SRDP: Securing Route Discovery in DSR

Jihye Kim and Gene Tsudik
Computer Science Department
University of California, Irvine
{jihyek, gts}@ics.uci.edu

Abstract

Routing is a critical function in multi-hop mobile ad hoc networks (MANETs). A number of MANET-oriented routing protocols have been proposed, of which DSR is widely considered both the simplest and the most effective. At the same time, security in MANETs – especially, routing security – presents a number of new and interesting challenges. Many security techniques geared for MANETs have been developed, among which Ariadne is the flagship protocol for securing DSR.

The focus of this work is on securing the Route Discovery process in DSR. Our goal is to explore a range of suitable cryptographic techniques with varying flavors of security, efficiency and robustness. The Ariadne approach, while very efficient, assumes loose time synchronization among MANET nodes and does not offer non-repudiation. If the former is not possible or the latter is desired, an alternative approach is necessary. To this end, we construct a Secure Route Discovery protocol (SRDP) which allows the source to securely *discover an authenticated route to the destination using either aggregated Message Authentication Codes (MACs) or multi-signatures. Several concrete techniques are presented and their efficiency and security are compared and evaluated.*

1. Introduction

Multi-hop Mobile Ad Hoc Networks (MANETs) have been studied extensively in recent years and a large body of relevant research has been accumulated, especially pertaining to routing security.

One of the key MANET characteristics – absence of fixed infrastructure – makes it difficult to re-use results from more traditional wired networks. In particular, popular IP routing protocols (used both in the Internet and in private intranets) are not suitable for MANETs, due mostly to node mobility. Consequently, a lot of effort has gone into developing MANET-geared routing protocols. Most of these protocols have, for various reasons, remained on paper, only a few have been implemented and even fewer have made into real MANETs.

Since the focus of this paper is on security, rather than routing, we do not review the relevant routing literature. Suffice it to say, that the most popular MANET routing protocol is also one of the conceptually simplest, Dynamic Source Routing (DSR), developed by Johnson and Maltz. The centerpiece of DSR is the Route Discovery (RD) protocol which uses flooding to discover routes on-demand. (See section 2 below for a detailed description.)

Like most network protocols, MANET routing protocols (including DSR) are often designed for non-adversarial networks and thus forgo security features. This follows the traditional model of first designing a protocol and later (sometimes much later) retrofitting it with security features.

Being a popular protocol, DSR has received a lot of attention from the security community. The state-of-the-art of MANET routing security is represented by Ariadne [11] which is a DSR-specific security mechanism based on the earlier TESLA protocol [17]. Ariadne's security is based on Message Authentication Codes (MACs) and loose time synchronization among nodes is required; the latter feature is inherited from TESLA.

The motivation for the work presented in this paper is very similar to Ariadne's. Our goal is to efficiently secure the Route Discovery process in DSR.[1] However, in doing so, we aim to address the needs of MANETs where either (or both) stronger security is necessary or loose time synchronization is not possible. Protocol ef-

[1]Once a route or a set of routes is securely discovered, security of subsequent data transmission is out of scope of this paper.

ficiency is also one of our goals, especially, the minimization of communication (bandwidth) overhead. This contrasts with Ariadne which focuses more on lowering computation costs.

With the above goals in mind, we develop Secure Route Discovery Protocol (SRDP). It is a generic protocol which works with a range of cryptographic primitives, some based on aggregated MACs and others – on digital signatures amenable to aggregation. (Aggregation is essential as it allows us to compress authentication tags thus saving bandwidth and reduces verification costs.) We explore five cryptographic techniques and evaluate/analyze their respective security features and efficiency features. One of the interesting aspects of our work is the novel application of aggregated signature and multi-signature schemes.

Viewed from the higher-level perspective, SRDP enhances the functionality of DSR with the feature we term *Route Integrity*. Informally, this means that all nodes in a putative route agree on the exact sequence (order) of nodes traversed in that route. Moreover, the source is able to ascertain that all intermediate nodes vouch for the integrity of the *same* route. (See section 4 for further details.) However, *Route Integrity* does not imply viability of the discovered route, since an adversarial node that behaves honestly during Route Discovery may behave in an arbitrarily malicious manner during subsequent forwarding of data packets.

Organization: the remainder of this paper is organized as follows: section 2 summarizes the basic operation of DSR. Then, section 3 presents our network assumption, attack model and defines necessary security properties. Section 4, describes SRDP and several cryptographic techniques. Their efficiency is discussed in section 5. Finally, section 6 overviews relevant prior work.

Remark: due to length restrictions, this paper does not include security proofs for the proposed cryptographic techniques. For the same reason, the descriptions of the schemes are quite terse and lack the ideal level of detail. Also omitted is the discussion of future work and an in-depth treatment of prior results (although section 6 provides a short summary).

2. DSR Overview

Since our work is specific to DSR, this section provides a brief re-cap of the DSR Route Discovery process. For further details we refer to [12].

DSR is a purely on-demand ad hoc network routing protocol. This means that a route is discovered only when it is needed and no pre-distribution of connectivity is performed. Since route discovery is done via flooding, nodes do not accumulate network topology information except for cached routes.

DSR includes two main mechanisms: *Route Discovery* and *Route Maintenance*. *Route Discovery* is used to discover a route from a given source to a given destination, while *Route Maintenance* is used to manage (cache, expire, switch among) previously discovered routes. Since our focus is on *Route Discovery*, we do not further discuss *Route Maintenace*.

Route Discovery is composed of two stages: *Route Request (RREQ)* and *Route Reply (RREP)*. Whenever a source needs to communicate to a destination and does not have a route in its Route Cache, it broadcasts a *RREQ* message to find a route. Each neighbor receives the *RREQ* and (if it has not already processed the same request earlier) appends its own address to the address list in the *RREQ* and re-broadcasts the packet. This process continues until either the maximum hop counter is exceeded (and *RREQ* is discarded) or the destination is reached. In the latter case, the destination receives the *RREQ*, appends its address and generates a route reply packet (*RREP*) back towards the source using the reverse of the accumulated route. Unlike *RREQ*, *RREP* percolates towards the source via unicast. When the source finally receives *RREP*, it stores the route in its Route Cache.

Figure 1 illustrates an example of *Route Discovery* and figure 2 shows the processing of *RREQ* and *RREP* packets.

