, the "right" route remains indistinguishable from redundant ones, considerably decreasing his ability to analyze the traffic. Furthermore, the encrypted payload is changing as it progresses along the route(s), making it hard for attackers to identify the packet (unless by recognizing the header).

Similar to the packet header coding methods, the onion payload can be made using symmetric shared keys, however this comes at the cost of revealing the source ID.

Note that multicast would not have been possible without the separation of the routing list from the payload onion: encrypting the receivers' IDs inside the onion layers will make

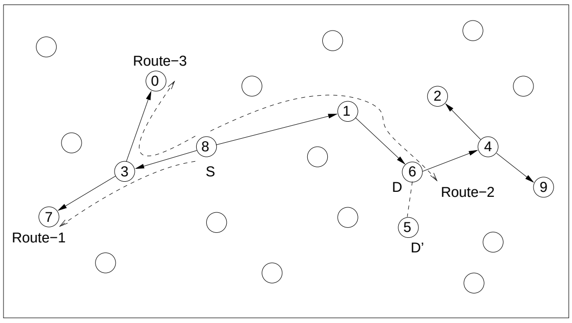

Fig. 6. Example of packet routing concealed by multicast trees.

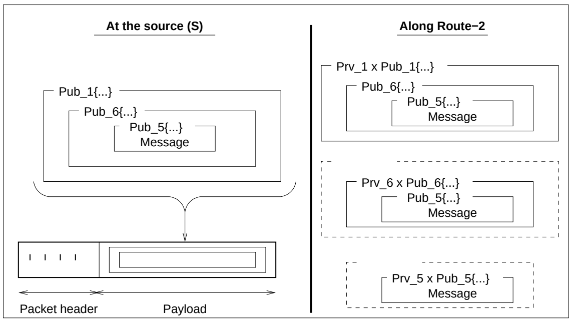

Fig. 7. Decryption of the payload. $Prv_j \times Pub_i\{M\}$ stands for the decryption of $Pub_i\{M\}$ using the public key of node j. If $j = i$ decryption is successful, else it results in random text.

the packet go on the correct path while being decrypted correctly. On redundant paths, though, the packet gets decrypted with the wrong private keys resulting in random outputs, making it impossible to include the IDs in the onion layers. One possible alternative is to include several payloads (as in Fig 3(b)), one per forwarder, in the onion layers. This makes the overall payload onion tremendously large, without bringing any advantages. The separation between the routing list and the payload onion is therefore better adapted to multicast.

D. Avoiding routing loops

Changing the payloads was described in the previous section. Still, packet headers remain unchanged, resulting in two problems:

- Routing loops occur at any segment of the whole packet path(s)
- With constant packet headers, all packets are easily traceable by an attacker

In this section we solve the routing loops issue, and in the next section we show mechanisms to make packet headers untraceable.

Since the routing list in a packet header is disordered, does not contain “from” and “to” fields, and since packets should not contain fixed identifiers, packets will “bounce” backward and forward on every segment of the packet path(s). The following techniques help avoiding routing loops/bounces:

- Using TTLs (Time To Live). However, this would not avoid the packet from continuously “bouncing” until the TTL expires
- Every forwarding node “marks” the packet by adding its own “reminder” to the routing list. However, when using Bloom filters adding new entries will increase false positives, therefore forwarding the packet more than necessary and consuming energy.
- Every forwarding node removes its entry from the routing list

We will describe this last option with respect to the two node models we are considering.

1) Removing entries from Bloom filters: Removing an entry from a Bloom filter solves the problem of routing loops/bounces. However, the implementation method brings another problem: A routing node cannot set any of the filter bits to zero, even the ones that are referred to by its hashed own ID. In fact, the same bit positions can be referred to by other hash functions of other IDs of nodes supposed to forward the packet further. Setting bits to zeros will exclude those nodes from the routing list, stopping the packet from being forwarded.

