MAT-3900

MASTER'S THESIS IN

INFORMATION SECURITY

Distributing a Private Key Generator in Ad hoc Networks

Eystein Måløy Stenberg

May, 2009

FACULTY OF SCIENCE
Department of Mathematics and Statistics
University of Tromsø

Abstract

A Mobile Ad hoc Network (MANET) is a wireless network that does not rely on a fixed infrastructure. These characteristics make algorithms that route network traffic particularly vulnerable to attack. Mechanisms used to protect against such attacks often depend on cryptographic keys.

Since the nodes in a MANET have limited resources, designing methods for cryptographic key management is particularly challenging. Because the network infrastructure is unstable, assuming that authorities used in key management are implemented using any single node is not realistic. Threshold cryptography can be used to distribute an authority, such that it is implemented by multiple nodes. This makes the authority more robust against network failures and harder to compromise.

However, the bandwidth limitations in a MANET result in that public key distribution becomes very challenging. Identity-based cryptography (IBC), where any identity may serve as a public key, makes public keys and their certificates superfluous. The authority issuing private keys corresponding to an identity is called a Private Key Generator (PKG).

This thesis considers the issue of distributing a PKG to the nodes in a MANET. It gives a description of a generic distributed PKG, including a definition of security. An example of a distributed PKG is also given. This distributed PKG is compatible with some of the most prevalent IBC systems. It is shown that the security properties of the base IBC systems are preserved when this distributed PKG is used instead of the original one.

Threshold cryptography and identity-based cryptography are found to result in very efficient key management systems, compared to other methods. It is however important to consider which security properties a distributed authority has, especially with respect to any leakage of information on the authority's secret key. However, the main challenge in connection with key management in a MANET is to authenticate nodes without requiring preestablished trust.

Contents

List of Figures

List of Tables

List of Abbreviations

ANMA	Advertised Neighbour Main Address
AODV	Ad-hoc On-Demand Distance Vector Routing
ARAN	Authenticated Routing for Ad hoc Networks
BDH	Bilinear Diffie-Hellman
BF-IBE	Boneh and Franklin identity-based encryption scheme
CA	Certificate Authority
CC-IBS	Cha and Cheon identity-based signature scheme
CDH	Computational Diffie-Hellman
DDH	Decision Diffie-Hellman
DLP	Discrete logarithm problem
DPKG	Distributed Private Key Generator
DSA	Digital Signature Algorithm
DSR	Dynamic Source Routing
ECC	Elliptic curve cryptography
ECDLP	Elliptic curve discrete logarithm problem
IBC	Identity-based cryptography
IBE	Identity-based encryption
IBS	Identity-based signature
IETF	Internet Engineering Task Force
MAC	Message authentication code
MANET	Mobile Ad hoc Network
MPR	Multipoint Relay
OLSR	Optimized Link State Routing
PK	Public key (of asymmetric key pair)
PKG	Private Key Generator
PKI	Public Key Infrastructure
RDP	Route Discovery Packet
SK	Secret key (of asymmetric key pair)
SNEP	Sensor Network Encryption Protocol
SPINS	Security Protocols for Sensor Networks
TESLA	Timed Efficient Stream Loss-tolerant Authentication

Preface

This thesis concludes a master programme in Information Security at the University of Tromsø. I would like to thank the Norwegian Defence Research Establishment (Forsvarets forskningsinstitutt) for their cooperation on this thesis. It resulted in a thesis with an exciting combination of mathematics and information technology. I have had the pleasure of three excellent supervisors helping me with the thesis, namely Ragnar Soleng, Tormod Sivertsen, and Eli Winjum. Thank you for the motivating feedback I have received, it has increased the quality of the thesis considerably.

Studying at the Department of Mathematics and Statistics has been a rewarding experience academically, but has also given strong generic skills in structured problem solving and analytical thinking. The employees were always happy to spend time helping me when I had questions, which had a strong motivating effect during the years I have studied here. I would in particular like to thank Loren Olson and Ragnar Soleng in this respect.

My family has also given much support during the time I have been a student. I would especially like to thank my parents, Erling and Kristine, for their support, and for teaching me to never give up when facing problems. My girlfriend Mikaela has also played an important role for me during my time as a student.

Tromsø, 13th May 2009
Eystein Måløy Stenberg

1 Introduction

A Mobile Ad hoc Network (MANET) is a self-configuring wireless network. Nodes participating in the network do not rely on a fixed infrastructure, but use each other to communicate outside their own transmission range. Thus, all nodes have a responsibility as routers when they forward traffic for other nodes, in addition to being communication endpoints. The nodes may join and leave the network at arbitrary points in time. Being mobile, they may also move, which also results in changes in the network topology. But since the nodes are also being used as routers, it is important that these special characteristics are respected when routing traffic. This implies that the routing protocol used in a MANET must be designed for such an environment. Several MANET routing protocols are going through a standardisation process by the IETF[1], examples include the Optimized Link State Routing (OLSR) protocol [19], the Ad-hoc On-Demand Distance Vector Routing (AODV) protocol [53], and the Dynamic Source Routing (DSR) protocol [40].

MANETs have so far mainly been used in military applications, for example to facilitate communication on a battlefield. Their special characteristics also make them well suited in emergency and rescue operations, for example during the search for a missing person. In recent years, wireless networks have become increasingly pervasive in civil applications as well. Although most civil wireless networks today are based on a fixed infrastructure, it is not difficult to find scenarios where a MANET is better suited. A corporate meeting where the participants want to exchange documents residing on their laptops is one example. The broad applicability of a MANET in many problem areas make it probable that such networks will gain widespread civilian use in the future.

The most prevalent MANET routing protocols assume that all participating nodes are well-behaved, i.e. act according to the protocol instructions. Examples include OLSR, AODV and DSR. However, the characteristics of a MANET, especially wireless communication and lack of infrastructure, make the routing protocol used in these networks particularly vulnerable to attack. Therefore, routing security in MANETs has gained attention. This has resulted in extensions to existing MANET routing protocols, and design of new protocols that alleviate certain security problems. However, these protocols often assume that a central Certificate Authority (CA) is available at all times, a problematic assumption in a MANET. In addition, security mechanisms create an extra overhead on the network, mainly caused by cryptographic key distribution and transmission of authentication data. However, Hegland et

[1] See http://www.ietf.org/html.charters/manet-charter.html.

al. [36] show that the OLSR protocol in particular, and probably MANET routing protocols in general, cannot handle more than a few hundred bits of extra overhead per message.

To solve these demanding challenges, cryptographic tools that were previously little used are proposed utilised. Threshold cryptography may be used to distribute an authority in such a way that one does not rely on any single node. When distributing an authority, multiple nodes are used to implement it. Furthermore, identity-based cryptography makes public key distribution obsolete because an identity, an arbitrary bit string, may serve as the public key. The authority that issues private keys corresponding to identities is called a Private Key Generator (PKG). There are many proposed key management systems designed for MANETs in which distribution of a CA or PKG plays a central role [35, 69, 43, 42].

We will consider some prominent proposals for distribution of an authority in a MANET and compare them. In particular, we will discuss similar characteristics of a distributed CA and a distributed PKG, together with advantages and disadvantages of the two approaches. Based on this, we will try to make a generic description of a distributed PKG. This description will include a discussion of algorithms required and common issues that arise when distributing a PKG. The description will make it easier to compare proposals for a distributed authority in general, and a distributed PKG in particular. We evaluate all our work in the context of a MANET.

We initiate our work by gaining an understanding of how routing can be done in a MANET, through an in-depth discussion of the OLSR protocol in Chapter 2. After this, we consider a broad range of proposals for securing MANET and sensor network routing protocols. We consider proposals for sensor networks as well because sensor networks share many characteristics with MANETs, although they are generally more resource-constrained. This will give an overview of the often used cryptographic mechanisms, together with the assumptions on key management. Elliptic curves and pairings often lead to very efficient cryptographic mechanisms. Pairings are functions which may be defined on elliptic curves. We explore both some theory on pairings and cryptographic applications of them in Chapter 3. We are especially interested in efficient implementations of pairings, and the identity-based cryptography systems and short signature schemes they are used to construct. In Chapter 4, we consider threshold cryptography, which is the building block when distributing a cryptographic algorithm. Using threshold cryptography, we can create functions which require the consent of t out of n nodes to succeed. We describe existing proposals for key management in a MANET in Chapter 5. In particular, we discuss a system based on a combination of threshold cryptography and identity-based cryptography in depth. A generic description of

a distributed PKG follows in Chapter 6. The description includes a discussion of algorithms that are required and common issues that arise when distributing a PKG. An example of a distributed PKG evaluated in this description is given in Chapter 7. This distributed PKG is compatible with some of the most prevalent IBC systems, for example the Boneh and Franklin identity-based encryption system [12], the Cha and Cheon identity-based signature system [15] and the Chen and Malone-Lee identity-based signcryption system [17]. In Chapter 8, we consider a method for authenticating nodes without relying on preestablished trust. This method assumes that all nodes have about the same computational resources available. We compare our distributed PKG from Chapter 7 with a system based on a distributed CA in Chapter 9. We also summarise and conclude in Chapter 9.

2 Routing

Through this Chapter, we will gain a better understanding how routing can be done, especially in a MANET. We start with a short introduction to routing in Section 2.1. In Section 2.2, we give an in-depth description of the MANET routing protocol that is closest to standardisation by the IETF (Internet Engineering Task Force), namely the Optimized Link State Routing protocol. After this, we will consider proposals for securing a broader range of MANET routing protocols in Section 2.3. This will give an overview of the often used cryptographic mechanisms, and the assumptions on key management. In order to not get lost in details, we will frequently give summaries of our discussions.

2.1 Introduction to routing

In this Section, we will briefly discuss what routing is, or what the goal of a routing algorithm is. We will also describe standard classifications of routing algorithms.

To formulate our discussion of routing, we will use the concept of a graph $G = (N, E)$, where N is a set of nodes, and E is a set of edges connecting some of these nodes. The nodes in the set N are called routers, and the edges in E are links between the routers, either physical (e.g. twisted pair connections) or wireless. An example graph is shown in Figure 1.

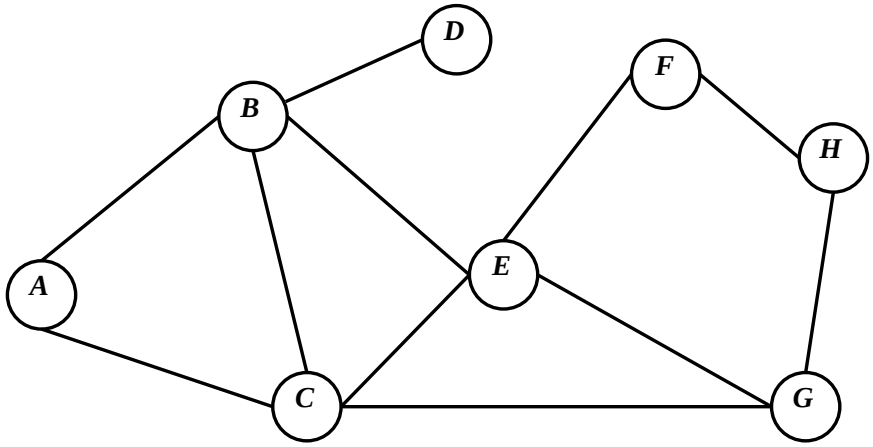

Figure 1: Example of a routing graph.

The whole purpose of routing is to establish good paths (also known as routes) between a given source and destination node, through the network

of routers. However, it is not obvious how to measure if a path is better than another. But the main goal is usually to find a path that carries traffic fast from source to destination (i.e. has low latency), and at the same time has high bandwidth. We can associate costs with all links (e.g. a positive integer), such that links with a low latency and high bandwidth get a low cost, while bad links get a high cost. Finding a good path is now equivalent to finding a *least-cost* path.

However, determining a realistic cost for all links is a difficult task — and it might not even be possible in practice because the latency and bandwidth vary with the network load. Thus, some simple approximation is often used, such as identifying the *shortest* path. This is equivalent to finding the least-cost path when all links have the same cost. For example, in Figure 1, the shortest path from node A to H is $A \rightarrow C \rightarrow G \rightarrow H$.

Since a routing algorithm is usually a distributed algorithm run by the N nodes in the network, we will often refer to a routing algorithm as a *routing protocol*.

Proactive routing A *proactive* routing algorithm keeps routing information up-to-date at all times. Every node stores a routing table with information on all the destinations it can reach, and which of its neighbours it should forward traffic to in order to reach these destinations. An example routing table is shown in Table 1.

Proactive routing algorithms can be divided into two types. As the first type, we have algorithms that are based on the *link-state* algorithm. The link-state algorithm requires all nodes to obtain complete topology information, and the cost associated with all links. Each node can detect links to its neighbours. The other links are detected by receiving broadcast messages, which contain information about the sender's neighbours, from other nodes in the network. When complete network topology is known, each node can independently compute the least-cost path from itself to any other node in the network, by using for instance Dijkstra's algorithm.

Distance-vector algorithms comprise the second type. They do not require that complete network topology is known by every node. Each node detects its neighbours and informs each of them which nodes it can reach. If one of these neighbours discovers a destination that it previously could not reach or a destination which it can reach with a lower cost through the node that published the information, it will record that this destination should be reached through the publishing node. In this way, nodes learn how to reach new destinations and publish the destinations they can reach to their neighbours, which may then also reach these new destinations. After some

period of time, all routers will discover the best next-hop for all destinations.

Reactive routing A *reactive* (or *on-demand*) routing algorithm establishes paths only when they are needed. This is in contrast to a proactive routing algorithm, which maintains every path at all times. When data traffic is to be sent to a destination, a path to that destination needs to be established first. This is usually obtained by broadcasting a message containing the address of the destination node. If the destination node receives the broadcast message, it will reply to it, using the same path as the broadcast message travelled, but the opposite direction. Thus, the path of the broadcast message must be stored by intermediate nodes as it travels through them, either by appending it to the message itself or having the intermediate nodes recognise the broadcast message when it travels back. If the source node receives the response to the broadcast, it will have a valid path to the destination.

Examples of reactive routing algorithms include the Ad-hoc On-Demand Distance Vector Routing protocol and the Dynamic Source Routing protocol. In these algorithms, the broadcast message is called ROUTE REQUEST, while the response to it is called ROUTE REPLY.

2.2 Optimized Link State Routing protocol

The Optimized Link State Routing (OLSR) protocol is standardised by IETF [19]. OLSR is based on the classical link-state routing algorithm, but is specifically developed for MANETs.

2.2.1 Routing

Like the link-state algorithm, OLSR is proactive, which means that it keeps track of possible routes to and from all nodes. Every node keeps a routing table which contains an entry for each of the other nodes in the network that it can reach — either directly or via some intermediate node(s). An example of a routing table is given in Table 1. In the routing table, *R_dest_addr*

R_dest_addr	R_next_addr	R_dist	R_iface_addr
10.0.0.3	10.0.0.2	2	10.0.0.1
10.0.0.2	10.0.0.2	1	10.0.0.1
...	...	...	...

Table 1: Example of an OLSR routing table.

is the address of a reachable node, and *R_next_addr* is the address of the neighbour node[2] that is the next one in the chosen path to that node. Furthermore, *R_dist* is the estimated distance (in terms of hops) from the local node to the node specified by *R_dest_addr*, and *R_iface_addr* is the address of the local interface that can communicate with the neighbour specified by *R_next_addr*[3]. For example, the first row of Table 1 specifies that the node with address 10.0.0.3 can be reached via the neighbour 10.0.0.2, and is two hops away. The local interface with address 10.0.0.1 can be used to reach the neighbour 10.0.0.2.

So if every node has such a routing table, communicating is easy. The real work lies in creating this routing table.

2.2.2 OLSR packet format

The specification of OLSR defines the format of a few messages that any node participating in the OLSR network must understand. These are the `HELLO` message (see Figure 3), the Topology Control message (see Figure 6) and the Multiple Interface Declaration message, and they will be discussed successively. However, all these messages are transmitted inside an OLSR packet, shown in Figure 2.

2.2.3 Neighbour detection

Nodes periodically emit `HELLO` messages. A `HELLO` message contains the addresses of the neighbours that the sender has detected. The sender has, in turn, detected these neighbours by *receiving* `HELLO` messages from them. The format of a such message is given in Figure 3.

