UNIVERSITÀ DEGLI STUDI DI PARMA

Dottorato di Ricerca in Tecnologie dell'Informazione

XXIII Ciclo

SECURITY IN PEER-TO-PEER MULTIMEDIA COMMUNICATIONS

Coordinatore:

Chiar.mo Prof. Carlo Morandi

Tutor:

Chiar.mo Prof. Luca Veltri

Dottorando: *Riccardo Pecori*

Gennaio 2011

A tutti coloro
che mi hanno a cuore

To everyone
who cares about me

Table of Contents

List of Figures

Introduction

Peer-to-peer (P2P) architectures became very popular in the last years as a consequence of the great variety of services they can provide. When they were born, they were mainly deployed as a simple, decentralized and scalable way to exchange files, but they have now become very popular also for a lot of different services, exploiting the possibility of sharing bandwidth, computing power, storage capacity and other resources between peers.

Among the possible uses such an architecture can be deployed for, an emerging field of study is the application of P2P technologies to VoIP communication scenarios in order to overcome some of the current issues centralized SIP-based platforms suffer of.

Unfortunately, security issues in P2P networks are still an open field of investigation both because of the recent development of such a platform and for the inherent risks of a distributed environment itself.

This thesis is meant to investigate the security issues and the possible solutions in order to setup a secure P2P communication. The research was conducted into two directions:

- Security issues at routing level;
- Security issues at application level.

They represent the two steps of a possible communication scenario: first of all, one must find in a secure way the location of the callee (maybe stored in a peer-to-peer network), this is a problem of secure lookup; then one must ensure that the person he is going to talk with is really who he wanted and that the communication itself is secret and cannot be tampered, these are problems of authentication and confidentiality.

As regards the first point, we studied several possible attacks to structured and unstructured peer-to-peer networks particularly focalizing onto the disruptive Sybil attack [1] from which many other attacks can be derived. After an analysis of the possible countermeasures presented over the years, we focalized onto the Kademlia algorithm, one of the most used in the world, studying through simulations the degradation of performances in presence of malicious nodes. We also studied trust and reputation countermeasures and tried to apply them to a Kademlia-based network operating in an environment where there is a growing number of malicious nodes.

For the second point, first of all we studied current key agreement protocols focusing on the number of messages and trying to find out possible drawbacks even in widely accepted protocols and algorithms. In a second time we proposed a new key agreement protocol based upon MIKEY [2] and ZRTP [3] integrating them into the standard SIP invite procedure. An analysis of the proposed protocol is also provided. On this basis we got further, adding also certificate-based authentication to our model and a way to manage in a P2P way certificates and signatures. Finally we also provided an architecture where certificates are stored in a P2P network itself with the use of a DHT.

Chapter 1

Peer-to-peer Networks

Working as peers is better
than always serving a client

– Riccardo Pecori

A P2P communication structure is made of multiple autonomous devices that interact as equals acting both as client and server, so there is no a centralized server or at least its usage is limited.

A P2P network is a quite complex system that represents a synthesis of several technological components, and one of them is the overlay network, that is a subgroup of the network composed by hosts running the same P2P client software. This virtual network overlaps the underlying network (e.g.: the Internet) with its own logical connections between nodes, maybe physically very far in the real world (see Figure 1.1). The logical links map onto maybe several physical links of a transport network.

P2P networks can be not structured or structured. In the first case arbitrary logical links exist among peers that can be all involved in sharing operations; in the latter case each node in the P2P network has its own identifier and connections according to a specific, firm algorithm that renders storage and retrieval operations simpler and faster.

In the history at least four [4] generations of P2P networks have been developed:

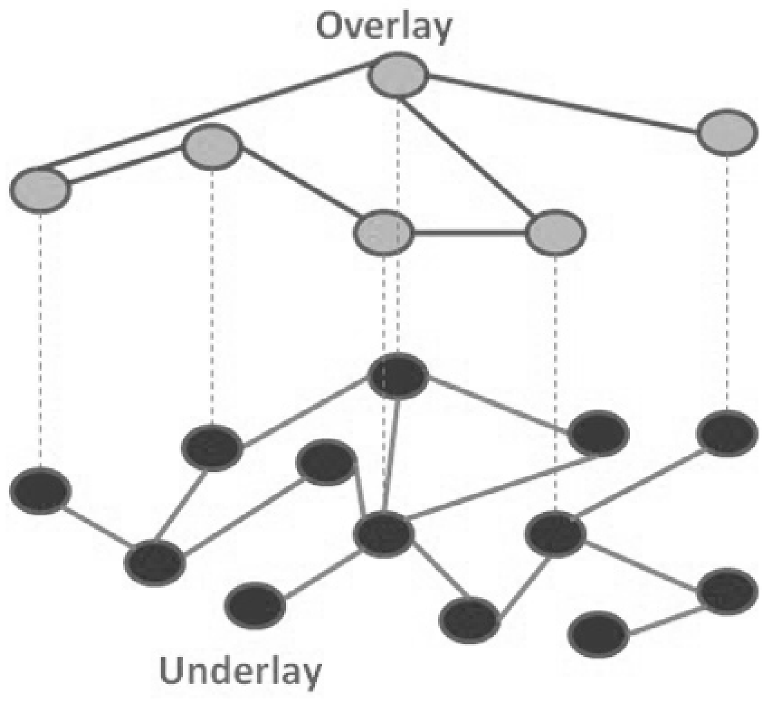


Figure 1.1: *Overlay and underlay networks.*

1. Centralized P2P networks like the by now dead Napster. They exploit a central server to provide indexes of distributed resources, and then the communication is peer-to-peer.

2. Pure P2P like Gnutella. In a pure P2P network each node implements functions of both client and server, and either peer can initiate a communication session at any moment. The research takes place flooding the network, and then the communication is only end-to-end between the initiator peer and the responder peer.

3. Hybrid P2P like JXTA or Skype. In this case there is a combination between the previous two cases; there are superpeers acting as servers towards a reduced part of the network. This improves bandwidth and grants signaling operations

saving.

4. Distributed Hash Tables (DHT) based P2P networks, which are a building block of many nowadays peer-to-peer applications like KAD. DHT mechanisms provide structured P2P networks and guaranteed data retrieval, moderate lookup times, automatic load balancing and self-organizing data storage and lookup system.

The first three are unstructured P2P networks, whereas the last one is usually defined as a structured network. The research work of the thesis was focused mainly onto structured p2p networks with particular care to DHT-based networks and their resource look-up mechanisms. P2P platforms that adopted a DHT structure are for example: eMule (Kademlia [5]), BitTorrent (Kademlia), CFS (Chord [6]), OceanStore (Tapestry [7]), etc.

1.1 DHT Networks

The general structure of DHT-based platforms provides a mapping (the hash table) between keys and values, where values are stored in a distributed manner across the p2p networks. The mapping is maintained in a distributed routing table in which nodes are addressed according to particular IDs. Both IDs/keys and values are the result of applying a hash function to some form of identification and the entire or a piece of the actual value.

Responsibility for maintaining the mapping from names to values is distributed among all nodes, in such a way that a change in a set of participants causes a minimal probability of disruption. This allows DHTs to scale to extremely large numbers of participants and to handle continual joins, leaves, and failures of nodes.

The lookup procedure, the core part of this type of networks, allows to find a certain value position in the network and so to store or retrieve the corresponding value that, in turn, can be a file, the Address Of Record of a SIP user, etc.

IDs distribution in the network and the lookup procedures vary a lot according to the particular DHT algorithm chosen, e.g. Chord distributes IDs in a ring, CAN [8] in rectangles, Kademlia in a tree-like manner, etc.

We now briefly describe two of the most used DHT, namely Chord and Kademlia, as these will be the starting points from which our research kicked off.

1.1.1 Chord

Chord is a lookup protocol based on consistent hashing that maps keys to nodes responsible for them in a circular virtual-ring structure. Every node and key are assigned a unique *m-bit* identifier using a hash function, e.g.: SHA-1, applied to the node's IP address, or the key's value.

Chord uses consistent hashing [9] to assign keys to its nodes as it tends to balance load, since each node receives roughly the same number of keys, and it requires relatively little movement of keys when nodes join and leave the system.

Chord views the identifier space as a circle formed by no more than 2^m nodes (where $m = 160$) with identifiers/keys ranging from 0 to $2^m - 1$ [6].

Each node of a Chord network maintains two data structures:

- successor list;
- finger table.

The first one is necessary to keep the correct organization of the network; in fact it contains peers immediately succeeding the node's key in the identifier circle in a clockwise direction.
The second one is needed to accelerate the process of lookup of a resource.

A node with the smallest ID that is greater than or equal to i represents the *successor* of a key (or node identifier) i. The *successor* node of a key is responsible for that key.
Figure 1.2 shows an example of a Chord ID ring. For example node 14 is successor of node 8 and of key 10, so it is responsible for key 10; node 32 is the successor of node 21 and keys 30 and 24 of which it is the responsible, etc.

When a node n leaves a network, all the keys it is responsible for should be reassigned to its successor. In the case when a node n joins a network, certain keys previously assigned to n's successor pass to n.
Once a new node joins the network, the node occupies an appropriate position in the identifier circle, and the predecessors of the newly joined peer update their successor lists.

A finger table is a routing table which contains IP addresses of peers halfway around the ID space from the node, a quarter-of-the-way, an eighth-of-the-way and so forth. Given N nodes in the Chord network, the size of a Chord routing table is $log_2 N$. If a node is looking for a resource with key k, it forwards the query to a node in its finger table with the highest ID not exceeding k reaching it in at most $O(log_2 N)$ steps.

The lookup in Chord can be implemented in both iterative and recursive modes; in the first case the control remains always to the initial requesting node who has, at every step, to send new request messages, in the second case the control passes,

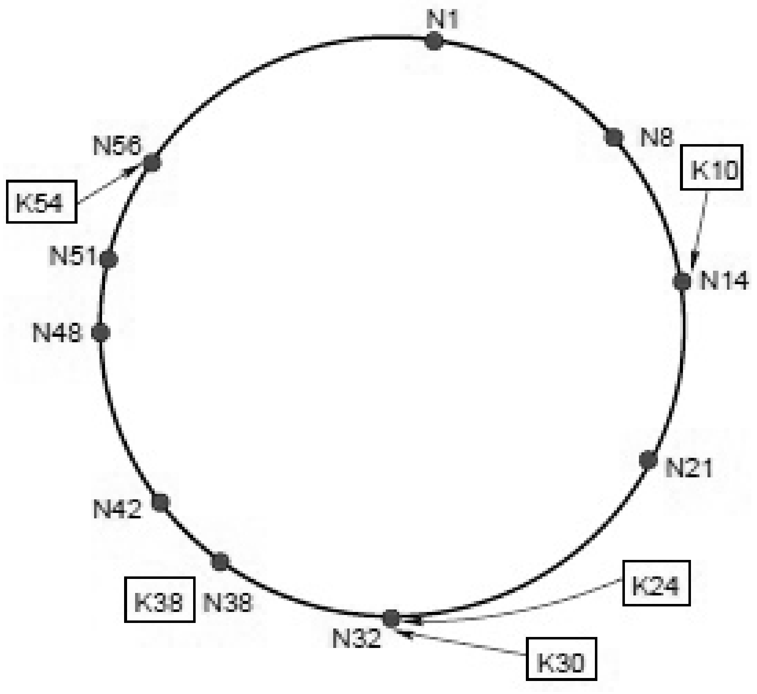


Figure 1.2: *Chord ID assignement [6].*

at every step, to the node at that moment closer to the searched resource who is in charge of sending the next requests to the next closer node.

1.1.2 Kademlia

Kademlia [5] networks are arranged basing upon a *XOR metric*, as the distance between two kademlia IDs is defined as: $d(x,y) = x \oplus y$. As in Chord IDs of both nodes and resources (keys) are usually made of 160 bits. Data are replicated in the k (the recommended value for k is 20) closest nodes to a key where key/value pair is stored. Kademlia, differently from Chord, has a binary tree-like data structure and nodes are considered as leafs on this tree (see Figure 1.3). Routing processes are implemented in prefix-matching mode.

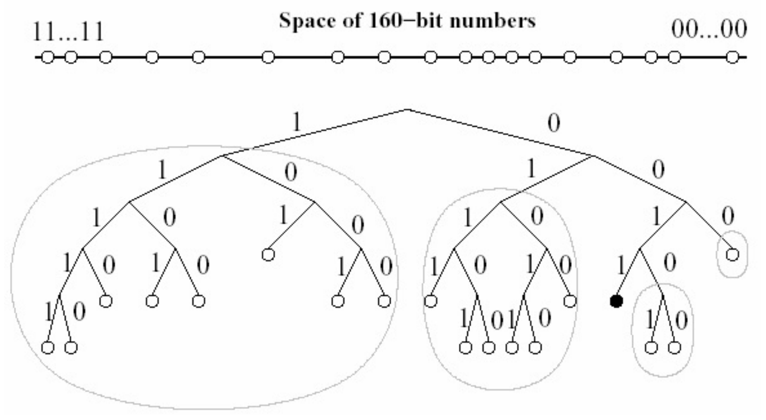


Figure 1.3: *Kademlia tree structure [5].*

Kademlia nodes store contact information to route query messages, keeping lists of *<IP address; UDP port; Node ID>* triples for nodes of distance from itself between 2^i and 2^{i+1} (in base 10) and for i ranging from 0 to 160 excluded. These lists are known as *k-buckets*, where k is both the length of the lists and the abovementioned system parameter that is usually chosen such that any given k nodes are very unlikely to fail within an hour of each other. Given the binary structure these *k-buckets* contain at least one node.

Kademlia routing is *iterative*, i.e., the node querying for a resource receives back, from the queried one, a list of nodes to ask at the next step until the node responsible for the searched resource is found; moreover, given the binary structure, a lookup procedure usually takes at most $O(log_2 N)$ steps where N is the number of nodes in the network.

In classical Kademlia both in storing and in retrieving resources the requesting peer runs α parallel asynchronous queries for a given key at the same time, α is another system parameter usually set to 3. The α peers are chosen by the requesting peer according to a certain policy among its k peers nearest to the searched resource that are stored in a temporary list to be updated at each iterative step. Each of the α peers contacted responds to the querying process initiator with its own k nodes nearest to the resource key to be find. The initiator peer can have as a response at most $\alpha \times k$ nodes; among these it selects for the next step only the k nearest to the researched resource and inserts in the temporary list the peers not already present. For the next step it iteratively selects in this temporary list, always in accordance to a certain policy, α to query, and so on. This iterative process ends up when at step n, among the at most $\alpha \times k$ responses, the initiator is unable to find at least one peer nearer to the resource than any other already present in the temporary list updated at step *n-1*. As a side effect of a lookup process, both for nodes or for keys, *k-buckets* of the contacted nodes are populated and updated.

Differently from Chord, Kademlia uses only iterative lookup procedures peforming them in parallel mode with α as parallelism degree.

Chapter 2

Security issues and countermeasures in peer-to-peer networks

Peer-to-peer networks are a special type of networks as, given their unstable structure, unreliable nodes may join or leave frequently, moreover they are decentralized so centralized solutions can be applied with difficulty [4]. P2p networks share the following weaknesses [10]:

- peers use the same client software, so a bug in the software can easily propagate in the whole network;
- there are many interconnections, so the possibility of getting a link with a malicious node becomes higher;
- in this type of network there is usually less attention from Intrusion Detection Systems;
- this type of networks usually exploit simple PCs that are usually more vulner-

able than server machines;

- users of p2p networks have less inclination to report unusual behaviors and often do not know who they should report to.

P2P networks are subject to both attacks of any network and attacks specifically made for p2p networks [10]; in particular DHT-based p2p networks, given their rigid structure, are subject to potentially disruptive attacks.

