Integrating a Trust Framework with a Distributed Certificate Validation Scheme for MANETs

Giannis F. Marias, Konstantinos Papapanagiotou, Vassileios Tsetsos, Odysseas Sekkas and Panagiotis Georgiadis

Dept. of Informatics and Telecommunications, University of Athens, Panepistimiopolis, Ilissia, Greece, GR15784

marias@mm.di.uoa.gr, {conpap, b.tsetsos, o.sekkas, georgiad}@di.uoa.gr

Abstract. Many trust establishment solutions in Mobile Ad Hoc Networks (MANETs) rely on public key certificates. Therefore, they should be accompanied by an efficient mechanism for certificate revocation and validation. Ad hoc Distributed OCSP for Trust (ADOPT) is a lightweight, distributed, on-demand scheme based on cached OCSP responses, which provides certificate status information to the nodes of a MANET. In this paper we discuss the ADOPT scheme and issues on its deployment over MANETs. We present some possible threats to ADOPT and suggest the use of a trust assessment and establishment framework, named Ad hoc Trust Framework (ATF), to support ADOPT's robustness and efficiency. ADOPT is deployed as a trust-aware application that provides feedback to ATF, which calculates the trustworthiness of the peer nodes' functions and helps ADOPT to improve its performance by rapidly locating valid certificate status information. Moreover, we introduce the TrustSpan algorithm to reduce the overhead that ATF produces, and the TrustPath algorithm to identify and use trusted routes for propagating sensitive information, such as third parties' accusations. Simulation results show that ATF adds limited overhead compared to its efficiency in detecting and isolating malicious and selfish nodes. ADOPT's reliability is increased, since it can rapidly locate a legitimate response by using information provided by ATF.

Keywords: MANET, certificate revocation, caching, trust, reputation, OCSP

1. Introduction

Mobile Ad hoc Networks (MANETs) are dynamically configured, multi-hop wireless networks with varying topology. Mobile nodes in such networks are continuously associated or disassociated with each other, according to their topological arrangements. Thus, the network topology varies with time due to the ataxic locomotion of the participating nodes. In such a dynamic environment, cooperation between nodes is essential for the network's well-being. In order to enforce cooperation, adjacent nodes should build up trust with time. Such trust establishment procedure can improve security, connectivity and quality of service in the network, and, thus, enhance its performance. Applications in a MANET can take advantage of such trust relationships, by exploiting them to improve their performance and efficiency.

MANETs are dynamic and open: nodes join or leave the network at any time, are constantly moving in any direction, while the electromagnetic spectrum, being the transfer medium, is considered publicly available. In such a context selfish and malicious nodes are likely to appear. Selfish nodes are characterized by their reluctance to spending resources to maximize social welfare (e.g. by forwarding packets that are not destined for them). Conversely, they demand from other nodes to exploit resources for their profit. Malicious nodes attack directly the network's robustness and the nodes' availability through common techniques such as Denial of Service (DoS), flooding and sleep deprivation torture [1]. Hence, security schemes that are based on symmetric or public keys are considered essential for the establishment of authenticity, confidentiality and integrity services in MANETs, being focused on the avoidance of the malicious nodes' actions. On the other hand, if the primary goal is the maximization of the network's availability, robustness and overall throughput, then cooperation enforcement schemes fit better. They mainly encourage the collaboration between the nodes of a MANET, trying to overcome egoistic behaviours that jeopardize the network's operation.

In order to prevent malicious attacks, several proposals based on the usage of public key cryptography and digital certificates signed by Certificate Authorities (CA) have been made, such as the threshold cryptography schemes proposed in [2], [3] and [4]. For public key cer-

tificates a scheme for retrieving CSI (Certificate Status Information) is essential. In [10] we have proposed a distributed scheme based on OCSP (Online Certificate Status Protocol) called ADOPT, that can provide on demand up-to-date CSI to any node in a MANET. ADOPT is based on the concept of distributing and caching pre-issued and pre-signed OCSP responses in selected nodes of the network. Its goal is to provide an efficient, lightweight and always-available scheme for determining the status of certificates within a MANET.

On the other hand, when the availability of nodes and network, and the overall throughput are required, then the cooperation enforcement techniques might fit better. These models face mainly the question of encouragement of collaboration between the nodes of a MANET so that the right implementation of routing and forwarding tasks is achieved. In [24] we have proposed the Ad hoc Trust Framework (ATF), a generic, distributed, framework for self-evolving trust establishment. ATF incorporates self-evidences, recommendations, subjective judgment and historical evidences to continuously evaluate the trust level of peers. To capture such semantics, ATF is armed with a novel trust computation model. The model consolidates user's natural behaviour, through a Trust Policy. ATF does not use any symmetric or public cryptography for trust building, or message authentications schemes. Thus, it avoids complex computations and the expenditure of resources (power, CPU and memory). In that sense, ATF is usable in different types of distributed, or peer systems (ad hoc networks, pervasive computing devices, autonomic systems), although in this work it is primarily exploited in the context of ad hoc networking.

In this paper we propose the integration of these two frameworks, namely ADOPT and ATF, in order to demonstrate how ATF can be used to increase ADOPT's efficiency. ATF, being a general purpose, self-evolving trust scheme can support trust aware applications. Such an application could take advantage of trust values assigned by ATF and use them to improve its efficiency. In ADOPT's case, as we will see in the following sections, ATF can be used to prevent attacks that could flood the MANET with useless messages. Threats to ADOPT, not only affect the usage of this service, but may also cause exhaustion of node's resources. Hence, ADOPT, could rely on ATF for improving its robustness. ATF provides trust information for specific nodes, so that selfish and malicious nodes can be detected and isolated as soon as possible. Additionally, ATF can disseminate trust metrics concerning the behaviour of nodes when executing ADOPT, providing more accurate trust information for ADOPT, but also assessing the trustworthiness of a node in a MANET in a more complete way.

The structure of this paper is as follows: First we present and explain the functionality of ADOPT. Details of proposed caching alternatives are also given as well as some characteristics of the ADOPT protocol and messages. In Section 3 we describe ATF, its architecture, how it manages trust and reputations using a specified trust policy. Furthermore, the details of the ***TrustSpan*** algorithm are given. Subsequently, in section 4 we analyse the threats and attacks that ADOPT may have to face. In addition, we describe the reasons that led us into integrating ADOPT and ATF and the specifics of this integration. Moreover, the ***TrustPath*** algorithm and ***Cerberus*** function are analyzed. In Section 5 we discuss details of this integration, providing and assessing several simulation results. Finally, we provide some examples of related work and conclude with our remarks and suggestions for future research.

2. ADOPT

ADOPT [10] is an on-demand, distributed OCSP scheme, based on cached OCSP responses, and designed in such a way, so that it can be successfully and efficiently deployed in MANETs. Its purpose is to create a fast, light, distributed and always-available certificate revocation protocol for MANETs.

2.1 ADOPT Architecture

OCSP is a simple protocol involving requests and responses that provide the current status of one or more certificates. A client can send a request to a server (usually called OCSP Responder) asking for information on the status of one or more certificates. This request can be digitally signed and contains a reference to the queried certificate(s) (certID). The server responds with a signed message that contains the status of the referenced certificate(s). The re-

sponse message also contains time and date information. OCSP responses are always digitally signed either by the CA, a trusted or an authorized responder.

We distinguish three different kinds of nodes in ADOPT: *ServerNodes*, *CachingNodes* and *ClientNodes*. *ClientNodes* request the status of a certificate by broadcasting a message similar to an OCSP request. *CachingNodes* cache pre-issued and pre-signed OCSP responses and act as OCSP responders by providing such responses when needed. *ServerNodes* are nodes that announce the revocation status of the certificates, such as OCSP responders. They issue and sign certificate status responses which are then stored in *CachingNodes*.

A *ClientNode* wishing to determine the status of a certificate forms an OCSP-like request message. In traditional OCSP, this message should then be sent to an OCSP responder, which would be identified by the *authorityInfoAccess* extension [19] of the X.509 certificate. However, MANETs are highly dynamic in nature, as nodes may enter or leave the network anytime and may be moving constantly. Therefore, a DSR-like [20] mechanism has been proposed [10], more appropriate for such environments. Thus, the request message is broadcasted by the *ClientNode*. Intermediate nodes that receive the message re-broadcast it if they do not act as *CachingNodes*. On the other hand, *CachingNodes* examine their cache for a pre-issued response corresponding to the requested certificate. If such a response is found, it is forwarded back to the *ClientNode*; else the request message is re-broadcasted. Similarly, if the message reaches a *ServerNode*, a corresponding response is issued.

Clearly, a request message may be circulating in the MANET without ever getting a corresponding response. To avoid this problem, ADOPT proposes a solution [10] similar to the one proposed for resolving routing loops in DSR [20]. In detail, the *ClientNode* determines the maximum number of hops that the request message is allowed to travel through. This *TTL (Time-To-Live)* parameter is included in the request message. Every intermediate node that receives the message decreases *TTL* by one, until the maximum number of hops is reached and the message is dropped.

2.2 ADOPT protocol

The main advantage of ADOPT lies on the fact that the nodes of a MANET can receive up-to-date CSI anytime, using a distributed scheme that ensures the availability of this service. In addition, this information is delivered with a minimum cost in terms of both network and node resources. As a matter of fact, OCSP messages are rather small in size [21] and *CachingNodes* do not need to re-sign cached information status. The authenticity and integrity of the responses can be verified by the OCSP responder's signature on the response. However, the freshness of CSI depends on the freshness of the cached responses and thus, on the mechanism used for cache updating.

OCSP requests reference the queried certificate using a hash of the issuer's name and key as well as the serial number of the certificate. These fields uniquely identify a certificate [11]. OCSP responses include three time parameters, critical to OCSP's operation. The first one, indicated by the field *producedAt*, denotes the time when the OCSP response was issued. Two additional parameters specify the validity interval of the OCSP response. In detail, *thisUpdate* indicates when revocation information regarding the queried certificate was last obtained, while *nextUpdate* is the time when the responder is expected to have new information concerning this certificate. The most important fields contained in an ADOPT request (*AReq*) and response message (*ARes*) can be seen in Figure 1:

ARes freshness can be of a great importance to *ClientNode*s as some critical user applications may require up-to-date responses. ADOPT [10] introduces a parameter (*updateTime*) in the request message that allows a *ClientNode* to specify how fresh the expected response should be. In such terms, if a *CachingNode* does not have a fresh enough response, it rebroadcasts or drops the request, depending on the *TTL* parameter.