Note that it may well be possible for a node to *receive* a `HELLO` message from a node that it cannot *send* to, or the other way around. This is because the the sending radius of the nodes may differ, for reasons such as different transmission power, geographical topology, etc. OLSR only defines routes on links that can transmit in both directions, i.e. are symmetric. But asymmetric links are also stored by nodes and published in `HELLO` messages — if they were not, symmetric links could not be detected.

A receiver of a `HELLO` message with an asymmetric link defined to it, can deduce that the link actually is symmetric and store and publish it as such. Since the link was set as asymmetric, the sender has received a `HELLO` message

[2]I.e. a node that is one hop away — it can be communicated with directly, without using intermediate nodes.

[3]This is only useful if the local node is using more than one radio transmitter in the same OLSR network.

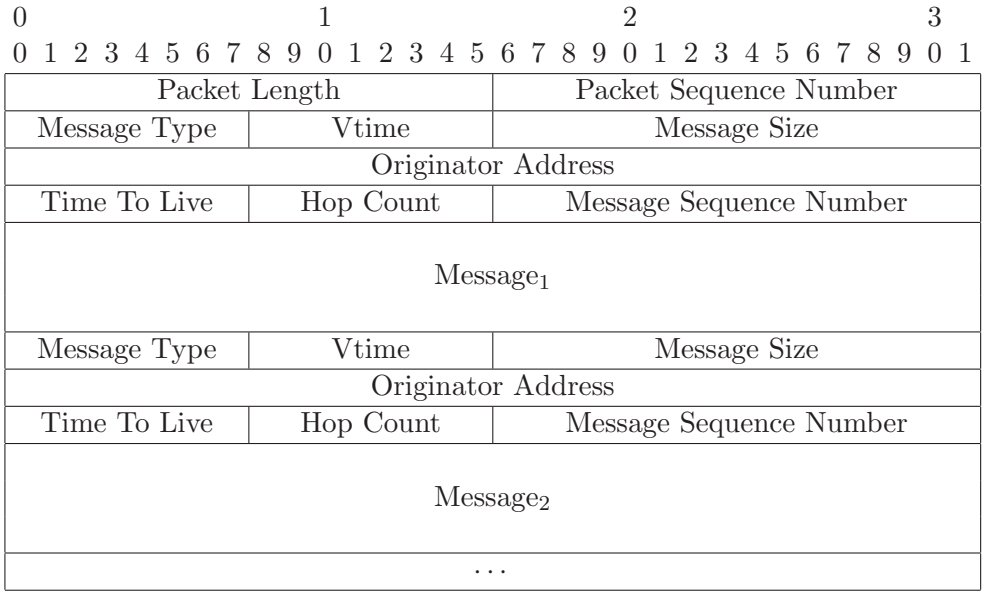

Figure 2: OLSR packet format.

from the receiver earlier, and now the receiver also got a `HELLO` message back. This process is shown in Figure 4, with A acting as the receiver in step b).

In total, a `HELLO` message serves three tasks:

- **Link sensing.** When receiving a `HELLO` message, a node detects a link, as shown in step a) in Figure 4.
- **Neighbour detection.** When a node receives a `HELLO` message, it may deduce that it is a symmetric neighbour to the sender, as illustrated in Figure 4, step b) and c).
- **MPR selection signalling.** The sender advertises which of its neighbours it has selected as Multipoint Relays (MPRs) in a `HELLO` message. MPRs are discussed below.

2.2.4 Multipoint Relays

By receiving `HELLO` messages, a node will also obtain a list of neighbours of its neighbours, i.e. 2-hop neighbours. A node chooses a subset of its neighbours in such a way that *all* its 2-hop neighbours are reachable through (any) one of the nodes in this subset. The nodes selected by at least one node are called Multipoint Relays (MPRs). This concept is illustrated in Figure 5, where node A has selected three MPRs (shown in gray in the illustration).

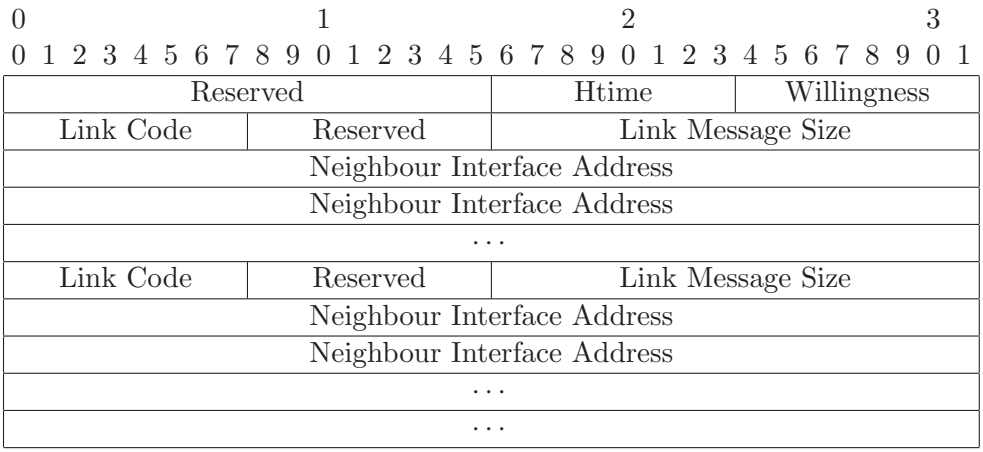

Figure 3: HELLO message format.

All nodes advertise which nodes they have selected as multipoint relays in the `HELLO` message, by listing their addresses under an entry with *Link Code* = *SYM_LINK* | *MPR_NEIGH* (see Figure 3). Therefore, the nodes selected as MPRs will know that they are in fact selected as such, and also which nodes selected them.

Note that a node may select *all* its neighbours as multipoint relays, since all 2-hop neighbours are reachable through at least one of its neighbours. But for efficiency, it is beneficial that each node selects as few multipoint relays as possible. However, it is shown that the problem of computing a minimal set of multipoint relays is NP-complete [66]. Therefore, efficient heuristic algorithms are used to compute a small set of multipoint relays.

It might be worthwhile to note the *Willingness* field in a `HELLO` message (see Figure 3). The sender of a `HELLO` message uses this field to advertise how willing it is to forward traffic for other nodes. This information is used when a node considers selecting the sender as an MPR. The sender may use this field to ensure that no neighbours select it as an MPR, but also force all neighbours to select it an MPR.

2.2.5 MPRs in broadcasting

The point of MPRs is to reduce the amount of traffic in the network. Only MPRs forward broadcast messages, i.e. messages from one node to all the other[4]. However, all nodes still receive the broadcast message, which can be

[4]A sender may however bypass this "MPR-broadcasting" mechanism, and force *all* the nodes to forward the message.

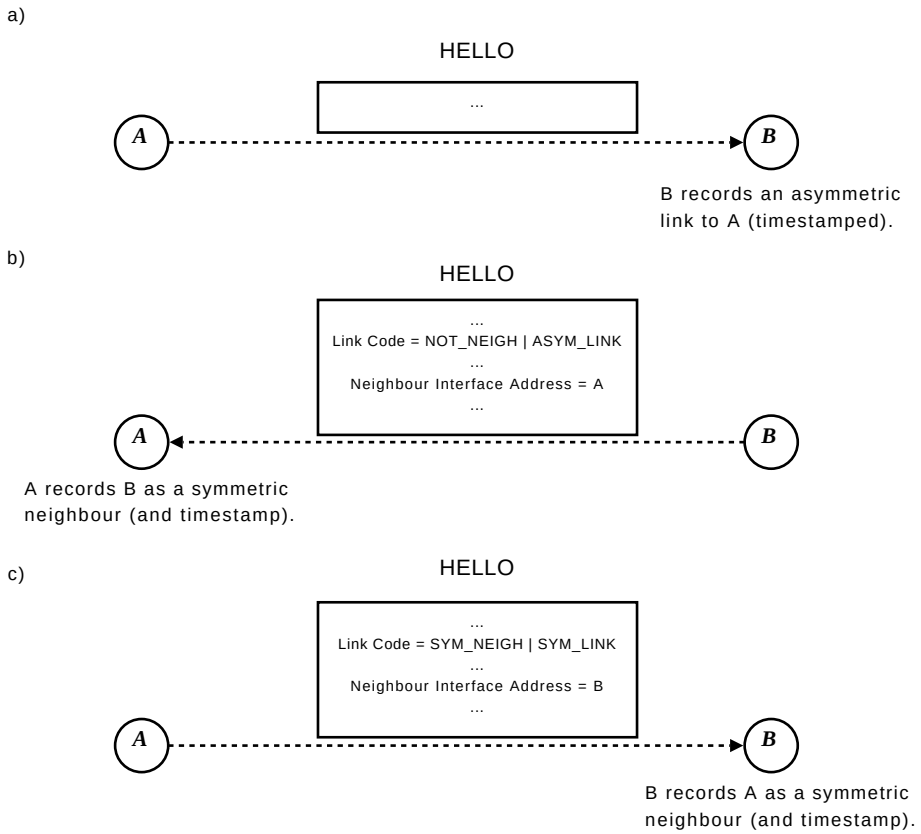

Figure 4: Neighbour detection in OLSR.

seen as follows.

Firstly, the sender will reach all neighbours directly. The MPRs of the sender will then forward the message to all the 2-hop neighbours — they were selected in this way by the sender. The 3-hop neighbours will be reached because the MPRs of the sender have selected MPRs from the 2-hop neighbours such that all 3-hop neighbours can be reached through these. This continues to all the nodes in the network.

Thus, allowing only MPRs to do the forwarding during a broadcast reduces the number of times the message is forwarded, while it still assures that all nodes in the network receives the message.

2.2.6 Disseminating network topology

As explained in Section 2.2.1, messages are routed in accordance with a routing table computed locally in each node. Obviously, this routing table is

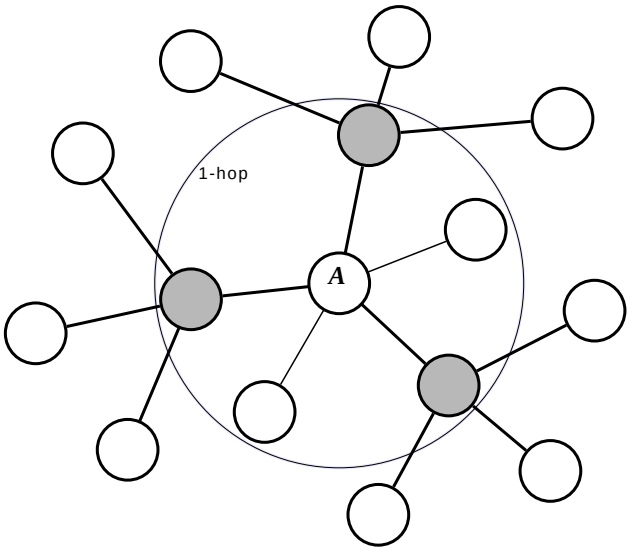

Figure 5: Multipoint relays.

computed from the topology of the network. As we have seen, information about neighbours and 2-hop neighbours can be obtained directly by receiving `HELLO` messages. Information about other nodes in the network is obtained by receiving Topology Control (`TC`) messages. The format of a `TC` message is shown in Figure 6.

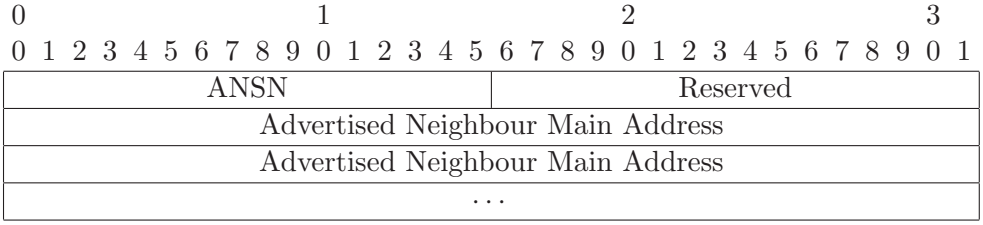

Figure 6: TC message format.

Every node may generate `TC` messages and broadcast them using MPRs as discussed above. The *ANSN* field is merely a sequence number, incremented every time the sender broadcasts a new `TC` message. A *Advertised Neighbour Main Address* (*ANMA*) field contains the main address[5] of a neighbour of the sender.

In the *ANMA* fields, a node is only required to include the addresses of

[5]The main address concept is explained in Section 2.2.7.

nodes that have selected it as an MPR, but it may include all its neighbours. Therefore, it is usually only the MPRs that generate `TC` messages. Essentially, what the sender of a `TC` message informs, is that all the nodes listed in the *ANMA* fields can be reached through it.

Thus, when other nodes in the network receive this message, they might construct their routing table such that the creator of the `TC` message is used for forwarding messages to other nodes. As a result, MPRs are usually, in addition to being used for forwarding broadcast messages, also being used to forward unicast messages.

2.2.7 Multiple interfaces

Nodes may have multiple network addresses (i.e. multiple radio transmitters) within the same OLSR network. To support this, a node must choose one of its addresses as the *main address*. Only the main address is allowed to be used as the *Originator Address* in an OLSR packet header (see Figure 2). Next, the node must inform the other nodes in the network of all the addresses it uses. It does this by creating and broadcasting a Multiple Interface Declaration message. This message merely contains all the addresses of the node. As a result, other nodes know that these addresses belong to the same node and that the node has the main address as specified by the *Originator Address* in the OLSR packet header.

2.2.8 Extensions

We have now covered all that is required for an OLSR network to operate — known as the *core functionality* in the OLSR standard [19]. However, it is worthwhile to note that OLSR may be extended with additional message types, also when some nodes do not know how to handle these new message types. As long as a node supports the core functionality, it will forward these new message types correctly. Nodes that do understand the messages will receive and process them. This might prove useful for security mechanisms; for instance, a secured network may exist within an unsecured network. New message types may be defined to transmit keys or signatures.

2.3 Secure routing protocols

A number of proposals have been published within the subject of creating secure ad hoc routing protocols. We will discuss some popular representatives, in order to understand the mechanisms and ideas that are used in these.

But before we consider proposals for secure routing protocols, we will briefly discuss what it means for a routing protocol to be "secure".

2.3.1 Defining secure routing

Unfortunately, "secure" is a word that may have very different meanings in different topics, and even in publications within a specific topic. This is also true within the topic of routing protocols. A common way to define security of a routing protocol is to outline a taxomony of possible attacks against it, and define it to be secure if it can prevent these particular attacks. Of course, this definition of security does not give any guarantees of what happens if an attack that is not considered is being used.

However, from a holistic point of view, we want the routing protocol to be able to do its job: create and maintain valid routes, and transport application data through those routes. Thus, we are more concerned with the authenticity of routing packets than the confidentiality. Encryption on the network layer is only useful to hide the topology of the network, since sensitive application data should be encrypted end-to-end by the application itself[6]. The routing protocol should be robust, in the sense that it continues to function even if some nodes deviate from the protocol.

Andel and Yasinsac [3] have written an excellent survey on security analysis techniques in MANETs. Their work covers the issue of defining secure routing.

2.3.2 Authenticated Routing for Ad hoc Networks

Authenticated Routing for Ad hoc Networks (ARAN) is proposed by Sanzgiri et al. [59]. The paper presents possible security exploits against ad hoc routing protocols, and specifically details attacks against the Ad-hoc On-Demand Distance Vector Routing (AODV) [53] and Dynamic Source Routing (DSR) [40] protocols. The route discovery in ARAN is based on AODV and DSR, but the described attacks are mitigated by a number of additions to the protocol. See Section 2.1 for an introduction to route discovery in reactive routing protocols like AODV and DSR.

ARAN route discovery ARAN requires ROUTE REQUESTs and ROUTE REPLYs to be digitally signed. The goal is to achieve *authentication*, *integrity*

[6]The application should handle it by the *end-to-end argument* [57]. For example, the application can potentially run on top of different routing protocols, some of which may not behave as expected (e.g. decrypt packets in each node).

and *non-repudiation*. Authentication prevents spoofing of node identity, integrity ensures that the packet has not been modified since it was created and non-repudiation makes it possible to catch malicious insiders (i.e. nodes possessing a valid certificate and the corresponding key pair).

We will be using an example when describing ARAN more precisely. Imagine that node S wants to create a route to node D. The only intermediate nodes are A and B. So the route obtained in the end should be $S \rightarrow A \rightarrow B \rightarrow D$. First, node S creates a *route discovery packet* (RDP), the analogy of a ROUTE REQUEST, and broadcasts it to all nodes:

$$S \rightarrow all\text{: } \{RDP, IP_D, Cert_S, N_S, t\}_{sk_S}$$

The packet contains a packet identifier (RDP), the IP address of the destination (IP_D), and the public key certificate of the sender ($Cert_S$). Furthermore, a nonce[7] (N_S) and a timestamp (t) ensures the freshness of the packet, i.e. protects against replay attacks. This whole message is signed by the sender (S).