2.1 General Malicious Activities

General malicious attacks, that are in common with other types of networks, can be classified as:

- Denial-of-Service (DoS) and Distributed DoS Attacks
- Poisoning Attacks
- Tracking and Man-in-the-middle (MITM) Attacks
- Malware injection and propagation

The first one (DoS) consists usually into flooding the network with bogus packets or committing a peer into long computations, so preventing correct traffic to reach destination. If multiple hosts are involved into the attack we then speak of Distributed denial of service (DDoS).
A widely used technique to prevent DOS is “pricing”, that is, forcing requesters to solve computational puzzles before requesting a service. This method is not very effective in a DDoS scenario moreover it can be hard to implement for lightweight legitimate devices.

The second type consists of inserting into the network false information such as fake resource indexes, IP addresses, etc. these are not deadly attacks, whose effect can be only momentary, however some solutions involve replication, and thresholds.

The third type of attacks involves the finding out of some type of packet information, like IP addresses, and their manipulations. MITM attacks provide a fake entity inserting between two communicating nodes and listening undetected or pretending to be, from time to time, the other counterpart. This can be done wiretapping messages from both peers and acting as responder to these messages on both directions. A proved-to-work solution to these attacks is currently the deployment of Certification Authorities that grants authenticity of information and identities of peers, but that is not in accordance with the p2p paradigm.

The fourth attack involves malicious code spreading into the Internet like worms propagation. The best solution is to make users aware of the risks and to encourage them to deploy firewalls and antiviruses.

2.2 Specific Attacks to P2P Networks

As regards specific attacks to p2p networks, it is possible to consider [10][11]:

1. *rational attacks*, caused by the selfish behavior of peers willing only to take advantage of the network and with little cooperation with other nodes. On a large scale this can cause a system destabilization;

2. *routing attacks*, involving routing lookup schemes and algorithms as well as routing table maintenance and updating;

3. *storage and retrieval attacks*, with nodes denying the ownership of resources they actually have or on the contrary pretending to have resorces they do not really have;

4. *eclipse attack*, with an adversary having the control over some strategic routing path and succeeding into splitting the network in subnets it can controls;

5. *sybil attack*, with an entity having multiple identities;

6. *miscellaneous attacks*, such as inconsistent behaviors, overload of targeted nodes, rapid joins and leaves (churn), etc.

2.2.1 Sybil Attack

Particularly in this thesis work we focused onto the disruptive Sybil attack [1], as from this many other attacks, also the abovementioned, can be perpetrated. A sybil behavior exploits the concept of overlay and virtual IDs breaking the one-to-one mapping between overlay and underlay (see Figure 2.1).

This type of attack exploits a p2p networks usual custom, that is relying onto the existence of multiple, independent remote entities to mitigate the threat of bogus peers, since it is normally started by a faulty entity masquerading with *multiple identities*. The key idea behind the Sybil attack, or however a Sybil behavior, is in fact to insert in the p2p network malicious nodes identities, the *sybils*, which are all controlled by one single entity. This permits, positioning the sybils into strategic places,

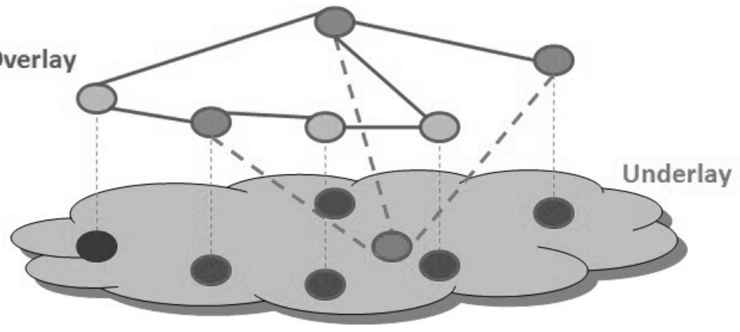


Figure 2.1: *Sybil behavior.*

to gain control over a part of the network or even over the whole network, to monitor the traffic or to abuse of the eventually present DHT protocol in other ways.

At present, there is no general solution to this type of attack despite the many approaches presented in literature. These ones can be classified, according to [12], into five categories: trusted certification, resource testing, costs and fees and trusted devices.
The most used anti-sybils P2P solutions are based on: the usage of diverse routing schemes, the limitation on the number of peers that a physical node can admit to the network, the verification of some types of requirement (computational resource, possession of some signed data, etc.), the periodical refresh of the routing tables or of the IDs. On the other hand the presence of a centralized authority (CA) is also claimed to be one of the best solutions if not the only one [1].

We made a detailed analysis and classification of the current types of solutions given to the Sybil behavior. Classification is done according to different levels: considering specific threatened DHT operations (mainly find_node/find_value and join procedures), then the malicious activities involved and the general approach of the solution.

2.2.1.1 Find node/ find value operations

The malicious activity involving this type of operations could be *incorrect routing* or *incorrect routing updates*.
As regards the first one all the studied approaches deal with some forms of *redundancy*. The studied possible solutions are:

- *Diversity routing* [13]: use of an ID lookup trust profile (histogram representing nodes appearance frequency on the lookup path towards the ID). The advantage is that adversary nodes introducing a lot of sybils may acquire high values in the trust profile, so nodes behind them will not be used. Disadvantages are the inefficiency under normal circumstances and the absence of any progress toward the target node.

- *Mixed routing* [13]: a balance between classic routing and diversity routing. It does not perform better than the diversity strategy in case there are a lot more sybils than honest nodes.

- *Zig-zag routing* [13]: an alternation of diversity and classic routing. It outperforms classic and mixed routing when the number of malicious nodes is large but it reduces lookup speed.

- *Secure routing primitive* [14]: failure test and redundant routing are employed to ensure, with very high probability, at least one message copy arrival to each correct replica root for a key. It solves a constrained routing problem: even a few malicious nodes in the routing table can reduce the probability of successful delivery, but several attacks have been demonstrated to weaken the failure test.

- *Random routes* [15] with predefined hops number; the identity verification process is based on two random routes number. It does not provide centralization elements and trust and reputation policy are applicable due to the character of nodes relationships (social network). However routing loops can reduce routes effective length and intersections probability, a small number of attack edges is assumed, honest users may be compromised and the number of hops is a critical parameter.

As regards incorrect routing updates instead the possible solutions concern *redundancy* like:

- *2 routing tables* usages [14]: one exploiting network proximity, another exploiting a further constraint on closeness. Constrained routing tables have, in fatc, an average fraction of only few random entries that point to attacker controlled nodes. Disadvantages are: routing process overhead, the possibility for an attacker to reduce the successful delivery probability by not forwarding messages; no flexibility in neighbor selection.
- *periodic reset* [16] of optimized routing tables to constraint tables: this method includes routing-tables updates rate limitation and unpredictable ID assignment. The adversary is constrained to render more intensive its poisoning activity in order to maintain its foothold. Disadvantages are: the short-term sybil-proof effect; only Bamboo tested, the periodic reset is effective at low routing tables poisoning level, the system adaptability and routing security trade-off and the randomness server availability.

or *resource testing* like:

- *Identity legitimacy control* [17] by sending check messages at predefined time intervals on channels preassigned to each neighbor node. No centralization elements are involved but a node cannot communicate simultaneously on more than one channel.
- *node degree bounding* [18] to a certain threshold: nodes anonymously audit each other's connectivity through a certificate binding ID and a public key. This method leaves flexibility in the selection of neighbors nodes but is mainly addressed to the eclipse attack, mechanisms for maintaining auditor's anonymity are required and a localized eclipse attacks could still work.

or *trusted certification/trusted devices* like:

- *Random assignment of key-sets* [17] to each node. Sybils detection is carried out through checking the number of coincided keys in sets of different nodes. It is a distributed control mechanism but there is still the possibility to steal IDs (masquerading attacks).

2.2.1.2 Join operations

The malicious activitities cocerning join operations could be summarized into the *false ID creation*.
The possible solutions concern *resource testing* like:

- Hierarchy of cooperative *admission control* nodes based on the bootstrap graph [19]. Challenging new nodes by puzzles solving to have a globally verifiable cryptographic proof. This method provides also an upper bound on the number of IDs. With this solution targeted attacks are more difficult, malicious node removal and attack localization are easier. However, as simple Diffie-Helman is used to share a secret between nodes in the hierarchy: man-in-the-middle attack is possible. Moreover signature verification is computationally expensive and the tree root is a single point of failure.

- Requiring the prospective (joining) node to solve *crypto puzzle* [14]. It is a distributed access control but there is a costs trade-off in puzzles solving: not too expensive for legitimate nodes but sufficient to attack preventing.

- *binding nodeIDs to real-world identities* [14]. Effective in “virtual private” overlays (enterprise networks) but a Certification Authority is a single point of failure.

- *Random computational puzzles*, with incorporated distributed locally generated challenges, required to participate in the network [20]. Challenge broadcasting from each peer toward others. Locally challenges prevent puzzles to be reused by attackers over time, the method gives a node flexibility to choose its position. A disadvantage is the opposite effect when legitimate users are stuck with out-of-date hardware whereas attackers have access to high performance computing.

or *trusted certification/trusted devices* like:

- *Release of new identities by Certification Authorities* [1]. For the moment, it is the only way to combat definitely the sybil attack and moreover it introduces centralization elements into the system that are a single point of failure.

- ID obtaining through *double-key encryption* of newcomer's IP address by a central agent that returns ID via SMS [21]. This method grants a significant reduction of the sybils number; however there is a single point of failure, different SIMs use is possible, there are PKI elements, it requires a IP addresses and phone numbers list, there is the need of central agents to avoid multiple IDs issue to the same peer.

- *Registration* through a centralized authority [17], with a list of trusted identities. This method grants a secure bootstrap, but it also introduces a single point of failure, centralization elements, and there is the possibility of stolen IDs.

- *Central trusted certification authority* assigning and signing nodeId certificates that binds ID to a public key and to the IP address [14]. Randomly chosen nodeIDs and ID forging are prevented, binding IDs and IP addresses renders harder for an attacker to move ID across the nodes. There is, however, a single point of failure and if an IP address is changed, the corresponding nodeId and certificate become invalid.

- *Periodical invalidation* of node IDs by a trusted entity that broadcasts a different initialization vector for some hash computations [14]. It is hard for an attacker to accumulate many nodeIDs over time and to reuse them, but legitimate nodes must periodically spend additional time to maintain their membership.

- New node's *ID* (IP address and port hash) *registration by other nodes* chosen according to IP address prefix [22]. This method provides also a maximum number of nodes per participant bound. There is an external ID usage, and no centralization elements, no need to obtain new separate IDs and it is easily applicable for other DHT than Chord. Anyway distributed ID assignment is Sybil-proof only for a limited time, there can be only one node behind a NAT and in theory nodes can obtain a huge number of IPv6 addresses.

or *costs and fees* methods like:

- Certificates binding the ID to a public key that must be paid [14]. The cost of an attack grows with the size of the network, moreover fees are supposed to fund the operations of a Certification Authority. However, as already said, there is a single point of failure.

Chapter 3

Analysis of Sybils Effects on Kademlia and Possible Countermeasures

In this chapter a detailed campaign of simulations is carried out to study Kademlia behavior in presence of bogus nodes with various malicious behaviors. Moreover trust and reputation techniques are studied to be applied to Kademlia in order to find a method to decrease the negative effects of the malicious nodes on the lookup procedures. For the simulations we deployed DEUS [23], a discrete event simulator, where simulation time is sampled in virtual seconds (vs), developed in Java language by the Distributed System Group of the Information Engineering Department of University of Parma. In particular we deployed its embedded implemented Kademlia model using XML simulation scripts.

3.1 Simulations conditions

We worked in conditions of steady state network, so that despite node joining and leaving, a constant number of nodes is always present in the network, this is obtained starting from 11000 vs (virtual seconds). Particularly the number of good nodes, that is, nodes that behave according to the standard Kademlia algorithm, are always *100*, whereas the number of bogus nodes, named *sybils*, was variable from simulation to simulation: *0*, *2*, *10*, *50*, *100*, *200* and *400* nodes. Sybils may have two different behaviors during a value lookup: they can answer back with an empty list or they can respond with a list made of randomly chosen peers.

Nodes join the network or leave the network till 11000vs time is elapsed according respectively to a periodic process of period 100 for good nodes and variable period for bogus nodes. Other parameters of the Kademlia network regard α equal to 1 and k equal to 20 as suggested in [5] and the key space size is set to 2000. In the simulation model of Kademlia another parameter is present, namely *discoveryMaxWait*, fixed to 1000 vs; this is needed to simulate nodes answering too slowly.

In our considerations we will only look at correctly ended value lookup procedures that start only since the network is steady, that is after 11000 vs; moreover we put log events at different times, namely at 50000 vs, 100000 vs, 150000 vs and 200000 vs in order to verify the influence of unended searches. As the results confirmed that the same percentage of unended searches is present at the four time of logging and the monitored metrics are about constant in time we concentrated only on the first logging time, that is 50000 vs, corresponding to 500 *real* seconds.

Finally for each simulation set of parameters an average over 100 seeds of the considered features was made.

3.2 Effects of sybil nodes

We looked at some features in order to evaluate the effects of a growing number of *sybils* into the network. They are: the percentage of filling of the k-buckets averaged over all the nodes in the network, the average number of successful lookup up proce-

dures and the average number of nodes known by a peer.
The results depicted in figures 3.1, 3.2 and 3.3 refer to sybils node responding with a null list and show the dramatical decrease of performances in all the three analyzed metrics as the number of sybils increases.

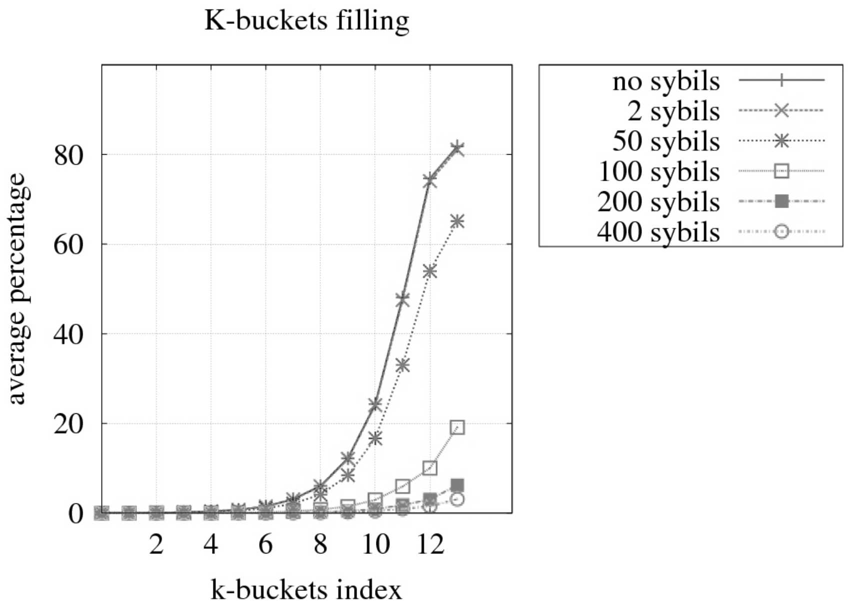


Figure 3.1: *Kbucket filling function of kbucket index.*

In figure 3.4 instead we studied the difference in performance degradation between two different malicious behaviors in a value look up procedure, namely the cases in which sybils answer lookup queries with a null or a random list of nodes. How it is possible to see from the figure the difference is very slight.