Ideally, a *CachingNode* should deliver the most recently issued cached response. Nevertheless, it is possible that its cache is not updated. An efficient mechanism for cache updating should be in place to ensure that *CachingNodes* get the most updated responses. ADOPT suggests that *CachingNodes* get updated directly from an OCSP responder (*ServerNode*), either periodically or on demand. Even when communication with *ServerNodes* occurs using out-of-band means (e.g., through a GSM or GPRS bridge), it is possible that these OCSP responders

may not be always available. Thus, ADOPT also suggests that *CachingNodes* can eavesdrop on messages that they forward in order to detect *ARes* designated for other nodes but also useful to them, for updating their cache.

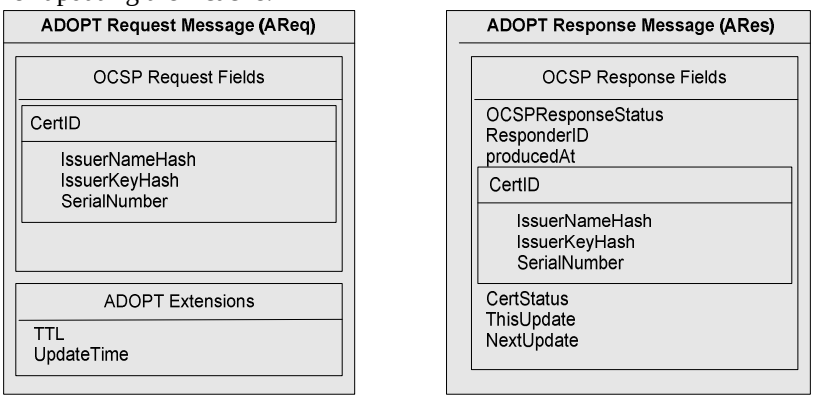

Figure 1. ADOPT messages

Efficient cache placement policies ensure that there is no unwanted flooding of OCSP requests in the MANET. Cached *ARes* may be placed in strategic elements within a MANET, for example in high mobility nodes [22]. Each node in the network should be able to reach a *CachingNode* with a cached response to his request in a few hops, so that *AReq* do not have to travel far in the network. Moreover, each node chooses for itself a cache update and deletion policy. Its decision depends on its position within the MANET its resources in terms of processing power, memory capacity and power autonomy. Thus, a node may choose between a greedy, selective and no-caching policy [22]. In ADOPT, the candidate states of a *CachingNode* are:

- Greedy Caching State. The node caches every *ARes* that passes through it.
- Selective Caching State. The node caches a response after ***m*** appearances, with ***m*** being the popularity index of that response.

Overall, the caching strategies ensure that network resources are wisely spent in legitimate protocol runs, initiated by good-willing entities. Some corresponding time thresholds have been proposed in [22]. ADOPT's *TTL* and *Waiting Window (WW)* parameters ensure that request propagation stops once a response has been located. A *ClientNode* has to set the *TTL* parameter, which specifies the maximum number of nodes that the request can pass through. If a response is not found within the specified number of nodes, the request will be dropped. The *ClientNode* will be able to resend the same request with a different TTL parameter, depending on the *WW*. The *Waiting Window* parameter indicates the time a *ClientNode* has to wait until receiving an *ARes* and its calculation is based on a node's observations of network delays. Evidently, a legitimate request message will only reach a specific number of hosts, without forming any loops, thus consuming only the necessary network resources.

3. ATF

ATF uses a layered architecture and consists of the following components: *Trust Sensors (TS), Trust Builder (TB)* and *Reputation Manager (RM)*. The proposed framework is fully distributed. Every node hosts these components and provides a number of routine functions (services), such as packet forwarding, routing, etc. Moreover, every node implements a special function, called *Recommendation Function (RF)*. This is a simple service that provides recommendations to third parties, upon request.

3.1 ATF Architecture Components

ATF follows [23] for the definition of the reputation of a node's function: Reputation is expressed through the triple *{NodeId, Function, Trust Value}*. Thus, the reputation of a function f of node n is defined as $R(n,f)= \{n, f, TV_{n,f}\}$, where $TV_{n.f}$ is the *Trust Value* (*TV*) for the function ***f*** of node ***n***.

In the context of ATF, a *detector* is a node that directly monitors the behaviour of another node's functions, called *target*. In such a case the *detector* captures *Direct Evidence (DE)* about the trustworthiness of a particular function of the *target*. A *requestor* is a node that asks for recommendations. A *recommender* issues recommendation responses (upon request).

In general, a trust building mechanism could be laid out based on two diverse architectural directions. The first relies on an on-demand, and the second on an event-driven reputation mechanism. The difference between these two approaches lies in the way that nodes are being informed for changes in *Trust Values (TVs)*. The ATF architecture is capable of supporting on-demand recommendations. The ATF architecture consists of the following modules (see Figure 2):

Trust Sensors (TS). The majority of the proposed reputation systems agree that the most significant factor for trust building is the evidence. In ATF, for every function offered by an adjacent node there is a *Trust Sensor (TS)* that monitors its execution/operation. A *TS* is an abstraction of common physical sensors: translates a (physical) phenomenon in a machine interpretable form. In our case this phenomenon is the trustworthiness of a node. A *TS* monitors the behaviour through real-time measurements or statistical analysis of logs, and compares it to a predefined reference attitude (i.e., expected functionality). In that sense, the ATF scheme uses *TSs* to assist a node in defining the credibility of other collaborating peers. A *TS* maps the captured evidence (i.e., observation) to a numerical value and forwards this value to the *Trust Builder* for further trust computation.

Trust Builder (TB). This component computes the *TV* of other nodes' functions. A *TV* value is distinct for each discrete function per node. A node, for example, may be trustworthy to perform packet forwarding, but unreliable to contribute on routing. *TV* computation depends on several factors, such as direct evidence, recommendations, historical data and subjective criteria. Each node follows its own Trust Policy, which specifies the user's subjectivity (e.g., distrustful, deceivable) and the weights.

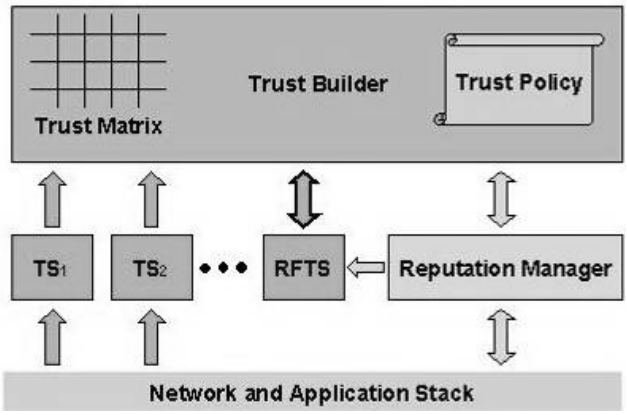

Figure 2. ATF Architecture

Reputation Manager (RM). The *RM's* main role is to provide recommendations from third parties to the *TB* in order for the latter to compute the *TVs*. *TB* requests recommendations for a *target* when it has inadequate information for it. Next, *RM* selects the *recommenders* in order to obtain requested values. These should be as trusted as possible and close enough so as to limit communication overheads. For that purpose, the *RM* takes into account the *TVs* of the *recommenders*' *Recommendation Function*.

Recommendation Function Trust Sensor (RFTS). This special trust sensor evaluates the trustworthiness of a node regarding its recommendation function. *RFTS*, like any other *TS*, categorizes a direct observation as Success or Failure. The *RM* of a *detector* asks from *recommenders* the recommendations that correspond to a specific function of a *target*. A recommendation is returned only when the *recommender* has adequate *DE* about the *target*, so as to reduce rumour spreading.

3.2 Trust computation in ATF

ATF incorporates several user-defined weights and time-dependent parameters, defined separately for each entity in its *Trust Policy* (*TP*). Time-dependence is important, since it allows the modeling of temporal trust strategies, which can be followed by the participating nodes. In the context of ATF, time is discrete and is measured through counting successful interactions.

The majority of the trust computation approaches acknowledge that for the evaluation of *TV* two main components should be taken into consideration: the *DE* and the Recommendations (*RECs*) from third parties. The *DE* is calculated from the *TS*s' feedbacks and is useful for evaluation of adjacent nodes' functionality. *REC*s are communicated between the entities participating in the trust network, according to a reputation dissemination protocol, implemented in *RM*. Many socio-cognitive approaches for trust, e.g. [30], dictate that trust computation should also include a subjective component. Thus, a *Subjective-factor component* (*SUB*) is introduced in the trust computation model. This time-dependent component enables time-variant trusting behaviour of peers. *SUB* is defined by the *TP* of each individual node, and can model typical trust characters, such as unwary, suspicious, disbelieving, etc. This component provides flexibility in the trust strategy of a user, without imposing significant complexity in the overall trust computation. History is an additional concept that has drawn attention in the trust community. Several researchers use history as an implicit component in the trust computation, as in [14], [15] and [26]. In ATF, a significant weight is assigned in the current observation. This enables a peer entity to rapidly identify when a *target* starts to misbehave. Additionally, the history of the observations that will be considered for the evaluation of a trust value of a *target*, when a new observation is produced, depends on the subjective behaviour of the requestor. Thus, a distrustful peer entity might need past observations, whilst a deceivable peer entity might need only the newer ones. Thus, when a new observation (i.e., *TS(n,f)*) occurs, a distrustful entity takes into account the history *h1* of the direct evidences, as seen in Figure 3, whilst a deceivable entity takes into account the history *h2*.