When this packet is received by an intermediate node (A), the intermediate node checks its correctness (e.g. certificate, signature, freshness, etc.). If it is correct, the intermediate node signs it and appends its own certificate. The next node that receives the packet (B) will again check it for correctness, including the signature of the previous node in the path (A). If everything is OK, it will *remove* the signature and certificate of the previous node, and append its own signature of the message together with its own certificate. So only the signature and certificate of the current node will be sent in addition to the packet itself, and signatures are checked and created by each node in the path.

If D receives the RDP, it creates and unicasts a reply packet (REP) along the reverse path:

$$D \rightarrow B\text{: } \{REP, IP_S, Cert_D, N_D, t\}_{sk_D}$$

The meaning of the fields in this packet are the same as the ones in the RDP packet. The same actions as described for the RDP packet, with signature checking and creation, are carried out by intermediate nodes.

A key point in ARAN is that it does not make use of hop counts[8] to select a route. If there are two or more possible routes from a source to

[7]Nonce stands for "number used once" and is usually a random number, or a counter that is incremented after it is used. This is a standard mechanism to protect against replay attacks, where an adversary simply resends a correct message at a later point in time.

[8]A *hop count* is a counter in the packet that is incremented by every node which the packet travels through. The destination node may then see the distance to the source, in terms of hops, if the intermediate nodes have behaved according to the protocol.

a destination, the route on which the *RDP* packet travels *fastest* on will be selected (i.e. the route where the *RDP* packet arrives first at the destination). This helps to prevent an attack where cooperating malicious nodes make every path seem shorter if it travels through them. In this attack, also known as a *tunneling attack*, a malicious node will encapsulate a ROUTE REQUEST and send it to another malicious node close to the destination for the ROUTE REQUEST. The other malicious node will decapsulate the ROUTE REQUEST and send it to the destination. The hop count of this ROUTE REQUEST will then seem to be very low, and this route has good chances of being selected if the hop count is used to measure the quality of the route.

Summary ARAN is a reactive routing protocol based on AODV and DSR. The main security enhancement over these protocols is achieved by signing routing packets. To manage the asymmetric keys, the standard approach of a `PKI` is taken, where it is assumed that all nodes have an authentic copy of the public key of a Certificate Authority (`CA`) and that the `CA` create certificates for nodes. Certificates are distributed in every *RDP* and *REP* packet. Certificate revocation is achieved by having a `CA` online to create revocation lists. These lists are distributed to all nodes, using best effort.

2.3.3 Ariadne

Ariadne [38] is a routing protocol based on DSR. This implies that a route is created and maintained only when data needs to be sent over it. A route is created by the source node by broadcasting a ROUTE REQUEST. When intermediate nodes receive this ROUTE REQUEST, they append their own address and rebroadcast it. The destination node responds with a ROUTE REPLY, which follows the reverse path of the ROUTE REQUEST. Ariadne enables the source and destination node to authenticate each other, but also the intermediate nodes. A key mechanism of this authentication is the use of the TESLA broadcast authentication protocol. We will therefore briefly discuss TESLA.

TESLA (Timed Efficient Stream Loss-tolerant Authentication) [54] is a broadcast authentication protocol, which means it allows the receiver of a broadcast message to authenticate all the nodes that have forwarded this particular broadcast message. Thus TESLA must support the ability for multiple nodes to authenticate data that is created by one node, which is the same function as a digital signature. The special thing about TESLA however, is that it utilises only *symmetric* cryptography.

TESLA is based on one-way *key chains*. A key chain is a list of keys, with the property that any key in the list can be used to compute all the keys *following* it in the list, but not the keys *preceding* it. As a concrete implementation, a node chooses a random *initial* key, K_0. All nodes agree on a publicly known one-way hash function $\mathcal{H}$, and timestamps in the future t_i, sorted reversed chronologically. The node chooses how many keys it wants to compute, n, and computes them as $K_i = \mathcal{H}(K_{i-1})$, for $i \in [1, n]$. Now the node must have authenticated channels to send K_n to all the other nodes. In addition, the clock reaches t_m, the node broadcasts K_m. Since the timestamps in the nearest future were the last ones in the list, the node first publishes K_{n-1} at time t_{n-1}, then K_{n-2} at time t_{n-2} and so on.

The idea is that when a node wants to authenticate data, it computes a *message authentication code* (`MAC`) over the data using a key K_m that is *not yet published*. The receivers of this data and `MAC` will then check that K_m is in fact not yet published based on the current time, and wait until it is published. When they receive K_m, they authenticate it by checking that $K_m = \mathcal{H}(K_{m+1})$ — they have already obtained K_{m+1} from the node. If the check succeeds, they will compute the `MAC` over the data with K_m as key, and check that it matches the `MAC` that they received. If it does, the data is successfully authenticated. The rationale behind this is that only the sender could have computed a correct `MAC` at the time the data was received, since it was the only one which knew the correct key then.

As TESLA utilises only symmetric cryptography, it is very efficient compared to digital signatures. A `MAC` is generally more efficient than a digital signature both in terms of computation and length. However these benefits come at a cost: the nodes must have clocks that are quite synchronised, all nodes must obtain an authentic copy of the last key in a key chain K_n and TESLA does not provide non-repudiation. But the second problem is also present with digital signatures: all nodes must have an authentic copy of the sender's public key. Nevertheless, with TESLA, an authentic copy of a new key $K'_{n'}$ must be distributed after all the n keys in the initial key chain are published, while there typically no need for a node to choose a new public key regularly.

Ariadne with TESLA Ariadne route discovery may use either TESLA, digital signatures or only `MAC`s. We will discuss only the former. Using TESLA, nodes must obtain an authentic copy of a key in the key chain of all other nodes, as explained above. Furthermore, all end-to-end communicating nodes, S and D, must share two symmetric keys, K_{SD} and K_{DS}. These two keys are used when computing `MAC`s over data that is sent between these

nodes: K_{SD} is used when S sends data to D, and K_{DS} is used when D sends to S.

The process of route discovery is shown in Figure 7. The source node S creates a ROUTE REQUEST and generates a `MAC` over it, using K_{SD}. Intermediate nodes (B and C) use a hash function to embed their own identity into the *Hash Chain* field. Furthermore, they append their addresses to the *Node List* field. Then they compute a `MAC` over this modified ROUTE REQUEST, using a not yet published key from the TESLA protocol, and append it to the *MAC List*. When the destination node (D) receives the ROUTE REQUEST, it checks that the TESLA keys used by intermediate nodes have not yet been published and recomputes the *Hash Chain* field and checks it correctness. If everything is correct, it creates a ROUTE REPLY which also contains the *Node List* and *MAC List* from the ROUTE REQUEST. It then computes a `MAC` over the whole ROUTE REPLY, using K_{DS}, and sends it back the reverse route. Intermediate nodes add the TESLA key they used to compute the `MAC` in the ROUTE REQUEST (waiting until they can publish it, if required) and forward the packet. As the source node (S) receives the ROUTE REPLY, it checks that the TESLA keys used by intermediate nodes are valid based on previous keys it has received from these nodes, as discussed in the TESLA paragraph. It then uses K_{DS} to check the `MAC` computed by D (*Target MAC*), and checks the `MAC`s in the *MAC List*, which were computed by intermediate nodes over the ROUTE REQUEST, by using the keys in the *Key List*. If every check succeeds, it accepts the ROUTE REPLY.

When a packet cannot be forwarded by an intermediate node because the next hop in the route is not available (e.g. has moved), a ROUTE ERROR packet is returned to the sender. This packet is also authenticated by the sender using TESLA.

Another point we should take away from Ariadne is how it handles non-forwarding or only partly forwarding nodes. In Ariadne, an intermediate node that knows multiple routes to the destination will forward a fraction of the data on every route. It is assumed that acknowledgements are sent back when a packet is successfully received by the destination (e.g. ACK in TCP). The nodes will forward more data on the routes on which a high fraction of acknowledgements on sent packets are received. If a malicious non-forwarding node is part of a route, the other intermediate nodes will thus attempt to use different routes.

Summary As Ariadne is based on DSR, it is a reactive routing protocol. By using TESLA, all nodes must send authentic TESLA keys K_n to all other nodes, and this must be repeated when all n keys have expired. End-

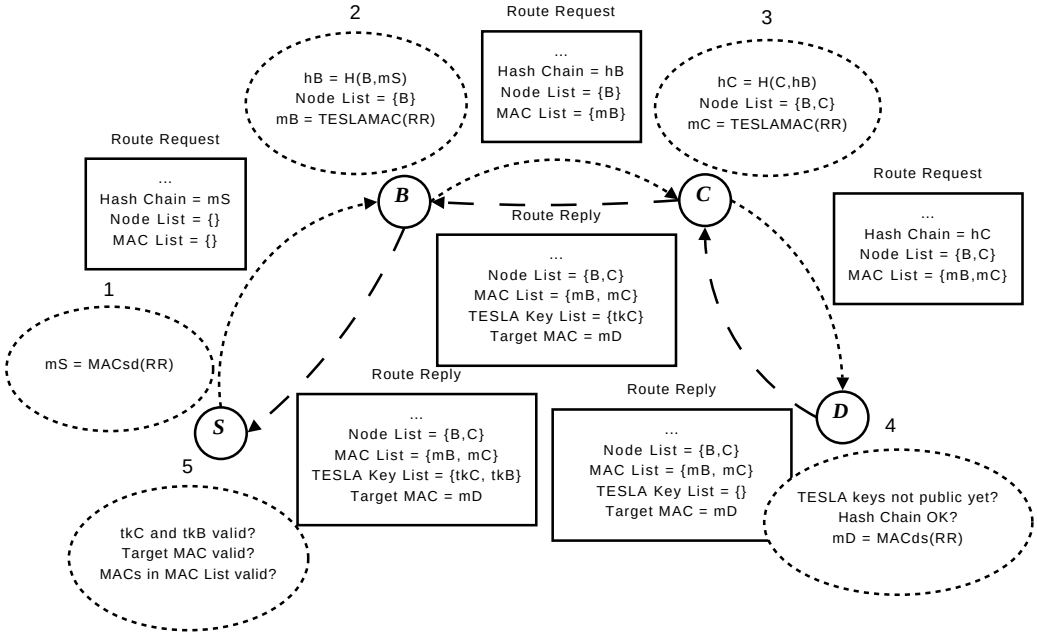

Figure 7: Route discovery in Ariadne.

to-end communicating nodes (potentially all) must exchange authentic keys K_{DS} and K_{SD}. Furthermore, the ROUTE REQUEST and ROUTE REPLY messages grow with the path length. TESLA also requires clocks to be well synchronised.

If used with only symmetric cryptography, Ariadne is efficient, both in terms of computing power and message sizes. The messages will be smaller than if asymmetric cryptography is used because `MAC`s are generally smaller than signatures. But much keying material must be exchanged and stored by each node: $3N^2$ keys if N is the network size. This limits the scalability of Ariadne when used with only symmetric cryptography. Asymmetric cryptography can be utilised by exchanging TESLA with digital signatures.

2.3.4 Security Protocols for Sensor Networks

Security Protocols for Sensor Networks (SPINS) [55] consists of two protocols: SNEP and μTESLA. SPINS requires very little computing power and network bandwidth by utilising only symmetric cryptography.

SPINS assumes that every node can reach a base station and that a base station will not be compromised. Each node shares a *master key* with the base station, K_M. All other keys are derived from the master key.

SNEP The Sensor Network Encryption Protocol (SNEP) provides point-to-point confidentiality, authentication and freshness. It relies on a shared counter between the two communicating nodes in order to offer semantic security[9]: the counter C is shared and incremented each time it is used. The master key K_M is used to derive two keys: the *encryption key*, K_{enc}, and the *MAC key*, K_{MAC}. The encrypted message node A sends to node B is on the following form:

$$A \rightarrow B\text{: } \{D\}_{\{K_{enc},C\}}, MAC(K_{MAC}, C \mid \{D\}_{\{K_{enc},C\}})$$

The message contains two components. The first is the plaintext D symmetrically encrypted with the key K_{enc} and the counter C. The counter is used in the *CTR* block cipher mode[10] to create a semantically secure encryption. The second component is created by first concatenating the counter with the first component and computing the `MAC` over this concatenation, using the `MAC` key.

The nodes may securely agree on a counter C by creating an encrypted and authenticated message containing it, using K_{enc} and K_{MAC}. The reason for not sending the counter as part of the message is to save on the message length. Note that only the base station knows the master key K_M of a node. Therefore, if two nodes want to communicate securely, the base station must be involved to send a shared key to the two nodes, by using SNEP.

μTESLA Like TESLA (see Section 2.3.3), μTESLA provides authentication for data broadcast. μTESLA is very similar to TESLA, but emphasises on low communication overhead and little computing power.

As a fist distinction, it defines a way for nodes to obtain an authentic copy of a key in the key chain of another node using only symmetric cryptography. The sender computes a `MAC` over the μTESLA key, by using K_M (details are omitted).

Second, the nodes may do broadcasting through the base station. The nodes use SNEP to communicate securely with the base station, and the base station broadcasts the data to all the nodes. Alternatively the nodes may do the broadcasting themselves, but rely on the base station to store distribute the keys in all key chains, saving memory and bandwidth in each broadcasting node. Thus a node needing a μTESLA key will contact the

[9] *Semantic security* is a strong confidentiality property: an adversary cannot derive any information about a plaintext by seeing only the ciphertext. For example, if the same plaintext is encrypted twice, the adversary cannot see from the resulting ciphertexts that the plaintexts are equal.

[10] Refer to for example Ferguson and Schneier [25] for a description of the *CTR* block cipher mode.

base station, and the base station will send the key, if the node is allowed to receive it yet.

Summary SPINS is designed for nodes which have extremely little computing and battery power. Nodes rely on a base station to save memory and minimise the amount of data they need to send. Each node share a symmetric key with the base station. If a node needs to communicate with another node, it initiates a protocol with the base station to establish a shared key with the other node. Furthermore, the base station is needed during a broadcast. Thus, the base station is a single point of failure: it is required for forwarding, to establish secure channels between nodes, and can decrypt all network traffic.

In a sensor network, which SPINS is designed for, this may be a good trade-off. However, in a MANET, it will probably be better to make the nodes do more computation (e.g. by allowing asymmetric cryptography) in order to relax the assumptions on the base station.

Furthermore, SPINS is not a routing protocol per se, as it assumes that nodes can reach a base station. However, it might be generalised by having nodes act as base stations and relay data to other nodes acting as base stations.

2.3.5 Securing OLSR

Adjih et al. [1] discuss security threats against the OLSR protocol, and suggest OLSR extensions to alleviate some of these attacks. Insider nodes, i.e. nodes possessing a cryptographic key corresponding to the system (e.g. signed by a trusted `CA`), are assumed not to be compromised.

Vulnerabilities Recall that `HELLO` and `TC` messages comprise the control traffic in OLSR. From an abstract point of view, a node in an OLSR network has two responsibilities:

1. Correctly generate control traffic
2. Correctly forward control and data traffic

Consider responsibility 1 with respect to `HELLO` messages (see Section 2.2.3). The first type of attack is to spoof the identity, i.e. the sender claims to be another node. The second type of attack is to send an incomplete list of neighbours or include nodes that are not neighbours in the list.

Regarding responsibility 1 and `TC` messages (see Section 2.2.6), the sender may here too do identity spoofing. The other type of attack is to send an

incomplete list of MPR selectors or include nodes that are not neighbours in the list.

Consider responsibility 2. Two types of attacks exist here too. One possibility is to alter control or data messages in a way that conflicts with the protocol before forwarding them. Only *Time To Live* and *Hop Count* is allowed to be changed by a forwarding node (see Figure 2), and these fields should be decremented and incremented, respectively. The second type of attack is to not forward all packets that should be forwarded.

Security mechanisms Adjih et al. propose an additional OLSR message type (see Section 2.2.8), a signature message, for transmitting a timestamp[11] and signature. A control message without a corresponding signature message will be rejected. Note that the original OLSR protocol already has mechanisms for rejecting old control messages: the *Message Sequence Number* field in the OLSR packet (see Figure 2) and the *ANSN* field in the `TC` message (see Figure 6). However, these mechanisms are not considered good enough for protecting against replay attacks because sequence numbers are only 16 bits and thus reused too often.