This can be explained considering the implementation of Kademlia we had. At first the querying node has a local *temporary list* filled with the k nodes nearest to the resource to find at its knowledge. At each iterative step it receives from the node it queried (only one as we concentrated on the case $\alpha = 1$) at most k responses updating the local list with the nodes it did not have before. If the FIRST node changes the

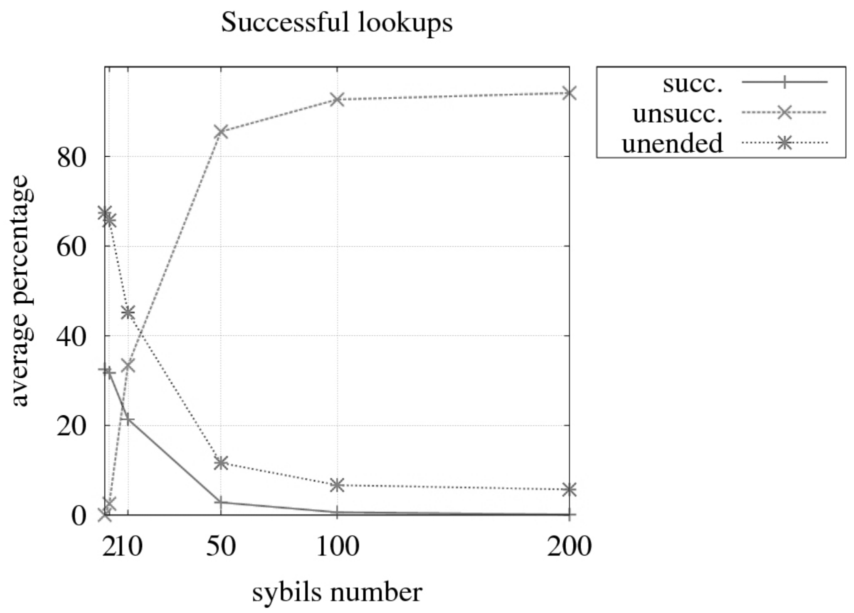


Figure 3.2: *Successfull, unsuccessfull and unended lookup procedures.*

initiator goes on querying other α nodes among the ones not already queried in the local list, otherwise it proceeds to query k nodes in the same list. If also in this case the first node does not change and no nodes answered with the searched value the lookup fails, otherwise the procedure proceeds querying α nodes as described above. The procedure ends successfully if a node responds with the searched resource.

In case a node answers with a null list the FIRST node of the temporary list does not surely update (we remember that we consider the case $\alpha = 1$) so this behavior forces to go in the searching k nodes phase. From this point the procedure, as described above, could stop or proceed with only one step more than if no malicious nodes where found on the path.

The same happens if we consider a malicious node answering with a random list even if in theory there could be the possibility the local list acquires anyway a first node. This could explain why the graphs in Figure 3.4 are very similar.

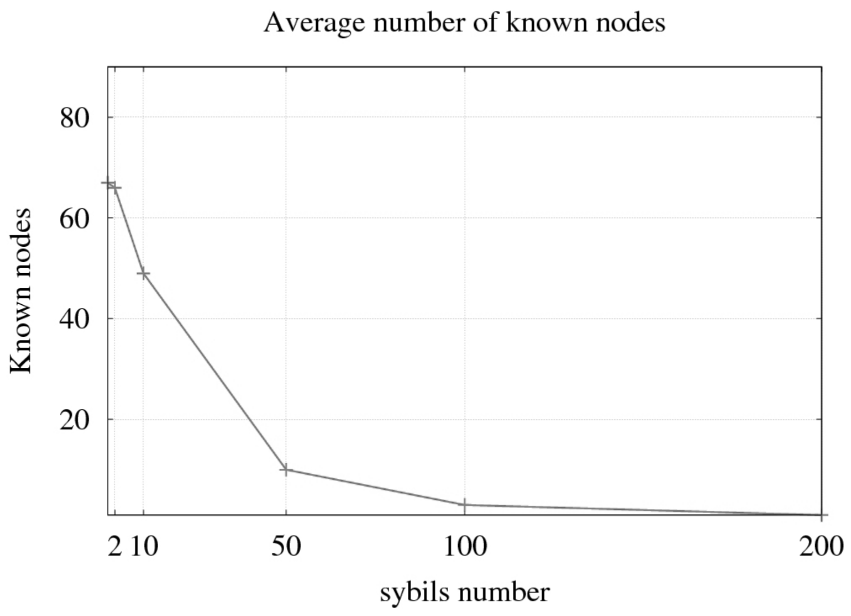


Figure 3.3: *Average known nodes function of number of sybils.*

After this considerations we addressed towards the research of some solutions, or better, some ways to soothe away the effects of malicious nodes on the lookup procedures, this can be seen as a countermeasure at routing level.
With this aim in mind we addressed to the study of the current solutions presented in literature concerning trust and reputation, this seems a suitable technique in P2P networks, as, like in any human community, nodes interact with each other, create new contacts, and progressively gain their own experience and reputation about each other. These factors can help them to evaluate trustworthiness of other nodes and to understand what kind of behaviour can be expected from a certain node.

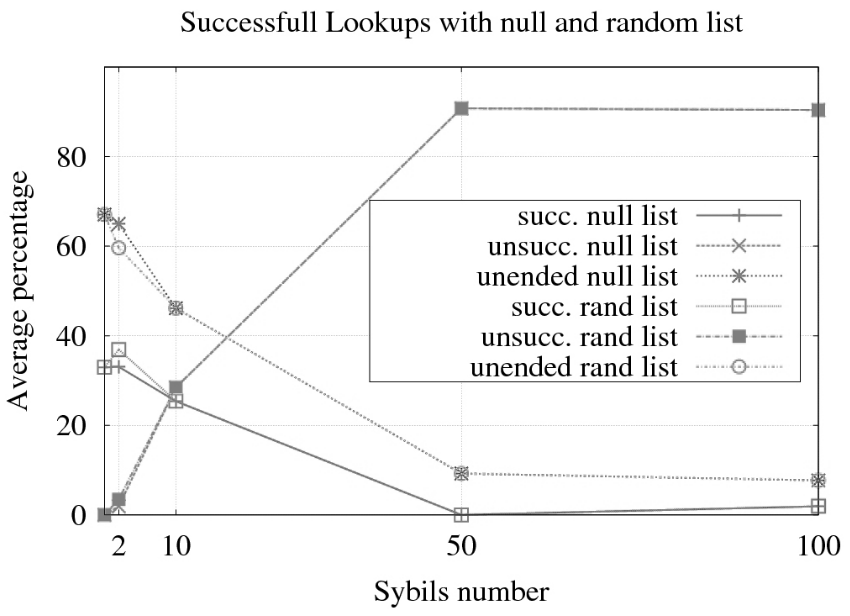


Figure 3.4: *Comparison between two malicious behaviors.*

3.3 Reputation in P2P networks

We made an analysis of different recent trust and reputation techniques and schemes [24][25][26][27][28][29][30][31] according to the four point depicted in the graph in figure 3.5, namely the model and its factors, the threats faced and the metrics to evaluate the model.

3.3.1 Model

As regards the model of trust and reputation, this can present various basic features:

- direct experience or recommendations;
- credibility of peers, that can be considered in peer reccomandations;
- reputation of shared resources, that may come before the evaluation of peers;

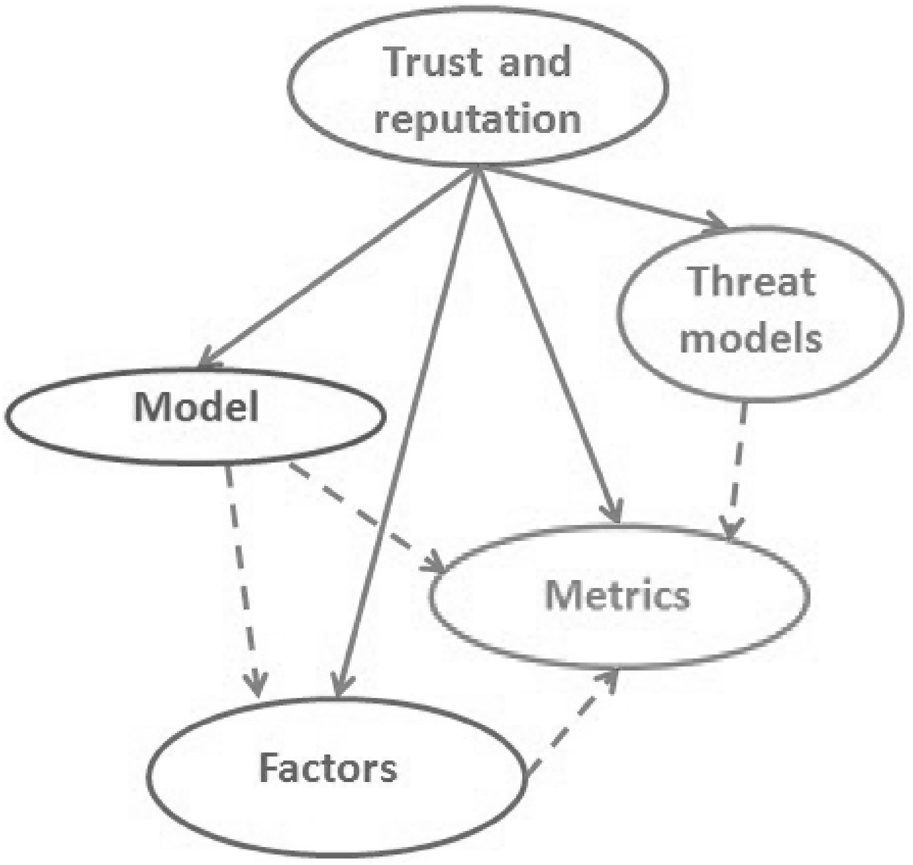


Figure 3.5: *Trust and reputation factors.*

- presence of a few pre-trusted peers;
- indipendence of peer reputation from the willingness of cooperation;
- limited sources of reputation like a few most trusted peers or neighbors;
- evolution of links according to experiences.

3.3.2 Factors

On the second point, trust and reputation can be computed acording to different factors like:

- the satisfaction of a transaction;
- the number and "size" of the interactions;
- the "age" of a transaction, that is a *time factor* weighing the trust according to a decreasing function like in figure 3.6;
- the similarity of interests of the evaluated peers;
- a punishment factor or debit for "negative" actions;
- the slowness in trust increase for good actions and the sharp decrease for malicious actions;
- the environment's risk.

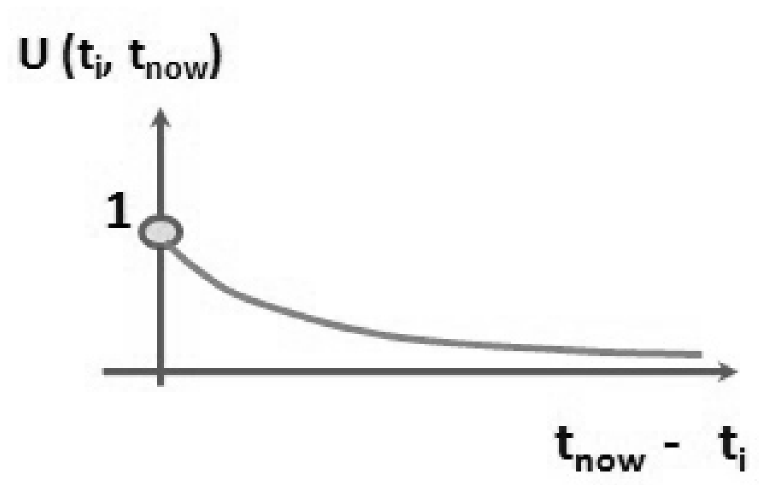


Figure 3.6: *Time factor. t_{now} is current time whereas t_i time of i^{th} interaction.*

A *trust score* is also often used in order to combine in some way the previous factors, a simpler ratio between good actions and total actions of a peer may be alternatively used. The specific implementation, however, depends upon specific threat models or proposed solutions.

3.3.3 Threat models

The opportune trust model to use depends also on the environment on which this is deployed and the threats it has to face. There can be in fact scenarios in which:

1. malicious peers always provide inauthentic resources and give high trust values to any other MP;
2. bogus nodes always behave badly but collude with only a limited set of collaborators peers;
3. bad nodes provide inauthentic resources only in a certain percentage of cases;
4. part of the malicious peers behaves like in the second case, part, the *spies*, answer correctly to a certain percentage of queries but increase trust only of standard malicious peers. This creates a division in the network as shown in figure 3.7;
5. bad peers downgrade trust of good peers/neighbours;
6. bogus nodes leave the network after a bad behavior re-entering with a new identity (*smart churn*);
7. *spies* assign positive trust values only to colluding malicious peers, whereas these ones give trust also to good peers. This renders them more difficult to discover, and it is even more dificult if malicious peers provide inauthentic resources only in a certain percentage of cases.

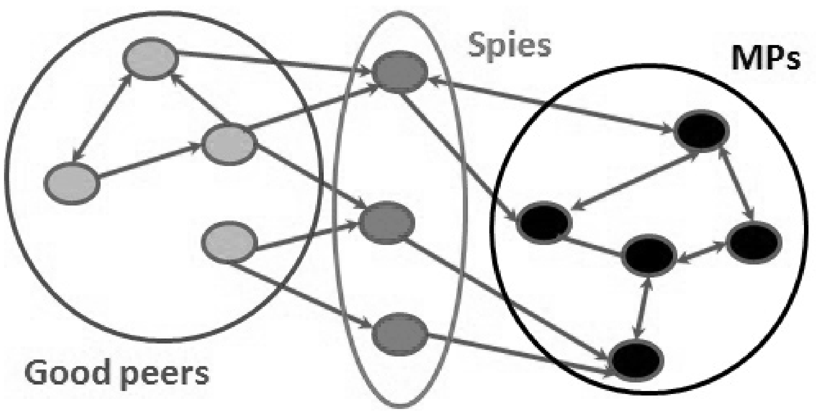


Figure 3.7: *Spies in a p2p network.*

3.3.4 Metrics

Also the metrics useful to evaluate and judge the goodness of a trust model may vary according to the model itself and the threats it is called to cope with. They however can be quickly summarized as:

- reputation of certain good peers VS number of peers in the network, may be varying the percentage of malicious peers and the credibility factor;
- fraction of inauthentic provided resources VS fraction of malicious peers in the network or number of total interactions;
- comparison among different threat models and eventually file and peer reputation;
- comparison between trust model and trust and reputation model.

3.3.5 Studied solution

Our approach consisted in taking into account the trust value when ordering the temporary local list of the initiator peer. Since the use of a brute force ordering of such list just on the basis of trust could not work (also demonstrated by our simulations), we tried to define a new metric for the the DHT distance function, called "new distance" (nd), and computed, in a first approximation, according to the following equation:

$$nd = od \times b + (1-b) \times \frac{1}{t} \tag{3.1}$$

where *nd* is the new distance, *od* the old distance computed according to the XOR operation as in standard Kademlia, *b* a balancing factor ranging from 0 to 1 and *t* the trust factor ranging from a very small number near zero (say ε) and infinite. We plotted the function in figure 3.8 with different values of b.
The trust increases for all nodes that led to a right resource in a lookup procedure whereas it decreases for all nodes involved in an unsuccessful lookup. The increase or decrease quantities can be varied according to what one want to study.
Future works may be carried out in order to better study and determine the optimum formula for the new distance metric, and the best value of the possible *b* coefficient that balances the weight of trust respect to the "old" distance value.