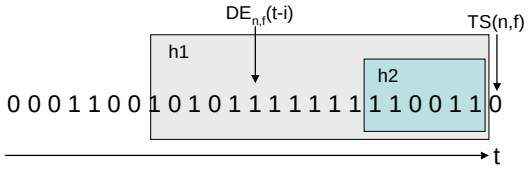

Figure 3. Different observation windows for the assessment of the DE trust value

Thus, for each new observation or recommendation (i.e., NewValue) the following equation is used to relate historical data with current observations (**Eq. 1**):

$$NewValue_{n,f}(t) = w * CurrentValue_{n,f} + (1-w)\left[\sum_{i=1}^{H} NewValue_{n,f}(t-i)\right]/H \qquad \textbf{(Eq. 1)}$$

For simplicity we write $NewValue_{n,f}(t)=WA_H(n,f,t)$ to denote the weighted average over the last H data and the new value recorded at time t. According to the analysis in [24], moderate values of w (e.g., 0.2) ensure that the calculated values belong to a range that is not deviated from the real behaviour. Thus, moderate values of *w* in **Eq. 1** ensure that the direct evidence will capture the actual behaviour of the *target*, without significant deviations due to sporadic misbehaviours or rational errors. Additionally, the historical average component of **Eq. 1** introduces different sharpness on the estimation. When only the recent observations are required (e.g., H=3, corresponding to an impressionable entity) then the *DE* approaches faster the minimum or maximum values, illustrating sudden fluctuations. On the other hand, when the *requestor* takes into account long history (e.g., H=8, for a disbeliever entity) then the *DE* approaches less rapidly the threshold values, illustrating smooth fluctuations.

Each node maintains an *NxF* Trust Matrix (*TM*), representing the *TV* that the entity computes per monitored function of a *target*, where N is the number of nodes in the network, and F corresponds to the overall number of supported functions. Each element $TM_{n,f}$ ($1 \le n \le N$ and

$1 \le f \le F$) refers to a specific function *f* of a particular node *n*, and it varies with time. The formulae for *TM* and *TV* are:

$$TV' \equiv TV'(n, f, t) = (a * DE_{n,f} + b * REC_{n,f}) * SUB_{n,f}(t) \quad \textbf{(Eq. 2)}$$

$$TV(n, f, t) = TV' \cdot u(1 - TV') + u(TV' - 1),$$

$$TM \equiv (TM_{n,f}),\ and\ TM_{n,f} = TV(n, f, t) \in [0,1]$$

where, $DE_{n,f} \in [0,1]$, $REC_{n,f} \in [0,1]$, and $SUB_{n,f}(t) \in [0,2]$, and, thus, $TV' \in [0,2]$. The parameters *a* and *b* (see **Eq. 2**) are step increasing and decreasing functions. In order to map the *TV* values within the [0,1] interval we use a unit step function *u(t)* (see **Eq. 3**). The range of *TV(n,f,t)* is [0,1], where 0 means that the *detector* distrusts a *target n* for a specific function *f*, and 1 means that it fully trusts *n* for *f*.

$$u(t) = \begin{cases} 0, & t < 0 \\ 1/2, & t = 0 \\ 1, & t > 0 \end{cases} \quad \textbf{(Eq. 3)}$$

DE_{nf} is the *DE* for a *target n* and its function *f*, as observed by the corresponding *TS* of the *detector*. A *NxF* matrix *DE*, stored in every node, is defined as $DE \equiv (DE_{n,f}) \in [0,1]$, where the matrix elements $\mathbf{DE_{n,f}}$ are computed according to **Eq. 1**, as follows:

$$DE_{n,f}(t) = WA_H(n, f, t)\ ,\ TS(n, f) \in \{0,1\} \text{ and } TS(n, f) \in \{0,1\}$$

The lower value (i.e. 0) denotes *Failure*, while the higher (i.e. 1) denotes *Success*. The coefficients w_1 and w_2 adjust the weights assigned to recent and historical *DE* values, respectively. $REC_{n,f}$ stands for the aggregated recommendations we have for the function *f* on node n from third parties. The *SUB* component of the *TV* computation formula incorporates the node's subjectivity. This subjectivity is a key differentiator between the various nodes and stems from the socio-cognitive approaches to trust modelling [27]. *SUB* is an *NxF* matrix with elements in the $\{f : T \to [0,2]\}$ domain. Thus, its elements are time-functions. The range [0,2] allows the *detector* to distrust (i.e. value 0) the *target*, trust it (i.e. value 1), be enthusiastic about the *target* (i.e. value 2) or develop any other intermediate form of subjective trust strategy. We have chosen the value 2 as an upper bound to allow enthusiastic entity but not to such a degree that would endanger the network's rationality. Thus, we might consider nodes with diverse trust strategies, but we want to restrict the deviation of this diversity. An example SUB time-function could be defined as:

$$SUB_{n,f}(t) = u(t - 2) \quad \textbf{(Eq. 4)}$$

This function indicates that no matter what *DEs* or *RECs* a *requestor* has for a function of a *target (n,f)*, it will not trust the latter until two successful (positive) direct interactions have been observed. In case the aforementioned defined *SUB* component is used indiscriminately for all *targets* and provided functions, it will be identical for all the elements of the *detector's SUB* matrix. The set of the *SUB* functions is defined in the *TP* and it can be adjusted depending on the *target* and the monitored function. However, in practice, it is highly unlikely that a node will have *NxF* different *SUB* functions. Instead, a *detector* will usually use identical *SUB* function for every *target* or every function.

3.3 Trust Policy

For each node, a *TP* defines the functionality of its *RM* and *TB*. A *TP* captures the conceptual subjective behaviour of the entity (e.g., end-users). The parameters of the *TP* are summarized in Table 1. The parameter *HFI* is used by the proposed *RM* module and its application is described in the following section.

The parameters *a* and *b* (see **Eq. 2**) are step increasing and decreasing functions on *MI*. This policy was chosen because when a *detector* realizes the existence of a newcomer only the recommendations of the trusted *recommenders* should be used (high values of parameter b). Thus, in the initial phase the *RECs* are essential. With time, the *DEs* become more important, and this happens only when the *MI* value is exceeded.

3.4 Trusted Recommenders

As illustrated in **Eq. 2** a *requestor* is based on recommendations to compute the *TV* of other nodes. However, trusted *recommenders* (i.e., nodes illustrating a high *TV* value for the *RF*) should be conducted to provide these recommendations. This will minimize the effects of rumour spreading and avoid potential DoS attacks [13]. Moreover, the selected *recommenders* should be as close as possible to the *requestor* so as to have limited communications overhead. The work in [12] ranks routes according to security metrics (e.g., reputation of nodes in the path), and avoids paths containing malicious nodes. Path ranking is also proposed in [29] to mitigate the effects of routing misbehaviour. In [24] we have introduced a mechanism, called *TrustSpan* that is used by the *RM* of a *requestor* to consult only trusted nodes whenever recommendations for the evaluation of newcomer nodes' trustworthiness are required.

Table 1. The parameters of the Trust Policy

Parameter	Semantics
MI	The Minimum Interactions required for being confident about the *TV* of a *target*
a	The impact (weight) of the *DE* on the *TV* ($0 \le a \le 1,\ a+b=1$), in **Eq. 2**
b	The impact (weight) of the *REC* on the *TV* ($0 \le b \le 1, a+b=1$), in **Eq. 2**
w	The weight of history **(Eq. 1)**
TTH(RF)	The minimum allowable TV_{RF} assessed to a node in order to be a *recommender*
HFI	Honourable Friend Index is the minimum number of trusted *recommenders*, required to be consulted by a *requestor* to reliably evaluate the trustworthiness of an unclassified *target*

We consider as newcomer the node for which the *requestor* has no or inadequate experience (i.e., zero or few *DEs*). When a newcomer becomes adjacent to a *requestor*, the latter does not know a priori whether it is trustworthy and, therefore, it executes *TrustSpan* in order to identify the nearest trusted *recommenders*. A *requestor* characterizes a node as a trusted *recommender* if the *TV* for its *RF* is higher than *TTH(RF)*, defined in *Trust Policy*. We remind here that the *RF* is monitored and evaluated, just like any other function, via the *RFTS* sensor. The *TrustSpan* procedure is presented in Listing 1. The *Distance Matrix (DM)* stores the hop-distance of known nodes from the *requestor*.

Listing 1. TrustSpan algorithm

```
Inputs:
    int HFI ;              -- Honorable Friend Index
    int DM[];              -- The distance of the recommender from the requestor, 1 ..., M
    int TM[][];            -- Trust Matrix, MxN
    int TTH(RF);           -- The minimum TV_RF of the recommender in order to be trusted
Output:
    int trusted_IDs[]      --  IDs of nearest, HFI trusted recommenders

procedure TrustSpan (){
    int TV_RF[], NodeID[];
    int NumberOfTrustedNodesForRF;

    for (j=0; j<numberOfNodes; j++)
        if ( TM[j][RF] ≥ TTH(RF)) {    -- Trust Value for the RF for the recommender j
            NodeID[j] = j;
            NumberOfTrustedNodesForRF ++;
        }
    if (NumberOfTrustedNodesForRF ≤ HFI )
        return (NodeID[]);
    order(DM[], NodeID[]);                  -- orders NodeID[] according to DM[]
    for (j=0; j<HFI; j++)
        trusted_IDs[j] = NodeID[j];
    return trusted_Ids[];
}
```

TrustSpan tries to minimize both the delay in the propagation of the requested recommendations and the communication overhead. It is invoked only when a node has inadequate number of direct interactions with a newcomer. This number is defined in *TP* through the pa-

rameter *MI*. *MI* also affects which nodes reply to recommendation requests. In particular, the *recommenders* selected by *TrustSpan* that have less than *MI* direct interactions with a *target*, will not give recommendations about it. When the *TrustSpan* returns the IDs of the nearest trusted *recommenders*, the *RM* asks for recommendations (through unicasting) from the corresponding *RMs* of the trusted *recommenders*. When the required recommendations arrive or a respective timeout expires, the *requestor*'s *RM* forwards the incoming information to the *TB* where the actual trust computation takes place.

4. ADOPT over ATF: Integration Approach

In this section we show how ADOPT can take advantage of ATF in order to ensure the availability and reliability of its service. ATF can also include TVs that are calculated according to a node's behaviour when executing ADOPT, regardless if it is a *ClientNode*, a *CachingNode* or simply an intermediate node forwarding ADOPT messages. Furthermore, we will examine how ATF and ADOPT can be integrated using a trust plane and also suggest some functions that will optimize the performance of the integration approach.