The signature message protects against identity spoofing, including replay, in responsibility 1. It also protects against unauthorised modification of control traffic, with the exception of the *Time To Live* and *Hop Count* fields. Since the insider nodes are assumed to behave according to the protocol, control messages will be generated correctly, and traffic is forwarded correctly.

Summary Extending OLSR with a signature message efficiently counters external attackers (i.e. attackers not holding a cryptographic key corresponding to the system). However, trusting the insider nodes completely for the whole lifetime of the system might be unrealistic in practice. There exist mechanisms that detect insider attacks in OLSR as well [67].

2.3.6 Summary

The secure routing protocols discussed are summarised in Table 2. The "Keying material" column states how many keys must be stored in total, where N is the number of nodes in the network, $\mathcal{C}$ means certificate and $\mathcal{S}$ means symmetric key.

[11] Note that a timestamp does not need to be real time. Logical time, a monotonically increasing counter, is also applicable.

Name	Route discovery	Key mgm.	Key dist.	Keying material	Crypt. mech.
ARAN	reactive	PKI	Append cert.	$N \cdot \mathcal{C}$	signature, nonce, timestamp
Ariadne	reactive	preest. or PKI	preest.	$3N^2 \cdot \mathcal{S}$ or $N \cdot \mathcal{C}$	MAC, hash chain, key expiry
SPINS	base station	preest.	preest.	$2N \cdot \mathcal{S}$	MAC, nonce, enc., key expiry
Secure OLSR	proactive	preest. or PKI	preest. or online CA	$N \cdot \mathcal{C}$ (PKI)	signature, timestamp

Table 2: Comparison of secure routing protocols.

All the protocols assume preestablished keys, either symmetric keys or the public key of a Certificate Authority. Authentication is the main concern of the secure routing protocols. This results in that MACs and signatures are common security mechanisms. Additionally, the order of events is important (e.g. to protect against replay), which leads to timestamps, nonces and expiry techniques also being frequently used mechanisms. None of the protocols use encryption in the routing protocol (SPINS use SNEP to establish a secure channel end-to-end, not to hide network topology).

The protocols essentially partition nodes into two sets: trusted and untrusted. The nodes that are able to prove possession of a secret key (e.g. by computing a MAC) are trusted and allowed to take part of the routing. This model is a simplification of real life since trusted nodes may be captured by an adversary. Thus, a secure routing protocol should include or be coupled with *intrusion detection* mechanisms that try to distinguish honest holders of a secret from compromised ones. Additionally, when a node is known to be compromised, a mechanism for removing that node from the network should be available. If a PKI is used, a certificate revocation list[12] is a possibility for achieving this.

The vast majority of secure routing protocols proposed are reactive, as opposed to proactive. For example, in a survey discussing twelve secure routing protocols [4], ten are reactive. Proactive routing protocols, and OLSR

[12] A certificate revocation list is a list of certificates that are revoked. This list is usually constructed and signed by the CA.

in particular, have not been researched in a security context to that extent.

3 Pairings and Elliptic Curve Cryptography

3.1 Mathematical aspects of pairings

Elliptic curves will play a central role for us during the discussions in this Section. Loosely speaking, an elliptic curve is the zeroes of a polynomial. Such a polynomial is often written as $Y = X^3 + aX + b$, where $a, b \in \mathbb{F}$ for some field $\mathbb{F}$ [13]. It is possible to define a method for adding two zeroes (x_0, y_0) and (x_1, y_1), such that the sum is also a zero, and in this way create a group structure. It turns out that this group structure is well suited for cryptographic applications, since the discrete logarithm problem (DLP) seems to be especially hard in such groups. In practical applications, this means that one may use shorter keys, but still attain the same security level, which increases efficiency. Many cryptographic algorithms that are based on the hardness of the DLP in a multiplicative group of a finite field may use elliptic curves instead to enjoy these benefits. Examples include Elliptic Curve Diffie-Hellman, and Elliptic Curve Digital Signature Algorithm. There are quite a few introductory text on elliptic curves, for instance Nagaraj and Sury [49], and Silverman [63].

Pairings are functions with some special characteristics, that may be defined on elliptic curves. We will discuss pairings and properties of them that will be of importance to us. In addition, we will consider how pairings can be efficiently implemented on devices with limited computing power, such as the nodes in a MANET.

Notation When describing algorithms, we often need to draw elements uniformly at random from a set. We will henceforth use the symbol $\in_R$ to denote this operation. For example, choosing an integer r, less than 42, uniformly at random is denoted $r \in_R \mathbb{Z}_{42}$.

3.1.1 Preliminary definitions

In order to discuss pairings, we will need some standard definitions related to elliptic curves

Definition 3.1. Let E be an elliptic curve over a finite field $\mathbb{F}_q$, with $q = p^s$ and p prime. Then E is *supersingular* if one of the following equivalent conditions holds:

1. $\#E(\mathbb{F}_q) \equiv 1 \pmod{p}$

[13] The polynomial $X^3 + aX + b$ must have distinct roots (to define a *non-singular* curve), which creates some restrictions on the choice of a and b.

2. E has no points of order p over $\overline{\mathbb{F}}_q$.

3. The endomorphism ring of E over $\overline{\mathbb{F}}_q$ is non-commutative.

If E is not supersingular, then it is called *ordinary*.

Definition 3.2. Let E be an elliptic curve defined over $\mathbb{F}_q$. Let $n \in \mathbb{Z}$, with $n \mid \#E(\mathbb{F}_q)$ and $\gcd(n, q) = 1$. Define $\mu_n = \{x \in \overline{\mathbb{F}}_q^* \mid x^n = 1\}$, the set of nth roots of unity of $\overline{\mathbb{F}}_q$. The field $\mathbb{F}_q(\mu_n)$ is some finite extension $\mathbb{F}_{q^k}$ of $\mathbb{F}_q$. The integer k is called the *embedding degree* (or *security multiplier*).

Equivalently, k is the smallest integer such that $n \mid (q^k - 1)$. Thus, the embedding degree k is a function of n and q.

Definition 3.3. Let E be an elliptic curve defined over $\mathbb{F}_q$, $n \mid \#E(\mathbb{F}_q)$, and k be the embedding degree corresponding to n and q. Let $P = (x, y) \in E(\mathbb{F}_{q^k})$. The *trace map*

$$Tr : E(\mathbb{F}_{q^k}) \to E(\mathbb{F}_q), \text{ is defined by}$$
$$Tr(P) = \sum_{i=0}^{k-1} (x^{q^i}, y^{q^i})$$

where the sums are elliptic curve point additions.

Proposition 3.4. *The trace map is a group homomorphism.*

Proof. Let $P, Q \in E(\mathbb{F}_{q^k})$ and write $P = (x_1, y_1), Q = (x_2, y_2)$. Recall that the Frobenius endomorphism is a map $\phi : E(\mathbb{F}_{q^k}) \to E(\mathbb{F}_{q^k})$, defined by $\phi(P) = (x_1^q, y_1^q)$. Thus, evaluating the trace map at P can be written as $Tr(P) = P + \phi(P) + \phi^2(P) + \cdots + \phi^{k-1}(P)$. By using the homomorphic properties of the Frobenius endomorphism, we have

$$\begin{aligned} Tr(P+Q) &= P + Q + \phi(P+Q) + \phi^2(P+Q) + \cdots + \phi^{k-1}(P+Q) \\ &= P + \phi(P) + \phi^2(P) + \cdots + \phi^{k-1}(P) \\ &\quad + Q + \phi(Q) + \phi^2(Q) + \cdots + \phi^{k-1}(Q) \\ &= Tr(P) + Tr(Q) \end{aligned}$$

□

3.1.2 Pairings

Our pairing definition will be a bit more strict than common definitions [8]. We require that the groups involved have prime order r to avoid certain

weaknesses in the cryptographic schemes we construct later[14]. Furthermore, prime order groups allows us to use pairings to separate the CDH and DDH problems, as explained in Section 3.1.3. In order for a pairing to be applicable in practice, we also require that the pairing is efficiently computable.

Definition 3.5. Let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_3$ be groups with prime order r. A *pairing* is a map

$$e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_3$$

with the following properties:

- **Bilinear**
  $e(aP,bQ) = e(P,Q)^{ab}$, $\forall\ P \in \mathbb{G}_1,\ \forall\ Q \in \mathbb{G}_2$ and $\forall\ a, b \in \mathbb{Z}$.
- **Non-degenerate**
  - $\forall\ P \neq O \in \mathbb{G}_1,\ \exists\ Q \in \mathbb{G}_2$ such that $e(P,Q) \neq\ 1$.
  - $\forall\ Q \neq O \in \mathbb{G}_2,\ \exists\ P \in \mathbb{G}_1$ such that $e(P,Q) \neq\ 1$.
- **Computable**
  For all $P \in \mathbb{G}_1, Q \in \mathbb{G}_2$, there is an efficient algorithm to compute $e(P,Q)$.

The following proposition shows that we often obtain a non-trivial pairing computation.

Proposition 3.6. *Let e be a pairing and $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_3$ groups as in Definition 3.5, and let $P \in \mathbb{G}_1, Q \in \mathbb{G}_2$. If $P \neq O$ and $Q \neq O$, then $e(P,Q) \neq 1$.*

Proof. Since $Q \neq O$ and $\#\mathbb{G}_2$ is prime, $\langle Q \rangle = \mathbb{G}_2$. Because e is non-degenerate, $\exists k \in \mathbb{Z}^+$ such that $e(P,kQ) \neq 1$ (note that $k \neq 0$ since $e(P,O) = 1$). But if $e(P,Q) = 1$, then $e(P,Q)^k = e(P,kQ) = 1$, which is a contradiction. □

3.1.3 Computational problems

Any secure asymmetric cryptographic system must depend on the hardness of at least one computational problem. The problems discussed next form the fundament for many pairing based cryptographic schemes. In all the definitions below, $a, b, c \in \mathbb{Z}$.

[14]Most notably, the Silver-Pohlig-Hellman algorithm can be used to efficiently solve the DLP in a group by solving it in some of its subgroups. However, if the group has prime order, it will not have any non-trivial subgroups, which renders the Silver-Pohlig-Hellman algorithm useless.

Definition 3.7. Let E be an elliptic curve defined over a finite field $\mathbb{F}_q$ and let $P, Q \in E(\mathbb{F}_q)$, with $Q \in \langle P \rangle$. The *elliptic curve discrete logarithm problem* (ECDLP) is to find the smallest non-negative $l \in \mathbb{Z}$ such that $Q = lP$.

Definition 3.8. Given a group $\mathbb{G}$ and $P, aP, bP \in \mathbb{G}$, the *Computational Diffie-Hellman* (CDH) problem is to compute abP.

Definition 3.9. Given groups $\mathbb{G}_1$ and $\mathbb{G}_2$, let $P \in \mathbb{G}_1$ and $Q, aQ \in \mathbb{G}_2$. The *Computational co-Diffie-Hellman* (co-CDH) problem on $(\mathbb{G}_1, \mathbb{G}_2)$ is to compute $aP \in \mathbb{G}_1$.

Definition 3.10. Given a group $\mathbb{G}$ and $P, aP, bP, cP \in \mathbb{G}$, the *Decision Diffie-Hellman* (DDH) problem is to decide if $cP = abP$.

Definition 3.11. Given groups $\mathbb{G}_1$ and $\mathbb{G}_2$, let $P, aP \in \mathbb{G}_1$ and $Q, bQ \in \mathbb{G}_2$. The *Decision co-Diffie-Hellman* (co-DDH) problem is to decide if $a = b$.

Definition 3.12. Given groups $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_3$, with $\mathbb{G}_1 = \mathbb{G}_2$, and the pairing e as in Definition 3.5, let $P, aP, bPcP \in \mathbb{G}_1$. The *Bilinear Diffie-Hellman* (BDH) problem is to compute $e(P, P)^{abc}$.

Definition 3.13. Given groups $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_3$ and the pairing e as in Definition 3.5, let $P, aP, bP \in \mathbb{G}_1$ and $Q, aQ, cQ \in \mathbb{G}_2$. The *Bilinear co-Diffie-Hellman* (co-BDH) problem is to compute $e(P, Q)^{abc}$.

All these definitions only informally lead to security assumptions. A security assumption states that any polynomial-time algorithm will have only a negligible probability of obtaining a correct solution to the problem. In addition, the probability will be lower as the parameter sizes (e.g. group orders) increase.

Note that the (co-)BDH problem would become trivial if $e(P, Q) = 1$, which is one reason why we try to avoid the trivial result. But in addition, if $e(P, Q) \neq 1$, pairings may be used to solve the DDH and co-DDH. The co-DDH may then be solved using pairings as follows: check if $e(P, bQ) = e(aP, Q)$ in $\mathbb{G}_3$. If all the groups are of prime order r, the check will succeed if and only if $a \equiv b \pmod{r}$. This gives us the ability to construct groups where the CDH (or co-CDH) is hard while the DDH (or co-DDH) is easy. Such groups are sometimes referred to as *Gap Diffie-Hellman groups* [14]. This construction is the main reason why pairings have become an important tool for cryptographers.

3.1.4 Pairings on Elliptic Curves

All known ways to construct pairings relies on abelian varieties. We will be using elliptic curves in our implementations. But there are three ways to select the group $\mathbb{G}_2$, as described by Galbraith, Paterson and Smart [29]. In the description of all types, k is the embedding degree, $\mathbb{G}_1$ is a subgroup of $E(\mathbb{F}_q)$ of order r, and $\mathbb{G}_3$ is a subgroup of $E(\mathbb{F}^*_{q^k})$ of order r.

1. $\mathbb{G}_1 = \mathbb{G}_2$
   This type uses supersingular elliptic curves. The pairing is computed as $e(P, \phi(Q))$ in this case, where ϕ is a distortion map (as described below).

2. $\mathbb{G}_1 \neq \mathbb{G}_2$ **and there is an efficiently computable homomorphism** $\phi : \mathbb{G}_2 \rightarrow \mathbb{G}_1$
   The elliptic curve is ordinary (i.e. not supersingular). Pick a random element $Q \in_R E(\mathbb{F}_{q^k})$ of order r, and set $\mathbb{G}_2 = \langle Q \rangle$. The homomorphism ϕ is the trace map.

3. $\mathbb{G}_1 \neq \mathbb{G}_2$ **and there is no efficiently computable homomorphism between** $\mathbb{G}_1$ **and** $\mathbb{G}_2$
   The elliptic curve is ordinary, and $\mathbb{G}_2$ is the kernel of the trace map[15].

As discussed in Section 3.1.3, it is often important that the pairing does not frequently map to the identity in $\mathbb{G}_3$ (e.g. it renders the BDH problem trivial in this case). For a type 1 pairing instantiation, which uses supersingular curves, we may actually enforce that $e(P, P) \neq O, \forall P \in \mathbb{G}_1$. This is achieved by using a *distortion map*, as defined by Verheul [65]. A distortion map ϕ is a non-rational endomorphism of E, with the property that $e(P,\phi(P)) \neq O$, if $\phi(P) \notin E(\mathbb{F}_q)$ (see Blake, Seroussi and Smart [8] for details). A distortion map can be defined for supersingular elliptic curves with all possible embedding degrees. Thus, we may use $\hat{e}(P,Q) = e(P,\phi(Q))$ in this case, which is called a *modified pairing*.

3.1.5 Implementations of pairings

The most well known pairings used in elliptic curve cryptography are the Weil and Tate pairings, which we will discuss next. But researchers have also constructed new pairings that are more efficient to compute, for example the χ-based Ate pairing [2].

[15] This makes the trace map trivial on $\mathbb{G}_2$, and this is why we have no efficiently computable homomorphism.

Before considering the pairings, we will briefly discuss divisors. Throughout this discussion, let E be an elliptic curve and K be a field. A *divisor* D is a formal sum $D = \sum_{P \in E(K)} a_P(P)$, where $a_P \in \mathbb{Z}$ and $a_P \neq 0$ for only a finite number of a_P. The *degree* of a divisor D is $\sum_{P \in E(K)} a_P$, and its *support* is the set of all points P with $a_P \neq 0$. A divisor of a non-zero function f, written as (f), is $\sum_{P \in E(K)} ord_P(f)(P)$, where $ord_P(f)$ is the order of f at P. A *principal divisor* is a divisor which is equal to (f), for some function f. Two divisors D, D' are said to be *equivalent*, written $D \sim D'$, if $D - D'$ is a principal divisor. We define the value of a function f at a divisor D as $f(D) = \prod_P f(P)^{a_P}$.