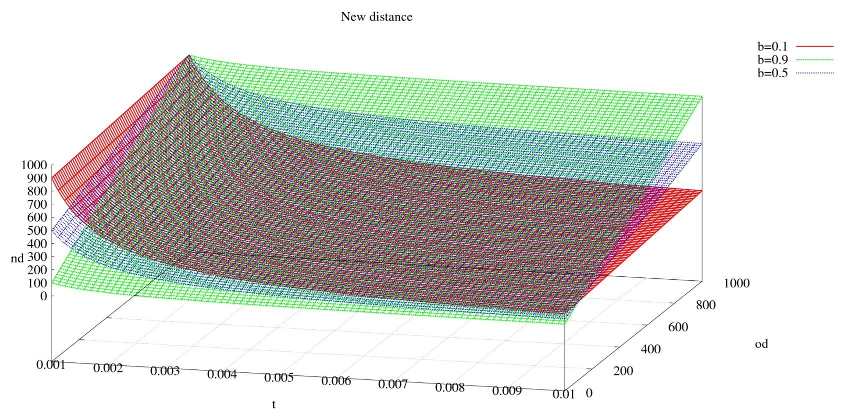


Figure 3.8: *New distance function.*

Chapter 4

Security issues and security protocols at application level

When two peers want to establish a secure end-to-end communication channel, they need to have previously established a so called *Security Association*. A Security Association (SA) is a logical connection between two or more peers that specifies all algorithms, parameters, and key materials needed for providing security services (e.g. authentication, integrity protection, confidentiality) to the traffic exchanged between the peers. Usually a SA is defined as a simplex connection (i.e. unidirectional), and two different SA are required for bidirectional communications.
Such a SA can be, moreover, statically setup through a manual configuration or can be (more conveniently) dynamically established by a proper SA management protocol.

Usually secure communication protocols such as IPSec [32], TLS [33], SSH [34], etc. have their own mechanism in order to establish a SA between two endpoints. Some protocols use some complementary SA management protocols before starting the actual secure communication, while other protocols perform SA establishment

during the connection setup phase at the beginning of the communication.
During the SA setup the two peers have to negotiate and agree on the security services (authentication, confidentiality, etc.), security algorithms (encryption/decryption, message authentication, hash functions), parameters (e.g. initialization vectors) and public or secret keys or digital certificates. Keying material is an important component of the SA and it is usually obtained by a one-way pseudo-random function (PRF) applied to a master or pre-master key. In turn, the agreement on such a master key is one of the crucial aspects of the SA establishment and requires the generation of such a key at the two end points (e.g. through Diffie-Hellman) or its secure transmission over an insecure channel. In both cases such a master key agreement should be authenticated someway. e.g.: through a pre-shared key or signed through a private-key and verified through a public key (or the corresponding digital certificate). However in both cases (use of secret pre-shared key or private/public key) both parties have to already agree on some shared keys, secret or public. Unfortunately this is a very strong assumption when applied to a peer-to-peer scenario in which peers want to communicate with each other without any pre-established relationship. Digital certificates, Certification Authorities (CAs), and Public Key Infrastructure (PKI) are also often used in order to overcome the lack of a directly pre-established relationship between peers; however this approach requires a sort of trust delegation between peers and CAs (or between peers in case of user-signed certificates like in PGP [35]).

In our research work we firstly analyzed the current key agreement protocol both standardized and presented in literature and the we proposed a method for overcoming the problem of lack of a prior shared secret or a PKI for SA establishment for P2P multimedia communications.

4.1 Current Security Protocols Comparison

The main used approaches in the scientific literature for key-agreement on a master key are:

1. *pre-shared key derived* - the master key is derived from a pre-shared key and it is influenced by random values and generated and exchanged by both parties in order to avoid the reusing of previously generated keys. The main problem of this approach is that it provides a previous relationship between the two parties;

2. *secret key or public key encryption* - the master key is generated by one peer and transmitted to the other party encrypted through a shared secret key or a public key. The main drawback is that both parties still have to previously agree on a pre-shared secret key or on the receiver's public key. In the latter case the initiator need to own the responder's public key or the responder's digital certificate or the certificate of the Certification Authority that signed the responder's certificate;

3. *Diffie-Hellman* - the master key is generated through the Diffie-Hellman (DH) exponential key exchange, in which two parties jointly exponentiate a generator with random numbers. In such a way an eavesdropper cannot guess what the key is. The main disadvantage of this approach is that it does not guarantee protection against a MITM attack, that is a third party tries to trick both parties forcing them to agree to two different keys shared with itself. To prevent MITM attacks authentication is needed, but this in turn requires a pre-shared secret or public/private key pair in order to provide and verify a MAC (message authentication code) or a digital signature, so the problem still arises.

In the following various key agreement protocols are analyzed in order to figure out a common authentication scheme from which to develop a new proposal.

4.1.1 General data exchange security protocols

In this category we briefly describe the key agreement features of protocols of various layers with interesting features of authentication and key agreement.

4.1.1.1 IPSec

IPSec [32] grants security at network level establishing unidirectional Security Associations (SA) between two parties.
It employs two security protocols: Authentication Header (AH) and Encapsulating Security Payload (ESP).

The first one grants the authenticity of the packets source, their integrity and eventually protection against replay attacks. The latter is achieved through a Sequence Number in the AH, whereas the first two ones are accomplished computing an Integrity Check Value (ICV) with a double MD5 hash with secret pre-shared key over the whole IP packet except for the mutable fields that are set to zero before the computation.
The second one grants confidentiality through data encryption and optionally integrity, authentication and replay protection. They both support transport or tunnel mode, the latter protecting the whole IP original packet.

For the key management and agreement IKE, Internet Key Exchange, is used, an application level protocol using UDP as transport protocol with port 500.
IKE [36] is needed to establish a protected environment and to set up a Security Association (SA) between two parties, that is, in turn, made of protection mechanisms, protocols, keys and encryption algorithms and a shared session secret with Diffie Hellman. A SA is made of a Security Parameter Index (SPI), IP destination address and security protocol (AH or ESP) and it is unidirectional.
It is based upon three protocols: ISAKMP (Internet Security Association Key Management Protocol), the Oakley algorithm (actually an improvement of Diffie Hellmann) and SKEME. There are two pahses:

1. bidirectional IKE security association set up: needed to generate a shared secret from wich other keys will be computed. It is usually executed in MAIN MODE (4 messages in total, 2 for security parameters negotiation and 2 for identity proof).

2. unidirectional IPSec security association set up: every IPSec SA is a child SA of the IKE SA of phase one. The aim is to create new keys from the shared

secret in QUICK MODE.

There are 5 main ISAKMP [37] exchanges:

- *Basic exchange*: no identity protection because ID info are exchanged at the same time of the shared secret. The initiator sends SA payload and the Responder makes a choice, then there are other two messages with the Key Exchange (KE) payload and the AUTHentication (AUTH) payload.
- *Identity protection exchange*: two more messages compared to basic exchange, separating the KE and the authentication that so can be encrypted.
- *Authentication only exchange*: three messages, no encryption if it is not used in the second phase.
- *Aggressive exchange*: SA, KE, Auth payloads, sent altogether, no choices on a proposal.
- *Informational exchange*: only one notify or delete message.

4.1.1.2 Transport Layer Security (TLS)

TLS is a client-server oriented protocol laying between the transport and the application level using the 443 port (HTTPS). It supports server identification through the usage of certificates and optionally clients identification. It grants *confidentiality* through a secret key defined in an initial handshake (symmetric encryption like DES, RC4), authentication through asymmetric cryptography (RSA, DSS, X.509 certificates) and integrity through a transport level Message Authentication Code (MAC) check using hash functions (SHA, MD5). It is further subdivided into two sublayers: at the lower layer there is the TLS Record Protocol used for encapsulating the various higher level protocols among which its three application layer protocols (TLS Handshake, TLS Change Cipher Spec and TLS Alerts) besides obviously other application level protocols such as HTTP, SMTP, etc.

TLS Record Protocol grants compression, confidentiality through symmetric encryption and message integrity through a MAC with shared secret key added to the message before encryption. The keys for encryption, HMAC and the eventual Initializing Vector (IV) are exchanged with the Handshake protocol.

Among its three specific application protocols the most important is surely TLS Handshake that permits:

- server and optionally client identification;
- encryption and MAC algorithms negotiation;
- crypto keys negotiation.

All these operations are made before an application protocol transmits or receives its first data. TLS Handshake provides usually 4 phases:

1. establishing security capacities through a ClientHello message conveying a list of supported cipher suites (key exchange algorithm, symmetric and MAC algorithm) and a ServerHello response with the choice made by the server picked from the client's list.

2. the server sends its certificate signed by CA and sends its KeyExchange message part, eventually requiring a client's certificate. The exact meaning of the key exchange depends on the server previous choice about the cipher suite: for RSA the server sends its public key, for DH modulus p generator g and $x = g^a mod p$ are sent. Necessary if no public key is sent in the Certificate message.

3. the client eventually sends its certificate to prove its identity and its KeyExchange message, variable always according to the server previous choice.

4. change in the cipher suite message, that indicates that since now the communication is encrypted and handshake finish message. These messages have the same meaning both coming from the client and from the server.

At this point a pre-master key (PMK) is owned by both parties, the full-blown master key (MK) is derived from the PMK according to the always same pseudorandom function (PRF). The MK is used to generate key material such as client and server MAC secret, client and server write key, etc.

4.1.1.3 Secure SHell (SSH)

SSH [34] is a client/server application protocol that can be used like telnet to log into a remote machine running the ssh server process. It is a very secure protocol unlike telnet because it uses algorithms to encrypt the data stream, ensure data stream integrity and even perform authentication in a safe way.
The authentication takes place usually in two forms: the standard password based one and the public key one.

- The *default standard password authentication*: when a user logs in to a certain machine, the user is required to prompt its username and password for its account on that machine. This exchange takes place through an encrypted channel.

- *Public key authentication*. The user generates a key pair, the public key is copied in the server, the private key is held by the user and protected through a passphrase. The client prove the server to have the private key decrypting some data previously encrypted by the server with the user public key and sending back the plain data to the server. In this way the user is not requested every time to give its own password at each connection. In a similar but reciprocal way also the server can be authenticated by the client before sending the user credentials.

The required symmetric cipher is 3des-CBC, the required MAC algorithms hmac-sha1, the required key exchange are DH group1 (using Oakley group 2) and DH group14 (using Oakley group 14), whereas required public key algorithm is only ssh-dss.

Each side has a preferred algorithm in each category, and it is assumed that most implementations, at any given time, will use the same preferred algorithm. Each side may guess which algorithm the other side is using, and may send an initial key exchange packet according to the algorithm, if appropriate for the preferred method.
After this the true key exchange begins by each side sending a packet with a list of key exchange algorithms, server host keys algorithms, encryption algorithms client to server and viceversa, MAC algorithms client to server and viceversa, compression algorithms client to server and viceversa, etc. The first algorithm in each list must be the preferred one.
If both sides make the same guess, that algorithm havs to be used, otherwise the chosen algorithm is the first one satisfying the following conditions:

- the server also supports the algorithm;
- there is an encryption capable algorithm on the server host key algorithms also supported by the client if needed;
- there is a signature capable algorithm on the server host key algorithms also supported by the client if needed.

Otherwise the connection fails.

The server lists the algorithms for which it has host keys, the client lists the algorithms that it is willing to accept. There may be multiple host keys for a host, possibly with different algorithms.
The chosen to each direction encryption algorithm, MAC algorithm and compression algorithm MUST be the first algorithm on the client's name-list that is also on the server's name-list. If there is no such algorithm, both sides MUST disconnect.

After receiving the SSH_MSG_KEXINIT packet from the other side, each party will know whether their guess was right. If the other party's guess was wrong, and the first_kex_packet_follows is true, the next packet is silently ignored, and both sides act then as determined by the negotiated key exchange method. If the guess was right, key exchange continues using the guessed packet. After this exchange is performed, the key exchange algorithm is run. It may involve several packet exchanges, as specified by the key exchange method. The only following messages could be transport layer generic messages, algorithm negotiation messages, specific key exchange messages.

The key exchange produces a shared secret K and an exchange hash H from which encryption and authentication keys will be derived. Each key exchange method specifies a hash function used in the key exchange and that is used in key derivation for encryption key client to server and viceversa, integrity key from client to server and viceversa.
The key exchange ends by each side sending an SSH_MSG_NEWKEYS message. All messages sent after this message must use the new keys and algorithms.

4.1.2 Multimedia key exchange protocols

In this section we briefly summarize the main features of protocols more specifically used for key agreements in VoIP (Voice over IP) or Multimedia applications. From this protocol we will borrow most of our ideas for creating a novel solution.

4.1.2.1 Multimedia Internet KEYing (MIKEY)

MIKEY [2] is a key management protocol to support SRTP in real-time applications both for peer-to-peer and for group communications. The objectives of MiKEY are end-to-end security, simplicity, efficiency, possibility of tunneling into session establishment protocol like SDP and independency from security specific functionalities accounted by the lower transport layer protocols. MIKEY allows to produce a Data Security Association (SA) for the security protocol, e.g. SRTP, including a key to encrypt the traffic called TEK (Traffic Encryption Key) used as input for SRTP. In point of fact, with MIKEY it is possible to establish keys and parameters for more than a security protocol at the same time, that is, it is possible to create a Crypto Session Bundle (CSB), a collection of one or more Crypto Sessions sharing a key generator called, Traffic Generation Key (TGK) and a set of security parameters allowing to derive different TEKs. The following are the step to generate a TEK:

1. the two parties agree on a set of security parameters and onto a key generator TGK, useful to the CSB;

2. TGK is used to derive a TEK for each Crypto Session;
3. TEK and security protocol parameters determine a SA, that is finally delivered to the security protocol.

MiKEY can moreover be used to update TEKs and CS or other specific parameters of the cryptographic context. There are three different methods to get a TGK: pre-shared key, public key cryptography and Diffie-Hellman exchange.

In the first method the pre-shared secret key, is used to derive key material for both the encryption and the integrity protection of the MIKEY messages. The main objective of the Initiator's message (I_MESSAGE) is to transport one or more TGKs (carried into KEMAC) and a set of security parameters (SPs) to the Responder in a secure manner. The KEMAC payload contains a set of encrypted sub-payloads and a MAC. The main objective of the verification message from the Responder is to obtain mutual authentication. This is the most efficient way to handle a common secret transport, since only symmetric encryption is used and only a small amount of data has to be exchanged. However an individual key has to be shared with every single peer, so some scalability problems may arise.

In the second method the objective of the Initiator's message is always to transport one or more TGKs and a set of security parameters to the Responder in a secure manner. This is done using an envelope approach where the TGKs are encrypted (and integrity protected) with keys derived from a randomly/pseudo-randomly chosen "envelope key". The envelope key is sent to the Responder encrypted with the public key of the Responder in the PKE field. If the Responder has several public keys, the Initiator can indicate the key used in the CHASH payload. The KEMAC contains a set of encrypted TGK and a MAC, the encryption and MAC keys are derived from the envelope key. There will be one encrypted ID_i and possibly also one unencrypted ID_i in the initiator's message. The encrypted one is used together with the MAC as a countermeasure for certain man-in-the-middle attacks, while the unencrypted one is always useful for the Responder to immediately identify the Initiator.
It is possible to cache the envelope key, so that it can be used as a pre-shared key; then, the pre-shared key can be used, instead of the public keys. The disadvantage

of this method is that it is required a Public Key Infrastructure to handle the secure distribution of public keys.

The DH exchange is based upon a two messages communication as in the previous two cases (I_MSG from the communication Initiator and R_MSG from the Responder) but the responder's answer is now necessary and not optional.
Supposing that g is a predetermined generator, known both to the Initiator and the Responder through the reception of the SP payload from the Initiator; I_MSG contains a part called DH_i, that is the Initiator's value for the DH-exchange, namely g^{x_i}, with x_i randomly chosen. In the I_MESSAGE there is also a $SIGN_i$, that is a signature covering the whole I_MSG. The Responder's message contains instead a DH_r, that is g^{x_r}, with x_r always randomly chosen. Both parties then compute with the exchanged values the TGK, that is $g^{x_i x_r}$. The Initiator doesn't need the Responder's certificate before the setup, whereas it is sufficient that the Responder sends its own with the response.