4.1 Attacks and threats against ADOPT

The presence of malicious or selfish nodes is substantial, and should be considered when providing CSI. In this section we analyze how selfishness and malicious behavior can be dealt with using ATF as a trust component. However, we do not take into account selfish or malicious behaviours against the robustness of other protocol layers, such as Route Request flooding, or routing table fabrication, materialized in the network layer of a MANET, since we assume that these attacks will be prevented on the corresponding layer of the stack. We only discuss selfishness and malicious behaviour on the ADOPT layer. We first consider a malicious node that either initiates flooding attacks to cause serious disruption to ADOPT, or fabricates the valid status information of a certificate.

***AReq* flooding**. A malicious node starts flooding the network with an invalid *AReq*, i.e., asking for the status of a certificate that does not exist. This can be easily achieved by including a random certificate serial number in an *AReq*. Intermediate nodes receiving this request will look for a corresponding response in their caches and, after finding none, forward the request. This way, *CachingNodes* consume resources to look for a cached response that does not even exist. Moreover, a more important issue is that the network will be flooded with invalid requests that can never be replied. Assigning a high *TTL* to the invalid requests can easily scale this attack. Such requests will not be withdrawn, as a corresponding response will never be located. As a result, these requests will be circulated in the entire network, reserving the scarce bandwidth and resources.

***ARes* flooding.** This similar type of attack might cause more damage as far as robustness is concerned. In an attempt to find an updated response as quickly as possible, ADOPT broadcasts *AReqs*, which are propagated in the network. Malicious nodes can take advantage of this, by issuing and propagating false *ARes*. Such nodes can be acting as *CachingNodes*. In such a case they keep in their repository a variety of cached *ARes*, issued by various OCSP responders. These *ARes* might be valid, but probably are out-of-date, or they correspond to different serial numbers from the ones which a status was requested for. Each issued *ARes* will traverse the reverse path, until it is discarded as invalid by the requestor *ClientNode*.

***ARes* Fabrication.** Malicious caching nodes can fabricate responses in order to trick *ClientNodes* into acting as if they were valid. In detail, a malicious *CachingNode* can randomly pick a valid cached response from its repository and alter the referenced certificate in order to match the queried one. Alternatively it can keep only aged responses that, for example, make a currently revoked certificate appear as valid. Actually, such responses are valid, but out-of-date. In any case, a *ClientNode* receiving a cached *ARes* will first try to verify the signature on it. If the response has been altered by a malicious node, the signature will be found invalid and the *ClientNode* will drop the response and broadcast a second request message, probably with a higher *TTL*, when the *WW* will expire. In case of an aged response, the signature will be found valid and it is left to the *ClientNode* to decide if it will look for a fresher status or regard it as valid. Otherwise, the digital signature will fail verification. Nevertheless, the *Cli-*

entNode will have verified the signature spending its power in processing public-key algorithms. A node flooded by such responses may soon consume a lot of its resources in vain (i.e., without ever getting a valid response). Several malicious nodes located near a *ClientNode* can flood it with false responses until completely exhausting its power, materializing a sleep deprivation torture attack [1].

A large number of malicious nodes, strategically placed within the MANET may cause serious disruptions to ADOPT and the network as a whole. These nodes could manage to provide invalid responses to most request messages, before legitimate nodes could manage to respond. This kind of attack resembles to DoS attacks. The availability of ADOPT is affected as practically valid responses cannot be delivered. In extreme cases the nodes' availability may be in risk too, since if they continue to look for valid responses, they will soon run out of resources. In addition, ADOPT's reliability and robustness become questionable too, as it appears unable to operate properly.

Apart from being malicious, some nodes may be selfish as well. Such nodes do not wish to spend their resources for the profit of other peers, or even for the social welfare. Additionally, they demand from others to use resources for their own profit. In terms of ADOPT, selfish nodes may decide to always follow a non-caching policy. Alternatively, selfish nodes may choose not to process request messages at all. In the latter case, they relay the *AReq* messages without checking their cache for responses or reducing the *TTL*. Thus, a request message will travel in the MANET more than intended. Finally, selfish nodes may decide not to forward any request or response message. In any one of the aforementioned cases, such nodes affect the performance of ADOPT or may even cause disruption of the service.

4.2 Motivation

The attacks analyzed in the previous section are rather simple but if deployed on a large scale in a MANET, they could result in network congestion and partitioning as well as significant node resource consumption. In order to prevent and deal with these attacks we propose the use of the ATF framework as a trust component. Such a component would evaluate nodes' trustworthiness according to their behaviour so that peers could decide whether they should trust each other. In this section we examine how the use of ATF can enhance ADOPT's efficiency and prevent the aforementioned attacks. We also propose some customizations for ATF in order to match ADOPT's needs.

We consider here a MANET in which ATF is already deployed and *Trust Values* are estimated for certain node functions (e.g., packet forwarding) [24]. Even though a generic trust framework which estimates nodes' trust based on their performance on fundamental functions would be useful for any MANET application, the calculation of application-specific TVs would increase the efficiency of the application and of the network overall. ATF supports any trust-aware function and application, and, thus can also support ADOPT. A node receiving an *AReq* or *ARes* can retrieve the *TVs* of the node that issued these messages from its Trust Matrix. Then, a decision should be made, based on the retrieved *TVs*, whether the originator should be trusted or not. In case of an *AReq*, the message should be processed as normal if the originator is considered trusted and, thus, the intermediate node should seek for a cached response in his cache and act accordingly. Similarly, an *ARes* should be forwarded to the next node if the originator is considered trusted. Conversely, if the node which issued the corresponding message should not be trusted, the request or response should be dropped.

Using this approach *AReq* and *ARes* flooding attacks can be successfully mitigated. A legitimate node is expected to soon collect enough *TVs* in order to characterise a node as malicious. Hence, it will ignore and drop any requests or responses that originate from such nodes. Furthermore, in section 4.4 we introduce optimisations in order to facilitate the detection and isolation of misbehaving nodes. Thus, the *ARes* fabrication attack is also faced, as nodes that continuously fabricate invalid responses will be eventually isolated.

This approach solves the general issue of attacks to ADOPT but may result in many false positive if *TVs* are not based on the node's behaviour while executing ADOPT, but on other, possibly routine functions. In some cases a node may not be behaving rationally in terms of routine tasks, but may legitimately request the status of a certificate. This request should not be dropped by the network nodes, nor should a request indicating an unusually high *TTL* pa-

rameter or very recent *updateTime* parameter. Such a request might have been constructed by a node unable to find sufficiently recent cached responses in *CachingNodes* of its neighbourhood. All similar, legitimate requests should be replied as knowledge of certificate revocation status may be critical for some applications.

A combination of general and ADOPT-specific *TVs* is considered necessary for ADOPT's efficient operation. Taking into account parameters concerning how nodes behave when they execute ADOPT results in having a more complete view of trust for each node. Regarding requests, an intermediate node receiving *AReq* messages should check its *TTL* and *updateTime* parameter. If these two values are considered reasonable (e.g. low *TTL* and recent *updateTime*), the request may be legitimate even if other *Trust Values* indicate otherwise. Of course, the specific message may have travelled a lot in the MANET before reaching the specific node, so that its initially large *TTL* may now appear low. As a result, node proximity should also be considered. A node's Trust Builder wishing to evaluate another node's trust, in terms of its behaviour regarding *AReq* messages, should, thus, consider the *TTL* parameter in each request message combined with the node's location. On the other hand, it is very hard to decide if a queried certificate actually exists, as it is impossible for a node to have knowledge of all issued certificates. Regarding *updateTime*, values indicating a time in the future demonstrate a possibly illegitimate request message. However, messages containing values that are near to the current time cannot be regarded malicious by default. In such inconclusive cases, other *TVs* should be also examined in order to provide the most possible accurate result.

Concerning response messages, the following steps are required to verify their validity:

1. the digital signature on them should be valid,
2. the referenced certificate should be the one that was queried, and
3. the *thisUpdate* and *producedAt* fields should be recent enough, while the *nextUpdate* field should indicate a time in the future [11].

As we have already mentioned, signature verification requires considerable resources and, thus, should not be performed in every node for a large number of messages. Similarly, checking the referenced certificate for every response is also impractical, as each node would also have to cache the corresponding request message. Consequently, the only parameter that should be considered by the *Trust Builder* is the combination of the two OCSP date fields. Nodes sending aged cached responses should be considered suspicious for malicious behaviour. Likewise, messages whose *producedAt* or *thisUpdate* values indicate a future time, must have been originated from malicious nodes.

Ideally, some trusted nodes in the MANET should be able to perform the aforementioned tasks in order to verify responses and provide trust feedback to other nodes. ATF already supports a similar function as it introduces the notion of trusted *recommenders.* These are nodes that have proved their trustworthiness and provide trust recommendations to other nodes on demand. In ADOPT's case, such nodes could verify response messages and provide recommendations to nodes that suspect malicious behaviour. Normally, *recommenders* should be nodes with increased processing, memory and energy resources and thus should be capable of performing the required cryptographic functions.

A common problem in trust and reputation systems is false accusations, where a network entity is given a wrong classification (e.g., malicious, or selfish). Inadvertently, ATF, and the ADOPT over ATF framework, may face such situations due to unseemly estimation of nodes' trustworthiness. To minimize false accusations, ATF relies on several mechanisms that have been already described. Firstly, the *TrustSpan* algorithm aims at filtering out recommendations from "inexperienced" nodes in order to limit rumour spreading. *TrustSpan* relies on the predefined *RF*. The value update for this function is performed automatically through the *RFTS* or can be triggered by explicit testing mechanisms (i.e., ask nearest nodes about the trustworthiness of a node T for which you have already a very strong evidence).

Additionally, as we have already mention, the weights of *REC* and *DE* in the *TV* computation depend on *MI*. Thus, a node can flexibly control how much it should rely on recommenders. In general, the minimization of false depends, among others, on the Trust Policy followed by each node. An explicit evaluation of the false accusations effects as a function of possible *TPs* is left as future work.