To simplify our discussion of the Weil and Tate pairings slightly, we will only consider the pairings over finite fields, which is the only interesting case for us. For the discussion of the pairings, let E be an elliptic curve defined over a finite field $\mathbb{F}_q$, where $q = p^s$, p is prime and $s \in \mathbb{Z}^+$. Define $n \in \mathbb{Z}$, with $\gcd(n, q) = 1$.

Weil pairing Let $K = \mathbb{F}_q(E[n])$, the field extension of $\mathbb{F}_q$ generated by the coordinates x, y of the points $P = (x, y) \in E(\overline{\mathbb{F}}_q)$ with $nP = O$. The set of nth roots of unity of K^* is denoted μ_n, so $\mu_n = \{x \in K^* \mid x^n = 1\}$.

Let $P, Q \in E[n]$, and let D_P and D_Q be divisors with disjoint support such that $D_P \sim (P) - (O)$ and $D_Q \sim (Q) - (O)$. D_P may be constructed by choosing an arbitrary point $S \in_R E(\mathbb{F}_{q^k})$ and setting $D_P = (P + S) - (S)$ (the same applies for D_Q). Since $nP - nO = nQ - nO = O$ and $n - n = 0$, nD_P and nD_Q are principal divisors, so let f_P and f_Q be functions with $(f_P) = nD_P$ and $(f_Q) = nD_Q$. The *Weil pairing* of P and Q is

$$\omega_n : E[n] \times E[n] \to \mu_n, \text{ defined by}$$
$$\omega_n(P, Q) = \frac{f_P(D_Q)}{f_Q(D_P)}$$

Miller's algorithm (discussed below) can be used to construct the functions f_P and f_Q in polynomial time. Refer to Blake, Seroussi and Smart [8] for a discussion on the Weil pairing with respect to the other properties we required by a pairing in Definition 3.5.

Tate pairing Let μ_n be the set of nth roots of unity of $\overline{\mathbb{F}}_q^*$, so $\mu_n = \{x \in \overline{\mathbb{F}}_q^* \mid x^n = 1\}$. If we let k be the embedding degree corresponding to q and n, we have $\mathbb{F}_{q^k} = \mathbb{F}_q(\mu_n)$. We define $E(\mathbb{F}_{q^k})[n] = \{P \in E(\mathbb{F}_{q^k}) \mid nP = O\}$ and $nE(\mathbb{F}_{q^k}) = \{nP \mid P \in E(\mathbb{F}_{q^k})\}$. The set of elements of $\mathbb{F}_{q^k}^*$ raised to the nth power is written $(\mathbb{F}_{q^k}^*)^n$, so $(\mathbb{F}_{q^k}^*)^n = \{x^n \mid x \in \mathbb{F}_{q^k}^*\}$.

Let $P \in E(\mathbb{F}_{q^k})[n]$ and $Q \in E(\mathbb{F}_{q^k})$. Let D_P be a divisor with $D_P \sim (P)-(O)$. Since $nP = O$ and $n-n=0$, nD_P is a principal divisor, so let f_P be a function with $(f_P) = nD_P$. Let D_Q be a divisor defined over $\mathbb{F}_{q^k}$ with $D_Q \sim (Q)-(O)$ and disjoint support of D_P. The *Tate pairing* of P and Q is

$$\tau_n : E(\mathbb{F}_{q^k})[n] \times E(\mathbb{F}_{q^k}) \to \mathbb{F}^*_{q^k}/(\mathbb{F}^*_{q^k})^n, \text{ defined by}$$
$$\tau_n(P,Q) = f_P(D_Q)$$

However, for practical purposes, we would like unique representatives of the equivalence classes $\mathbb{F}^*_{q^k}/(\mathbb{F}^*_{q^k})^n$. A simple way to achieve this is shown by the following proposition.

Proposition 3.14. *Let $a, b \in \mathbb{F}^*_{q^k}$. Then $a(\mathbb{F}^*_{q^k})^n = b(\mathbb{F}^*_{q^k})^n$ if and only if $a^{(q^k-1)/n} = b^{(q^k-1)/n}$.*

Proof. 1. Assume that $a \in b(\mathbb{F}^*_{q^k})^n$, that is, a and b are representatives of the same coset. This implies that $\exists c \in \mathbb{F}^*_{q^k}$ such that $a = bc^n$. By raising both sides of this equation to $(q^k-1)/n$, we obtain the desired result;

$$a^{(q^k-1)/n} = (bc^n)^{(q^k-1)/n} = b^{(q^k-1)/n}$$

2. Recall that $\mathbb{F}^*_{q^k}$ is cyclic, and let g be a generator. Write $a = g^s$ and $b = g^t$ for some $s, t \in \mathbb{Z}$. From our assumption, $a^{(q^k-1)/n} = b^{(q^k-1)/n}$, so $g^{s(q^k-1)/n} = g^{t(q^k-1)/n}$. This implies that

$$s\frac{q^k-1}{n} \equiv t\frac{q^k-1}{n} \pmod{q^k-1}$$
$$s \equiv t \pmod{n}$$
$$s = t + kn, \text{ for some } k \in \mathbb{Z}$$

Thus, $a = g^s = g^{t+kn} = b(g^k)^n$.

□

Thus, by raising the values of the Tate pairing to $(q^k-1)/n$, we obtain unique representatives: the nth roots of unity of $\mathbb{F}^*_{q^k}$.

Again, Miller's algorithm can be used to construct the function f_P. A proof of bilinearity is given by Blake, Seroussi and Smart [8], while non-degeneracy is shown by Frey and Rück [27].

Miller's algorithm In order to compute the Weil or Tate pairings, we would like to construct a function f_P with the property that $(f_P) = nP - nO$, for some point P. However, we do not need an explicit expression of f_P in either of the pairings, but rather the evaluation $f_P(D_Q)$, for some divisor $D_Q \sim (Q) - (O)$. Miller's algorithm [48] computes this evaluation step-wise, by using a method for doubling and adding an intermediate value, much like the square-and-multiply method for computing powers of an integer.

Write $(f_i) = i(P) - (iP) - (i-1)(O)$. We will now consider how f_i can be constructed. Our goal is to arrive at f_n, because $(f_n) \sim (f_P)$. We can take $f_1 = 1$, so we will look at how to construct f_{i+j} given f_i and f_j. Let l_1 and l_2 be the lines used in the computation of $iP + jP$. Then

$$\begin{aligned}(l_1) &= (iP) + (jP) + (-(i+j)(P)) - 3(O)\\ (l_2) &= (-(i+j)P) + ((i+j)P) - 2(O)\\ \left(\frac{l_1}{l_2}\right) &= (iP) + (jP) - ((i+j)P) - (O)\end{aligned}$$

Next, consider the divisor of $f_i f_j \frac{l_1}{l_2}$;

$$\begin{aligned}\left(f_i f_j \frac{l_1}{l_2}\right) &= i(P) - (iP) - (i-1)(O) + j(P) - (jP) - (j-1)(O)\\ &\quad + (iP) + (jP) - ((i+j)P) - (O)\\ &= (i+j)(P) - ((i+j)P) - (i+j-1)(O)\end{aligned}$$

From this, we can see that $(f_{i+j}) = (f_i f_j \frac{l_1}{l_2})$. Pick a random point $S \in_R E(\mathbb{F}^*_{q^k})$, and define $Q' = Q + S$ and the divisor $D_{Q'} = (Q') - (S)$. Note that $D_{Q'} \sim D_Q \sim (Q) - (O)$. At each step of the algorithm, we will calculate the evaluation of f_i at $D_{Q'}$, without finding f_i explicitly.

Miller's algorithm loops over the binary expansion of n, from the most significant to the least significant bit. For each iteration of the loop, a "double" is done. If the bit at the current bit-position of n is 1, then an "add" is also carried out. This is analogous to the square-and-multiply algorithm. During the computations, we keep track of $T = iP$.

Now assume that we have computed $f_i(D_{Q'})$, and we want to double to $f_{2i}(D_{Q'})$. From the discussion above, we know that we should find the lines l_1 and l_2 which are used in the computation of $2T$, and calculate;

$$f_{2i}(D_{Q'}) = f_i(D_{Q'})^2 \frac{l_1(D_{Q'})}{l_2(D_{Q'})} = f_i(D_{Q'})^2 \frac{l_1(Q')l_1(S)^{-1}}{l_2(Q')l_2(S)^{-1}} = f_i(D_{Q'})^2 \frac{l_1(Q')l_2(S)}{l_2(Q')l_1(S)}$$

For the case when the current bit position of n is 1, we want to compute $f_{i+1}(D_{Q'})$ from $f_i(D_{Q'})$. From the preceding discussion, this is the case where $j = 1$, so we should find the lines l_1 and l_2 used when computing $T + P$, and calculate;

$$\begin{aligned} f_{i+1}(D_{Q'}) &= f_i(D_{Q'})f_1(D_{Q'})\frac{l_1(D_{Q'})}{l_2(D_{Q'})} = f_i(D_{Q'})\frac{l_1(Q')l_1(S)^{-1}}{l_2(Q')l_2(S)^{-1}} \\ &= f_i(D_{Q'})\frac{l_1(Q')l_2(S)}{l_2(Q')l_1(S)} \end{aligned}$$

We continue in this fashion until we have iterated over all the bits in the binary representation of n and arrive at $f_n(D_{Q'}) = f_P(D_Q)$. The complete algorithm is stated in Figure 8.

`INPUT:` $P, Q \in E(K)$, with $| \langle P \rangle | = n$ for some field K
`OUTPUT:` $f_P(D_Q)$

1. Choose $S \in_R E(K)$ at random, and compute $Q' = Q + S$ [a]
2. Set $T = P$, $m = log_2(n) - 1$ and $f = 1$
3. while $m \geq 0$:
   - 3.1. Find the lines l_1 and l_2 used for doubling T
   - 3.2. $T = 2T$
   - 3.3. $f = f^2 \frac{l_1(Q')l_2(S)}{l_2(Q')l_1(S)}$
   - 3.4. If the mth bit of n is 1: [b]
     - 3.4.1. Find the lines l_1 and l_2 used for computing $T + P$
     - 3.4.2. $T = T + P$
     - 3.4.3. $f = f \frac{l_1(Q')l_2(S)}{l_2(Q')l_1(S)}$
   - 3.5. $m = m - 1$
4. Return f

[a] The algorithm may fail if Q' or S appear in any intermediate calculations. But since S is chosen randomly, the probability for this is small when K is large.

[b] We index the bits of n from right to left, starting with 0 (i.e. the least significant bit is bit 0).

Figure 8: Miller's algorithm.

The efficiency of a pairing computation depends on the number of iterations in Miller's algorithm. We see that for the Weil and Tate pairings, this is $log_2(n)$. The recently created and more efficient pairings [2] generally require less iterations than the Weil and Tate pairings.

3.1.6 Attacks using pairings

The security of a cryptographic system based on elliptic curves is clearly based on the assumption that the ECDLP is infeasible to solve in the given group. The best algorithm for solving the general ECDLP (i.e. without making any assumptions on the parameters of the elliptic curve) is the Pollard ρ, with exponential running running time. But for any DLP-based cryptographic system, one must ensure that the group order is a multiple of a large prime. This is to avoid an attack by the Silver-Pohlig-Hellman algorithm.

However, by using pairings, additional attacks on the ECDLP are made possible. The probably most important one is due to the MOV/Frey-Rück algorithm. This algorithm maps the ECDLP into a finite field DLP. The motivation for doing so is that the Index Calculus algorithm, with sub-exponential running time, may be used to solve the DLP in a finite field. Menezes, Okamoto and Vanstone demonstrated this by using the Weil-pairing [46], while Frey and Rück showed how to do it by using the Tate-pairing [26]. The algorithm is given in Figure 9.

`INPUT:` $P, Q \in E(\mathbb{F}_q)$, $\#E(\mathbb{F}_q) = r$, with r prime, such that $Q = l \cdot P$ for some unknown l

`OUTPUT:` l

1. Construct $\mathbb{F}_{q^k}$, where k is the embedding degree
2. Find[a] a point $S \in E(\mathbb{F}_q)$ such that $e(P,S) \neq 1$
3. $a = e(P,S)$
4. $b = e(Q,S)$
5. Use Index Calculus in $\mathbb{F}_{q^k}^*$ to compute l such that $a^l = b$
6. Return l

[a] E.g. by choosing at random.

Figure 9: The MOV/Frey Rück algorithm.

3.1.7 Choosing a key size

We will discuss the issue of choosing a reasonable key size for a pairing-based elliptic curve system. It is instructive to compare the key sizes required by ECC- and finite field DLP-based cryptographic systems to attain a certain security level, both to observe the benefits of ECC and in order to choose a reasonable embedding degree.

Recall the definition of the embedding degree in Section 3.1.1 (in particular, $n \mid \#E(\mathbb{F}_q)$). The embedding degree k plays an important role both

for the security of a pairing-based cryptographic system, and the efficiency of the pairing computation. If the embedding degree is large, computation in $\mathbb{F}_{q^k}^*$ will become unfeasible, but if it is too small, the ECDLP will become vulnerable to the MOV/Frey Rück algorithm (see Section 3.1.6). For randomly chosen elliptic curves and finite fields, the embedding degree is usually about the same size as n. However, for our purposes, optimal values for the embedding degree will be around 6 – 12, while n should be a prime in the order of $\#E(\mathbb{F}_q)$. If we are given a desired security level, measured in bits, and know how large an elliptic curve group and a finite field group must be in order to obtain this security level, we may compute an optimal embedding degree. The embedding degree should be exactly large enough such that the MOV/Frey Rück algorithm does not yield an advantage to an adversary: the optimal embedding degree k will be such that $\mathbb{F}_{q^k}^*$ provides the desired security level. Thus, we can compute an optimal embedding degree for any security level by taking the required finite field size and dividing by the required elliptic curve size.

The comparison found in Table 3 is composed from a paper by NIST [5] and www.keylength.net[16]. Each row in the table gives a security level, measured in bits, and indicates the minimal (and thus optimal) sizes of finite fields and elliptic curve groups needed to attain the given security level. To avoid that the adversary gains any advantage in solving the ECDLP from the MOV/Frey Rück algorithm, we should use the embedding degree given in the row. Note however that these numbers are approximate, and differ slightly based on which methods are used to obtain them.

Security (bits)	Years of protection	Finite field	Elliptic curve	Embedding degree (k)
80	2008–2010	1024	160	7
96	2008–2018	1500	192	8
112	2008–2028	2048	224	10
128	2008–2038	3072	256	12

Table 3: Comparison of key sizes.

Blake, Seroussi and Smart [8] show that supersingular elliptic curves have a maximal embedding degree of 6. From Table 3, we can thus conclude that supersingular elliptic curves should only be used for a security level of 80 bits or less if we do not want the finite field DLP to dominate our choice of key

[16]This web site uses mathematical formulas from different publications to indicate minimum recommended key lengths.

size.

3.2 Applications of pairings

The number of published papers related to pairings has dramatically increased over the last years. In 2007, even an annual conference dedicated to pairing based cryptography was created[17]. However, we will merely cover what is interesting to us, which is how to make signatures as short as possible, and how to create identity-based encryption and signature schemes.

For each scheme we consider, we will verify that it is correct (e.g. decryption is the inverse of encryption) and informally discuss why it is secure. The security discussions are informal in the sense that we do not prove that solving the computational problem (e.g. the BDH) is the only way of breaking the given system. A common way to construct such a proof is to show that any algorithm that breaks the system can also be used to solve the computational problem (this is also known as a *security reduction* — we reduce the security of the system to the hardness of the computational problem). A formal proof also includes a model for security, i.e. a description of what the adversary is allowed to do[18].

3.2.1 Short signatures

There are nice applications of pairings outside the area of identity-based cryptography. A prominent example is the BLS (Boneh, Lynn, Shacham) short signature scheme [14], which is shown in Figure 10.

To verify the correctness of the signature scheme, we use the bilinearity property of the pairing and compute:

$$e(H, U) = e(H, sQ) = e(H, Q)^s = e(sH, Q) = e(\sigma, Q)$$

Now let us informally consider the security of the scheme. An adversary wanting to sign a message M can compute $H = \mathcal{H}(M)$, so he has $H \in \mathbb{G}_1$ and $Q, sQ \in \mathbb{G}_2$. To sign the message, $sH \in \mathbb{G}_1$ should be computed. But this is exactly the co-CDH problem on $(\mathbb{G}_1, \mathbb{G}_2)$, as defined in Section 3.1.3.