To solve the above problem, we consider using a variant of Bloom filters used in [17], where the bits are replaced by counters (bytes). All counters in the filters are initially set to zeros. The source node constructs the routing list and inputs each entry to the hash functions, as described before. However, instead of setting the bits where the hash functions refer to, it will increase the corresponding counters (by 1). To check whether it belongs to the routing list, a receiving node will test if the counters (instead of the bits) its hashed ID refers to are > 0 or $= 0$. If all corresponding counters are > 0, the node forwards the packet. The approach looks quite similar to the description of the basic Bloom filter, however now a forwarding node can extract its hashed ID from the filter by reducing the corresponding counters by 1. Since we use counters instead of bits, decreasing the counters at some positions will not introduce any false negatives, where routing nodes get excluded from the list and packets stop being forwarded.

On the other hand, the compression rate will decrease with respect to the basic Bloom filter, depending on how large a counter is (in bits).

2) Removing entries from CRT headers: For each routing node to remove its entry from the aggregated routing list X, a node uses all public p_i values to extract the encrypted entries of all nodes (regardless of whether they are valid or not). The node then removes its own entry from the list, keeps other nodes' entries unchanged and computes a new value X to be used for packet forwarding. This approach, though simple, shows two main drawbacks:

- The source node has to (compute and) include an entry for *every* node in the network using the corresponding p_i, a time/processing consuming procedure.
- Every forwarding node that needs to remove its entry from the X header should extract *all* entries in X, whether their decryption is valid (for forwarding nodes in the routing list) or invalid (for other nodes in the network), delete or invalidate its entry, and recompute

TABLE I
SUMMARY OF THE PROPOSED CODING TECHNIQUES

	Extensive routing list	Aggregate routing list	Avoid routing loops	Change packet on path	Decorrelate packets with similar paths
Tamper-resistant devices	Sec. III	Sec. IV-A	Sec. IV-D.1	Sec. IV-E.1	Sec. IV-E.1
Non-tamper-resistant devices	Sec. IV-B	Sec. IV-B.2	Sec. IV-D.2	Sec. IV-E.2	Sec. IV-E.2

a new X, another time/processing consuming procedure.

Even though only nodes that need to change their entries are subject to the above drawbacks, we are currently working on alleviating this procedure, taking more advantages of the mathematical properties of the CRT.

E. Making packet headers untraceable

Whenever a forwarding node removes its entry from the routing list (Bloom filter, or the CRT X), it makes the packet header change. However, such changes are not enough to make the packet header untraceable:

- Using Bloom filters with counters, constantly decreasing counters and constant positions of 0 counters give attackers a potential tool to trace packets as they progress in the network.
- Using CRT routing lists, deleting entries from a routing list X will reduce the number of included p_i. Therefore the value of X decreases since $0 < X \leq \prod_{i=1}^{n} p_i$, increasing the capability of an attacker to trace the packet.

1) Changing the Bloom filters:

- The zeros in Bloom filters with counters can be converted to “redundant counters” that help nodes make packets untraceable by randomly changing them. Each forwarding node, after removing its entry from the routing list, randomly sets the values of the “redundant counters”. Several methods can be used to indicate the positions of the “redundant” counters in the Bloom filter, such as encrypting the list of positions using the shared key Z, or constructing another Bloom filter to include the list of redundant counters’ positions.
- Instead of decreasing (respectively increasing) the corresponding Bloom filter counters by 1, a forwarding node i (resp. the source node) can decrease (resp. increase) the counters by $\delta(Z, ID_i, r(t), counter\_position)$, a shared function of (for example) the shared key Z, the node ID, the clear-text nonce and the counter position in the filter. This makes the packet headers change irregularly as they progress along their route(s), and the tracking/analysis harder to perform.
- To avoid that different packets with the same route use the same header (refer to Fig. 4), the hash functions can use the nonce $r(t)$ (transmitted in clear) as an additional input, to generate totally different bit/counter positions for similar routing lists.

2) Changing the CRT headers: To avoid that the integer value of CRT list (X) constantly decreases when deleting a nodes’ entries as it progresses on the path, nodes should rather insert an invalid entry instead of deleting it. This ensures that the number of p_is does not decrease, and therefore X ($0 < X \leq \prod_{i=1}^{n} p_i$) changes randomly, regardless from the direction the packet goes in, or from its distance to the intended destination.