Currently there are also two further methods that are improvements of the previous ones: DH-HMAC, that is a light-weight version of DH using HMAC to authenticate the two parts instead of certificates and RSA signatures; RSA-R where TGK is exchanged with public key cryptography without requiring any PKI: the Initiator sends the Responder its public key and the Responder chooses the TGK and sends it back encrypted with the public key just received.

4.1.2.2 ZRTP

ZRTP [3] is a cryptographic key-agreement protocol specifically designed to negotiate an encryption key for securing VoIP or multimedia sessions using SRTP, for providing confidentiality and authentication and protect against MITM attacks without centralization elements.
It performs a DH key exchange during call setup in-band in the Real-time Transport Protocol (RTP) media stream which has been established using some other signaling protocol such as Session Initiation Protocol (SIP). One of ZRTP's characteristics is so that it does not rely on SIP or other signaling protocols for the key management.
ZRTP does not require prior shared secrets or rely on a PKI or on CAs, since the

ephemeral DH keys are authenticated by means of the use of a Short Authentication String (SAS), which is essentially a cryptographic hash of the two DH values, to be read aloud by the two users. More precisely the two end users verbally compare the shared SAS value displayed at both ends, if both values do match, it means with high probability that the DH succeeded and no MITM attack has been performed, otherwise it indicates the presence of a MITM wiretapper. This has however to be present since the first communication and must be active for all the subsequent ones to successfully continuing its attack.
The key agreement takes place according to a message exchange that in turn can be divided into four steps [38]:

1. discovery,
2. hash commitment,
3. Diffie-Hellman exchange,
4. switch to SRTP and confirmation.

These are shown in figure 4.1. The first phase consists of the capability exchange of algorithms, parameters, etc. and is made of the first four messages.
In the second phase one of the two counterparts sends the COMMIT message containing its choice of algorithms, keys, parameters, etc. It also generates its part of the DH secret.
The third phase consists of the true DH exchange and the master key computation.
Finally, the last phase immediately exploits the just established master key and gives confirmation both parties arrived to the same key and everything works properly.

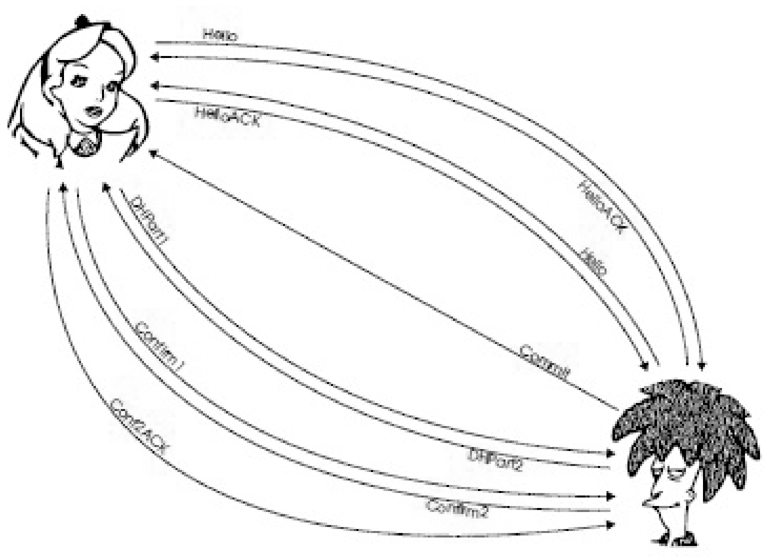


Figure 4.1: *ZRTP message exchange [38].*

4.2 Common key agreement scheme

Summarizing the instauration of a security association takes roughly the following form depicted also in Figure 4.2:

1. an initial exchange or Initiator imposition of encryption, MAC, signature algorithms;
2. an exchange of the master key through symmetric, asymmetric or DH procedures, according to what established in the previous phase;
3. an eventual verification of the two parties identities and SA establishment termination.

We now show the correspondence of these three phases into the analyzed protocols.

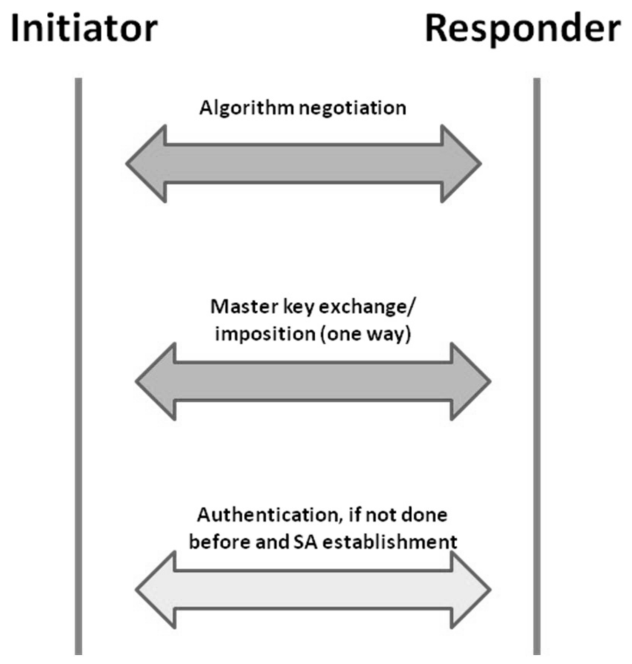

Initiator
Responder
Algorithm negotiation
Master key exchange/ imposition (one way)
Authentication, if not done before and SA establishment

Figure 4.2: *Common phases of authentication.*

4.2.1 IPSec

The three-phase negotiation takes place through the second phase of an ISAKMP exchange, application level protocol using port UDP 500. There are five exchanges types in ISAKMP [37] according to the DOI (Domain of Interpretation) in which the three basic phases of a negotiation described above can be in various way compressed till reducing to one single message exchange.

Phase 1 can be performed with the sending of one Security Association payload followed by one or more ISAKMP Proposal payload(s) and one or more Transform payload(s). In the proposal payload there is the ID of the security protocol(s) proposed (e.g.: AH, ESP, ecc.) whereas in the Transform Payload(s) there are the Inititator's supported or Responder's chosen list of hash algorithms (SHA, MD5), encryption algorithms (DES, 3DES, IDEA, etc.) or key exchange algorithms (Oakley, etc.)

The second phase takes place with the exchange of Key Exchange payloads. The content of the Key Exchange Data field depends upon the DOI (Domain of Interpretation) transported in the SA payload. The key exchange can be performed through the Oakley, DH or RSA-based algorithms.
The authentication phase can be performed using the ID payload and the Hash or the Authentication Payloads. The first one (ID) according to the specific DOI may contain IP address, IP subnet, etc. whereas the last two ones contain respectively the hash of a message's part or data from the signature function.

4.2.2 TLS

The three phases are performed using the TLS Handshake Protocol at application level.
The first one can be seen through the ClientHello and ServerHello message exchange, where the client is the initiator and the server the responder.
The second phase can be seen through the ServerKeyExchange and ClientKeyExchange message exchange.
The third phase through the Certificate messages from the Server and potentially from the Client, and through the ChangeCipherSpec and Finished messages.

4.2.3 SSH

The first phase, the negotiation, begins by each side sending a packet with a list of key exchange algorithms, server host keys algorithms, encryption algorithms client to server and viceversa, MAC algorithms client to server and viceversa, compression algorithms client to server and viceversa, etc.
The second phase, the key exchange, takes place according to the previously negotiated key exchange algorithm, usually DH.
The third phase, authentication, is performed with the help of a hash H over which the pre-negotiated signature algorithm is applied.

4.2.4 MIKEY

Here the three phases are described according to the three different methods.

- In the pre-shared key method there are only two messages exchanged, one from the initiator and one from the responder. There is not an agreement but the initiator imposes with its own message the TGK encrypted with an encryption key derived by a pre-shared secret (KEMAC payload). There is not so a phase 1 negotiation but only a one way phase 2 of key exchange or better imposition. The third phase (authentication) is performed attaching to the two messages a MAC computed using an authentication key derived always from the pre-shared secret (KEMAC payload for the Initiator or V field for the Responder).

- In the public key method the TGK is transmitted (KEMAC payload) by the initiator to the responder encrypted through an envelope key that is also contemporary transmitted to the responder encrypted with the responder's public key (PKE payload). In reality the TGK is encrypted through an encryption key derived from the envelope key as in the previous pre-shared case. From the envelope key an authentication key is also derived, it is used for the MAC payloads.
  The first negotiation phase is so avoided as in the pre-shared method, the exchange phase is the imposition by the initiator of the TGK(s), whereas the authentication phase can be seen on the initiator's side in the $SIGN_i$ (a signature covering the entire MIKEY message) and CERTIFICATE payloads, on the responder's side in the V payload, computed as in the pre-shared method.

- DH method: also here the first negotiation phase is avoided, whereas the exchange starts with the initiator's message sending its part in the DH exchange. This and the responder's answer represent the second negotiation phase. Moreover the third phase, the authentication, is performed always with this only aforementioned exchange; it is preformed using the SIGN and CERTIFICATE payloads from both parties.

All the aforementioned cases avoid the first negotiation phase supposing the responder always support the initiator's intentions, this is because MIKEY is not intended to have a broad variety of options, as it is assumed that a denied offer should rarely occur. In case this does happen the responder can together with an error message (indicating that it does not support the parameters), send back its own capabilities (negotiation phase) to let the Initiator choose a common set of parameters. This is done by including one or more security policy payloads in the error message sent in response.

Chapter 5

A New Key Agreement Protocol for P2P VoIP Applications

Of the previously described protocols, we mainly focalized upon MIKEY and ZRTP, as the most suitable for peer-to-peer multimedia communications. Even if they are both by now widely adopted, they however have some disadvantages:

- In ZRTP, it is possible for an attacker to force participants to use SAS authentication also in case they have shared secrets, by convincing the peers that they have lost such secrets [39]. In fact, according to the protocol specification, if the set of shared secrets does not match or is empty, the procedure may proceed with SAS. This may be more risky if SAS authentication cannot be performed for lack of display-equipped devices or if the two parties do not know each other voices. SAS authentication is also vulnerable to some types of voice forgery attacks [3], such as Bill- Clinton attack, six-month attack, etc. that require a lot of computational resources but that in theory are still possible.
- An obvious disadvantage of MIKEY is that it requires either prior shared secrets, or a separate PKI, for authentication, with all attendant problems such

as certificate dispersal, revocation, and so on. Moreover it does not grant key secrecy at all, as this would require that the key is undistinguishable from a random bit-string, whereas the DH result is directly used as session key. A simple application of a deterministic hash function to the joint Diffie- Hellman value does not provably produce an output which is indistinguishable from random. By contrast, in protocols like TLS and IKE, the key is derived by hashing the Diffie-Hellman value together with some (authenticated) random values generated by one or both participants.

In the following we propose a new key agreement protocol, inspiring to MIKEY and ZRTP but alternative to the ones described in the previous chapter. During our research work we developed three schemes of this protocol, the first one with no certificates, the second one with certificates and certificates management, the third one making use also of a Distributed Hash Table (DHT).

5.1 First Scheme: no certificates

The objective of the proposed protocol, in its first implementation [40][41] is to securely establish a master key between two multimedia SIP UAs that may or may not have already communicated with each other.
The proposed protocol, in this first scheme, has been designed in such a way that it does not rely on any PKI, user certificates or public keys, or pre-shared secret keys.

At the beginning of a new multimedia session, two UAs have to setup a new master key. This is done through DH exchange. In order to protect such a key against MITM attacks the DH exchange is authenticated by means of the vocal reading of a Short Authentication String (SAS), hash-generated from the master key, like in ZRTP. Actually the SAS authentication is used only the first time the two UAs try to initiate a session, as successive attempts (actually the corresponding master keys) will be authenticated with the previously established master keys in a security-chain way.

The proposed approach is similar to the one used by the ZRTP protocol, however the two methods differ in some aspects and in particular:

1. in the information exchanged during the key setup,

2. in the way that such information is effectively encapsulated and exchanged.

Particularly, ZRTP establishes a new master key directly at media level, by using the RTP protocol as transport support for the key negotiation. Instead, our solution uses MIKEY as negotiation protocol, opportunely encapsulated within SIP messages used for session setup.
In order to support a full authenticated DH exchange, the MIKEY protocol has been extended to consider a *MIKEY 3-way handshake* (MIKEY originally supported DH in a 2-way request/response transaction). The standard MIKEY message format was not modified, whereas the available payloads have been extended in order to support new fields and value types.

The new *offer/answer/confirm handshake* between the initiator (the caller) and the responder (the callee) is depicted in figure 5.1.

The initiator starts the procedure by sending a MIKEY offer message including:

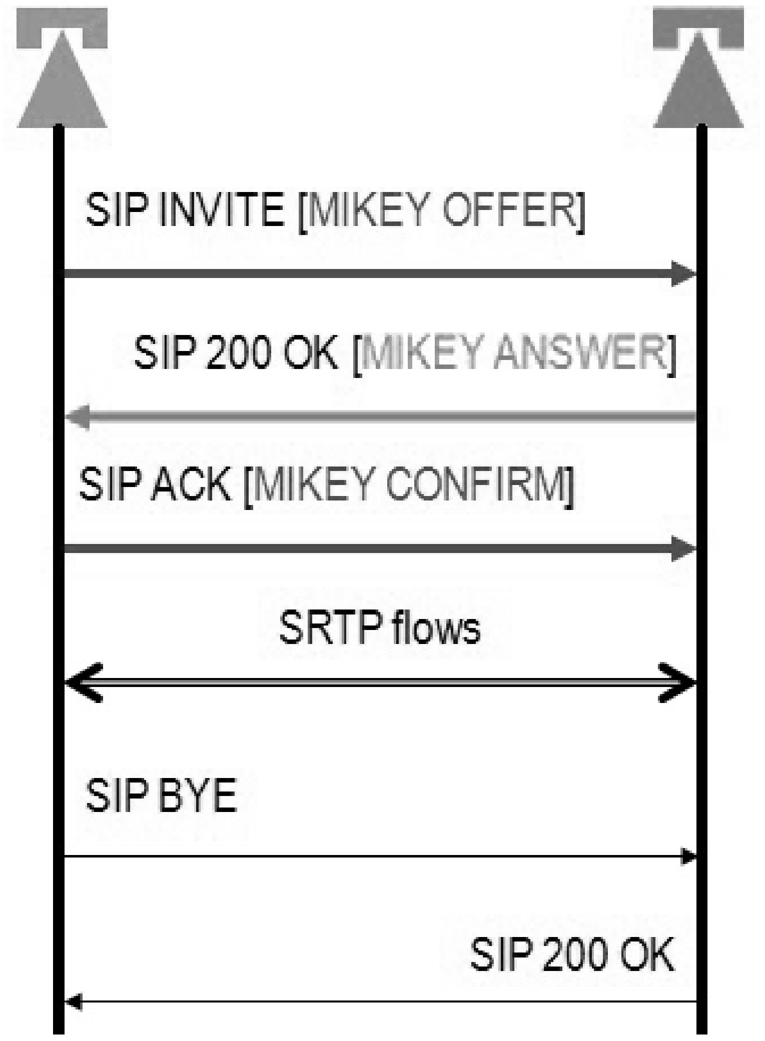


Figure 5.1: *SIP/MIKEY Session setup.*

1. a MIKEY header (HDR),

2. identities of both the initiator (ID_i) and responder (ID_r),

3. a random value ($RAND_i$),

4. the list of offered encryption and hash algorithms (SA_i),

5. the DH part of the initiator (DH_i),

6. a list of HMAC of the last five retained secrets (i.e. the last previously established master keys) ($RS = rs_1, rs_2, rs_3, rs_4, rs_5$).

The retained secrets are computed as follows:

$$rs_j = HMAC(MK_j, \, "RetainedSecret") \tag{5.1}$$

If no previous sessions have been already setup between the two UAs, such list is empty and SAS verification is requested in order to authenticate the DH exchange.