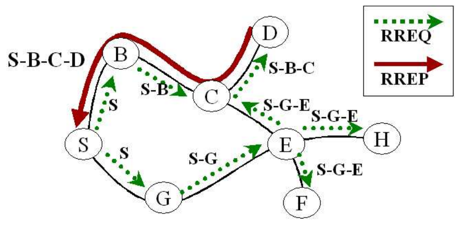

Figure 1. Route Discovery example: S **broadcasts** $RREQ$ **and each intermediate node re-broadcasts only if** $RREQ$ **has not been seen before. When** D $RREQ$**, it sends** $RREP$ **back via unicast. In this example,** D **replies to** S **with a route: <B, C, D>.**

DSR Route Discovery also includes some optimization measures: when processing a route request, an intermediate node can be authorized to issue a complete route reply if it already has a valid route to the destination in its route cache. An intermediate node can also switch its network interface into promiscuous mode, in order to harvest routes from passing route replies. How-

RREQ Broadcast	$S \rightarrow *$:	$< RREQ, S, D, S_{id}, () >$
	$B \rightarrow *$:	$< RREQ, S, D, S_{id}, (B) >$
	$C \rightarrow *$:	$< RREQ, S, D, S_{id}, (B, C) >$
RREP Unicast	$D \rightarrow C$:	$< RREP, D, S, S_{id}, (B, C, D) >$
	$C \rightarrow B$:	$< RREP, D, S, S_{id}, (B, C, D) >$
	$B \rightarrow S$:	$< RREP, D, S, S_{id}, (B, C, D) >$

Figure 2. RREQ and RREP processing example.

ever, in this paper, we focus on the basic Route Discovery, without considering these optimizations. (In fact, the first optimization mentioned above is incompatible with our proposed mechanism.)

3 Security Setting

In this section, we discuss our attack model and associated threats.

As usual, we distinguish among passive and active adversaries. A typical passive adversary only eavesdrops and aims to compromise communication privacy. Since routing is not usually a private function (except in military and other critical settings), we do not consider passive threats in our model.

An active adversary has far stronger capabilities. It can introduce its own packets as well as delete, delay and modify packets before forwarding them. We focus on protection against active adversaries.

The adversary's power is characterized by the combined coverage of compromised nodes. For example, to affect a message at a certain point, at least one compromised node needs to be present within the radio range of that point.

We say that nodes are *physically* connected if they can communicate directly, while nodes are *logically* connected if they communicate indirectly, via other nodes.

The connected adversaries can cause *unavoidable* attacks by colluding. Specially, they can delete or add a list of honest nodes sandwiched between them. To formalize it, we define *feedback loop*.

Definition 1. A *feedback loop* is an ordered sequence of linked honest nodes where both end-points are closed by compromised nodes to be able to communicate each other through their physical or logical channel.

For example, figure 3 shows two compromised nodes, *A1* and *A2*, as the compromised end-points of a feedback loop *N3-N4*. We note that feedback loop end-points do not need to be distinct, i.e., both end-points can be represented by the same compromised node. Also, the same set of compromised end-points can correspond to multiple feedback loops.

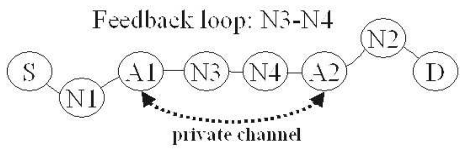

Figure 3. Feedback Loop Example.

In the context of DSR Route Discovery, controlling a feedback loop allows the adversary to control the presence of honest nodes (within the loop) in a source route. This, in turn, allows the adversary to create a fraudulent route. However, the adversary cannot make the source accept a counterfeit route which contains honest node(s) other than feeback loops it owns. In other words, each source(i.e. intermediate node) in the source routing protocol should be authenticated.

Definition 2. We say that Route Discovery is *secure* if, given a putative route: (1) the source can securely verify the presence of each honest node that appears in the route, and (2) for all honest nodes appearing in the route, their view of the route is either the same or, if not the same, the discrepancy is unambiguously detected by the source.

Note that the above definition implies that an honest node can not appear in the route unless it actually took part in Route Discovery that led to that route.

In contrast, the adversary model in Ariadne [11] is based on the number of nodes the adversary owns in the network as well as the number of honest nodes that the adversary has compromised. Our model is different since the distinction between "owned" and "compromised" nodes does not apply to our setting. Basically, we assume that each node has its own private/public key pair, and once a node is compromised it will be owned by the adversary including its key pair. Thus, there is no difference between "owned" and "compromised"nodes in our assumption.

4 Secure Route Discovery Protocol (SRDP)

We begin by stating some environmental assumptions and summarizing our notation.

We assume bidirectional communication on each link: if node S is able to send a message to node D, then node D is able to send to node S. This assumption is justified, since many wireless MAC-layer protocols, including IEEE 802.11, require bidirectional communication.

We **do not** assume that a node is aware of the exact set of its current immediate neighbors. Some MANET types have built-in neighbor discovery but we choose to err on the side of flexibility. (However, secure neighbor discovery would only serve to strengthen the security of our techniques.)

We assume that both the source and the destination are honest. (We use the generic terms *source* and *destination* to mean, respectively, the initiator and the target of the Route Discovery protocol in DSR.)

The cryptographic techniques that we propose further below fall into two categories: shared-key MACs and public key signatures.

Although sometimes implemented with the aid of conventional ciphers, MAC functions are often constructed using cryptographically suitable keyed hash functions (e.g, SHA-2). The most common MAC construct is the HMAC [3]. Clearly, any MAC function used for authentication purposes in a MANET setting would require $O(n^2)$ pairwise shared keys.

For schemes based on MACs, we assume the existence of a secure key distribution mechanism. For signature-based schemes, we likewise assume an appropriate mechanism for the issuance, distribution and revocation of public keys, i.e., a Public Key Infrastructure.

The table below summarizes the notation and formats used in the rest of the paper.

S, D, B, C, I, J	network nodes
$RREQ, RREP$	DSR Route Request & Route Reply packets
S_{id}	unique id assigned by S to Route Request
K_{IJ}	secret key shared between I and J
(x_i, y_i)	node I's private and public key pair
$MAC_{IJ}(M)$	MAC on message M under key K_{IJ}
δ, σ	authentication tags

4.1 Forward vs Backward Authentication

Recall that the goal of the DSR Route Discovery protocol is to discover a viable route. A route is accumulated incrementally via flooding until the destination is reached, at which point, the route is confirmed by revisiting it by unicast in reverse order.

One natural security strategy is to perform "forward authentication" of Route Request (RREQ) packets as they propagate from the source to the destination. Each node can compute and add its authentication tag to the RREQ before re-broadcasting it.

The chief advantage of this approach is that it would allow the destination to authenticate the accumulated source route before it generates a RREP back towards the source. However, there are also some drawbacks or issues:

- First, a node that processes a RREQ packet has no assurance of being on the eventual route. In fact, in a large MANET, it is safe to say that many nodes that process a given RREQ will not be part of the route. Thus, computing an authentication tag can be wasteful for two reasons: (1) it requires computation that may wind up being unnecessary, and (2) it costs in terms extra bandwidth since each new authentication tag makes RREQ longer.[2]

- Second, even is the above is justified, the authentication tags must be eventually verified. This can be a very expensive procedure (only if signatures are used) since each node in the route would authenticate a distinct route prefix. For example, given an actual route: $S - B - C - D$, node B would authenticate a route prefix $S - B$, node C would authenticate $S - B - C$, and so on.

- Third, we note that, if a particular sequence of nodes winds up forming a viable route, the destination generates a Route Reply (RREP) which then traverses the very same sequence of nodes in the reverse order.

These issues motivate us to explore the alternative: "backward authentication" of RREP packets.