4.3 Integration of Trust

Trust management for MANETs raises, as happens with every new computing paradigm, questions regarding its seamless integration with current network protocol stacks. In particular, one should define how the introduction of trust affects the architecture and operation of current services, in our case of ADOPT, and identify potential difficulties or problems during such integration. In general, three types of modifications are necessary for the desired integration (see also Figure 4):

1. **Introduction of a trust plane.** This is a vertical plane similar to the user/control and management planes of other networking specifications. It includes all the necessary components in order to assess trust of third entities: sensing mechanisms, policy-driven evolution of trust, interaction history etc. Trust plane operates also as a broker since it disseminates to all other interested layers the observations of each specific layer. For example, it may give feedback to the networking layer (i.e., routing) about the physical layer operation of peer entities.
2. **Recommendation exchange through a trust protocol.** Such protocol could be implemented in the application layer and is responsible for the request and receipt of recommendations (i.e., in ATF it would be part of the *Reputation Manager*).
3. **Trust-aware versions of current protocols.** The observations collected by the trust plane or the recommendations collected through the trust protocol should somehow affect the operation of the network stack. This can be only performed if the protocols support trust-driven re-configurability.

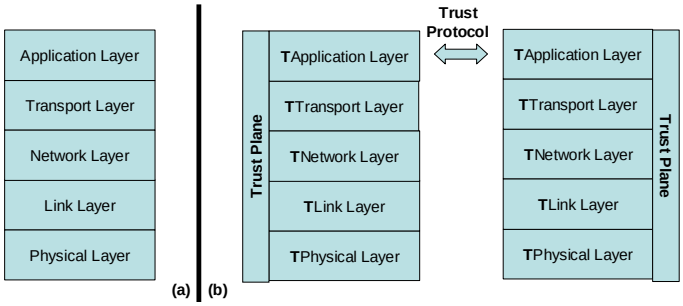

Figure 4: (a) Traditional network stack (b) New stack elements imposed by incorporation of trust (bold text in figure)

An example of a trust-aware, self-adapted and reconfigurable system is the software radio. Specifically, software radio serves as a radio communication system technology that uses software methods for the modulation and demodulation of the transmitted signal. The interface to the physical layer is no more hardware-based and fixed, but a set of interfaces provided by the deployed software. A straight-forward result of a trust-aware version of the software-defined radio is that the physical layer can be altered to any supported protocol by simple software redeployment, triggered by the trust plane. The initial triggering may be caused by an application that identifies poor Quality of Service, such as high bit error rate. Alternatively an upper layer might demand switching to a more secure version of the physical layer protocol, e.g., modified hopping sequences of the CDMA, possibly due to distrusted neighbours.

The trust plane that deals with trust management issues is distributed, and resides in each MANET node. It is similar in nature to the knowledge plane proposed in [18]. The similarities consist of the autonomic and distributed nature of the planes, and the "subjective" approach the proposed constructs follow. The trust plane, however, imposes requirements, which are harder and of narrower scope, on the design. The representation and reasoning of trust entities and relationships is strict and the operation of the whole autonomic systems network is very sensitive to any weak interpretation of trust.

In our case, ATF acts as the trust plane and ADOPT as a trust-aware application. When integrating the ATF trust plane with the ADOPT scheme, several performance measurements should be considered, such as:

- The communication overhead introduced when ADOPT invokes the ATF component and produces on-demand recommendation requests, and the corresponding responses.

- The time required by each node to identify the actual behavior of a peer (i.e., how rapidly ATF enables peers to identify the real *TV* that corresponds to an ADOPT misbehaving node)

4.4 Optimizations: TrustPath algorithm and Cerberus function

As mentioned in Section 2.2, a *ClientNode* might receive multiple *ARes* during a *WW*. During this time window, invalid responses might be received. From the captured responses, the *ClientNode* should select a valid one, which contains the most recent *producedAt* field. This process might introduce significant delays, since the verification of each response requires modulo arithmetic due to the RSA signatures. Thus, energy resources might be consumed unnecessarily. On the other hand, if the *ClientNode* is aware of the identity and the trustworthiness of the nodes that are located on the reverse path, i.e., the path that each response traverses, then it might drop those responses that arrive through a distrusted path. Such mechanism aids the *ClientNode* to avoid wasting time and scarce resources to verify each *ARes*. For this reason, we have developed the *TrustPath* algorithm to identify the trustworthiness of the reverse path. *TrustPath* determines if the response is originated by a legitimate *CachingNode* or *ServerNode* and if it passes through trusted nodes, as far as the *Forwarding Function (FF)* is concerned. For the calculation of the reliability of paths, the *TrustPath* defines the following rules concerning the admission of *ARes* at the destination (i.e., *ClientNode*):

Rule 1. If it was produced by a malicious *CachingNode* then it is dropped (*Caching Function, CF*) in Listing 2.

Rule 2. If it traverses through a path that consists of at least one distrusted node concerning the FF function, then it is dropped.

Rule 3. If it traverses through a path that consists of many (i.e., more than *HFI*) unclassified nodes concerning the trustworthiness of the *FF* function, then it is dropped.

Rule 4. If it traverses through a path that consists of few unclassified (i.e., less than *HFI*) concerning the trustworthiness of the *FF* function, but far-off nodes, then it is dropped.

Rule 5. If it traverses through a path that consists of few unclassified (i.e., less than *HFI*) concerning the trustworthiness of the *FF* function, but nearby nodes, then it is stored as pending. The *ClientNode* asks recommendations for the unclassified nodes, using the *TrustSpan* algorithm (section 3.4). If all nodes are reported to be trusted concerning the *FF* function, then the pending *ARes* is ranked as trusted in terms of forwarding and candidate to be examined as valid, otherwise is dropped.

Rule 6. If it traverses through a path that consists of trustworthy nodes, then it is accepted.

The *TrustPath* procedure is presented in Listing 2. The *MU* is defined as the maximum tolerated hop distance between an unclassified node and the *ClientNode* in order for the latter to ask for recommendations.

Moreover, in order to further minimize the number of *ARes* that a node has to evaluate, but also the total network traffic overhead that ADOPT produces, we have introduced the *Cerberus* function. *Cerberus* takes into account ATF measurements concerning the trustworthiness of nodes, and drops packets originated from or forwarded to malicious nodes. In our case, *Cerberus* drops *AReq* forwarded to malicious *ServerNodes* or *CachingNodes*. Additionally, it drops responses initiated by malicious *ServerNodes* or *CachingNodes*. Its purpose is to isolate these malicious nodes from the MANET, in order to reduce the total number of malicious messages that circulate in it. Malicious nodes that attempt to deploy attacks similar to those we described in section 4, are first flagged by ATF as malevolent and then the *Cerberus* function ensures that they are isolated from the MANET. The possibility of *AReq* or *ARes* flooding is thus minimised. Nodes that fabricate *ARes* are also flagged as malicious by ATF and isolated. The *TrustPath* algorithm additionally ensures that even responses that have been forwarded by such nodes will be ignored.

5. Simulation results and assessment

In a prototype simulation implementation of the ADOPT and ATF schemes, we have evaluated the performance of the integrated scheme, using the J-SIM wireless package simulator

[25]. We used the IEEE 802.11 protocol as MAC layer and the AODV as an IP routing protocol. Nodes were deployed in a 500m x 500m terrain, whereas their radio transmission range was set equal to 50m. Initially, the nodes were distributed randomly on the terrain grid.

Listing 2 – TrustPath algorithm

```
Definitions:
FF, CF      -- Forwarding and Caching Function, respectively
Vc          -- ClientNode id
Vr          -- CachingNode id
Inputs:
Vp          -- Vector of nodes, {V1, ..., VK, Vr}, in the path traversed by the ARes
TTH(RF); -- The minimum TVRF of the recommender in order to be trusted

Output:
Boolean valid   -- If the response is valid

procedure TrustPath () {
Θ[];              -- positions of unclassified nodes in Vp
TrustedRecs[]; -- Trusted Recommenders
K                 -- Path length
KU                -- Number of unclassified nodes in Vp (i.e., nodes with no TV assigned), KU < K
KD                -- Number of distrusted nodes in Vp (i.e., nodes with TV less that TTH),  KD < K

   if ( TV(Vr, CF) < TTH(CF) ) return –1;          -- rule 1
   for (j=1 to K) {
        if  (TV(Vj, FF) < TTH(FF)) KU ++;
        if  (TV(Vj, FF) == UNKNOWN) {
           KU ++;
           Θ[j] ++;
   }

   if (KD > 0) return –1;                          -- rule 2
   if (KU  < HFI) return –1;                       -- rule 3
   for (j=1 to K)                                  -- rule 4
        if ((Θ[j] > 0 )  && ( j  < MU ))
          return –1;
   TrustedRecs[]=TrustSpan();                      -- find nearest HFI trusted recommenders
   for (j=1 to K)                                  -- rule 5
        if (Θ[j] > 0 ) {
           Ask_Recommendations();                          -- ask for HFI recommendation for Vj's FF
           RECvj,FF()=WAH(j, FF, t);                       -- calculate REC for node's Vj FF
           TV(Vj, FF)=(a*DE (Vj, FF)+ b*REC(Vj, FF))*SUB (Vc, FF) ;    -- TV for Vj's FF
           If (TV(Vj, FF) <  TTH(FF))
             return –1;                                    -- Node is distrusted and rule 2 applies
        }
   return 1;                                       -- rule 6
}
```

In the simulations we used topologies of different density, using 30, 40 and 50 nodes. Nodes' mobility was simulated according to the random waypoint mobility model [20], in which each node moves to a randomly selected location at a configured speed (i.e., 2 m/s) and then stops for a configured pause time (i.e., 5 sec), before choosing another random location. For our tests we used the test certificates that gave us 115 byte *AReq* and 460 byte *ARes*, including ADOPT extensions. We assumed that one certificates was issued for each node. Each node was assumed to support a cache of 30, 40 and 50 OCSP entries. The number of malicious *CachingNodes*, Nc, was set to 5 and 8, and each one of them was configured to alter all the cached OCSP entries. The number of malicious *ServerNodes*, Ns, was set to 5 and 8, and each one of them was configured to provide only malicious *ARes* for any queried certificate. Table 2 summarizes the parameters of the simulation environment.