Once the responder receives the MIKEY offer message, it controls that the retained secrets match the list stored at its own side: in case it succeeds, such a list is further used to authenticate the DH exchange, otherwise the SAS authentication will be used.
The responder calculates the new master key MK_0 as the hash function of the result of the DH algorithm (DHres) concatenated with other values as follows:

$$MK_0 = hash(DH_{res}||ID_i||ID_r||RAND_i||RAND_r||MK_1||..||MK_5) \tag{5.2}$$

and replies with a MIKEY answer message similar to the offer, in which the RS list is replaced by a $HMAC_r$ calculated on the MIKEY answer message without the $HMAC_r$ field:

$$HMAC_r = HMAC(MK_0, \, MIKEYanswer) \tag{5.3}$$

In case the SAS is required, it is calculated as the 32 most significant bits of:

$$sas\_hash = HMAC(MK_0, \, "SAS") \tag{5.4}$$

Once the initiator receives the MIKEY answer message, it checks the correctness of the $HMAC_r$ and, if it succeeds, it sends a MIKEY confirm message including a

$HMAC_i$ field for authenticating the original offer and confirming the correctness of the new master key.

$$HMAC_i = HMAC(MK_0, MIKEYoffer) \tag{5.5}$$

The complete *offer/answer/confirm* transaction between initiator (caller) and responder (callee) is depicted in figure 5.2.

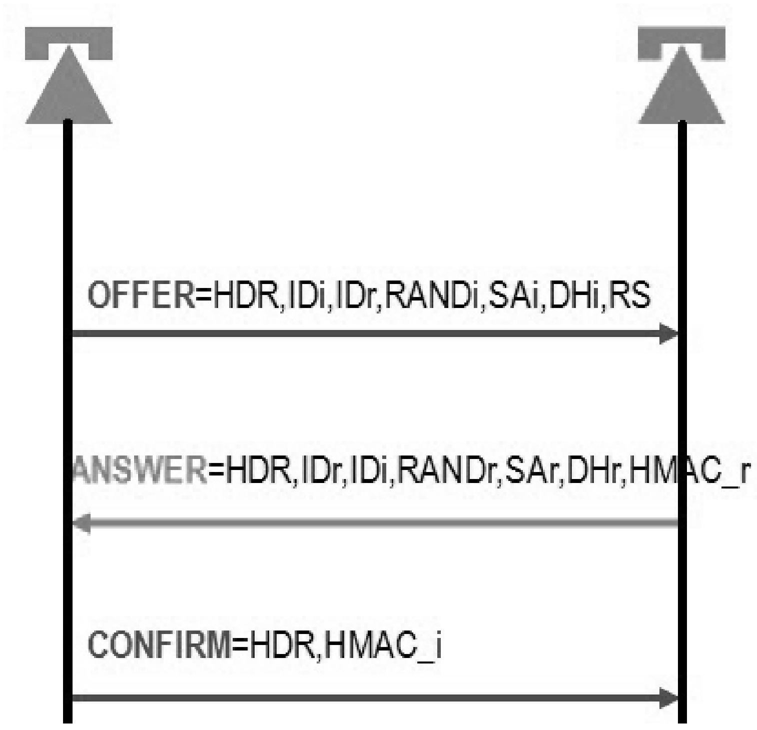

OFFER=HDR,IDi,IDr,RANDi,SAi,DHi,RS
ANSWER=HDR,IDr,IDi,RANDr,SAr,DHr,HMAC_r
CONFIRM=HDR,HMAC_i

Figure 5.2: *MIKEY offer/answer/confirm transaction.*

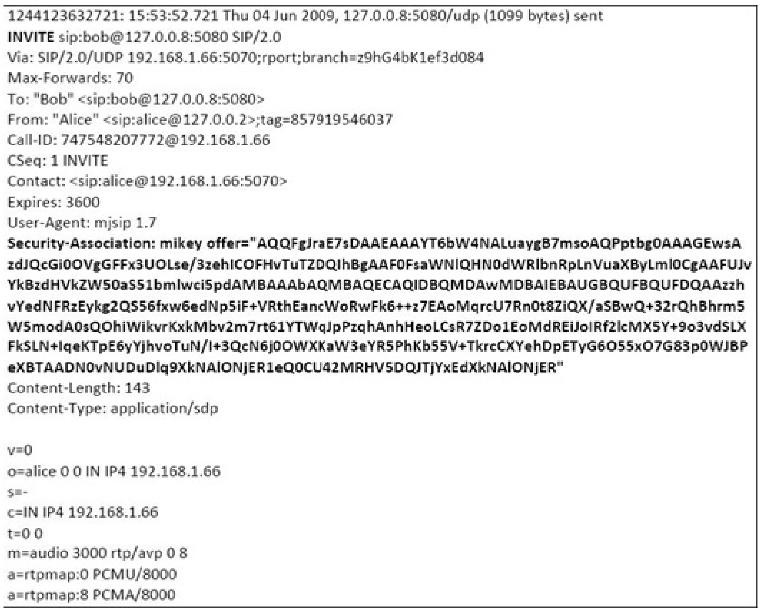
If the SAS authentication is ever necessary, both users are invited to read the SAS aloud. If the SAS, showed at both UA sides, matches and each one recognizes the counterpart voice, the new master key is considered secure and is saved in order to be used for authenticating successive key agreements between the same pair of UAs.

This protocol has been further integrated into the SIP [42] INVITE procedure: when two UAs want to securely establish new media sessions, e.g.: a VoIP call, they starts a new SIP INVITE transaction comprising the MIKEY *offer/answer/confirm* exchange for key agreement.
In order to provide key agreement functionality we extended the SIP protocol by simply adding a new header field and mapping the MIKEY *offer/answer/confirm* exchange onto the standard SIP INVITE 3-way handshake. The main motivations for integrating the key management with SIP session setup were:

- the possibility to negotiate at the beginning of a session the security credential for any desired media flows, accordingly to what is done for other media parameters (media types, codecs, transport ports, etc.);

- the possibility to reuse the same established master key for any successive media flows (no re-key negotiation is needed).

The MIKEY messages are exchanged inside SIP messages through a newly defined header fiel we named "Security-Association", in a similar way to that defined in [43]. Our new header field is embedded into the SIP INVITE (carrying the MIKEY offer), 200 OK (MIKEY answer) and ACK (MIKEY confirm) messages, in order to implement the complete key management protocol.
The content of the new header field varies according to the SIP message: in the INVITE message it includes the value *mikey offer*= followed by the *base64* encoding of the MIKEY offer message. The same happens in the 200 OK and ACK messages with the respective only difference of the initial parameters *answer*= and *confirm*=. In figure 5.3 an INVITE message with embedded MIKEY offer is shown, whereas a screenshot of the implemented User Agent is depicted in figure 5.4. The protocol has been completely developed in Java and mapped onto the SIP signaling according to the specifications provided in the previous lines. The implementation is based on the open source MjSip stack [44] that is a complete Java-based implementation of the layered SIP stack architecture as defined by RFC 3261 [42].



```
1244123632721: 15:53:52.721 Thu 04 Jun 2009, 127.0.0.8:5080/udp (1099 bytes) sent
INVITE sip:bob@127.0.0.8:5080 SIP/2.0
Via: SIP/2.0/UDP 192.168.1.66:5070;rport;branch=z9hG4bK1ef3d084
Max-Forwards: 70
To: "Bob" <sip:bob@127.0.0.8:5080>
From: "Alice" <sip:alice@127.0.0.2>;tag=857919546037
Call-ID: 747548207772@192.168.1.66
CSeq: 1 INVITE
Contact: <sip:alice@192.168.1.66:5070>
Expires: 3600
User-Agent: mjsip 1.7
Security-Association: mikey offer="AQQFgJraE7sDAAEAAAYT6bW4NALuaygB7msoAQPptbg0AAAGEwsA
zdJQcGi0OVgGFFx3UOLse/3zehlCOFHvTuTZDQlhBgAAF0FsaWNlQHN0dWRlbnRpLnVuaXByLml0CgAAFUJv
YkBzdHVkZW50aS51bmlwci5pdAMBAAAbAQMBAQECAQIDBQMDAwMDBAIEBAUGBQUFBQUFDQAAzzh
vYedNFRzEykg2QS56fxw6edNp5iF+VRthEancWoRwFk6++z7EAoMqrcU7Rn0t8ZiQX/aSBwQ+32rQhBhrm5
W5modA0sQOhiWikvrKxkMbv2m7rt61YTWqJpPzqhAnhHeoLCsR7ZDo1EoMdREiJoIRf2lcMX5Y+9o3vdSLX
FkSLN+IqeKTpE6yYjhvoTuN/I+3QcN6j0OWXKaW3eYR5PhKb55V+TkrcCXYehDpETyG6O55xO7G83p0WJBP
eXBTAADN0vNUDuDlq9XkNAlONjER1eQ0CU42MRHV5DQJTjYxEdXkNAlONjER"
Content-Length: 143
Content-Type: application/sdp

v=0
o=alice 0 0 IN IP4 192.168.1.66
s=-
c=IN IP4 192.168.1.66
t=0 0
m=audio 3000 rtp/avp 0 8
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
```

Figure 5.3: *SIP INVITE message including MIKEY offer.*

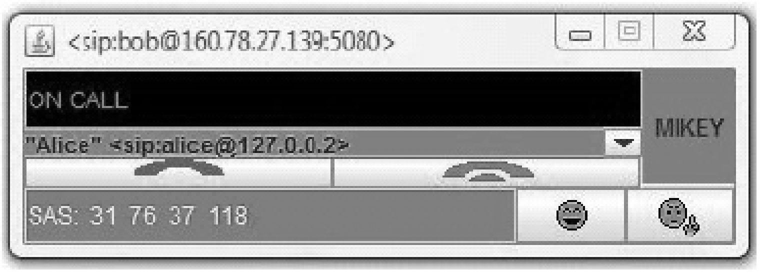


Figure 5.4: *Screenshot of the developed secure mjUA.*

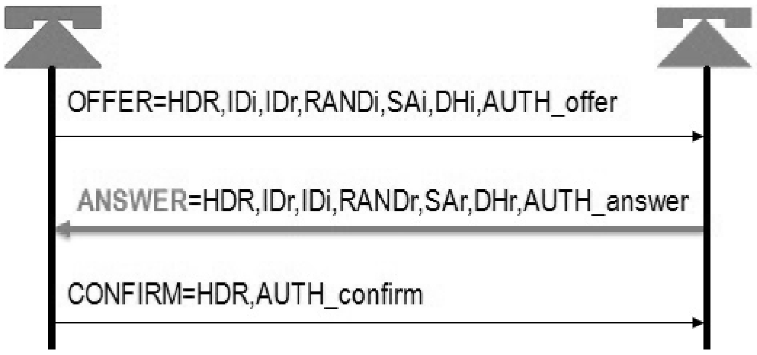

OFFER=HDR,IDi,IDr,RANDi,SAi,DHi,AUTH_offer
ANSWER=HDR,IDr,IDi,RANDr,SAr,DHr,AUTH_answer
CONFIRM=HDR,AUTH_confirm

Figure 5.5: *Three-way key agreement protocol, version two.*

5.2 Second Scheme: with certificates and their management

In its second implementation our proposal still bases upon MIKEY and ZRTP authentication both integrated into SIP, but it provides also other mechanisms for the master key authentication that include the use of: previously established session master keys (as in the first implementation), a pre-shared static secret, private-public keys and digital certificates. In case certificates are used our protocol does not rely upon a centralized scheme like a PKI, but on an architecture of “web of trust” where certificate signatures and public keys are exchanged between trusted peers.

The new three-way handshake is depicted in figure 5.5.

The $AUTH_{offer}$ is particularized depending on the selected authentication method. In case master keys from previously established sessions are used for authentication, the $AUTH_{offer}$ is formed by two values RS1 and RS2 (retained secrets) respectively obtained directly by the last two previously established master keys MK1 and MK2, in the same way as in equation 5.1:

$$RS_j = HMAC(MK_j, "RetainedSecret") \tag{5.6}$$

Such RS_js are used when computing the actual authentication field in the ANSWER and CONFIRM messages.
The reason of sending two retained secrets is to face the case when in a previous set up, one of the two parties succeeded into computing the correct master key and the

other one does not (e.g. caused by a fatal interruption during the session set up). We also thought that for this aim two secrets are enough instead of the five secrets of the first implementation.

In case private-public keys are used for authentication, the $AUTH_{offer}$ simply contains the certificate of the initiator ($CERT_i$) with the signatures of all peers that decided to trust the initiator (by signing its name and public key through their own private keys).

In case a pre-shared key (PSK) or passphrase are used, or in case of SAS-based authentication, the $AUTH_{offer}$ field contains no data, and it is left void. SAS or certificate-based authentication methods are used only if no previous secrets have been established and saved.

In the same way the list of master keys of equation 5.2 is in this version replaced by MK_j that is the previous master key corresponding to the more recent RS_j that matches one of the local stored retained secrets; if both given retained secrets do not match the locally stored ones, or no RS has been given at all, MK_j is left null.

The $AUTH_{answer}$, appended to the ANSWER message, depends on the type of authentication that is performed as well:

- if $AUTH_{offer}$ contained retained secrets, the $AUTH_{answer}$ includes the matching RS_j and a $HMAC_r$ computed according to equation 5.3;
- if $AUTH_{offer}$ contained a certificate that can be verified with some of local stored public keys (corresponding to other peers that may have signed the initiator's certificate), the responder composes an $AUTH_{answer}$ formed by the $HMAC_r$ signed by his private key;
- if $AUTH_{offer}$ was empty and if a pre-shared key PSK is available, the responder composes an $AUTH_{answer}$ formed by the $HMAC_r$ encrypted with the PSK;
- otherwise $AUTH_{answer}$ is simply formed by the $HMAC_r$, and SAS authentication is performed successively.

Finally the Initiator last message is a MIKEY CONFIRM message including a $AUTH_{confirm}$ built with the same rules used by the responder. The $AUTH_{confirm}$ in turn includes a $HMAC_i$ that authenticate the original MIKEY OFFER with the new master key, calculated as in equation 5.5.

5.2.1 Public Key Management and Verification

In case no previously retained secrets are available, peer's certificates are used for authenticating the DH exchange. Such authentication in turn is based on the capacity of each peer to verify the digital signature of the MIKEY message sent by the other peer, computed by such a peer by encrypting the HMAC of the message with his own private key.
In order to correctly verify such a signature, each peer should have (and trust) the public key of the other peer. Such a public key is also obtained during the key agreement procedure within the OFFER and ANSWER messages. However in order to properly use such a certificate-embedded public key the peer must verify at least one of the signatures, attached to the certificate, performed by all entities that decided to trust the given peer and sign his certificate.
This mechanism corresponds to the well-known Web of Trust approach used in other public key distributed architectures (e.g. PGP [35]). In order to fully benefit of the advantages of such an approach, a mechanism through which a peer can:

- obtain the public key (actually the certificate) of another peer (with all available signatures);
- ask another peer to sign (and trust) its certificate,

is needed.

We chose to follow a very simple and secure approach through which each peer may obtain public keys and signatures only from peers with whom it has already established a SA.
This approach has the following limitation that we will try to overcome in the third implementation: a node cannot setup a trusted connection with another node at distance longer than two steps apart from it. This is because for verifying a node's certificate, an entity needs the public key of one of the nodes that are one hop far from

the target node. For example, referring to Figure 5.6 node *I* may establish a SA with *D* (by verifying *D*'s certificate with the public key of *C*), but it cannot establish a SA with *K*.