However, we observe that, restricting authentication to RREP packets is not sufficient for secure Route Discovery. For example, consider an attack whereby two colluding compromised nodes F and E in the actual route: $S - B - F - C - E - D$ present an honest node C with prefix $S - F - C$ in a RREQ, but, later in a corresponding RREP, present C with a differently prefixed route, e.g., $S - B - C - E - D$. Unless C caches the route prefix (or a function thereof) received in a RREQ, it is unable to detect this attack.[3] To address attacks of this type, we require each node that processes and re-broadcasts a RREQ packet to compute and cache the hash of the route prefix (together with other values in $RREQ$ such as S, id_S, D). The added burden is truly minimal since DSR already requires each node to cache (for some pre-set interval) some information about each RREQ; this is done to prevent duplicate re-broadcasting.

[2]Of course, RREQ is supposed to become longer as the route is accumulated, however, depending on the underlying cryptographic primitive, a MAC can add at least 80 bits per hop and a signature – at least around 320 bits.

[3]Although we use the term *attack* here, it is debatable whether this can actually result in any real damage.

Backward authentication is conceptually very simple: each node in the route "sees" the entire route as it processes (via unicast) the RREP packet. It can thus easily compute an authentication tag (MAC or signature) and append it to the packet. Moreover, an intermediate node can also perform a "sanity check" on the route by checking for anomalies, such as loops, routes that are too long or impossible according to its own cache, etc. When a RREP with a route of length t finally reaches the source, the latter can easily verify each tag and, if all tags are verified, conclude that all nodes' view of the route is exactly the same.

Therefore, any modification of the route as it propagates back in a sequence of RREP packets, is ultimately detected by the source. Equivalently, route integrity is guaranteed once it reaches the destination.

We claim that the combination of prefix (hash) caching and backward authentication severely limits the scope of possible attacks on the DSR Route Discovery Protocol. The only attacks not addressed are those caused by *feedback loops* (see definition 1). Two types of attacks are possible: (1) the adversary can delete from the route honest nodes that are "sandwiched" between a pair of compromised nodes, or (2) the adversary can add to the route a set of compromised nodes as long as it inserts them between a pair of other compromised nodes in the route. However, the adversary is unable to manipulate any honest nodes in the route that are positioned outside any feedback loop.

4.2 Whither Tag Aggregation?

The purpose of an authentication tag is for each node in the route to independently convince the source that its view of the route agrees with that of all other nodes. It is thus quite natural to assume that the source needs to *explicitly receive* each authentication tag as part of the final RREP packet. Given t nodes in the route, the source can separately verify each corresponding authentication tag and discard the whole route if just one verification operation fails.

However, we assert that the verification of route integrity should ideally be an *all-or-nothing* operation: either all tags are valid or at least one is invalid and the whole route cannot be trusted. There does not seem to be any benefit in piece-meal authentication of individual tags.[4]

[4]One possible exception is to identify misbehaving nodes that purposely compute their own invalid tags. However, it is unclear how a distinction can be made between (1) such a misbehaving node and (2) an honest node that is "framed" by a malicious node closer to the source that modifies the honest node's tag.

Another reason to re-consider the need for explicit transmission of individual authentication tags is bandwidth overhead. As alluded to in section 4.1, a MAC costs at least 80 bits and a signature – around 320 bits (e.g., for DSA). Especially with longer routes, this represents significant overhead, as compared to, say, an IPv4 interface address which is a mere 32 bits.

Fortunately, there are a number of techniques (varying widely in complexity) that allow aggregation of authentication tags: both MACs and certain types of public key signatures. The term *aggregation* means that a collection of t tags can be transformed via an aggregation function into a fixed-sized aggregated tag which can be verified as a whole, in a single verification step. We consider such techniques in more detail in section 4.5.

4.3 Signatures or MACs?

Another important aspect of our design has to do with the choice of underlying cryptographic primitives. Route integrity, as we defined it earlier, necessitates authentication (verification) by the source of each intermediate node's view of the route. Broadly speaking, authentication can be obtained via MACs or signatures. The former requires a unique secret key for each distinct pair of nodes, and the latter – a public/private key-pair for each node in the network (plus, the public key of each node must be somehow made globally available).

Ultimately, the choice is determined by a combination of factors, based partially on the answers to the following questions:
* **Is pair-wise key pre-distribution (or on-demand pair-wise key distribution) possible?** If yes, MACs are ideal, unless the answer to any of the next two questions is yes.
* **Do intermediate nodes need to verify (partial) route integrity?** If yes, only signatures can be used.[5]
* **Is non-repudiation important?** If yes, only signatures can be used.
* **Is there a Public Key Infrastructure?** If not, either MACs or identity-based (certificate-less) public key signatures can be used. In either case, however, revocation is a major headache.
* **Is computation overhead a major issue?** If so, MACs are clearly preferable.

4.4 Generic SRDP

We now present a high-level description of SRDP. Figure 4 shows an example with a route of length 3.

[5]MACs are unsuitable since their use would dramatically increase bandwidth consumption. It is easy to see that, if implemented naively, roughly $t^2/2$ separate MACs would be needed. Even an optimized version (left as a trivial exercise) would require t MACs in each RREP packet.

We use two types of authentication tags, δ and σ. The former (δ) is optionally generated by the source and placed into the Route Request. Its purpose is to offer the intermediate nodes a chance to authenticate the origin and the contents of the original Route Request (as produced by the source). We say "optionally" since this tag only makes sense if the source is using public key signatures to produce it.[6] Even if δ is present, the decision to verify it is left up to the individual nodes. Alternatively, δ can be thought of as a MAC computed under K_{SD}, a key that the source already shares with the destination. In that case, only the destination node would verify δ. In any case, we consider δ to be relatively unimportant as a security feature; it serves only as a measure to prevent certain DoS attacks. We will discuss about source authentication issues in detail in the section 4.6.

The second authentication tag σ is computed incrementally by each node in the route as the Route Reply packet propagates back to the source. We use σ_{DCB} to denote an aggregated authentication tag of nodes D, C and B. (As described in the next section, depending on the cryptographic technique, the aggregated tag is either a multi-MAC or a multi-signature.) Note that σ_{DCB} implies the order of the nodes in the route indirectly, since the authenticated "message" (i.e., the route) lists all node names in order.

S:	generate $RREQ =< RREQ, S, D, S_{id}, \delta >$
$S \longrightarrow *$:	$RREQ_0 =< RREQ, (.) >$
B:	cache $RREQ_0$
$B \longrightarrow *$:	$RREQ_1 =< RREQ, (B) >$
C:	cache $RREQ_1$
$C \longrightarrow *$:	$RREQ_2 =< RREQ, (BC) >$
D:	optionally verify δ
	cache $RREQ_2$
	generate $RREP =< RREP, D, S, S_{id}, (B, C, D) >$
	compute σ_D
$D \longrightarrow C$:	$RREP_0 =< RREP, \sigma_D >$
C:	fetch $RREQ_1$, verify route prefix
	optionally verify σ_D
	compute σ_{DC}
$C \longrightarrow B$:	$RREP_1 =< RREP, \sigma_{DC} >$
B:	fetch $RREQ_0$, verify route prefix
	optionally verify σ_{DC}
	compute σ_{DCB}
$B \longrightarrow S$:	$RREP_2 =< RREP, \sigma_{DCB} >$
S:	verify σ_{DCB}

Figure 4. Generic SRDP Example.