Various different policies were evaluated concerning how often nodes examine ADOPT messages in order to verify *ARes* before caching them. In detail, we assumed that a node may choose to follow a "*LIGHT*" policy and check only 50% of the messages before caching them

or to follow a "*HARD*" policy performing all checks. Furthermore, in some scenarios ATF's *Cerberus* function was also enabled. In our simulations we have measured:

- The communication overhead introduced when ADOPT invokes the ATF component and produces on-demand recommendation requests and the corresponding responses
- The average number of malicious responses that a node receives and has to evaluate before receiving a legitimate response
- The profit, in terms of discarded malicious messages, gained from *Cerberus* function
- The impact of caching policies and cache size on the delay of receiving a valid response

During the simulations various nodes request CSI, at different points in time. We only observe two specific nodes' requests (IDs 10 and 20), concerning the certificates of 10 *targets* (with serial numbers 2, 5, 7, 9, 12, 15, 19, 22, 25, 29).

Table 2: Simulation environment parameters

Number of nodes	30, 40 or 50
Number of malicious caching nodes (Nc)	5 or 8
Number of malicious server nodes (Ns)	5 or 8
Maximum speed	2 m/sec
Pause Time	5sec
Terrain dimensions	500m x 500m
Radio transmission range	50m
OCSP Cache Size (entries)	30, 40, and 50
Caching Policies	Greedy (*GRE*), Selective (*SEL*, m=3)

5.1 Communication Overhead

The communication cost that the ADOPT produces over ATF due to recommendations (i.e., *TrustPath*) is depicted in Figure 5. For this scenario the *HFI* was set to 3. Additionally, the *MI* parameter was set to 20 and 30 direct evidences. As shown in Figure 5, the overhead decreases with time, and eventually reaches the 20% of the actual traffic. This happens because the number of direct evidences that the pairs of the nodes obtain increases with time, and, thus, fewer recommendations are requested from trusted peers. Additionally, for higher *MI* values the overhead is higher, since the minimum direct evidences required for being confident about the *TV* of a *target* is higher and more recommendations are requested from trusted peers.

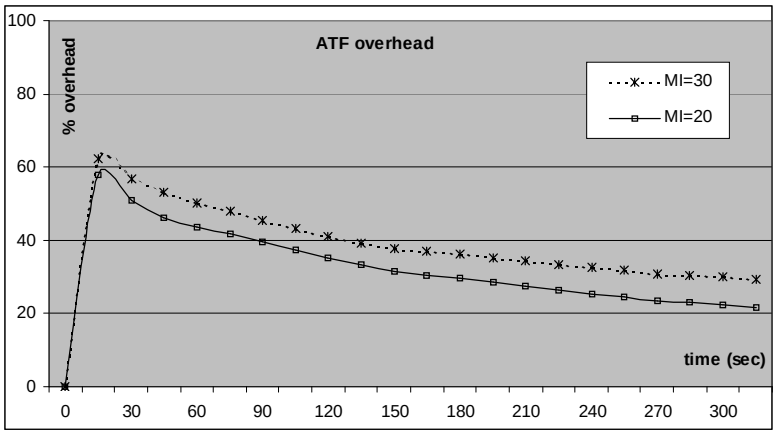

Figure 5. Overheads produced when ADOPT requires recommendations of peers

5.2 Detection Time

Figure 6 illustrates the detection times of ATF, i.e., the number of direct evidences required for a fair node i (i=0, ..., 19, 30,…,40) to detect a specific misbehaving node (here for the selfish node with id=27) in respect to the packet forwarding function. For this scenario we have preconfigured ten nodes to act selfishly (nodes with ids 21-30), and only the rational nodes assess the packet forwarding function of the ten selfish nodes. Each selfish node was

pre-configured to drop, without forwarding, 30% of the packets that it receives. Thus, we assume that a node has successfully detected selfish behaviour of node 27 when the *TV* that it maintains for the packet forwarding function of node 27 reaches 0.7. In this scenario, the weight of history, *w*, was set equal to 0.7 or 0.9 (**Eq. 1**). As Figure 6 depicts, moderate values of w (e.g., 0.7) produce low detection times, since less than 10 direct evidences are required for the recognition of the selfish node 27. Some nodes (except for the selfish ones, 21-30) did not manage to compute a TV for node 27 at all (e.g., nodes 2, 14, and 42). This is because they are far way from node 27 and, thus, do not have an adequate number of direct evidences to evaluate node's 27 *TV*. Finally, nodes' random mobility patterns also result in some time detection deviations, which have been measured for the two scenarios of Figure 6. For example, node 20 in the second scenario (i.e., *w*=0.9) was moving using a trajectory pattern that did not came near node 27 as the trajectory of the first scenario (i.e., *w*=0.7) , and, thus, it delayed in detecting node's 27 selfish behaviour.

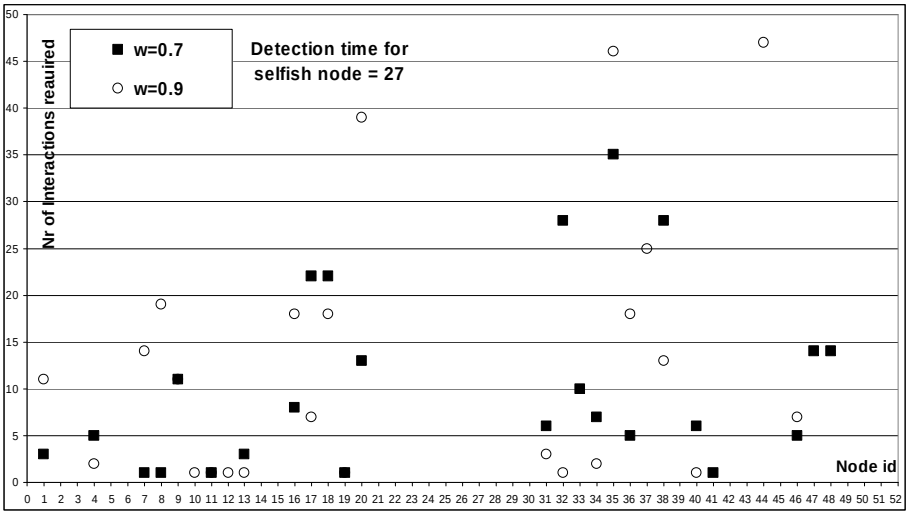

Figure 6. ATF Detection Time

5.3 Number of Responses and Trusted Paths

Let's assume that *ClientNode* 20 issues *AReq* for the CSI that corresponds to ten (10) *targets*. Figure 7 illustrates the total number of *ARes* received by the *ClientNode*, under different number of malicious caching and server nodes (values of Nc, and Ns, respectively), when the LIGHT policy is used. The large set of the captured responses consists of invalid (i.e., malicious) and valid responses. Thus, the *ClientNode* should parse and verify 250 (Ns=8), or 150 (Nc=8), or 245 (Ns=5), or 225 (Nc=5) responses in order to find the most recent and valid certificate. Such a procedure might require significant resources and time. As shown in Figure 7, for every simulation scenario, the number of trusted paths when *ClientNode* 20 issues *AReq* for the CSI that corresponds to ten *targets* is equal to or less than 10. Thus, the *ClientNode* takes advantage of the *TrustPath* mechanism to select the ten (when Ns=8, or 5, and Nc=5) or eight (Nc=8) valid responses, in order to identify the most recent one. Similar results appear when the *HARD* policy was used, as Figure 8 depicts. Trusted paths are much less than the received valid and invalid responses.

As illustrated in Figure 8, the *SEL* strategy minimizes the communication over-head, since fewer responses are produced by the *CachingNodes* and traverse the network to-wards the *ClientNode*. Thus, when nodes are caching selectively (using e.g., *m*=3 as the popularity index) valid responses are still found on peer entities, whilst the communication cost is minimized. Additionally, from the last two figures we observe that the malicious *ServerNodes* introduce more overhead of invalid responses than the malicious *CachingNodes.* For instance, from Figure 8 it is concluded that there are 50% invalid responses when Ns=8 and 35% invalid responses when Nc=8 (*GRE*). Additionally, there are 46% invalid responses when Ns=8 and 40% invalid responses when Nc=8 (*SEL*). Furthermore, the *TrustPath* algorithm introduces an overhead in terms of recommendations that nodes ask in order to evaluate a path. As

we have already mentioned a *ClientNode* using the *TrustPath* algorithm may ask for recommendations for unclassified nodes. The number of recommendations that *ClientNode* 10 requested during various simulation scenarios is depicted in Table 3. As Table 3 indicates, the overhead due to the requested recommendations is tolerable, especially when the number of invalid responses is maximized. For instance, when N=50, Nc=8, and the *GRE* and *HARD* policies apply, there is a huge need for validation of 300 CSI responses, whilst only 52 recommendations are asked before identifying a trusted path.

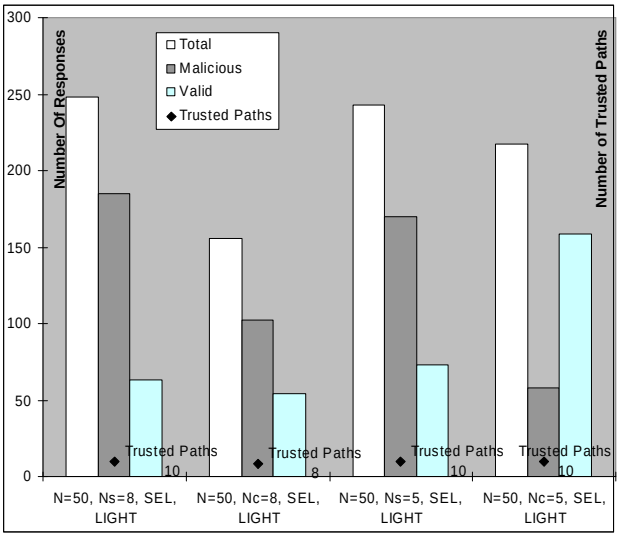

Figure 7. ADOPT total and valid responses in comparison with trusted paths for the *LIGHT* and Selective caching policies

5.4 Cerberus Function Gain

In this section we will discuss the effect of the *Cerberus* function. Table 4 illustrates the number of malicious ADOPT messages that were discarded by the *Cerberus* function. When the *LIGHT* policy applies the number of malicious responses that are dropped is quite high. This is because caching and server nodes maintain a larger number of altered entries in the *LIGHT* policy than in the *HARD* policy. Such entries are mapped to CSI requests, and then are returned by the malicious nodes. The first rightful node in the reverse path immediately discards these responses.