V. EAVESDROPPING DESTINATIONS

Leveraging the broadcast nature of node transmissions in an ad hoc network, the source can construct the packet route to *pass by* the intended destination, in a way that this last can eavesdrop the transmission, without necessarily being on the route. This will lead to a drastic increase of the anonymity set, at very negligible cost. It further slightly reduces the size of the routing list, making it less energy consuming for transmission and for reception. In fact, let v be the average number of distinct neighbors each node has, out of which r nodes relay the packets on the multicast tree. Using this technique multiplies the average size of the multicast anonymity set by $v - r$, at the cost of additional processing at each neighbor. With respect to *unicast*, the set size increases to

$$v \times \sum_{i=1}^{h} r^i$$

i.e. v times more efficient than a unicast that does not use eavesdropping destinations.

This technique, though simple, is highly efficient. However, one should carefully design the traffic pattern of eavesdropping destinations, for instance not to send one ACK per received packet, to avoid being recognized by the attacker.

VI. DISCUSSION

Mobility: One relevant issue that was not tackled in the previous sections is about networks with mobile devices. In the system model we assumed “global routing information is requested at an initial phase”, after which the traffic sources are able to establish their anonymous routing lists. If any of the forwarding node moves away from the path, the source cannot be notified of the route break, as in normal ad hoc routing, since it is assumed to be unknown to forwarding nodes. The destination cannot notify the source neither, unless it uses a different route that did not change in the meanwhile. In the same vein, a similar problem occurs at the MAC layer: since no MAC acknowledgments are used, a routing node has no direct way to see whether a link is broken. Listening to neighbor advertisements does not solve the problem since a routing node ignores which neighbor is the next forwarding node on the path.

Therefore, periodic route requests should be used to maintain a global view of the network topology. The impact of mobility on the performance of routing protocols has been analyzed in the literature [18], in the general context, not specific to anonymity. However, the results therein can be reused in our network model since the impact of mobility is somehow orthogonal to anonymity.

Anonymity performance evaluation: The coding methods we propose aim to combine generic anonymity techniques, multicast-based or onion-based, so they can thwart generic attackers, global or local. Evaluating the performance of a given protocol (e.g. multicast-based) in terms of anonymity level is specific to the protocol itself and not to our coding methods.

VII. Related work

Considerable work has been done on anonymous routing and untraceability. We limit this discussion to the ones of nearest link to our work. In [1] Kong and Hong proposed a novel technique called ANODR which introduces anonymous connection setup into the route discovery process using link pseudonyms. ANODR relies on the novel idea of broadcast with trapdoor information. ANODR provides excellent performance to thwart *local* attackers, but it does not diversify packet routes. Therefore, traffic analysis by a *global* attacker enables him to reveal the sender and destinations IDs. In [3] Jiang et al. proposed a method based on Mix routes. Mix is controlled by specific mix nodes, also called “dominators” that diversify routes. It relies on onion routing and has the following properties/limitations:

- Each Mix only knows the previous and next Mix
- The first and last Mix know the sender and recipient of the message, respectively
- High energy consumption at the Mixes
- For a global attacker, source and destination are still not well hidden. Stop-and-go, and dummy packet injection can help improving sender/recipient anonymity. This was proposed by the authors briefly, without getting into the details.

In [2], Deng et al. show how traffic analysis can localize traffic sinks in the network and propose using multicast trees to provide route diversity in the network and hide traffic sources and destinations. The approach is quite appealing, but the paper provides a high-level description, without dealing with implementation issues, such as how to combine multicast with onion routing (used in other anonymity techniques). In [16] Molva and Tsudik propose using the CRT to provide “secret sets” in generic communications. Our work differs from [16] by the fact that it applies the CRT to ad hoc multi-hop routing, combining it with onion routing, and by using considerably “lighter” mechanisms when tamper-resistant devices are used. Note that our coding techniques can be used in combination with most of the anonymity techniques we describe above and other ones like stop-and-go, varying packet sizes etc. Our techniques help combining separate anonymity techniques to reach higher levels of anonymity than the ones they reach separately.