[6]Since the route is unknown a priori, δ clearly cannot be computed using conventional cryptography, unless: (1) the source computes a distinct δ for each node in the network, or (2) all nodes in the network share the same key. We consider both cases to be unrealistic.

4.5 Cryptographic Techniques

SRDP can be used with either multi-MAC or multi-signature schemes. Both types of schemes allow us to generate fixed-size authentication tags (regardless of the number of signers) and facilitate all-or-nothing verification at the source.

Informally, a multi-MAC scheme is method of combining multiple MACs into a single fixed-size aggregated MAC is such a way that the aggregated MAC can be verified as a whole.

In a multi-signature scheme [14], a set of users sign the same message and the result is a single fixed-size multi-signature. The signing cost (for each signer) is the same as in a single-signer scheme. One benefit of multi-signature schemes is that the verification time is only slightly greater than that for a single-signer scheme. Thus, multi-signature schemes are very suitable for our purposes.

4.5.1 Conventional MACs

We illustrate this very simple scheme by an example. Assuming, as in figure 4, a route of length three: $S - B - C - D$, the scheme operates as follows: When D generates the first RREP packet, it computes: $\sigma_D = MAC_{DS}(RREP)$ and composes $RREP_0$ which it sends to C. In turn, C computes: $\sigma_{DC} = MAC_{CS}(RREP_0)$, composes $RREP_1$ and sends it to B. Next, B computes: $\sigma_{DCB} = MAC_{BS}(RREP_1)$, composes $RREP_2$ and sends it to S. Finally, S receives $RREP_2$, extracts RREP, and verifies σ_{DCB} by re-computing:

$$MAC_{SB}(RREP, MAC_{SC}(RREP, MAC_{SD}(RREP)))$$

Although the size of the multi-MAC stays constant, the verification cost increases with the route length. However, this is not a problem since MACs are very efficient, especially, when based on keyed hash constructs, such as HMAC [3]. The only problem is the assumed pre-distribution of the the pair-wise shared keys. While in some MANETs this is a reasonable assumption, it is impossible in others. For this reason, we consider another MAC-like scheme based on on-the-fly Diffie-Hellman key agreement.

4.5.2 MACs based on Diffie-Hellman (DH)

In this scheme each node is assumed to have its own public/private key-pair $(x, y = g^x)$ and any two nodes, S and D can compute a shared key on-the-fly as: K_{SD}=$g^{x_S x_D} \pmod p$ by simply exchanging each other's public keys, y_S and y_D. One modular exponentiation is needed to compute each key [8]. Once a shared key is established, it can be used for conventional cryptography, i.e., encryption and MACs. We slightly modify the DH key agreement scheme for MAC in $RREP$: K_{SD}=$g^{hx_S x_D} \pmod p$, where $h = H(RREP)$ and $H()$ is a hash function that yields elements of Z_p^*.

Continuing with our example, we assume that all nodes on the route either (1) possess the source's public key y_S or (2) that y_S is distributed as part of $RREQ$ (and copied into $RREP$). Thus, upon receiving $RREP$, each node can compute a shared key with the source. However, all shared keys are computed modulo $(p-1)$, instead of modulo p.

To minimize verification costs for the source, we take advantage of the following "trick": each node computes its MAC (in $RREP$) by exponentiating the MAC computed by the previous node with the Diffie-Hellman key it now shares with the source. This is best illustrated by an example.

Before generating the initial $RREP$, D computes $K_{SD} = {y_S}^{hx_D}$ and $\sigma_D = g^{K_{sd}}$, where h is computed as $h = H(RREP)$. The next node, C computes $K_{SC} = {y_S}^{hx_C}$ and $\sigma_{DC} = (\sigma_D)^{K_{SC}}$, and so on. When $RREP_2$ is received by the source, it contains:

$$\sigma_{DCB} = g^{g^{hx_S x_D} g^{hx_S x_C} g^{hx_S x_B}} = g^{(g^{x_D + x_C + x_B})^{hx_S}}$$

Note that in order to recompute the quantity in the exponent: $(g^{x_D+x_C+x_B})^{hx_S}$, the source needs to only multiply public keys of all intermediate nodes (which is very inexpensive) and exponentiate the result with the multiplication result of its own private key and the hash of $RREP$. Thus, to verify (re-compute) σ_{DCB}, the source needs to perform two exponentiations and $t+1$ multiplications (assuming t intermediate nodes). This is significantly cheaper than computing a pair-wise Diffie-Hellman key with each node in the route.

4.5.3 Accountable-Subgroup Multi-signatures (ASM)

Micali, et al. [14] recently proposed an elegant construction for multi-signatures based on the well-known Shnorr signature scheme. The resulting scheme – called Accountable-Subgroup Multi-signatures (ASM) – has been proven secure in the Random Oracle Model under the assumption that Schnorr signature scheme is secure. In an ASM scheme, three rounds of communication are required for signing:
Round 1. Each signer sends its commitment to a distinguished player D
Round 2. D computes the joint commitment and broadcasts it to all signers
Round 3. Each signer computes a partial signature (using the joint commitment) and sends it to D

Finally, D computes a multi-signature from the collected partial signatures.

In our setting, we can reduce the number of rounds to two by distributing the role of D to each signer (intermediate node) as follows: (1) each signer computes the joint commitment incrementally by multiplying its commitment with the previous joint commitment and sends it to the next signer until the last signer is reached, (2) The last signer completes the joint commitment, computes a partial multi-signature and sends the joint commitment and a partial multi-signature back to the previous signer. In other words, the instead of using broadcast, the partial signature is accumulated hop-by-hop as the set of signers is traversed in the reverse order. Each signer thus updates the multi-signature. The first signer (last to update the multi-signature) computes the final verifiable multi-signature.

The two stages of the ASM scheme are a perfect fit for DSR Route Discovery. Joint commitments can be collected as part of the $RREQ$ stage the multi-signature can be accumulated during $RREP$ processing.

In our example with a 3-hop route, the scheme operates as follows:
$RREQ$ **stage:**

1. B: compute $t_B = g^{r_B}$ for a random r_B
2. C: compute $t_{BC} = t_B g^{r_C}$ for random r_C
3. D: compute $t_{BCD} = t_{BC} g^{r_D}$ for random r_D

$RREP$ **stage:**

1. D: compute $e = h(RREP)$ and $\sigma_D = r_D + ex_D$
2. C: compute $e = h(RREP)$ and $\sigma_{DC} = \sigma_D + r_C + ex_C$
3. B: compute $e = h(RREP)$ and $\sigma_{DCB} = \sigma_{DC} + r_B + ex_B$
4. S: compute $y = \prod_{I \in \{B,C,D\}} y_I$
   and check $g^{\sigma_{DCB}} =? = t_{BCD} y^e \pmod p$

As in the DH-based scheme, each node is assumed to have a private/public key-pair $(x, y = g^x)$. Unlike others, this scheme requires additional overhead to gradually compute the joint commitment during the $RREQ$ stage. This might seem wasteful since we argued earlier that computation during $RREQ$ stage should be minimized in section 4.2. In general, a node that receives a $RREQ$ packet has no idea whether it will wind up being on the route. But, we point out that, for an intermediate node I, the bulk of the computation required during $RREQ$ stage is: (1) generating a random exponent r_I and (2) computing g^{r_I}. (The rest amounts to a single multiplication.)