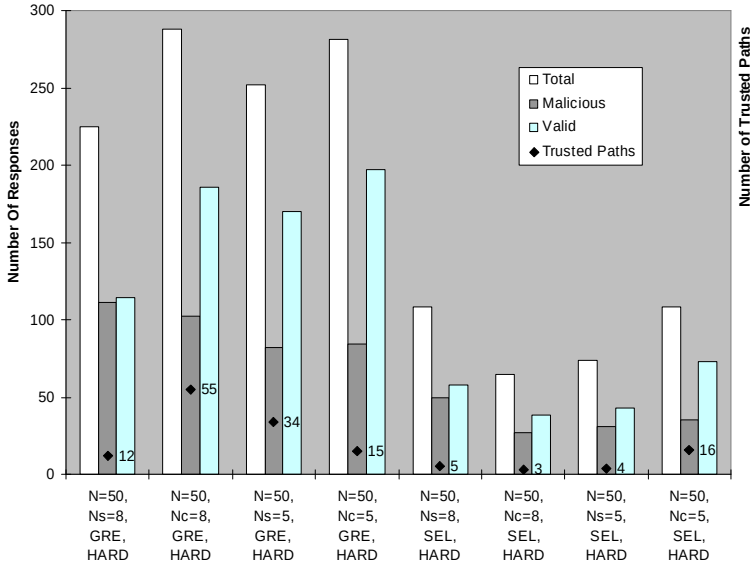

Figure 8. Total and valid *ARes* in comparison with trusted paths for the *HARD* and various caching policies

Additionally, Figure 9 shows that when *Cerberus* is activated the number of responses that *ClientNode* receives is significantly decreased. Additional measurements, not shown here, indicated that for the dense scenario (N=50) there was 25%, and 80% reduction of received *ARes*, for the *HARD* and *LIGHT* policies, respectively. Concerning the malicious responses, the corresponding reduction was 20% (*HARD*) and 75% (*LIGHT*).

Table 3. Recommendations asked for the TrustPath algorithm

Scenario	Number of RECs
N=50, Ns=8, *GRE, HARD*	52
N=50, Nc=8, *GRE, HARD*	25
N=50, Ns=5, *GRE, HARD*	50
N=50, Nc=5, *GRE, HARD*	32
N=50, Ns=8, *SEL, HARD*	38
N=50, Nc=8, *SEL, HARD*	81
N=50, Ns=5, *SEL, HARD*	24
N=50, Nc=5, *SEL, HARD*	56
N=50, Ns=8, *SEL, LIGHT*	44
N=50, Nc=8, *SEL, LIGHT*	46
N=50, Ns=5, *SEL, LIGHT*	81
N=50, Nc=5, *SEL, LIGHT*	86

5.5 Roundtrip Delay for receiving a response

Figure 10 depicts the round-trip hop delay that a *ClientNode* has to wait for until receiving a valid response, under various caching policies and cache sizes. The Greedy policy minimizes this delay, since each *ARes* is cached, while the cache is not depleted. The Selective policy introduces higher delays, since the responses are cached after m=3 appearances. Especially, when the cache size is relatively small (e.g., 30 entries for 50 available certificates) selective caching produces significant roundtrip delay (e.g., 17 hops for the certificate with ID 25), and, thus, it should be avoided if peer entities are equipped with efficient buffers.

Table 4. Cerberus function and packet discarding

Originated			Dropped Malicious Messages	
Requests	Responses	Scenario	Requests	Responses
3450	710	N=30, Ns=8, *GRE*, *HARD*	164	5
3480	929	N=30, Ns=8, *GRE*, *LIGHT*	143	532
3600	573	N=30, Nc=8, *GRE*, *HARD*	73	4
3480	534	N=30, Nc=8, *GRE*, *LIGHT*	69	149
3570	787	N=30, Ns=5, *GRE*, *HARD*	96	2
3390	856	N=30, Ns=5, *GRE*, *LIGHT*	95	511
3600	540	N=30, Nc=5, *GRE*, *HARD*	49	0
3570	666	N=30, Nc=5, *GRE*, *LIGHT*	70	126

6. Related Work

Public key cryptography schemes efficiently support confidentiality services to prevent passive attacks (e.g., eavesdropping), authentication of nodes, and integrity of messages to avoid fabrications. The architectures proposed in [2], [3] and [4] exploit threshold cryptography schemes to distribute the CA functionality over different nodes. In [5] GSM/GPRS technologies are proposed, enabling the nodes of a MANET to access CA services. An off-line CA is considered in [6], as well, to control an ad hoc network. Furthermore, in MANETs, digital certificates and signatures are employed to protect routing as well as packet forwarding. Secure routing protocols, such as ARAN [7], SAODV [8], and forwarding modules, such

as TRM [9], involve CAs. Some of the aforementioned proposals [4], [5], [7] include, or assume the existence of, a scheme for retrieving CSI, a service which is essential wherever digital certificates are involved. ADOPT was proposed in [10] to provide on demand up-to-date CSI to any node in a MANET. To the best of our knowledge no other scheme like ADOPT has been proposed so far.

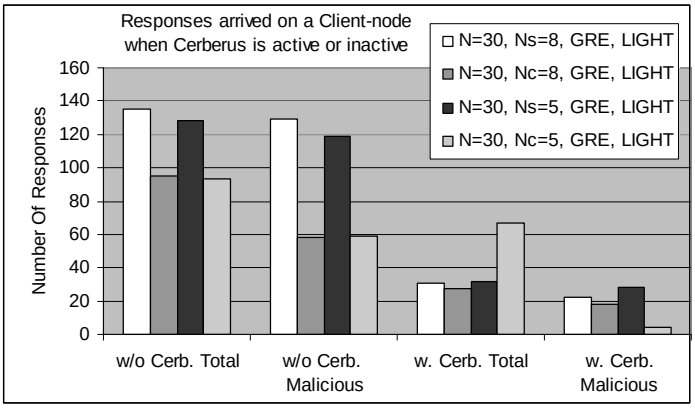

Figure 9. Responses on a *ClientNode* with and without *Cerberus*

On the other hand, key-based schemes are considered computationally hard for MANETs, due to complicated key management techniques, namely generation, distribution, verification, and revocation of keys or certificates. For this reason, many trust establishment solutions in MANETs rely on the trustworthiness of peers. These, self-evolving, reputation-based, schemes are considered suitable for avoidance of selfishness. They are practical if certificate distribution centres are ephemerally present and computation resources (e.g., energy, memory) are scarce. They are based on the determination of the trustworthiness of nodes, regarding their offered functionality. A primary goal of rational nodes is to cooperate in order to avoid, or even mutually isolate, notorious nodes (i.e., selfish, malicious) from routine network operations. Such cooperation requires the exchange of recommendations and the identification of trusted *recommenders*. Several cooperation enforcement schemes have been proposed in the literature, such as CONFIDANT [14], CORE [15], SORI [16] and OCEAN [17].

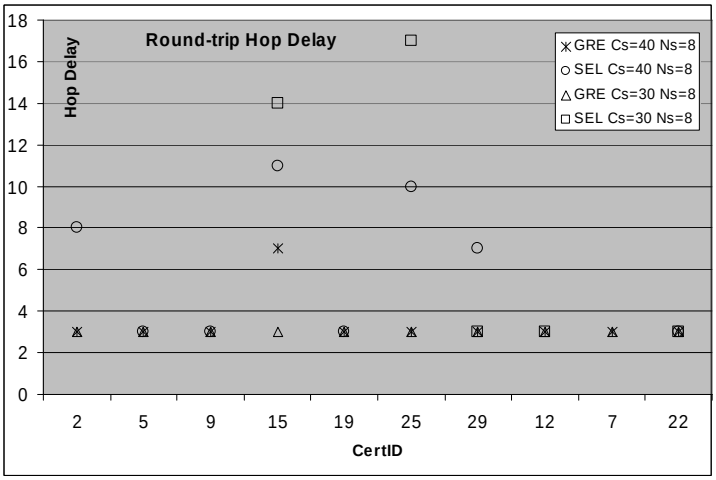

Figure 10. Responses on a *ClientNode* with and without *Cerberus*

CONFIDANT designed as an extension to DSR routing protocol, materializes monitoring and reporting of behaviours for route establishment that avoids selfish nodes [12], [13], [14]. It relies on four functional components: a) the monitor, b) reputation records for first-hand and trusted second-hand observations about routing and forwarding function of other nodes, c) trust records to control trust that is given to received warnings, and, d) a path manager to

take routing decisions. In CONFIDANT, node can detect selfish behaviour of the next node in the route either directly, by promiscuously sensing its transmission, or indirectly, by observing routing protocol misbehaviour. The monitor component registers these deviations. As soon as a specific misbehaviour occurs, the Reputation System is called, and ALARM messages are sent. Recipients of the ALARM messages are friends, which are maintained in a friends list. Incoming ALARMs that originate from "strangers" are checked for trustworthiness before triggering a reaction.

CORE relies on the DSR routing protocol, as well [15]. It uses first and second-hand experiences, and a Watchdog mechanism for the evaluation of other nodes' behaviour. Each node monitors the behaviour of its neighbours, with respect to the requested function, and collects observations about the execution of that function. If the observed behaviour is different than the outcome of a predefined function, then the rating of the observed node is altered. Observations are recorded on the Reputation Table (RT). Each row corresponds to a neighbour node and consists of three entries, regarding the monitored function: a collection of recent (first-hand) observations, a list of the recent second-hand TVs provided by other nodes, and the TV evaluated for the monitored function. The formula used to evaluate the TV avoids false detections by using an aging factor that gives more relevance to past observations. However, such an approach is vulnerable to an attack where a node builds up a good reputation before misbehaving.

The Secure and Objective Reputation-based Incentive Scheme (SORI) for Ad-hoc Networks, focuses on the packet forwarding function [16]. SORI, consists of three basic components: Neighbour Monitoring, Reputation Propagation and Punishment. A promiscuous mode is assumed, where a node is capable of overhearing the transmissions of its neighbours and maintains a neighbouring node list. SORI combines features of the fist-hand schemes and those that use reputation spreading. The nodes exchange reputation information only with their neighbours. This way a no-cooperative node will be punished by all of its neighbours (who share the reputation information about its misbehaviour), instead of just the ones who are directly affected by this node.