The security proof by Boneh, Lynn and Shacham [14] requires an efficiently computable isomorphism $\psi : \mathbb{G}_2 \rightarrow \mathbb{G}_1$, which will be the trace map. This corresponds to a type 2 instantiation of our pairing, described in

[17] http://www.pairing-conference.org/

[18] For example, it would be impossible to prove any security in a model where the adversary has access to all private keys. However, to make the model realistic, he should be allowed access to some keys. The model defines, among other things, which keys he is allowed to obtain and when.

- **Setup**
  Generate groups $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_3$, where $\mathbb{G}_1$ and $\mathbb{G}_2$ has prime order r.
  Define a pairing as in Definition 3.5.
  Let Q be a generator of $\mathbb{G}_2$.
  Define a cryptographic hash function $\mathcal{H} : \{0,1\}^* \to \mathbb{G}_1$.
  Pick a random private key $s \in_R \mathbb{Z}_r^*$, and compute the public key $U = sQ \in \mathbb{G}_2$.

- **Sign**
  Let $M \in \{0,1\}^*$ be a message and s the private key.
  Compute $H = \mathcal{H}(M)$ and the signature $\sigma = sH \in \mathbb{G}_1$.

- **Verify**
  Given a public key U, a message M and a corresponding signature σ, compute $H = \mathcal{H}(M)$.
  Check if $e(H, U) = e(\sigma, Q)$, and accept the signature if and only if this is the case.

Figure 10: The BLS short signature scheme.

Section 3.1.4. Also note that the security proof treats the hash function $\mathcal{H}$ as a random oracle[19], which is a problem from a theoretical point of view since no hash function behave exactly as a random oracle (although the outputs might be indistinguishable).

In practice, the BLS scheme allows for secure signatures of about 160 bits, which is half of what is required for the equivalent security level using DSA or ElGamal signature schemes with elliptic curves[20].

3.2.2 Introduction to identity-based cryptography

Any identity-based cryptography (IBC) scheme consists of four algorithms:

- **Setup**
  Generates system parameters and master-key.

- **Extract**
  Uses system parameters and master-key to generate a private key corresponding to any identity.

[19] A random oracle can be explained as a function with randomly mapped outputs [7].

[20] Recall that in the DSA and ElGamal signature schemes the signature is two group elements, while in BLS it is only one.

- **Encrypt / Verify**
  Uses the system parameters and an identity. For identity-based encryption (IBE), it encrypts to that identity, for identity-based signatures (IBS), it checks a signature claimed to be generated by the identity.

- **Decrypt / Sign**
  Uses the system parameters and a private key corresponding to an identity. For IBE, it decrypts a message for that identity, and for IBS, it signs a message.

We will discuss how IBC differs from classical public-key cryptography (e.g. RSA) on a high level. In a PKI, all the users have access to an authentic copy of the public key of a Certificate Authority (CA). The users then generate a key pair and request a certificate from the CA. In order for a user A to be able to encrypt a message to- or check the signature of another user B, A must first obtain the certificate of B. Certificate distribution is a serious problem for a routing protocol in a MANET, since every certificate is usually hundreds, or even thousands of bits long and needs to be sent to many (or all) nodes.

As a PKI, IBC require nodes to obtain authentic system parameters. But as soon as a node has obtained these, it may encrypt to all other nodes, and also check signatures generated by all other nodes — no certificates are required. The only requirement is that the node knows the *identity* of the other nodes. An identity may be any string, such as a name, email address or IP address. However, if the node wants to decrypt or sign a message, it will need the private key corresponding to its identity with respect to the system parameters. This private key is obtained from a so-called Private Key Generator (PKG). The PKG uses its knowledge of the master-key corresponding to the system parameters to generate the private key for any identity. This is somewhat similar to the situation where the CA issues a certificate to a user. However, two important differences exist. In an IBC, the PKG will know the *private* keys of all the users, so the PKG is more trusted in this sense. Furthermore, when the private key of a user is to be issued from the PKG, a confidential channel must be available. Otherwise, the private key may be compromised. This is not a problem in a PKI because only public keys are transmitted. Table 4 gives an overview of the procedures for setting up a PKI and a identity-based system.

IBC may be implemented using many different mathematical problems. IBS schemes were introduced as early as in 1984 by Shamir [62]; a signature scheme based on RSA. However, practical IBE schemes was an open problem until Boneh and Franklin presented their scheme based on the Weil pairing in 2001 [12]. Just after this, Cocks presented an IBE system based on quadratic residuosity problem modulo an RSA-composite [20]. This system, however,

Phase	Public Key Infrastructure	Identity-based cryptography
Setup	• `CA` generates key pair $\{pk_{CA}, sk_{CA}\}$ • users obtain authentic copy of pk_{CA} Users may now encrypt to `CA` and check its signatures.	• `PKG` generates system parameters • users obtain authentic copy of system parameters Users may now encrypt to anyone and check any signature.
Extract	• user authenticates to `CA` • user generates key pair • `CA` issues certificate to user User may now publish certificate, decrypt and sign.	• user authenticates to `PKG` • `PKG` generates user's private key • `PKG` issues private key to user, through a confidential channel User may now decrypt and sign.

Table 4: Overview of PKI and IBC setup.

produces long ciphertexts. A version with shorter ciphertexts is proposed by Boneh, Gentry and Hamburg [13], but this system is significantly less efficient than the system of Cocks. On the other hand, there exist both `IBS` and `IBE` schemes using pairings which may share the public parameters. These schemes produce short messages and are quite efficient, and are therefore the currently most appealing candidates for `IBC` implementations.

3.2.3 Identity-based encryption

The first practical identity-based encryption scheme was published by Boneh and Franklin [12]. Later, an `IBE` system without the random oracle assumption was proposed [68]. However, it is less efficient, and it does not seem to exist a signature scheme that works with the same parameters as this system (i.e. two sets of public parameters are required for allowing users to

both encrypt and sign messages). Furthermore, distributing the PKG seems easier in the scheme by Boneh and Franklin. Therefore, we will describe the first scheme, proposed by Boneh and Franklin [12]. We adapt the scheme to work with generic pairings as in Definition 3.5 [21], which results in the four algorithms shown in Figure 11.

First we should confirm that decryption is the inverse of encryption. Since XOR ($\oplus$) is its own inverse, the only thing we need to verify is that $g_{ID}^{\beta} = e(U, d_{ID})$ in $\mathbb{G}_3$. This is done merely by using the definitions of the variables, and the bilinearity property of the pairing e:

$$g_{ID}^{\beta} = e(P_{pub}, Q_{ID})^{\beta} = e(sP, Q_{ID})^{\beta} = e(P, Q_{ID})^{s\beta} = e(\beta P, sQ_{ID}) = e(U, d_{ID})$$

Now, let us informally discuss why decryption is impossible without knowing d_{ID} or solving the co-BDH [22]. The message is masked with $\mathcal{H}_4(\alpha)$ during encryption. To obtain α, one should try to find $g_{ID}^{\beta} = e(P, Q_{ID})^{s\beta}$ from the publicly known P, P_{pub}, U, Q_{ID}. Recall that $P_{pub} = sP$ and $U = \beta P$, where an adversary does not know s or β. Furthermore, $P, sP, \beta P \in \mathbb{G}_1$ and $Q_{ID} \in \mathbb{G}_2$. But computing $e(P, Q_{ID})^{s\beta}$ from this implies solving the co-BDH as stated in Definition 3.1.3 [23].

A model for security and a formal proof is found in the proposal by Boneh and Franklin [12]. Note that the proof relies on modelling hash functions as random oracles.

Adding repudiable authentication Lynn [45] shows how to extend the Boneh and Franklin IBE scheme to add repudiable authentication without any extra cost. The ciphertext itself serves as the message authentication code. This differs from a digital signature scheme, which is non-repudiable. The proposed scheme achieves repudiation through the fact that both the sender and receiver of the message can produce this authentication code, but no one else can. Thus, the receiver knows who sent the message since he did not do it himself, while the sender can claim that the receiver created the message to a third party. This might be desirable in some applications.

3.2.4 Identity-based signature schemes

Shamir published an IBS scheme based on RSA [62]. However, as we have an IBE scheme based on pairings, it is useful to have an IBS scheme based

[21] The "FullIdent" scheme described by Boneh and Franklin [12] assumes $\mathbb{G}_1 = \mathbb{G}_2$.

[22] If $\mathbb{G}_1 = \mathbb{G}_2$, we use the BDH problem instead.

[23] Assume there exists an algorithm $\mathcal{A}$ that on input $(P, sP, \beta P, Q_{ID})$ outputs $e(P, Q_{ID})^{s\beta}$. Given a co-BDH problem instance (P, aP, bP, Q, cQ), set $\tilde{Q} = cQ$ and run $\mathcal{A}$ on $(P, aP, bP, \tilde{Q})$. This gives $e(P, \tilde{Q})^{ab} = e(P, Q)^{abc}$, which solves the co-BDH

- **Setup**
  Generate groups $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_3$ of prime order r. Define a pairing e as in Definition 3.5.
  Pick at random $P \in_R \mathbb{G}_1 \backslash \{O\}$ and $s \in_R \mathbb{Z}_r^*$, and compute $P_{pub} = sP$.
  Choose $n \in \mathbb{Z}^+$ and define four cryptographic hash functions:

$$\begin{aligned} \mathcal{H}_1 &: \{0,1\}^* \to \mathbb{G}_2^* \\ \mathcal{H}_2 &: \mathbb{G}_3 \to \{0,1\}^n \\ \mathcal{H}_3 &: \{0,1\}^n \times \{0,1\}^n \to \mathbb{Z}_r^* \\ \mathcal{H}_4 &: \{0,1\}^n \to \{0,1\}^n \end{aligned}$$

  The system parameters are
  $\{\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3, e, P, P_{pub}, n, \mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3, \mathcal{H}_4\}$.
  The master-key is s.
  The plaintext space is $\{0,1\}^n$ and the ciphertext space is $\mathbb{G}_1 \times \{0,1\}^n \times \{0,1\}^n$.

- **Extract**
  Given an identity $ID \in \{0,1\}^*$, compute $Q_{ID} = \mathcal{H}_1(ID)$ and $d_{ID} = sQ_{ID} \in \mathbb{G}_2$.
  d_{ID} is the private key of ID.

- **Encrypt**
  Given a message $M \in \{0,1\}^n$, and an identity $ID \in \{0,1\}^*$, compute $Q_{ID} = \mathcal{H}_1(ID)$.
  Choose at random $\alpha \in_R \{0,1\}^n$, and set $\beta = \mathcal{H}_3(\alpha, M) \in \mathbb{Z}_r^*$.
  Define $g_{ID} = e(P_{pub}, Q_{ID}) \in \mathbb{G}_3$. The ciphertext is $C = \{U, V, W\}$, where

$$\begin{aligned} U &= \beta P \\ V &= \alpha \oplus \mathcal{H}_2(g_{ID}^{\beta}) \\ W &= M \oplus \mathcal{H}_4(\alpha) \end{aligned}$$

- **Decrypt**
  Given a ciphertext $C = \{U, V, W\}$ encrypted with ID, and the private key $d_{ID} \in \mathbb{G}_2$, compute:

$$\begin{aligned} \alpha &= V \oplus \mathcal{H}_2(e(U, d_{ID})) \\ M &= W \oplus \mathcal{H}_4(\alpha) \end{aligned}$$

  Set $\beta = \mathcal{H}_3(\alpha, M)$. Verify that $U = \beta P$, otherwise reject C.

Figure 11: The Boneh and Franklin IBE system.

on pairings as well, so that we may use the same public parameters for the two systems. An IBS scheme published by Cha and Cheon [15] may actually share public parameters with the IBE scheme we discussed in Section 3.2.3 when $\mathbb{G}_1$=$\mathbb{G}_2$. This scheme is described in Figure 12.

Correctness is easily checked:

$$\begin{aligned} e(P_{pub}, U + mQ_{ID}) &= e(sP, \alpha Q_{ID} + mQ_{ID}) = e(P, (\alpha + m)(sQ_{ID})) \\ &= e(P, (\alpha + m)d_{ID}) = e(P, V) \end{aligned}$$

Next, let us informally consider the security of the scheme. An adversary can easily compute the first component $U = \alpha Q_{ID}$ of the signature, since no secret is involved here. Consider the second component, V.

$$V = (\alpha + m)d_{ID} = (\alpha + m)sQ_{ID} = s((\alpha + m)Q_{ID})$$

Let $Q = (\alpha + m)Q_{ID}$, so $V = sQ$. Anyone can compute Q from the public parameters. However, an adversary wants to compute V to forge a signature. Recall that $P_{pub} = sP$ is a public parameter, so an adversary has $P, sP \in \mathbb{G}_1$ and $Q \in \mathbb{G}_1$ and wants to compute $V = sQ \in \mathbb{G}_1$. We can see that this clearly is the CDH problem from Section 3.1.3. The paper by Cha and Cheon [15] proves the security of this signature scheme in a model that is a natural generalisation of the existential forgery under an adaptive chosen message attack[24], to make the model suitable for IBS schemes. Their proof assumes the hardness of the co-CDH problem.

3.2.5 Identity-based signcryption

In practical scenarios, we often want both authentication and confidentiality of information. Usually, this is solved by first signing the information and then encrypting it and the signature. *Signcryption* combines signature generation and encryption into one monolithic operation. This operation is more efficient than first signing and then encrypting, in terms of computation and/or ciphertext size.

Many signcryption schemes are proposed in the literature, but not all are identity-based. The other main differences are efficiency, security assumptions and security guarantees (under the assumptions). A scheme by Chen and Malone-Lee [17] assumes the hardness of the BDH problem to give some of the stronger security guarantees [17, Definition 1,2,3 and 4] in the random oracle model. Additionally, this scheme may share system parameters with

[24]Existential forgery under an adaptive chosen message attack is the standard security model for signature schemes.

- **Setup**
  Generate groups $\mathbb{G}_1$, $\mathbb{G}_3$ of prime order r. Define a pairing e as in Definition 3.5, with $\mathbb{G}_1$=$\mathbb{G}_2$.
  Pick at random $P \in_R \mathbb{G}_1 \backslash \{O\}$ and $s \in_R \mathbb{Z}_r^*$, and compute $P_{pub} = sP$.
  Define two cryptographic hash functions:

$$\mathcal{H}_1 : \{0,1\}^* \rightarrow \mathbb{G}_1^*$$
$$\mathcal{H}_2 : \{0,1\}^* \times \mathbb{G}_1 \rightarrow \mathbb{Z}_r^*$$

  The system parameters are $\{\mathbb{G}_1, \mathbb{G}_3, e, P, P_{pub}, \mathcal{H}_1, \mathcal{H}_2\}$.
  The master-key is s.
  A signature is an element of $\mathbb{G}_1 \times \mathbb{G}_1$.

- **Extract**
  Given an identity $ID \in \{0,1\}^*$, compute $Q_{ID} = \mathcal{H}_1(ID)$ and $d_{ID} = sQ_{ID} \in \mathbb{G}_1$.
  d_{ID} is the private key of *ID*.

- **Sign**
  Given a message $M \in \{0,1\}^*$ and a private key d_{ID},
  pick at random $\alpha \in_R \mathbb{Z}_r$.
  Compute $U = \alpha Q_{ID} \in \mathbb{G}_1$, $m = \mathcal{H}_2(M, U)$ and $V = (\alpha + m)d_{ID} \in \mathbb{G}_1$.
  The signature is $\sigma = (U, V)$.

- **Verify**
  Given an identity *ID*, a message M and a corresponding signature $\sigma = (U, V)$, compute $Q_{ID} = \mathcal{H}_1(ID)$ and $m = \mathcal{H}_2(M, U)$.
  Accept the signature σ on M if and only if $e(P_{pub}, U + mQ_{ID}) = e(P, V)$.

Figure 12: The Cha and Cheon IBS system.

the Boneh and Franklin IBE and Cha and Cheon IBS schemes. By using these three schemes, we obtain the flexibility of being able to only sign or encrypt, while having a more efficient mechanism for doing both.

3.2.6 Revoking an identity

In a PKI, there are a few well-known ways to revoke a public key certificate. From these, we can construct equivalent (or better) techniques in an IBC system.