First, we note that both values can be pre-computed as they are not dependent on the specific $RREQ$. Thus, a pair (r_I, g^{r_I}) does not have to be generated on-the-fly. Each node can compute random pairs during its idle time and store them to its cache for the future use.

Second, the effort a node I invests to generate r_I and g^{r_I} is not wasted, since even if a corresponding $RREP$ never materializes, the very same (r_I,g^{r_I}) pair can be re-used in the context of another $RREQ$. (This is perfectly safe as long as the same r_i is never used twice in updating σ.)

For the re-usability, each node needs to store more information in its cache. Note that the basic DSR protocol requires a node to add an entry of the identification value and destination address into its cache for the source from $RREQ$ to avoid loops. In the integration of DSR and ASM, to save computation overhead, each node additionally keeps the (r_I, g^{r_I}) pair with a global time-out period after it uses it to process $RREQ$. The global time-out period determines if a node can safely re-use a random pair. (The time-out period should be long enough so that no $RREP$ for the $RREQ$ will not be generated by the destination after the time-out expires.) When the time-out in a random pair terminates, the random pair becomes availble for another $RREQ$ and the node saves the computation of generating a random pair by re-using the available one. Consequently, we consider the computation overhead in the $RREQ$ stage as rather negligible.

Overall, the ASM-based scheme is efficient in both generation and verification of the aggregated authentication tag σ. Computation overhead at each intermediate node is very low and the computation at the source is nearly equivalent to verifying a single Schnorr signature, regardless of the route length.

The ASM-based scheme is provably secure and, unlike the MAC schemes, provides non-repudiation in addition to authentication. Our version of this scheme is slightly different from the that in [14] since the group of signers in [14] is fixed in advance, while in our case it is determined as part of the signing process, i.e., during the $RREQ$ stage. This requires some adjustments to the security proof.

4.5.4 Multi-signatures based on Gap Diffie-Hellman (GDH) Groups

Another interesting multi-signature scheme was recently proposed by Boldyreva [4]. This scheme is based on an earlier Gap Diffie-Hellman (GDH) group signature scheme [6]. (A more general version was presented by Boneh, et al. in [5].) GDH groups are algebraic groups where the Computational Diffie-Hellman (CDH) problem is hard but the Decisional Diffie-Hellman (DDH) problem is easy.

A single-signer scheme can be obtained from such groups as follows: Each node has its own private and public key pair $(x, y = g^x)$. The signature on a message M is computed as $\sigma = H(M)^x$, where H is a cryptographic hash function. The validity of a signature σ on M under public key y is tested by checking if $(y, H(m), \sigma)$ is a valid DDH triple.[7]

When this basic scheme is extended to the multi-signature scheme, the salient property is that the product of two signatures on the same message M under two different public keys y_1, y_2 yields a signature of M under the combined public key $y = y_1 y_2$, since $H(M)^{x_1} H(M)^{x_2} = H(M)^{x_1+x_2}$. The following example illustrates the scheme:

$RREP$ **stage:**

1. D: compute $\sigma_D = H(RREP)^{x_D}$
2. C: compute $\sigma_{DC} = \sigma_D H(RREP)^{x_C}$
3. B: compute $\sigma_{DCB} = \sigma_{DC} H(RREP)^{x_B}$
4. S: compute $\tilde{y} = \prod_{I \in \{B,C,D\}} y_I = \prod_{I \in \{B,C,D\}} g^{x_I}$ and check if $(\tilde{y}, H(RREP), \sigma_{DCB})$ is a valid DDH triple (the latter required a pairing operation)

Unlike the ASM-based scheme, this scheme does not require any additional communication rounds and no pre-computation during the $RREQ$ stage. During the $RREP$ stage, the authentication tag σ is updated by each intermediate node using the BLS signature algorithm [6]. The source verifies the validity of σ under the combined public key of all nodes in the route.

The GDH multi-signature scheme is much more efficient than t instantiations of its single-signer counterpart. This is because the number of operations required for the verification of a multi-signature is almost same as in the single-signer scheme.

As mentioned earlier, this scheme works in very special algebraic settings, e.g., certain elliptic-curve groups, whereas, the Schnorr-based scheme works in a much

[7]Informally, (a, b, c) is a valid DDH triple iff $log_g(c) = log_g(a) log_g(b)$.

wider range of groups where the Discrete Logarithm Problem (DLP) is intractable. Basic operations in GDH, such as scalar multiplication and Tate pairing, are much more expensive then those in ASM-based signatures. However, like ASM, the GDH-based scheme is a signature scheme, and thus provides non-repudiation. Also, it offers provable security, as shown in [4].

4.5.5 Sequential Aggregate Signatures (SAS)

Shacham [19] developed a scheme for sequential aggregate signatures (SAS) based on homomorphic trapdoor permutations. In this scheme, each signer takes turns to (sequentially) add its signature to the current aggregate signature. Thus, the set of signers is explicitly ordered in the sequentially aggregate signature and the signers must communicate with each other during the aggregation process.

Unlike other aggregate signatures by Boneh, et al. [5], aggregation and signing are combined in a SAS scheme and both are performed incrementally by individual signers. Sequential aggregate signatures work as follows: signer 1 signs M_1 to obtain σ_1; signer 2 then combines σ_1 and M_2 to obtain σ_2, and so on. The final signature σ_t binds each signer i to M_i for all $i = 1, ..., t$.

Any SAS scheme can be easily applied to DSR Route Discovery since the route reply packet sequentially traverses every node in the route. We illustrate the concept with a SAS scheme based on RSA (as the trapdoor permutation).

Shacham presented two concrete SAS approaches based on RSA [19]. We only consider one of their approaches since the other assumes that the signers' RSA modulii are arranged in increasing order sequentially: $n_1 < n_2 < ... < n_n$. This is clearly unrealistic in an unpredictable MANET route. In the other approach, the only requirement is for all modulii to be of roughly the same length. The signature expands by t bits $b_1...b_t$, where t is the total number of signers for a given aggregate signature.

During signing, if i-th signature $\sigma_i \geq n_{i+1}$, set $b_i = 1$, otherwise, set $b_i = 0$. During verification, if $b_i = 1$, add n_{i+1} to σ_i before proceeding with the verification of σ_i.