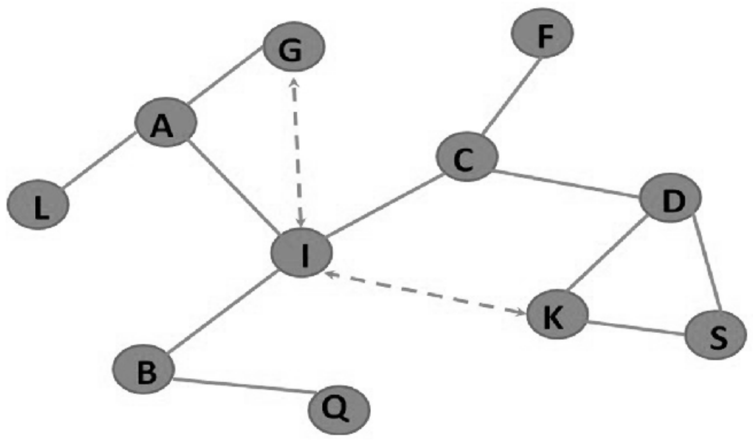


Figure 5.6: *Example of trust graph.*

In order to perform the previous two tasks (obtain a public key and a signature), we defined a mechanism still based on the SIP signaling, that applies to peers that have already established a master key, and use that key to *S/MIME* [45] protect payloads of SIP messages.
Such a mechanism expects that one peer that wants a public key and/or a signature from another peer sharing a SA, sends a *SIP SUBSCRIBE* message containing a certificate signature request (simply his certificate with eventually other peers' signatures). The target peer, if agrees, responds with a 200 OK and sends back a *NOTIFY* message containing the signed certificate and its public key. The described message exchange is depicted in Figure 5.7 where $CERT_i$ is the certificate of the Initiator, $sign_r$ and K_r^+ are respectively the signature and the public key of the responder. Note that such a mechanism is unidirectional: the peer initiator asks to the peer responder for its public key and signature, so two runs are required in case of mutual public keys and signatures exchange.

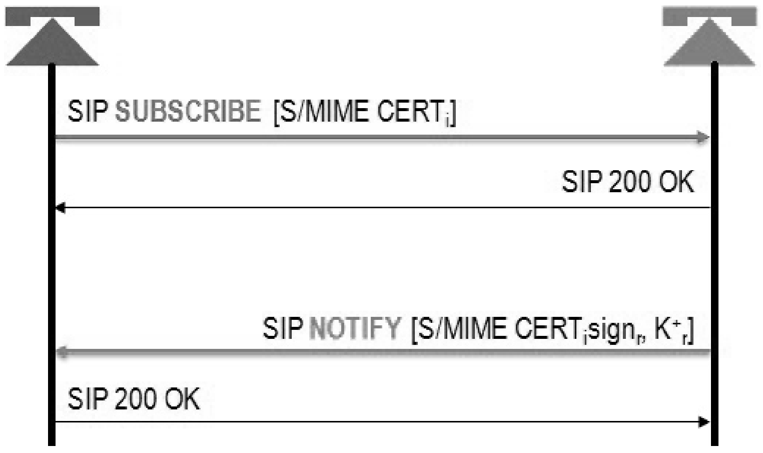

SIP SUBSCRIBE [S/MIME CERT$_i$]
SIP 200 OK
SIP NOTIFY [S/MIME CERT$_i$sign$_r$, K^+_r]
SIP 200 OK

Figure 5.7: *Exchange of public keys and certificates.*

5.3 Analysis of the proposed key agreement protocol

In this section a general analysis of the protocol in its second implementation and some comparisons with other protocols are given.

As regards the analysis, we made some considerations about desirable features a key agreement protocol should have:

- *Freshness*: the new master key is computed through a DH exchange and eventually through previously established master secrets. The ephemeral DH procedure provides freshness, as it depends upon random chosen inputs.

- *Unknown key share attack*: it happens when an adversary C makes one party, say A, believe it is communicating with B while in fact it is communicating with C itself. It is prevented providing the identities of both parties, both in plain text and ciphered through the MAC in the second and third messages. (This is a further explanation of the presence of a Confirm message).

- *Replay attack*: it is prevented with the presence of RAND fields.

- *Key confirmation*: through the second and third messages.

- *Computational costs:* the same as basic ephemeral DH plus signatures (two HMAC).
- *Entity Authentication*: achieved through certificates, voice recognition or pre-shared secrets, assuming the private keys or pre-shared secrets are not compromised and/or the voice has not been forged.
- *Liveness*: as long as the responder answers the initiator.
- *Key authentication*: the new master key is known only to the initiator and the responder. This is verified because only if entity authentication is granted the procedure goes on, otherwise it is stopped and the new just established key withdrawn. The second and third messages grant this property.
- *Forward secrecy*: perfect forward secrecy as the new master key could depend upon long term secrets (the previous master keys), but it surely depends upon a fresh ephemeral DH result.
- *Key control*: neither party can have full control over the master key as it depends upon ephemeral DH result.
- *Small subgroup attack*: mitigated with the implicit consideration that the group G of the DH algorithm is of prime order.
- *Key compromise impersonification attack*: it is an attack where an adversary stolen the private key of a party can impersonate another party. It is avoided as private keys are never transmitted.
- *Disclosure of identities*: possible because there are ID fields transmitted in a not encrypted way.
- *Key integrity*: this is granted if the new master key is given only by the input of both parties. This property holds recursively considering the previous master secrets.

- *Key consistency*: it holds for the ephemeral DH algorithm used. Both parties arrive to the same key as also the eventually previous master secrets are built in the same way.

5.3.1 Comparison with existing protocols

Compared to [46], our proposal has the advantage of not relying in any way on centralized servers.
If we see [47], it permits key agreement through a differentiated routing but it is not suitable for real time applications, it is attackable by the link corruption attack under certain conditions and moreover it needs a DHT structure; our proposal instead does not need such a superstructure but employs a self-maintained public-key distribution system. Differently from [48] our approach is integrated into SIP, and as stated above it is resilient to some attack that [49] proves [50] is subject to. [51] is the most similar work to ours in literature but it anyway bases upon the trustworthiness of third parties, namely SIP Proxies, and on an architecture for certificate distributions.

5.4 Third Scheme: use of a Distributed Hash Table (DHT)

The second implementation presents a temporal problem, that is, it is expected that all necessary connections (for connections we intend a bidirectional security association between peers) are already setup. Referring to Figure 5.8 all the connections are already setup in the communication path represented by the arrow. So node A could communicate with Z node with the second implementation of our protocol: in fact Z could ask D to sign its certificate and D could ask in turn to C to sign Z's certificate and so on. So, finally, Z could communicate with A sending to A its certificate signed by B.

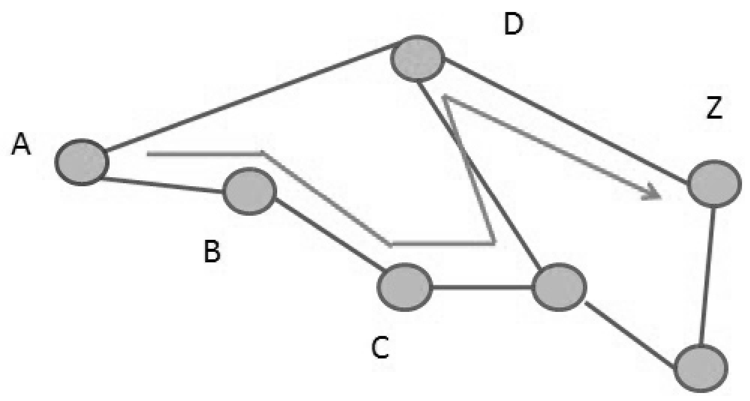


Figure 5.8: *Security associations between peers.*

However these issues are still open :

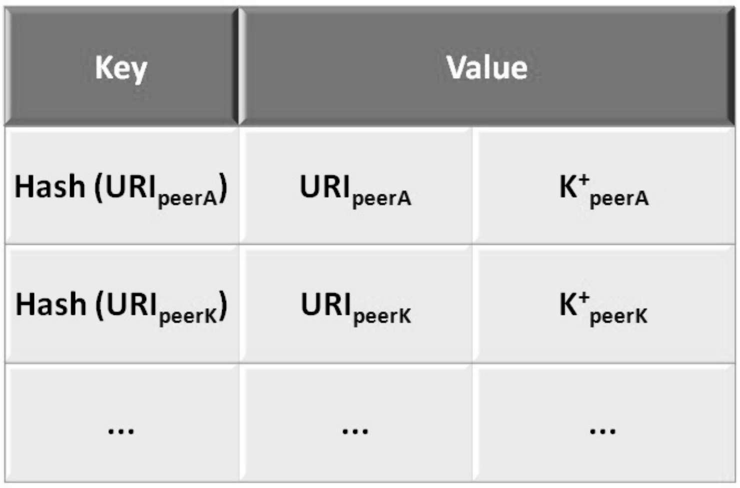
- there is a temporal problem, that is, all the nodes connecting Z with A must be online when Z needs to communicate with A and moreover must already have setup the SAs with all the other nodes in the path.
- there is a great number of bytes and data to be exchanged.

For this reasons we studied further towards a further improvement concerning the usage of a Distributed Hash Table (DHT).

We devised three different but similar solutions for this third implementation and we are still evaluating which one could be the best one.

5.4.1 Certificateless DHT

In this case a DHT is used to store for each peer a composed data formed by various peer information and his/her public key, as depicted in the table of Figure 5.9.



Key	Value	
Hash (URI_{peerA})	URI_{peerA}	K^{+}_{peerA}
Hash (URI_{peerK})	URI_{peerK}	K^{+}_{peerK}
...	...	...

Figure 5.9: *First possible usage of a DHT in the third implementation.*

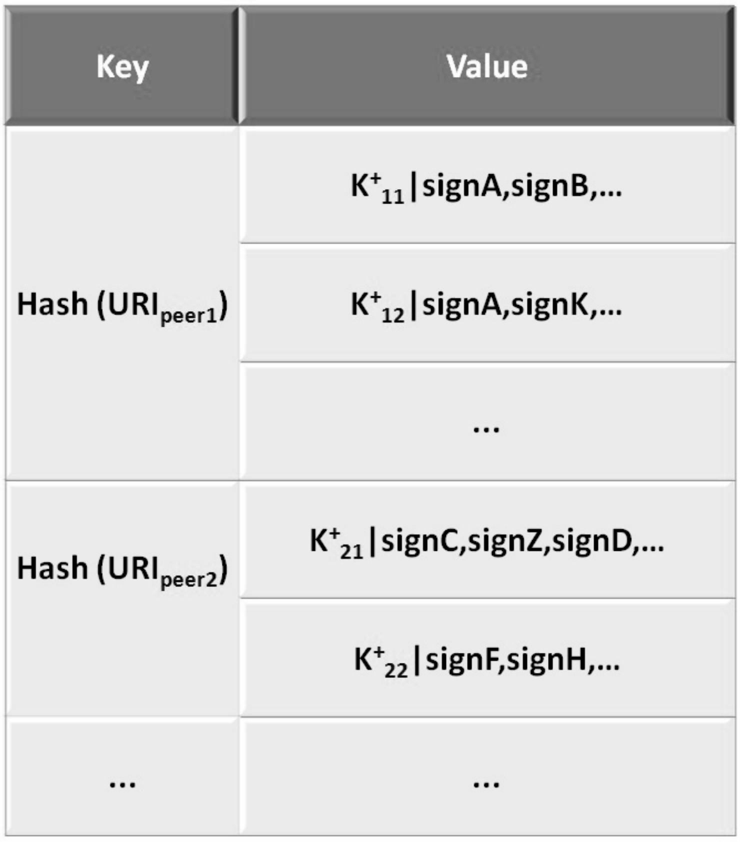
For each peer only one public key is maintained within the DHT, and its revocation and modification are possible only by the owner after proving that he/she has the corresponding private key. When revoking a possible compromised public key, the owner has also the possibility to (and should) republish a new valid key.
Although through the DHT it is possible to obtain a public key associated to a peer, the proposed key agreement protocol is still required in order to:

- be sure one is speaking with the person one wants
- verify in a more robust way the public key in the DHT (more precisely the DHT *get* operation).

There is still an open question on who is going to publish the public key in the DHT, probably the owner itself.

5.4.2 DHT with certificates

During our key agreement protocol direct exchange, the peers exchange their public key. Then if a SA is established they sign their counterpart public key and they publish it in the DHT that has the form of figure 5.10. The key is the hash of ID/URI, the value an array of public keys signed by various peers.



Key	Value
Hash (URI_{peer1})	$K^{+}_{11} \mid$ signA,signB,...
	$K^{+}_{12} \mid$ signA,signK,...
	...
Hash (URI_{peer2})	$K^{+}_{21} \mid$ signC,signZ,signD,...
	$K^{+}_{22} \mid$ signF,signH,...
...	...

Figure 5.10: *Second possible usage of a DHT in the third implementation.*

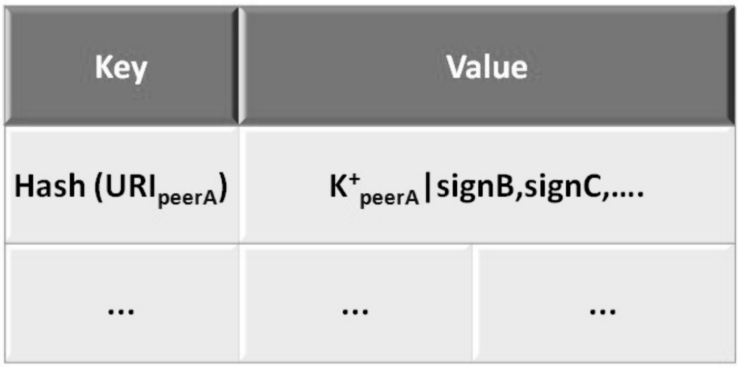
This solution provides that:

- more than one public key are possible for the same ID;
- it is up to the retriever to trust or not one or more signatures and in case to retrieve other public keys to verify the signatures.

Our key agreement protocol is needed to obtain the public key of the counterpart peer but not for robustness of the DHT *get* that depends on other things such as trust on the signatures. Maybe there can be a threshold or a quorum on the trusted signatures to decide whether to trust or not a certain public key.

5.4.3 DHT with certificates but only one possible public key per ID

This solution puts together ideas from the previous two subsections. In the DHT there can be only one public key per ID accompanied by all its signatures. The idea is depicted in figure 5.11.



Key	Value	
Hash (URI_{peerA})	K^{+}_{peerA} | signB,signC,....	
...	...	...

Figure 5.11: *Third possible usage of a DHT in the third implementation.*

Only one having the corresponding private key can edit the public key in the DHT but also signatures count: trusting one or more signature is up to the trust policy.
Our key agreement protocol is still needed to solve the problem of homonymous names like in subsection 5.4.1.

Conclusions

In this thesis work we prsented some possible study solutions to some security problems both at application and at routing level of likely multimedia peer-to-peer communications.

As regards the application level, following a study of current state-of-the-art key agreement protocols, both for multimedia communications and not, we provided a new one for establishing a security association between two peers willing to set up a SIP multimedia session. We based our research mainly upon MIKEY, ZRTP and SIP integrating the best features of each one into a new key agreement protocol that grants authentication according to various possibilities:

- pre-shared secret;
- certificates;
- Short Authentication String.

We also added to our key agreement protocol, which has been analyzed according to some desired features, two possible mechanisms to exchange in a p2p manner certificates and public keys: one based upon the *subscribe* and *notify* SIP methods, the other one exploiting a DHT structure and in case also trust or threshold concepts.

Trust was also the possible countermeasure we studied to cope with the sybil attack and all its possible negative consequences in a DHT environment at routing level. Particularly we mixed trust and standard mechanism in the Kademlia DHT to find out a method to alleviate the negative effects of an increasing number of sybil in the p2p network onto the resource lookup procedure.