The most prevalent mechanism for revoking a certificate is to create and publish a *certificate revocation list*. This list consists of revoked certificates together with the signature of a revocation authority, possibly the CA, to prove its authenticity. A revocation authority, possibly the PKG, can be created in an IBC system by designating an identity to it, e.g. "Revocation Authority". The revocation authority can then create a list of identities to revoke and sign it with the private key corresponding to its identity. The advantage of IBC over a PKI is that identities (e.g. email or IP addresses) are usually far smaller than certificates, resulting in a more efficient distribution of the revocation list.

Secondly, certificates usually have a *validity period*, which is defined as the period within two timestamps. Thus, the certificate expires, after which it is considered invalid. A new certificate must then be obtained. We can construct the same mechanism in an IBC system by requiring the validity period to be appended to the identity-string itself. For example, Alice may obtain the private key corresponding to the identity "Alice | 2009/01/01 - 2010/01/01" from the PKG any time between the 1st of January 2009 and 2010, and the users of the system will not use this public key after this period. This makes it possible for users to encrypt messages to Alice that she cannot decrypt until the start of the validity period. In fact, this can be generalised to include other conditions than a time period. For example, in a corporate scenario, Alice can only decrypt messages encrypted for "Alice | 2010/01/01 - 2010/01/02 | CEO" if she is the CEO of the corporation between 1st and 2nd January 2010. Note that Alice may always obtain private keys from the PKG when the conditions in the identity-strings hold, but it is generally inefficient that Alice has a lot of different keys. Thus, some conventions on the conditions should be applied, like that time periods should be one calendar month.

A third mechanism, perhaps less attractive, is to create a new set of system parameters and only give new keys to identities that are not revoked[25]. This can be done when some threshold of revoked identities is reached, a

[25]However, the new system parameters can be authenticated if the PKG has an identity

time interval is exceeded or some other condition is met. This mechanism can thus be used in combination with a revocation list to avoid that the list grows indefinitely. The equivalent in a `PKI` is that the `CA` changes public key.

and signs the new set of parameters.

4 Threshold Cryptography

4.1 Introduction

Practical cryptographic systems rely on the secrecy of keys to achieve any security. These keys may well be protected by encryption under other keys, but these encryption keys must also be protected. In the end, we must rely on that some keys are stored in a physically secure way.

More risk is associated with the storage of some keys than others. In a PKI, the whole system is compromised if the private key of the CA is compromised. This also holds true for the master-key of the PKG in an IBC system. In a MANET, it is not always plausible to assume that any one node is physically secure. If a node holds the secret key corresponding to the public parameters of the system, an adversary may compromise the whole system by capturing one node. We are therefore particularly concerned that privileged keys are kept *secret*.

But privileged keys also needs to be *available*. In an IBC system, the secret key corresponding to the system parameters is needed in order to generate keys for nodes. In a PKI, it is needed to generate certificates and revocation lists.

At first, the secrecy and availability requirements may seem contradictive: if we spread the key out on many locations to make it easily available, secrecy is degraded since the key is more prone to get compromised by an adversary. Threshold cryptography solves this problem by offering both secrecy and availability of information at the same time. Two important models for threshold cryptography are secret sharing and function sharing. These will be discussed successively.

4.2 Secret sharing

A *threshold secret sharing* scheme consists of a probabilistic algorithm, called the dealer, that takes as input a secret s, and outputs n shares $s_1, s_2, \ldots, s_n$. A threshold t, with $t \in [1, n-1]$, is also defined. The idea is that any t shares reveal no information about the secret s, while any $t+1$ shares determine s uniquely. More precisely, we require [21]:

- **Secrecy**
  Let I be any subset of indices in $\{1, 2, \ldots, n\}$, with $\#I \leq t$. Give the dealer a secret s and let it generate shares $s_1, s_2, \ldots, s_n$. Then the probability distribution of the shares $\{s_i \mid i \in I\}$ is independent of s.

- **Availability**
  Let J be any subset of indices in $\{1, 2, \ldots, n\}$, with $\#J \geq t + 1$. Give the dealer a secret s and let it generate shares $s_1, s_2, \ldots, s_n$. Then $\{s_i \mid i \in J\}$ uniquely determines s, and there exists an efficient algorithm, called the combiner, which computes s from $\{s_i \mid i \in J\}$.

The dealer creates n shares, but this would be quite pointless if it held them for itself. So the dealer will distribute the shares to n *shareholders*, using a confidential and authenticated channel. The secret s is guaranteed to be kept confidential as long as no more than t shares are obtained by the adversary. When the secret s is needed, the combiner will collect at least $t + 1$ shares, using a confidential and authenticated channel, and reconstruct s. The combiner may be one or more of the shareholders, or a separate entity. The whole process of secret sharing is illustrated in Figure 13.

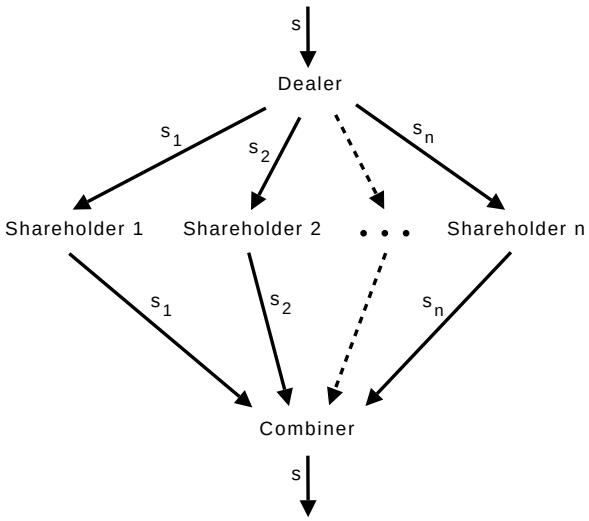

Figure 13: Secret sharing.

Non-threshold secret sharing In some applications, it may be useful to generalise threshold secret sharing such that we are not limited to specifying a threshold, but rather an *access structure* describing sets of entities, whose cooperation is necessary and sufficient to reconstruct the secret. To illustrate this concept, let A, B, C and D be entities. We want to create a secret sharing scheme such that the entities $\{A, B\}$, $\{B, C\}$ and $\{C, D\}$ may reconstruct the secret, but $\{A, D\}$ cannot. A threshold secret sharing scheme with $t = 1$ will not solve our problem, since this will allow the set $\{A, D\}$ to reveal

the secret alone. We will not discuss this any further, but merely note that it is possible to create a secret sharing scheme defined by access structures from a threshold secret sharing scheme (e.g. Shamir's secret sharing scheme described below), but that the resulting scheme may not always be efficient [21] [26].

The following proposition shows that the shares must be at least as large as the secret.

Proposition 4.1. *For any secret sharing scheme, the entropy of any share is at least the entropy of the secret. In particular, an optimal encoding requires at least as many bits to store a share as the secret.*

Proof. Let l be the number of bits of entropy of the secret s, and assume that we have obtained t shares $(s_1, s_2, \ldots, s_t)$ of s. By the secrecy property, we still have 0 bits of information on s. But if we obtain one more share s_{t+1} we have l bits of information on s by the availability property. This implies that s_{t+1} must have at least l bits of entropy. Since we can call any share s_{t+1}, this holds for all shares. □

4.2.1 Shamir's secret sharing

Shamir proposed an implementation of secret sharing [61], which we will discuss next. Note that there are also other implementations available, most notably a plane geometry based scheme due to Blakley [9]. This scheme is however less efficient than the one due to Shamir.

Shamir's scheme is described in Figure 14. The secret s is an element of a field $\mathbb{F}$, and the resulting shares will also be elements of $\mathbb{F}$. In particular, if $s \in \mathbb{Z}^+$, we may choose $\mathbb{F} = \mathbb{Z}_p$, for a prime $p > s$ and $p > n$. By Proposition 4.1, Shamir's secret sharing scheme produces as small shares as possible, since the shares and the secret are all elements of the same field.

First, we will elaborate a bit on why the shares are given names. As we soon shall see, in order to reconstruct the secret, one must know the elements x_i on which f evaluates to the given shares s_i. One way to get around this is to define a share as $s_i = (x_i, f(x_i))$. The only problem with this is that it doubles the entropy of the shares, leaving the sharing non-optimal in regard of Proposition 4.1. Furthermore, the elements x_i can, unlike the evaluations $f(x_i)$, be publicly known. So we will merely assume that there is some

[26] More precisely, we end up having an exponential number of shares in n which clearly does not scale very well. It can be shown that no efficient secret sharing scheme exist for some access structures.

INPUT:	A secret $s \in \mathbb{F}$, the number of shares n, a tuple $N = (x_1, x_2, \ldots, x_n) \in \mathbb{F}^n$ with distinct share names [a], and a threshold $t \in [1, n-1]$
OUTPUT:	Shares $(s_1, s_2, \ldots, s_n) \in \mathbb{F}^n$

1. Choose random coefficients $a_1, a_2, \ldots, a_t \in_R \mathbb{F}$.
2. Define the polynomial $f(x) = s + a_1 x + a_2 x^2 + \cdots + a_t x^t$.
3. Let the shares be $s_i = f(x_i)$, for $x_i \in N$.
4. Return $(s_1, s_2, \ldots, s_n)$.

[a] See the following discussion for an explanation.

Figure 14: Shamir's secret sharing scheme.

mechanism in place to keep track of that the share s_i is created from the element x_i, by saying that x_i is the name of share s_i [27].

To show that Shamir's secret sharing scheme satisfies the secrecy and availability requirements, we will need a classical result on Lagrange interpolation.

Theorem 4.2. *Let $\mathbb{F}$ be a field, and $(x_1, y_1), (x_2, y_2), \ldots, (x_{t+1}, y_{t+1}) \in \mathbb{F} \times \mathbb{F}$, where the x_i's are distinct. Then there exists a unique and efficiently computable polynomial $g(x) \in \mathbb{F}[x]$ of degree at most t, such that $g(x_j) = y_j, \forall j \in [1, t+1]$.*

Proof. Consider the polynomial

$$\delta_i(x) = \prod_{j=1, j\neq i}^{t+1} \frac{x - x_j}{x_i - x_j} = \frac{(x - x_1)}{(x_i - x_1)} \cdots \frac{(x - x_{i-1})(x - x_{i+1})}{(x_i - x_{i-1})(x_i - x_{i+1})} \cdots \frac{(x - x_{t+1})}{(x_i - x_{t+1})}$$

Consider what $\delta_i(x)$ evaluates to when $x = x_j$, with $j \in \{1, 2, \ldots, t+1\}$. It is not too hard to see that

$$\delta_i(x_j) = \begin{cases} 0 & \text{if } j \neq i \\ 1 & \text{if } j = i \end{cases}$$

Next, we construct the polynomial

$$g(x) = \sum_{i=1}^{t+1} y_i \delta_i(x) = y_1 \delta_1(x) + y_2 \delta_2(x) + \cdots + y_{t+1} \delta_{t+1}(x)$$

[27] For instance, the name of a share might be the name of its holder. In this way, we automatically know the name of a share by checking who sent it.

From this construction, we can see that $g(x_j) = y_j$, for $j \in \{1, 2, \ldots, t+1\}$ and that the degree of $g(x)$ is at most t.

To prove uniqueness, assume that some other polynomial, $h(x)$, with the same properties exists: its degree is at most t, and $h(x_j) = y_j$. This implies that the polynomial $k(x) = g(x) - h(x)$ has $t+1$ zeroes, while its degree is at most t. Only the polynomial $k(x) = 0$ has more zeros than its degree, so $g(x) = h(x)$.

From its construction, $g(x)$ is clearly efficiently computable. □

This shows that the availability requirement is satisfied, because if we are given $t+1$ points, we may find the unique polynomial $g(x)$ of degree at most t by using Lagrange interpolation and compute the secret $s = g(0)$ ($g(x)$ is the polynomial $f(x)$ constructed by the dealer). But there exists a more efficient method since we are not really interested in finding the polynomial $g(x)$, but rather one of its values, $s = g(0)$. We may instead compute $(\delta_1(0), \delta_2(0), \ldots, \delta_{t+1}(0))$, which is called a *recombination vector*, and from these values and the shares $s_i = y_i$ compute $g(0)$ directly as $\sum_{i=1}^{t+1} y_i \delta_i(0)$.

To show that the secrecy requirement holds, we will need the following proposition.

Proposition 4.3. *In Shamir's secret sharing scheme, any t shares are uniformly distributed.*

Proof. Fix any secret $s \in \mathbb{F}$ and a set I of t indices, where we know the shares s_i. In Shamir's scheme, $(a_1, a_2, \ldots, a_t) \in_R \mathbb{F}^t$ is chosen at random and $f(x) = s + a_1 x + a_2 x^2 + \cdots + a_t x^t$. f defines an evaluation map from $\mathbb{F}^t$ to $\mathbb{F}^t$ by mapping $(a_1, a_2, \ldots, a_t) \to (f(1), f(2), \ldots, f(t))$. This map is invertible by Lagrange interpolation: given $f(i)$, for $i \in [1, t]$ and the fixed secret $f(0) = s$, we know f on $t+1$ points, and may construct f to obtain the a_i's. Any invertible map maps the uniform distribution to the uniform distribution. Since the coefficients $(a_1, a_2, \ldots, a_t)$ are uniformly distributed, the shares $(s_1 = f(1), s_2 = f(2), \ldots, s_t = f(t))$ are also uniformly distributed. □

Since t shares are uniformly distributed, the distribution of the shares is clearly independent of the secret s. This shows that the secrecy requirement holds.

4.3 Function sharing

Secret sharing is a very powerful and versatile tool. However, if it is to be used to share a private key in a practical scenario, a problem arises: the combiner obtains the whole secret key when it is used in some computation.

So if the adversary can capture the combiner, he may also compromise the private key. Thus, secret sharing is very useful for secure physical *storage* of a secret, but as soon as the secret is to be used, it is again very vulnerable to a compromise.

Some thought on this issue reveals that we do not really need to *know* the private key explicitly as long as we may *use* it in a calculation of some function. For example, the function could be to sign a public key certificate or generate a private key for a user as a `PKG`.

It may seem a bit vague how we can use a secret without knowing it. But this is where the notion of secret sharing comes in: every shareholder has a share of the private key, but no one knows it. If we generalise this to functions, we could say that every shareholder has a share of the function, but no one knows the complete function. A common way to create shares of a function is to create a secret sharing of the input to the function, while the algorithm that is executed on the input is publicly known. In this scenario, the shareholders use their shares to do some local computations and then some entity combines the results of these computations to complete the function computation. Thus, by using function sharing, the combiner does not learn the secret s, but rather the output of a function where s is used.

It is probably possible to find cases where the algorithm executed in the shares of the function must be kept confidential, and the model that is presented next takes this into account. However, we will only construct shares of functions by using secret sharing and keeping only the shares secret, while the operations on the shares are publicly known. Our on-going discussion on secret sharing is motivated by that we use secret sharing to construct function sharing.

De Santis et al. [58] have formalised a model for function sharing, including a definition of security. Recall that n is the number of shareholders, while t is the threshold. Let f_{pk} be a random instance of a trapdoor function, with f_{sk}^{-1} being its inverse. pk is a public key, while sk is a private key (the trapdoor). A function sharing primitive has the following two algorithms:

- A probabilistic polynomial-time algorithm *share*, which given (pk, sk) computes $S_1, S_2, \ldots, S_n$, the *share-functions*.
- A probabilistic polynomial-time algorithm *rec*, given pk, and any $t+1$ evaluations $S_i(\alpha)$, where α is in the domain of f_{sk}^{-1}, computes $f_{sk}^{-1}(\alpha)$.

Note the similarity with secret sharing: the algorithm *share* corresponds to the dealer, while *rec* corresponds to the combiner.

In addition to this, we will also need a definition of security: we have not yet required that access to any t (or less) share-functions S_i is not sufficient

to compute $f_{sk}^{-1}(\alpha)$. We should also make sure that obtaining partial results $S_i(\alpha)$ does not make it easier to compute $f_{sk}^{-1}(\beta)$ for $\beta \neq \alpha$. A formal version of the security definition is given by De Saints et al. [58]. The process of function sharing is illustrated in Figure 15.