We assume that node i's private key is (x_i) and its public key pair is (n_i, y_i), where $x_i y_i = 1 \pmod{\phi(n_i)}$. Upon receiving a $RREP$ packet each node combines its signature with a previous signature. We continue with our 3-hop example:

$RREP$ **stage:**

1. D: compute $h_D = H(RREP, (n_D, y_D))$ and $\sigma_D = (h_D)^{x_D} \pmod{n_D}$
2. C: If $\sigma_D \geq n_C$ set $\sigma_D = \sigma_D - n_C$ and $b_1 = 1$, else set $b_1 = 0$
   compute $h_C = H(RREP, (n_C, y_C))$ and $\sigma_{CD} = (\sigma_D + h_C)^{x_C} \pmod{n_C}$
3. B: If $\sigma_{CD} \geq n_b$ set $\sigma_{CD} = \sigma_{CD} - n_B$ and $b_2 = 1$, else $b_2 = 0$
   compute $h_B = H(RREP, (n_B, y_B))$ and $\sigma_{CDB} = (\sigma_{CD} + h_B)^{x_B} \pmod{n_B}$
4. S: compute $h_B = H(RREP, (n_B, y_B))$,
   $\sigma'_{CD} = \sigma_{CDB}^{y_B} - h_B \pmod{n_B}$,
   $\sigma_{CD} = \sigma'_{CD} + b_2 n_B, \;\; h_C = H(RREP, (n_C, y_C))$,
   $\sigma'_D = \sigma_{CD}^{y_C} - h_C \pmod{n_C}$,
   $\sigma_D = \sigma'_D + b_1 n_C, \;\; h_D = H(RREP, (n_d, y_d))$,
   and finally check if $\sigma_D^{y_d} \pmod{n_d}$ equals h_D

Since the SAS-based scheme is based on plain RSA, its per-signer signature Generation cost is equivalent to that of a plain RSA signature, whereas, the verification cost increases linearly in the number of signers. In other words, the number of exponentiations computed by the source (verifier) is determined by the number of nodes in the route. However, this cost can be minimized by using very small public exponents (e.g., 3). Such small exponents can speed up verification by factor of ten or more. Thus, the SAS-based scheme can be made quite practical considering that most MANETs rarely exceed 5-6 hops in diameter.

4.6 Source Authentication

The SRDP scheme to authenticate the route list is designed for the source assuming that the source is honest. However, it is possible for a compromised node to originate RREQ. The goal of this attacker is to consume network resources such as bandwidth, or node resources such as computation power and cache memory. This attack is called a Denial-of-Service (DoS) attack. Specially, this DoS attack may cause a serious problem of node resources in a secure route discovery setting, where security is overhead. In case that the same node generates a lot of fake RREQs, the intermediate nodes computation power will be easily run out.

The main reason that the DoS attack by a source is possible is that the intermediate nodes do not authenticate RREQ. To prevent the DoS attack, we need to add an authentication mechanism of RREQ. When we consider possible authentication mechanisms, the signature scheme is the only solution since the source does not know who will authenticate RREQ before it broadcasts the message. In the signature scheme as a source authentication, the generation cost of signature should be more expensive than the verification. This is because we intend to give more work to the source and minimize additional overhead of intermediate nodes.

RSA signature scheme is perfect for RREQ authentication algorithm. Basically, the source cannot impersonate other honest nodes and intermediate nodes can verify that a received RREQ has origianted from the correct source. One of benefits in RSA is its cheap verification cost. Only two multiplications are required to verify a signature with a small exponent $e = 3$. Also, Comparatively high generation cost of RSA limits the attacker power to flood a lot of RREQs.

An interesting issue is when a node verifies a RSA signature. Verfication itself is an overhead although it mitigates DoS attacks. To maximize the performance, a node verifies the signature on the way back for two reasons: (1) the source verification is meaningful only if it is included in the eventual route, and (2) it needs to do a lot of work to perform security algorithms for RREP in SRDP.

Most algorithms in SRDP add security overheads for rewinded packets. If a node checks the signautre before starting a security algorithm, it will not lose its expensive computations through comparatively cheap verification. Only ASM scheme has to compute one exponentication to process a forwarded RREQ packet. Thus, it is better to verify the signature first before each node generates a (r_I, g^{r_I}) pair in ASM. In case that the source is flooding a lot of fake RREQs this will help to abbreviate the RREQ processing cost. However, the verification cost will be unnecessary if the node is not contained in the eventual route. To rescue more computations, we consider expired random pairs as we discussed in ASM. Each node verifies the RSA signature only if it has to generate a new random pair,i.e., there is no available random pair. Note that if we are concerning more on memory resource in each node, the verification should be done all the time before each node stores RREQ information to its route request table cache.

The authentication of RREQ is necessary to prevent the attacker from flooding RREQ packets and mitigate unprofitable computations in intermediate nodes. RSA is a good solution of RREQ authentication because its generation cost is comparatively high, while its verification cost is very cheap.

5 Performance Assessment

We now assess the efficiency of the schemes described above; first from the conceptual perspective and then, in section 5.2, based on experimental results.

5.1 Cost Analysis

We compare the schemes in terms of computation overhead since communication overhead is more-or-less[8] constant. For each scheme (except conventional MAC), table 1 shows the exponent size (in bits), the number of modular exponentiations and the number scalar multiplications (for GDH only) required to generate an authentication tag.

The ASM-based scheme is the most efficient in generation costs, requiring only one modular exponentiation with exponent of size $|q|$. The SAS-based scheme also requires one exponentiation, but, with a larger exponent of size $|n|$. For roughly equivalent security, the size of q is much smaller than that of n, e.g., a 160-bit q corresponds to a 1024-bit n. In ASM, the exponentiation operation for an RREQ broadcast message becomes overhead when the node computing the exponentiation does not belong to a route path. The exponentiation overhead of ASM on the RREQ is able to be optimized by preparing a list that contains pairs of a random number and its exponentiation values statically, and choosing one pair among the list. The DH scheme needs two exponentiations with the exponent of size $|p|$. This causes its costs to be about twice that of SAS since the size p in DH is same to the size n in SAS for the same level of security. (In reality, the gap between DH and SAS is actually greater as discussed in the next section.) The GDH scheme requires one scalar (elliptic curve) multiplication, which is a more expensive operation than a regular modular exponentiation.

Table 2 shows verification costs for all four schemes assuming t intermediate nodes. The same parameters are used here, except in case of GDH, which, instead of scalar multiplications, requires two Tate pairing operations. ASM exhibits the lowest costs: two exponentiations with the exponent of size $|q|$. To verify the tag, the source computes g^{σ} and ty^{e}. The length of each σ and e is 160 bits (to match the size of the hash function output, e.g., SHA). However, in practice, SAS is far more efficient than ASM especially for short routes (i.e., few signers). This is because SAS (like the RSA scheme it is built upon) can utilize very small public exponents. We

[8]Strictly speaking, the size of RREP in the SAS-based scheme grows by one bit at each hop.

also note that verification in GDH is very expensive in practice due to the high cost of Tate pairings.

operation type	DH	ASM	GDH	SAS
exponentiations	2	1	0	1
scalar mult-s	0	0	1	0
exponent size	$\|p\|$ for both	$\|q\|$		$\|n\|$

Table 1. Authentication Tag Generation Cost for intermediate nodes.

operation type	DH	ASM	GDH	SAS
exponentiations	2	2	0	t
tate pairings	0	0	2	0
exponent size	$\|p\|$ for both	$\|q\|$ & 160		$\|n\|$

Table 2. Authentication Tag Verification Cost for the source.