VIII. Conclusion

In this paper we first showed the relevance of using multicast for anonymity in ad hoc networks. Multicast considerably reduces the number of transmissions, with respect to anonymity techniques based on unicast, whenever a given anonymity set is to be reached. To make multicast work with anonymity techniques, such as onion encryption, we devised techniques for packet routing headers and the packet payloads separately. The resulting combination is a constantly changing/unrecognizable packet (header and payload), being routed on a multicast tree to reach a given anonymity set while reducing the transmission costs. Different novel techniques were introduced for different network models: for networks with tamper-resistant devices, the anonymity techniques we introduce alleviate the use of encryption, making it “light” enough without compromising the required level of anonymity. For networks with non-tamper-resistant devices, we introduced new techniques that guarantee anonymity, even when a large number of nodes gets compromised. Though novel, the techniques we present may be combined or adapted to existing security protocols to thwart their corresponding security attacks, without being constrained by our techniques. We evaluate each technique to show its performance costs and benefits.

References

[1] J. Kong and X. Hong, “ANODR: ANonymous On Demand Routing with Untraceable Routes for Mobile Ad-hoc Networks,” in *Proceedings of MobiHoc*, 2003.

[2] Jing Deng, Richard Han, and Shivakant Mishra, “Countermeasures Against Traffic Analysis Attacks in Wireless Sensor Networks,” in *Proceedings of IEEE/CerateNet Conference on Security and Privacy in Communication Networks (SecureComm)*, 2005.

[3] Shu Jiang, Nitin Vaidya, and Wei Zhao, “A mix route algorithm for mix-net in wireless ad hoc networks,” in *Proceedings of MASS*, 2004.

[4] Yanchao Zhang, Wei Liu, and Wenjing Lou, “Anonymous communications in mobile ad hoc networks.” in *Proceedings of IEEE Infocom*, 2005.

[5] Bo Zhu, Zhiguo Wan, Mohan S. Kankanhalli, Feng Bao, and Robert H. Deng, “Anonymous Secure Routing in Mobile Ad-Hoc Networks,” in *Proceedings of of the 29th Annual IEEE International Conference on Local Computer Networks (LCN)*, 2004.

[6] Marc Rennhard and Bernhard Plattner, “Practical anonymity for the masses with mix-networks,” in *Proceedings of IEEE WETICE*, 2003.

[7] D. Chaum, “Untraceable electronic mail, return addresses, and digital pseudonyms,” *Communications of the ACM*, 1981.

[8] Paul F. Syverson, David M. Goldshlag, and Michael G. Reed, “Anonymous connections and onion routing,” in *Proceedings of IEEE Symposium on security and privacy*, 1997.

[9] David B. Johnson and David A. Maltz, “Dynamic source routing in ad hoc wireless networks,” *In Imielinski and Korth, editors, Mobile Computing, volume 353. Kluwer Academic Publishers*, 1996.

[10] Y. Hu, A. Perrig, and D. Johnson, “A secure on-demand routing protocol for ad hoc networks,” in *Proceedings of MobiHoc*, 2002.

[11] Charles E. Perkins and Elizabeth M. Royer, “Ad hoc on-demand distance vector routing,” in *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, 1999.

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

[13] A. Broder and M. Mitzenmacher, “Network applications of Bloom filters: A survey,” in *Allerton*, 2002.

[14] Claude Castelluccia and Pars Mutaf, “Hash-based dynamic source routing,” in *Proceedings of Networking*, 2004.

[15] Sergio Marti, T. J. Giuli, Kevin Lai, and Mary Baker, “Mitigating routing misbehavior in mobile ad hoc networks,” *Mobile Computing and Networking, pages 255-265*, 2000.

[16] R. Molva and G. Tsudik., “Secret sets and applications,” *Information Processing Letters*, 1998.

[17] Li Fan, Pei Cao, Jussara Almeida, and Andrei Z. Broder, “Summary cache: a scalable wide-area web cache sharing protocol,” in *Proceedings of the ACM SIGCOMM*, 1998.

[18] Fan Bai, Narayanan Sadagopan, and Ahmed Helmy, “IMPORTANT: A framework to systematically analyze the Impact of Mobility on Performance of RouTing protocols for Adhoc NeTworks,” in *Proceedings of IEEE Infocom*, 2003.