Both the studied solutions need improvements, in the first case to verify accurately the performances of the protocol against various type of attacks and implementing in Java language also scheme two and three; moreover a choice of the better solution for the DHT usage in the third scheme is still an open issue. In the second case, finding out an appropriate balancing between trust and standard kademlia procedure and the optimum formula to both soothe away the effects of sybil and maintain quite high the performances is still an open question that could be the objective of possible future works.

Bibliography

[1] John R. Douceur. The sybil attack. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01, pages 251–260, London, UK, 2002. Springer-Verlag. Available from: `http://portal.acm.org/citation.cfm?id=646334.687813`.

[2] J. Arkko, E. Carrara, F. Lindholm, M. Naslund, and K. Norrman. MIKEY: Multimedia Internet KEYing. RFC 3830 (Proposed Standard), August 2004. Updated by RFC 4738. Available from: `http://www.ietf.org/rfc/rfc3830.txt`.

[3] P. Zimmermann, A. Johnston, and J. Callas. ZRTP: Media Path Key Agreement for Unicast Secure RTP. Obsolete Internet-Draft draft-zimmermann-avt-zrtp-22, Internet Engineering Task Force, June 2010. Work in progress. Available from: `http://www.ietf.org/id/draft-zimmermann-avt-zrtp-22.txt`.

[4] Lin Wang. Attacks Against Peer-to-peer Networks and Countermeasures. Technical report, Helsinki University of Technology, December 2006.

[5] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01, pages 53–65, London, UK, 2002. Springer-Verlag. Available from: `http://portal.acm.org/citation.cfm?id=646334.687801`.

[6] Ion Stoica, Robert Morris, David R. Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM*, pages 149–160, 2001.

[7] B.Y. Zhao, Ling Huang, J. Stribling, S.C. Rhea, A.D. Joseph, and J.D. Kubiatowicz. Tapestry: a resilient global-scale overlay for service deployment. *Selected Areas in Communications, IEEE Journal on*, 22(1):41 – 53, 2004. `doi:10.1109/JSAC.2003.818784`.

[8] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '01, pages 161–172, New York, NY, USA, 2001. ACM. Available from: `http://doi.acm.org/10.1145/383059.383072`, `doi:http://doi.acm.org/10.1145/383059.383072`.

[9] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, STOC '97, pages 654–663, New York, NY, USA, 1997. ACM. Available from: `http://doi.acm.org/10.1145/258533.258660`, `doi:http://doi.acm.org/10.1145/258533.258660`.

[10] Baptiste Pretre. *Attacks on Peer-to-peer Networks*. PhD thesis, Dept. of Computer Science, Swiss Federal Institute of Technology (ETH) Zurich, 2005.

[11] Emil Sit and Robert Morris. Security considerations for peer-to-peer distributed hash tables. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, Cambridge, MA, March 2002.

[12] Brian Neil, Levine Clay Shields, and N. Boris Margolin. A survey of solutions to the sybil attack.

[13] George Danezis, Chris Lesniewski-laas, M. Frans Kaashoek, and Ross Anderson. Sybil-resistant dht routing. In *ESORICS*, pages 305–318. Springer, 2005.

[14] Miguel Castro, Peter Druschel, Ayalvadi Ganesh, Antony Rowstron, and Dan S. Wallach. Secure routing for structured peer-to-peer overlay networks, 2002. Available from: `http://doi.acm.org/10.1145/1060289.1060317`, `doi:http://doi.acm.org/10.1145/1060289.1060317`.

[15] Haifeng Yu, Michael Kaminsky, Phillip B. Gibbons, and Abraham Flaxman. Sybilguard: defending against sybil attacks via social networks. In *Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '06, pages 267–278, New York, NY, USA, 2006. ACM. Available from: `http://doi.acm.org/10.1145/1159913.1159945`, `doi:http://doi.acm.org/10.1145/1159913.1159945`.

[16] Tyson Condie, Varun Kacholia, Sriram Sankararaman, Joseph M. Hellerstein, and Petros Maniatis. Induced churn as shelter from routing-table poisoning. In *In Proc. 13th Annual Network and Distributed System Security Symposium (NDSS*, 2006.

[17] Newso James, Elaine Shi, Dawn Song, and Adrian Perrig. The sybil attack in sensor networks: analysis & defenses. In *Proceedings of the 3rd international symposium on Information processing in sensor networks*, IPSN '04, pages 259–268, New York, NY, USA, 2004. ACM. Available from: `http://doi.acm.org/10.1145/984622.984660`, `doi:http://doi.acm.org/10.1145/984622.984660`.

[18] Atul Singh, Tsuen wan Ngan, Peter Druschel, and Dan S. Wallach. Eclipse attacks on overlay networks: Threats and defenses. In *IEEE INFOCOM*, 2006.

[19] H. Rowaihy, W. Enck, P. McDaniel, and T. La Porta. Limiting sybil attacks in structured p2p networks. In *INFOCOM 2007. 26th IEEE International Con-*

ference on Computer Communications. IEEE, pages 2596 –2600, May 2007. doi:10.1109/INFCOM.2007.328.

[20] Nikita Borisov. Computational puzzles as sybil defenses. In *Proceedings of the Sixth IEEE International Conference on Peer-to-Peer Computing*, pages 171–176, Washington, DC, USA, 2006. IEEE Computer Society. Available from: http://portal.acm.org/citation.cfm?id=1157740.1158254, doi:10.1109/P2P.2006.10.

[21] Moritz Steiner, Taoufik En-najjary, and Ernst W. Biersack. Exploiting kad: Possible uses and misuses. *ACM SIGCOMM CCR*, 37:2007.

[22] Jochen Dinger and Hannes Hartenstein. Defending the sybil attack in p2p networks: Taxonomy, challenges, and a proposal for self-registration. In *ARES '06: PROCEEDINGS OF THE FIRST INTERNATIONAL CONFERENCE ON AVAILABILITY, RELIABILITY AND SECURITY (ARES'06)*, pages 756–763, 2006.

[23] M. Agosti M. Amoretti. deus-project hosting on google code [online]. 2010. Available from: http://code.google.com/p/deus/.

[24] R. V. V. S. V. Prasad, Vegi Srinivas, V. Valli Kumari, and K. V. S. V. N. Raju. An effective calculation of reputation in p2p networks. *JNW*, 4(5):332–342, 2009.

[25] Xu Wu, Jingsha He, and Fei Xu. An enhanced trust model based on reputation for p2p networks. In *Sensor Networks, Ubiquitous and Trustworthy Computing, 2008. SUTC '08. IEEE International Conference on*, pages 67 –73, 2008. doi:10.1109/SUTC.2008.28.

[26] Debora Donato, Mario Paniccia, Maddalena Selis, Carlos Castillo, Giovanni Cortese, and Stefano Leonardi. New metrics for reputation management in p2p networks. In *Proceedings of the 3rd international workshop on Adversarial information retrieval on the web*, AIRWeb '07, pages 65–72, New York, NY, USA, 2007. ACM. Available from: http://doi.acm.org/10.

1145/1244408.1244421, doi:http://doi.acm.org/10.1145/1244408.1244421.

[27] So Young Lee, O-Hoon Kwon, Jong Kim, and Sung Je Hong. A reputation management system in structured peer-to-peer networks. In *Enabling Technologies: Infrastructure for Collaborative Enterprise, 2005. 14th IEEE International Workshops on*, pages 362 – 367, 2005. doi:10.1109/WETICE.2005.9.

[28] A. A. Selcuk, E. Uzun, and M. R. Pariente. A reputation-based trust management system for p2p networks. In *Proceedings of the 2004 IEEE International Symposium on Cluster Computing and the Grid*, CCGRID '04, pages 251–258, Washington, DC, USA, 2004. IEEE Computer Society. Available from: http://portal.acm.org/citation.cfm?id=1111683.1111799.

[29] Paul-Alexandru Chirita, Paul Alex, Ru Chirita, Wolfgang Nejdl, Mario Schlosser, and Oana Scurtu. Personalized reputation management in p2p networks, 2004.

[30] Minaxi Gupta, Paul Judge, and Mostafa Ammar. A reputation system for peer-to-peer networks, 2003.

[31] Yao Wang and Julita Vassileva. Trust and reputation model in peer-to-peer networks. In *Proceedings of the 3rd International Conference on Peer-to-Peer Computing*, P2P '03, pages 150–, Washington, DC, USA, 2003. IEEE Computer Society. Available from: http://portal.acm.org/citation.cfm?id=942805.943810.

[32] S. Kent and R. Atkinson. Security Architecture for the Internet Protocol. RFC 2401 (Proposed Standard), November 1998. Obsoleted by RFC 4301, updated by RFC 3168. Available from: http://www.ietf.org/rfc/rfc2401.txt.

[33] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008. Updated by RFCs 5746, 5878. Available from: http://www.ietf.org/rfc/rfc5246.txt.

[34] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Transport Layer Protocol. RFC 4253 (Proposed Standard), January 2006. Available from: `http://www.ietf.org/rfc/rfc4253.txt`.

[35] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer. OpenPGP Message Format. RFC 4880 (Proposed Standard), November 2007. Updated by RFC 5581. Available from: `http://www.ietf.org/rfc/rfc4880.txt`.

[36] H. Afifi D. Seret H. Soussi, M. Hussain. Ikev1 and ikev2: A quantitative analyses, 2005.

[37] D. Maughan, M. Schertler, M. Schneider, and J. Turner. Internet Security Association and Key Management Protocol (ISAKMP). RFC 2408 (Proposed Standard), November 1998. Obsoleted by RFC 4306. Available from: `http://www.ietf.org/rfc/rfc2408.txt`.

[38] Riccardo Bresciani. The ZRTP Protocol Security Considerations. Technical report, École Normale and Supérieure Cachan, may 2007.

[39] P. Gupta and V. Shmatikov. Security analysis of voice-over-ip protocols. In *Computer Security Foundations Symposium, 2007. CSF '07. 20th IEEE*, pages 49 –63, 2007. `doi:10.1109/CSF.2007.31`.

[40] R. Pecori and L. Veltri. A key agreement protocol for p2p voip applications. In *Software, Telecommunications Computer Networks, 2009. SoftCOM 2009. 17th International Conference on*, pages 276 –280, 2009.

[41] S. Cirani, R. Pecori, and L. Veltri. A peer-to-peer secure voip architecture. In *Proceedings of ITWDC 2010. 21st International Tyrrhenian Workshop on Digital Communications: Trustworthy Internet*, sept. 2010.

[42] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261 (Proposed Standard), June 2002. Updated by RFCs 3265, 3853, 4320, 4916,

5393, 5621, 5626, 5630, 5922, 5954, 6026. Available from: `http://www.ietf.org/rfc/rfc3261.txt`.

[43] J. Arkko, V. Torvinen, G. Camarillo, A. Niemi, and T. Haukka. Security Mechanism Agreement for the Session Initiation Protocol (SIP). RFC 3329 (Proposed Standard), January 2003. Available from: `http://www.ietf.org/rfc/rfc3329.txt`.

[44] L. Veltri. Mjsip project [online]. 2006. Available from: `http://www.mjsip.org/`.

[45] B. Ramsdell. Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification. RFC 3851 (Proposed Standard), July 2004. Obsoleted by RFC 5751. Available from: `http://www.ietf.org/rfc/rfc3851.txt`.

[46] Jen-Chiun Lin, Kuo-Hsuan Huang, Feipei Lai, and Hung-Chang Lee. Secure and efficient group key management with shared key derivation. *Computer Standards & Interfaces*, 31(1):192 – 208, 2009. Available from: `http://www.sciencedirect.com/science/article/B6TYV-4R8H1TR-6/2/fb531f2c68f8b92beebf1608a5a82746`, `doi:DOI:10.1016/j.csi.2007.11.005`.

[47] Yuuki Takano, Naoki Isozaki, and Yoichi Shinoda. Multipath key exchange on p2p networks. In *Proceedings of the First International Conference on Availability, Reliability and Security*, pages 748–755, Washington, DC, USA, 2006. IEEE Computer Society. Available from: `http://portal.acm.org/citation.cfm?id=1130897.1130998`, `doi:10.1109/ARES.2006.87`.

[48] Mengbo Hou and Qiuliang Xu. A secure two-party key agreement protocol with key escrow and perfect forward secrecy. In *Proceedings of the 3rd international conference on Anti-Counterfeiting, security, and identification in communication*, ASID'09, pages 501–504, Piscataway, NJ, USA, 2009. IEEE

Press. Available from: `http://portal.acm.org/citation.cfm?id=1719110.1719229`.

[49] Mengbo Hou and Qiuliang Xu. On the security of certificateless authenticated key agreement protocol (cl-ak) for grid computing. In *Proceedings of the 2009 Fourth ChinaGrid Annual Conference*, CHINAGRID '09, pages 128–133, Washington, DC, USA, 2009. IEEE Computer Society. Available from: `http://dx.doi.org/10.1109/ChinaGrid.2009.13`, `doi:http://dx.doi.org/10.1109/ChinaGrid.2009.13`.

[50] Shengbao Wang, Zhenfu Cao, and Haiyong Bao. Efficient certificateless authentication and key agreement (cl-ak) for grid computing, 2006.

[51] O. Jung, M. Petraschek, T. Hoeher, and I. Gojmerac. Using sip identity to prevent man-in-the-middle attacks on zrtp. In *Wireless Days, 2008. WD '08. 1st IFIP*, pages 1 –5, 2008. `doi:10.1109/WD.2008.4812920`.

Acknowledgments

These are not simply acknowledgements but sincere *thank you*s to some people I met during these years in Parma.

Annamaria and Stefano: thank you, mainly for the period I worked together with them, I felt myself very comfortable with them, learnt a lot and more important, they gave me the chance to write my first paper and to attend my first conference (Fotonica 2009). I hope they are a bit proud and happy for the work we made together, even if it was a small work.

Federica: thank you, mainly for the period I worked with her, she has been the perfect work-mate, I will never forget...thank you for letting me enrich her postcards wall,for having withstood me in the work together...I wish her all the best for her career.

Gianluigi: thank you for his company in Oahu when I was alone, in that dream trip to Hawaii. Thanks for his few but right words of advice in our trips for eCampus . They all have been very beautiful moments.

Armando: a special person, he is THE "teacher". I will always envy his proper parlance and his security when speaking. He was very supportive with me.

Giulio: a great person, one of the best people I have ever known, but with the courage to say things in front of others. I wish him all the best for him and his family.

Anila: thank you for her sweet and sensitive company, and for the English I learnt from her.

Simone: I would have liked to work well like he did, and I hope he will be luckier than me. A friend with whom I hope to keep in touch and make many things together that we did not make in these three years. I will always remember the trip to Hawaii, unforgettable. Thank you also for the words of advice on life in general.

Nicolò: thank you for the smiles and the light moments spent together, I hope he will be a great Ph.D. student.

Tommaso: one of the people I value more. A perfect researcher, temporary but perfect.

Luca: last but not least. I think I learnt a lot from him not only in work. Thank you to him is not enough to express the esteem and the admiration. He will be always welcome in my home, and my enduring support will be with him as long as he wants. Thank you for his infinite patience with me, I hope he is a bit proud of me.

Thanky you also to my family for its constant support.

Thank you also to *Erica*, who besides *Luca* has been my support every week,
thank you to Alfonso, Pasquale, MariaTeresa, Salvatore,
thank you to the guys and girls of Voladora Ultimate frisbee team,
thank you to my tennis girls like Giulia, Irene and Stefania
tank you to the guys and girls of the latin dancing course....
thank you to all the colleagues met at the conferences: Elisa, Alice, Mauricio, Amedeo, Rajiv who let me go to Miami, thank you...
thank you to past colleagues like Dario, Lorenzo, Aldo, Alan, Berto...
thank you to present colleagues like Amina, Andrea Modenini, Davide, Busa...
Special thanks also to *AMC*, *Emanuela* and *Stella*!