5.2 Experimental Results

In this section, we show measured (experimental) results for the four public key-based schemes. We do not include the conventional MAC scheme since it is clearly much more efficient and offers a very type of security. As before, only computation overhead is taken into account.

SAS, ACM and DH schemes were implemented using the popular OpenSSL library [2] and the GDH scheme – using the Miracl [1] library (geared for pairing and other special elliptic curve operations). Each implementation was measured in a Linux environment on PIV-2.66GHz workstation with 768 MB of RAM. All measurements are in msecs.

key size		Generation			
p or n (bits)	q (bits)	DH	ASM	GDH	SAS
1024	160	20.08	2.13	7.14	4.29
2048	224	131.53	10.25	22.78	26.10
key size		Verification			
p or n (bits)	q (bits)	DH	ASM	GDH	SAS
1024	160	20.00	4.47	89.00	2.53
2048	224	132.97	17.96	256.30	7.92

Table 3. Generation & verification costs for route of length 10.

Our experimental results are summarized in table 3. All measurements assume a route of length 10. Note that GDH key size is half of that for other schemes' for the same security level, e.g., a 512-bit p in GDH corresponds to a 1024-bit p in SAS. Generation is the cost for each intermediate node and verification is the cost for the source, assuming a 10-hop route. For a given key size, the cost of verification in all schemes (except SAS) is the same regardless of the route length. Verification cost in SAS increases linearly with the number of nodes (each extra signer costs an extra exponentiation for the source, albeit with a very small exponent.)

In terms of generation costs, ASM outperforms all other schemes. It is at least 10 times faster than DH and 2 times faster than GDH and SAS. The speed gap between DH and ASM increases as the size of p grows. For example, ASM is 10 times faster than DH with 1024-bit keys, but 13 times faster with 2048-bit keys. One reason is due to the exponent in ASM being restricted to the size of q, as described in section 5.1. Another reason is that ASM uses the Montgomery exponentiation algorithm [15] which improves the performance by 40%. This algorithm (in OpenSSL) can be used only if the modulus is odd. Thus, we cannot apply Montgomery algorithm to one modular exponentiation in DH since it is based on modulo $p-1$ which is even. Instead, we use the reciprocal-based remaindering algorithm for DH, but that only yields a modest improvement of about 5%. SAS showed better performance with a 1024-bit p than GDH with a 512-bit p, however, GDH was faster with longer key sizes. Also, the speed difference between DH and SAS was notably greater in the experiments than anticipated in section 5.1. This is because SAS takes advantage of the Chinese Remainder Theorem (CRT) which cuts its time roughly in half.

In terms of verification costs, SAS exhibits the best performance even for long routes mainly because it (like plain RSA) uses relatively small exponents, e.g., $e = 3$. Thus, the SAS-based scheme represents a fairly practical solution in a typical MANET where routes sizes are at most 4-6 hops. DH verification cost shows the same trend as DH generation cost since both operations require two full-blown exponentiations with the same size exponents. ASM still has relatively good performance in verification, and GDH is the slowest due to the high cost of Tate pairing operations, as discussed in section 5.1.

6 Related Work

In this section, we briefly overview relevant prior work. The most related prior work is the Ariadne scheme by Hu, et al. [11]. Ariadne is based on TESLA – an earlier broadcast authentication scheme. Ariadne is very efficient since it uses MACs and reduces the setup cost of pair-wise shared keys by using TESLA. Also, Ariadne offers some protection against DoS attacks by requiring the destination to authenticate the source.

Ariadne inherits from TESLA the requirement for loose time synchronization among all nodes. In addition, each node generates (and appends) a MAC using a yet-unpublished shared secret key in the defined TESLA time unit during *RREQ* broadcast. This increases overhead, since it is done regardless of whether or not the node is actually part of the eventual route. Also, in addition to accumulating the source route, the size of *RREQ* grows due to the accumulation of MACs. Furthermore, Ariadne does not offer non-repudiation which is attained in our schemes based on signatures.

Papadimitratos and Haas [16] proposed Secure Routing Protocol (SRP) that can be adapted to existing routing protocols such as DSR or ZRP in order to secure the route discovery mechanism. SRP, unlike Ariadne [11], requires only the two communicating end-nodes to form a security association. Neither the source nor the destination authenticates each intermediate's node presence in the route. This makes SRP vulnerable to attacks that involve adding (or deleting) honest nodes to (or from) the route.

To alleviate DoS attacks in MANETs, Hu, et al. proposed two protection mechanisms [9, 10] against wormhole and rushing attacks, respectively. These two types of attacks aim to disrupt data transmission after the adversary manipulates route discovery procedure to get itself included in the route. However, it is quite hard to detect nodes behaving honestly during route discovery but becoming adversarial later, during the transmission of data packets. Moreover, it is also hard to distinguish among adversarial actions and link failures. The work in [9] tries to detect possible adversarial behavior via packet leashes which, in turn, rely on loosely synchronized time across all MANET nodes. However, as with Ariadne, time synchronization is not a realistic assumption for all types of MANETs. In a so-called blackhole attack (based on a wormhole) the adversary creates a routing blackhole, with which it attracts and then drops data packets. Watchdog and Pathrater [13] proposed by Marti et al. can also mitigate blackhole attacks since each node is required to monitor whether its neighbors are forwarding their packets.

References

[1] Miracl project:http://indigo.ie/ mscott.

[2] Openssl project:http://www.openssl.org.

[3] M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. *Lecture Notes in Computer Science*, 1109, 1996.

[4] A. Boldyreva. Efficient threshold signature, multisignature and blind signature schemes based on the gap-diffie-hellman-group signature scheme.

[5] D. Boneh, C. Gentry, H. Shacham, and B. Lynn. Aggregate and verifiably encrypted signatures from bilinear maps, 2003.

[6] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. *Lecture Notes in Computer Science*, 2248, 2001.

[7] C. Castelluccia, S. Jarecki, J. Kim, and . Gene Tsudik. Verifiable and secure acknowledgement aggregation. In *Communication Networks Security*, 2004.

[8] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Trans. Inform. Theory*, pages IT–22:644?654, 1976.

[9] Y.-C. Hu, A. Perrig, , and D. B. Johnson. Packet leashes: A defense against wormhole attacks in wireless ad hoc networks. Tech report, Dep. Of Computer Science, Rice University, 2001.

[10] Y.-C. Hu, A. Perrig, , and D. B. Johnson. Rushing attacks and defense in wireless ad hoc network routing protocols. Tech Report TR01-384, Dep. Of Computer Science, Rice University, 2002.