The Observation-based Cooperation Enforcement in Ad hoc Networks (OCEAN) introduces an intermediate layer that resides between the network and MAC layers [17]. This layer helps nodes in making intelligent routing and forwarding decisions. OCEAN relies only on first-hand observations. Every node maintains ratings for each neighbouring node and monitors behaviours promiscuously. Positive or negative events are recorded through the reaction of a neighbour that is expected to forward a packet. In OCEAN, the absolute value of a decrement is chosen to be bigger than the value of an increment. When the rating of a node drops below a threshold, called faulty threshold, the node is added to a faulty list. This list is attached to the Route Request message of the DSR protocol in order to be flooded. A route is rated good or bad, based on whether the next hop in the route belongs to the avoid-list. The receiver of a RREQ decides to drop or to further process it (through relaying or a Route Reply), if the intersection of the avoid-list and the DSR route in the RREQ packet is void. This way, each node along a route, makes its own decision about the trustworthiness of other nodes, and has control only over routes that it belongs to. Every node rejects data packets arrived from nodes belonging to its faulty list. Thus, misbehaving nodes are eventually isolated.

CONFIDANT and CORE were designed as an extension to the DSR routing protocol for trusted route establishment, SORI focuses on the packet forwarding function, and, finally, OCEAN, introduces an intermediate layer between the network and the MAC layers. Thus, these network, DLC, and MAC layer specific schemes cannot be directly integrated with the ADOPT, which is an application layer trust-building framework. On the other hand, even though ATF illustrates similarities to the aforementioned schemes, it does not concentrate on routing and packet forwarding mechanisms and demonstrates significant innovations. These include the deployment of the ***TrustSpan*** algorithm to reduce the communication overhead that recommendations produce, and the ***TrustPath*** algorithm to identify and use trusted routes for propagating recommendations. It introduces the ***Cerberus*** function to isolate selfish nodes. It uses a Trust Sensor to monitor and evaluate the trustworthiness of the recommendation function provided by peers, using tests with predefined results. It incorporates historical, sub-

jective and trust policy parameters to evaluate the trustworthiness of nodes' functions. Finally, it provides a second chance mechanism, where isolated nodes always have the opportunity to re-enter, if they start to cooperate.

7. Conclusions

In this paper we discussed the integration of a trust assessment framework, ATF, with ADOPT, a distributed OCSP deployment for MANETs. ADOPT, implemented as a trust-aware application, can take advantage of ATF to improve its efficiency. We discussed some security weaknesses that ADOPT illustrates when malicious or selfish nodes are present, and described how the integration with the ATF mitigates such weaknesses. Details of integration of ADOPT and ATF were presented, according to which ATF acts as a trust plane, supporting trust-aware applications such as ADOPT. To enable seamless integration we suggested new parameters that ATF should take into account when calculating the trustworthiness of peers, that are brought about by the behaviour of nodes when executing ADOPT. Simulation results illustrate that ATF does not introduce substantial communication overheads. On the other hand, using ATF, legitimate nodes have sufficient means to isolate selfish or malicious peer nodes, significantly improving ADOPT's reliability and availability. Furthermore, simulation results show that it is preferable to pay an overhead in communication cost for detecting and isolating malicious *recommenders* and formulating trusted paths via the ATF procedures, than to propagate and process fabricated CSI responses. ADOPT can thus rapidly locate legitimate CSI. This information is transferred through trusted paths, established by the nodes when using ATF mechanisms. Processing overhead is also minimized, since *ClientNodes* do not need to evaluate responses that ATF identifies as having originated from, or forwarded through malicious nodes.

Currently we are enhancing the ATF protocol to include supplementary ADOPT-oriented recommendations. ADOPT will, thus, be able to take further advantage of ATF so as to provide fresher responses and distribute more efficiently cached responses. Our goal for ADOPT is to optimize performance metrics, such as the delay in the location of the certificate's status information, the minimization of cache capacity, as well as the decrease of the communication overheads of both ADOPT and ATF. In this direction we are further investigating optimised caching policies, which depend on nodes' mobility, connectivity, capacity and trustworthiness.

8. Acknowledgements

Part of this work was performed in the context of the project entitled "PERAS: PERvasive and Ad hoc Security" co-funded from the European Union by 75% and from the Hellenic Ministry of Development, General Secretariat for Research and Technology by 25% under the framework "PENED03".

References

[1] F. Stajano, and R. Anderson, "The Resurrecting Duckling: Security Issues for Ad hoc Wireless Networks", in Proc. 7th Intl. Workshop on Security Protocols, 1999.

[2] Zhou, and Z. Haas, "Securing Ad Hoc Networks", IEEE Network vol. 13, no.6, Nov-Dec. 1999.

[3] J. Kong, P. Zerfos, H. Luo, S. Lu, and L. Zhang, "Providing robust and ubiquitous security support for mobile ad-hoc networks", in Proc. of International Conference on Network Protocols (ICNP), 2001.

[4] S. Yi and R. Kravets, "MOCA: Mobile certificate authority for wireless ad hoc networks", In Proc. 2nd Annual PKI Research Workshop (PKI03), April 2003.

[5] J. Cheambe, J.-J. Tchouto, C. Tittel, T. Luckenbach, and M. Bechler, "Security in Wireless Ad-Hoc Networks", In Proc. 13th IST Mobile & Wireless Communications, Lyon, France, June 2004.

[6] S. Capkun and J.-P. Hubaux, "BISS: Building Secure Routing Out of an Incomplete Set of Security Associations", in Proc. ACM WiSe2003, San Diego, USA, Sept. 2003.

[7] K. Sanzgiri, B. Dahill, B. Levine, C. Shields, and E. Royer, “A secure routing protocol for ad hoc networks", in Proc, 10th IEEE International Conference on Network Protocols (ICNP’02), Paris, France, Nov. 2002.
[8] M. G. Zapata, and N. Asokan, “Securing Ad hoc Routing Protocols”, in Proc. ACM WiSe02, Atlanta, Georgia, USA, Sept. 2002.
[9] V. Leung, J-H Song, Y. Kawamoto, and V. Wong, “Secure Routing with Tamper Resistant Module for Mobile Ad Hoc Networks”, in Proc. ACM MobiHoc2003, Annapolis, Maryland, USA, June 2003.
[10] K. Papapanagiotou, G. F. Marias, P. Georgiadis and S. Gritzalis, " Performance evaluation of a distributed OCSP protocol over MANETs", in Proc. IEEE Consumer Communications and Networking Conference 2006 (CCNC06), Las Vegas, January 2006.
[11] M. Myers, R. Ankney, A. Malpani, S. Galperin and C. Adams, “RFC 2560 - X.509 Internet Public Key Infrastructure Online Certificate Status Protocol – OCSP”, IETF, June 1999.
[12] S. Buchegger and J.-Y. Le Boudec, "Performance Analysis of the CONFIDANT Protocol ", in Proc. 3rd ACM Intl. Symp., on Mobile Ad Hoc Networking and Computing, Jun ’02.
[13] S. Buchegger and J.-Y. Le Boudec, “The Effect of Rumour Spreading in Reputation Systems for Mobile Ad-hoc Networks”, in Proc. WiOpt03, Mar. ’03.
[14] S. Buchegger and J.-Y. Le Boudec, "A Robust Reputation System for P2P and Mo-bile Ad-hoc Networks", in Proc. P2PEcon2004, Jun. ’04.
[15] P. Michiardi and R. Molva, “CORE: A collaborative reputation mechanism to enforce node cooperation in mobile ad hoc networks”, In Proc. 6th IFIP Commun. and Multimedia Security Conf., Sept. ’02.
[16] Q. He, D. Wu and P. Khosla, "SORI: A Secure and Objective Reputation-based Incentive Scheme for Ad-hoc Networks", in Proc. IEEE WCNC2004, Mar. ’04.
[17] S. Bansal and M. Baker. "Observation-Based Coopera-tion Enforcement in Ad-hoc Networks", Techical Re-port, Stanford University, ’03.
[18] Clark, D., Partridge, C., Ramming, J.C., Wroclawski, J.: A Knowledge Plane for the Inter-net. SIGCOMM ’03, Karlsruhe, Germany, 2003.
[19] R. Housley, W. Polk, W. Ford and D. Solo, “RFC 3280 - Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile”, IETF, April 2002.
[20] D. Johnson and D. Maltz, “Dynamic source routing in ad hoc wireless networks”, Mobile Computing, Vol. 353, Kluwer Academic Publishers, 1996.
[21] A. Arnes, “Public Key Certificate Revocation Schemes”, PhD thesis, Norwegian University of Science and Technology, Kingson, Ontario, Canada, Feb. 2000.
[22] G. F. Marias, K. Papapanagiotou, and P. Georgiadis, "Caching Alternatives for a MANET-Oriented OCSP Scheme", in Proc. First IEEE/CREATE-NET Workshop on Security and QoS in Communication Networks, Athens, Greece, September 2005.
[23] A. Abdul-Rahman and S. Hailes, “A Distributed Trust Model”, in Proc. New Security Paradigms Workshop 1997, ACM, 1997.
[24] G. F. Marias, V. Tsetsos, O. Sekkas, and P. Georgiadis “Performance evaluation of a self-evolving trust building framework”, in Proc. IEEE/CREATE-NET Workshop on the Value of Security through Collaboration, Athens, Greece, September 2005.
[25] J-SIM simulator package, available at http://www.j-sim.org/
[26] Y. Wang, J. Vassileva, Bayesian Network Trust Model in Peer-to-Peer Networks, in: Proc. Agents and Peer-to-Peer Computing 2nd Intl Workshop, July 2003, pp., 23-34
[27] C. Castelfranchi, R. Falcone, Trust is much more than subjective probability. Mental components and sources of trust”, in: Proc. HICSS33, Hawaii, 2000
[28] C. Perkins et al. Ad hoc On-Demand Distance Vector (AODV) Routing. In IETF RFC 3561, July 2003
[29] S. Marti, T. J. Giuli, K. Lai, M. Baker, Mitigating routing misbehaviour in mobile ad hoc networks, in: Proc. of Mobicom2000, Boston, USA, Aug. 2000
[30] M. Cieslak et al. “Web cache coordination protocol v2.0,” IETF Internet draft, 2000, http://www.ietf.org/intenet-drafts/draft-wilson-wrec-wccp-v2-00.txt