[11] Y.-C. Hu, A. Perrig, and D. B. Johnson. Ariadne : A secure on-demand routing protocol for ad hoc networks. In *MOBICOM*, 2002.

[12] D. B. Johnson and D. A. Maltz. Dynamic source routing in ad hoc wireless networks. In *Mobile Computing*, 1996.

[13] S. Marti, T. Giuli, K. Lai, and M. Baker. Mitigating routing misbehaviour in mobile ad hoc networks. In *IEEE/ACM International Conference on Mobile Computing and Networking (MobiCom 2000)*, 2000.

[14] S. Micali, K. Ohta, , and L. Reyzin. Accountable-subgroup multisignatures. In *ACM Conference on Computer and Communications Security*, 2001.

[15] P. Montgomery. Modular multiplication without trial division. In *Mathematics of Computation*, number Vol. 44:519?521, 1985.

[16] P. Papadimitratos and Z. Haas. Secure routing for mobile ad hoc networks. In *SCS Communication Networks and Distributed Systems Modelling Simulation Conference CNDS*, 2002.

[17] A. Perrig, R. Canetti, J. D. Tygar, and D. X. Song. Efficient authentication and signing of multicast streams over lossy channels. In *IEEE Symposium on Security and Privacy*, pages 56–73, 2000.

[18] D. Pointcheval and J. Stern. Security proofs for signature schemes. *Lecture Notes in Computer Science*, 1070:387+, 1996.

[19] H. Shacham. Sequential aggregate signatures from trapdoor homomorphic permutations, 2003.

A Security Analysis

The proposed schemes are secure against the adversary who adds honest nodes to the route and delete them from the route, except in case of non-adjacent compromised nodes on the route, where the adversary can delete feedback loops as we discussed in section 3. A secure scheme implies that no honest node has been added or deleted by the adversary, i.e., each honest node really appeared in the route but was not forged.

A.1 Security of DH-based scheme

Theorem 1. *Under the Computational Diffie-Hellman (CDH) assumption, the multi-MAC scheme based on DH key agreement described in section 4.5 is secure against existential forgery.*

Proof (sketch). There is an adaptive chosen message attack adversary A which queries MACs to a simulator S. The A can compute a new MAC, $g^{g^{hx_1x_2}}$ from a polynomial number of n MACs queried. Suppose A' using A which queries additionally $g^{g^{h_1x_1x_2}}$, ..., $g^{g^{h_lx_1x_2}}$, where h_i is a l-bit string and only ith bit is 1. For example, h_1=(000...01). Let h the hash of a message on which A forges a MAC. Since any $h = h_{i_1} + h_{i_2} + ... + h_{i_k}$ for some k,

$$g^{g^{hx_1x_2}} = g^{(g^{h_{i_1}x_1x_2})(g^{h_{i_2}x_1x_2})...(g^{h_{i_k}x_1x_2})}$$

Therefore, we can solve a CDH problem of $(g, g^{g^{h_{i_1}x_1x_2}}, g^{g^{h_{i_2}x_1x_2}}, ..., g^{g^{h_{i_k}x_1x_2}})$ by using the adversary A'.

A.2 Security of ASM-based scheme

We consider an adversary F with the following capabilities: (1) F can fully control all packets exchanged in the network, i.e., it can read, modify or discard any packet as well as inject its own packets, (2) F can mount an adaptive chosen-message attack against any honest node I by requesting that I sign a message of F's choosing.

Definition (adapted from [14]): A multi-signature scheme is *secure* if every polynomial-time adversary F has a negligible chance of outputting a triple (σ, M, S) such that (1) σ is a valid multi-signature on message M by the subgroup S of players, and (2) there exists an honest player $P \in S$ who has never been asked by A to sign M in the context of S.

Theorem 2. *The multi-signature scheme (based on the Schnorr signature scheme) described in 4.5 is secure in the Random Oracle Model, under the Discrete Logarithm Assumption.*

Proof (sketch). In case of adaptive chosen message attack, the adversary F uses the signer as an oracle. We construct a simulator A which does not know the signer's secret key but can simulate the signer. Together with A, F can break the underlying signature scheme and solve the discrete logarithm.

Suppose that A is given a DLP instance: $< p, g, y >$ and is asked to compute x such that $g^x = y \mod p$. We assume that all nodes – except the source and the sole honest node I – are corrupt and F can forge many multi-signatures on some message M without asking I to sign. A assigns $y_i = y$ as the public key of I, while it selects all other nodes' private keys at random.

For F to forge a multi-signature, A needs to simulate I's response. This is done in a manner very similar to the security proof of the Schnorr signature scheme in [18]. The only difference is that A needs to rewind F to answer a signature query to I on M as in the security proof of the multi-signature scheme [14, 7], based on the Schnorr signature scheme.

Assume that F asks q^h hash queries and the answers to all hash queries are picked in advance and at random: $e_1, ..., e_{q_h}$ To answer a signature query to I on M, A selects e at random from hash values which are picked in advance at random and $\sigma \in [0, q-1]$ at random, computes $t_i = y_i^{-e} g^{\sigma}$ mode p and sends t_i to F as the response to the first round of the query. Upon receiving $\tilde{t}$ from F, A verifies if $e = (\tilde{t}, M)$. If so, A outputs σ. Otherwise, rewinds F and repeats picking e. Note that A will eventually succeed in expected number of at most q^h times, which degrades the simulation algorithm with a polynomial factor.

A then runs F, answering its queries as described above. Suppose that F outputs a forgery $(\tilde{t}, \tilde{\sigma})$ of the signature on a message M with a subgroup SG and the forgery was based on the j-th hash query. Following the "forking lemma" technique in [18], A resets F with the same random tape as the first time and reruns F from the point of the j-th hash query, answering with a new random number e'. As in the above description, A might need to rewind at most q_h times again.

In case A does not need to rewind (since the random tape and answers to all hash queris – up to j-th – are the same) we are assured that j-th hash query to the hash function will also be the same but with a different answer with a different subgroup SG': $e' = (\tilde{t}, M)$. Let $\tilde{y} = \prod_{k \in SG-\{i\}} y_k$ and $\tilde{y'} = \prod_{k \in SG'-\{i\}} y_k$ and let $\tilde{x} = \sum_{k \in SG-\{i\}} x_k$ and $\tilde{x'} = \sum_{k \in SG'-\{i\}} x_k$

So, if F again forges a signature $(\tilde{t}, \tilde{\sigma'})$, and forgery is based on j-th hash query, then: $g^{\tilde{\sigma}} = \tilde{t}(\tilde{y}y)^e \pmod p)$, $g^{\tilde{\sigma'}} = \tilde{t}(\tilde{y'}y)^{e'} \pmod p$. Thus, $\tilde{\sigma} - \tilde{\sigma'} = (\tilde{x} + x)e - (\tilde{x'} + x)e' \pmod q$. So, $x = (\tilde{\sigma} - \tilde{\sigma'} - \tilde{x}e + \tilde{x'}e')/(e - e') \pmod q$. Thus $x = DL_g(y)$ is solved.