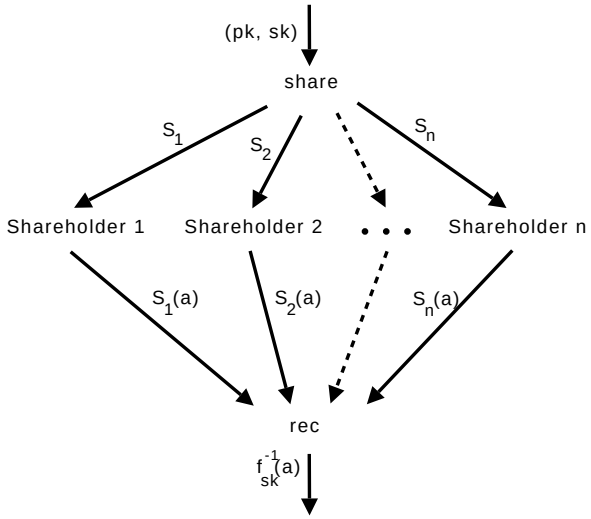

Figure 15: Function sharing.

As an example of function sharing, consider the RSA signature scheme: p, q are primes, $n = pq$, $e \in \mathbb{Z}_n^*$, and $d = e^{-1}$. We have $pk = (e, n)$ and $sk = (d, p, q)$. For $m \in \mathbb{Z}_n^*$, we have $f_{pk}(m) = f_{(e,n)}(m) = m^e \pmod{n}$ and $f_{sk}^{-1}(m) = f_{(d,p,q)}^{-1}(m) = m^d \pmod{n}$. The *share* algorithm is defined as follows: use Shamir's secret sharing to create shares $(s_1, s_2, \ldots, s_r)$ of $s = d$. For $m \in \mathbb{Z}_n^*$, define $S_i(m) = m^{s_i} \pmod{n}$. To construct the *rec* algorithm, assume we have $t+1$ tuples $(\delta_i(0), S_i(m))$ [28]. We can now compute $\prod_{i=1}^{t+1} S_i(m)^{\delta_i(0)} = m^{\sum_{i=1}^{t+1} s_i \delta_i(0)} = m^d = f_{sk}^{-1}(m) \pmod{n}$ [29].

4.4 Robustness

The availability requirement for a secret sharing scheme ensures that $t + 1$ shares is sufficient to reconstruct the secret. But as shareholders may be

[28] Recall that $(\delta_1(0), \delta_2(0), \ldots, \delta_{t+1}(0))$ is the recombination vector, as defined at the end of Section 4.2.1.

[29] There is a subtle problem with this: the exponent is reduced modulo $(p-1)(q-1)$, while we have only defined Shamir's secret sharing to work over fields. A possible approach to solve this is to generalise Shamir's secret sharing to work over a finite module. Note that if this is done, the share space, $\mathbb{Z}_{(p-1)(q-1)}$ must be kept secret in order not to compromise d. De Saints et al. [58] give the details.

corrupt, we cannot assume that any $t + 1$ shares obtained by the combiner are correct. Indeed, in Shamir's secret sharing scheme, if one of the $t + 1$ shares used by the combiner is incorrect, the secret cannot be reconstructed. But worse, the error might not be detected and the output from the combiner is taken as the secret. Detecting an incorrect share is the topic of *robustness*.

A generic solution is to use *Zero-knowledge protocols*. These protocols can be used to prove that any communication or computation is done correctly, without revealing anything else (hence the name Zero-knowledge). However, the protocols are generally less attractive in practice because they are interactive and require much communication. Thus, robustness is most efficiently handled in an application-specific way.

However, for groups where the DDH is easy and the CDH is hard, efficient verification schemes exist [10]. Note that such groups can be constructed by using pairings, as discussed in Section 3.1.3.

As an example using function sharing (discussed in Section 4.3), Gennaro et al. [33] show how to obtain robust function sharing for RSA. In their approach, the dealer creates *verification data* v_i along with the shares s_i. The combiner can use v_i to check if s_i is correct.

4.5 Proactive secret sharing

Recall the secrecy and availability requirements for secret sharing from Section 4.2. Secrecy ensures that the secret is kept confidential even though up to t shares are compromised, while availability ensures that the secret can be reconstructed if $t + 1$ valid shares are obtained.

These requirements naturally lead to two goals for the adversary. He can try to obtain $t + 1$ shares to find the secret. But he can also *destroy* enough shares such that the secret cannot be reconstructed: destroying $n - t$ shares leaves only t correct shares, which is insufficient to reconstruct the secret. Which one of the situations is the worst is hard to tell, but they are both clearly undesirable.

In general, it is impossible to reason about the probability for the adversary to succeed with any of these two goals. It clearly depends on the choice of n, t and the specific application where secret sharing is used. However, it is reasonable to say that the success probability for the adversary depends on *how much time* he has to his disposal, where more is generally better for him.

Proactive secret sharing defends against a *mobile* adversary, where an adversary attacks one shareholder, either steals or corrupts his share, and moves on to the next. The main idea is to periodically do *share refreshing*, where shareholders get new shares, leaving the old shares obsolete, while

the secret remains unchanged. This forces the adversary to control enough shareholders within the same time frame (e.g. a week, day or hour), in order to succeed.

Herzberg et al. [37] propose a protocol for proactive secret sharing using Shamir's scheme. An observation they use in their protocol is that a new sharing of a secret s is obtained by creating a sharing of 0 and adding corresponding shares of these two sharings. This can be seen as follows. If the polynomial f_a is used to share a, while f_b is used to share b, with the same t and n, then $f_a(0) = a$ and $f_b(0) = b$ by definition. Adding the shares from f_a and f_b is equivalent to creating a new sharing with polynomial $f = f_a + f_b$, since $f(0) = a + b$, $f(i) = f_a(i) + f_b(i)$ and f has degree at most t [30], so any $t + 1$ shares can be used to reconstruct the secret.

Their protocol for passive adversaries[31] is shown in Figure 16.

`INPUT:`	A share s_i from Shamir's secret sharing scheme, and the t and n used in the sharing of s_i
`OUTPUT:`	A refreshed share s'_i of the same secret

Shareholder i does the following

1. Use Shamir's secret sharing to create n shares of 0, $(z_1^{(i)}, z_2^{(i)}, \ldots, z_n^{(i)})$.
2. Send $z_j^{(i)}$ to shareholder j, using a confidential and authenticated channel.
3. When shares $(z_i^{(1)}, z_i^{(2)}, \ldots, z_i^{(n)})$ are received from the other shareholders, compute $s'_i = \sum_{j=1}^{n} z_i^{(j)}$.
4. Return s'_i.

Figure 16: Proactive secret sharing for a passive adversary.

Allowing the adversary to arbitrarily deviate from the protocol (i.e. be active) for the up to t shareholders he controls, makes it possible for him to publish sharings where the secret is not 0. The resulting sharing may then have a different secret, so s might be lost. To solve this problem, the protocol may be extended to ensure that all the secrets are 0 in the created sharings, without revealing the polynomials to the shareholders (which would render the protocol useless). Herzberg et al. [37] presents a protocol that does this, but it involves more communication than the protocol for passive adversaries.

[30] It may seem a bit strange why we allow a polynomial with degree *less* than t, but this is not a problem because the adversary does not know the exact degree of the polynomial, and thus cannot be sure that he has the right secret from less than $t + 1$ shares.

[31] A passive adversary merely reads messages, while an active adversary may in addition change and create messages.

The protocol in Figure 16 ensures that the shares a mobile adversary has learned in one time frame becomes obsolete in the next. However, mechanisms that *recover* shares that an adversary has destroyed in the previous time frame can also be constructed [37].

Applicability in a MANET In the scenario where the shareholders are MANET nodes and the MANET is *short-lived*, proactiveness is not beneficial [60]. In addition, there are two major problems with implementing proactiveness in a MANET. First of all, the scheme assumes that all shareholders can be contacted when distributing new sharings. Second, synchronised clocks are required in order for the nodes to refresh the shares at the same time. Neither of these assumptions are realistic in a MANET. However, if some event occurs that involves all MANET shareholders, then share refreshing can be done simultaneously.

4.6 Removing trust in the dealer

The dealer is trusted in two respects. Firstly, the sharing it creates should make the secret reconstructible. Secondly, it should keep the secret from leaking to the adversary.

Verifiable secret sharing A secret sharing scheme is *verifiable* if the shareholders may check that their shares can be used to construct an unambiguous secret, without revealing it. This notion was pioneered by Chor et al. [18]. Multiple implementations exist, but all have in common that the dealer distributes some auxiliary information to the shareholders. Note that a malicious dealer may nonetheless share a *different* value than its input secret s. Depending on the application, the exact value of the secret might not matter, as long as the sharings are consistent (e.g. when sharing a random value).

Distributed key generation Suppose we want to share a private key of an asymmetric key pair, but we have no specific key in mind. A private key needs to be random to be hard to guess, and some constraints on the key exist in order for it to be valid.

The idea of *distributed key generation* is to make each shareholder generate a share of the private key. This has the effect of removing the need for a dealer, and thus no entity knows the whole private key. It should however be possible to compute the corresponding public key without revealing the

private key. Different protocols must be used when generating keys for different cryptographic systems. Distributed key generation schemes exist for discrete log based systems [32] and RSA [11, 34].

4.7 Summary

Secret information that is supposed to be used once can be protected with secret sharing. Information used multiple times should be protected by a function sharing scheme instead.

As seen in Figure 13, secret sharing includes three entities: the dealer, shareholder and combiner. Each of these entities are possible victims for the adversary. We have addressed the threats that the adversary poses if he controls any of these entities, and given mechanisms to protect against them. By distinguishing between a passive and active adversary, Table 5 summarises the threats we face and gives remedies for them.

	Passive attack		Active attack	
Entity	Threat	Defense	Threat	Defense
Dealer	leak s	dist. kg. (Section 4.6)	inconsistent sharing	VSS, dist. kg. (Section 4.6)
Shareholder	leak $t+1$ shares	proactive (Section 4.5)	inconsistent shares	robustness (Section 4.4)
Combiner	leak s	function sharing (Section 4.3)	shares wrongly combined	combining for own purposes

Table 5: Threats by the entities in threshold cryptography.

5 Key Management in MANETs

There are many different ways to handle the issue of key management in a MANET, and different classifications of schemes exist. One possibility is to divide them into *contributory* and *distributive* schemes.

Contributory schemes are key agreement protocols, where a group of nodes agrees on a secret symmetric key after the protocol is run. Note that authentication may be required to participate. The distributive schemes include one or more trusted authorities, and can be further divided into symmetric and public key schemes, reflecting the type of cryptography utilised.

In general, contributory and symmetric distributive schemes are either less robust to insider attacks or do not scale as well as public key schemes. The reason lies in the fact that a group of nodes share a common secret. If large groups share a key, a lot of nodes will be compromised if one of the nodes are compromised[32]. On the other hand, if the groups sharing key are small (e.g. pairwise sharing), much storage space is needed to share a key with many MANET nodes.

Hegland et al. [35] survey 25 key management schemes and consider their applicability in a MANET. The authors conclude that the contributory schemes surveyed are inapplicable in a MANET, due to authentication or scalability issues. The distributive symmetric schemes are deemed too reliant on a static infrastructure. All the nine public key schemes surveyed utilise threshold cryptography to distribute a trusted authority.

Zhou and Haas [69] were the first to propose the use of threshold cryptography in MANETs. In their proposal, a distributed (threshold) `CA` issues certificates to joining nodes. A threshold `CA` is essentially a shared signature function (see Section 4.3). Threshold cryptography gives availability and secrecy to the signature function, but their approach has drawbacks in that a fixed set of nodes act as the threshold `CA`. These nodes may disappear or be far away when needed. Kong et al. [43] extends the model proposed by Zhou and Haas by allowing any node to take part of admission control, but their scheme is limited to RSA[33]. A `PKI` based approach requires certificate exchanges, which introduces heavy communication overhead.

In `IBC`, a distributed `PKG` entails a function sharing of the *Extract* algorithm (see Section 3.2.2). Of the public key schemes surveyed by Hegland et al., a protocol utilising `IBC` and threshold cryptography is viewed as one of the most interesting options. We will discuss this protocol in detail.

[32] This is also true for schemes where only one "master" node shares keys with all other nodes: if the master node is captured, the whole system is compromised.

[33] We will consider the scheme of Kong et al. more detailed in Section 9.2.

5.1 An identity-based threshold scheme

Khalili et al. [42] propose a key distribution protocol for MANETs that utilise IBC and threshold cryptography. The authors do not assume any preestablished trust, so no key distribution has taken place. Their protocol intends to provide a key infrastructure that other protocols, such as secure routing protocols, can be built upon.

Initially, some nodes decide to form a network and agree upon some security parameters. Nodes that do not agree on the parameters may refuse to take part of the network. This initial set of n nodes forms a Distributed Private Key Generator (DPKG), by using threshold cryptography (see Chapter 4). We will refer to a node in this set as a DPKG node. The authors require that the number of corrupted DPKG nodes, t, is less than $n/3$ in the network formation phase.

When a new node wants to join the network, it sends its identity, possibly along with some other information, to at least $t + 1$ of the DPKG nodes, which respond with a share of the node's private key[34]. If the node obtains $t + 1$ correct shares, it can construct its private key. It is assumed that an efficient local mechanism is in place to check the correctness of the shares of the private key and the private key itself. Since this protocol could be a prerequisite for a secure routing protocol, it may be the case that only direct communication is possible in this phase (if unsecured routing is deemed insufficient). If $t + 1$ DPKG nodes can not be reached directly, the authors suggest using mobility to reach more DPKG nodes.

One possible attack from the adversary is to present another node's identity to the DPKG nodes and request its private key. To counter this, two conventions are suggested by the authors. Fist, the DPKG nodes will not issue the same private key twice. This would stop the adversary from obtaining a private key after the right node has received it. Second, all nodes may use unpredictable identities. This can make it hard for the adversary to obtain another node's private key from the DPKG before the right node requests it[35].

Advantages As the protocol description is rather imprecise and abstract, we will focus on its two main properties: the PKG is distributed using threshold cryptography, and identity-based cryptography is used instead of tradi-

[34]It is not stated how the dealer, which creates the private key shares, is constructed. Probably, a function sharing scheme is assumed (see Section 4.3), where the *Extract* algorithm of the IBC system is the function being shared (see Section 3.2.2).

[35]However, if DPKG nodes keep a list of identities they accept and the adversary can control a DPKG, he might see which nodes he should impersonate. Additionally, man-in-the-middle attacks are possible.

tional public key cryptography.

First, by using threshold cryptography to distribute the actions of the PKG, we have no single point of failure, i.e. the PKG becomes more available. This is important in a MANET, since arbitrary nodes may become unavailable due to failures (e.g. hardware problems) or loss of connectivity. It will also make it harder for the adversary to cause denial of service by corrupting the PKG. In addition, the system is more resistant to compromise since $t+1$ nodes must be captured to construct the system's private key.

Second, identity-based cryptography makes the system more efficient than a PKI-based scheme. Most importantly, communication overhead for certificate distribution is eliminated. This is very important in a MANET, since public key certificates are usually 600 – 3000 bits in size, depending on the cryptographic algorithms used, which is several orders of magnitude the size of control messages used by routing protocols. In addition, certificates do not need to be stored (saving memory) or verified (saving computation)[36].

Challenges Khalili et al. stress not to make many assumptions when describing the protocol. This has the unfortunate effect that no guarantees on the security or efficiency can be made. In addition, a range of problems are not addressed in the proposal, some of which are pointed out by Merwe et al. [47].

The most important question is perhaps how joining nodes are authenticated before they receive private key shares. As the authors do not assume any preestablished trust relationship between the nodes, no cryptographic keys or certificates can be used for authentication. Thus no assumptions can be made on "external" identities of nodes (e.g. the name of the person operating a wireless device).

Related to the issue of authentication is also the question on how to avoid that a joining node's private key shares are compromised. Encryption is a useful tool to protect the confidentiality of the key shares, but confidential channels are not sufficient to keep the key shares secret. Authentication of both the DPKG nodes and the joining node is required to defend against man-in-the-middle attacks.

Only direct communication is used to reach other nodes when running the key management protocol, because secure routing protocols rely on key management. This implies that during network formation, all pairs of nodes must be within communication range. A node joining the network after its formation must directly reach at least $t + 1$ of the DPKG nodes, perhaps

[36]Key authenticity checks are not needed in an IBC system since the PKG is responsible for properly authenticating nodes before they receive their private key.

by utilising mobility. Relying only on direct communication makes the DPKG nodes less available. It may also weaken the security of the threshold scheme, because if $t+1$ DPKG nodes are within direct communication range, they can not be very far away, which might make it easier to compromise them (and thus the whole system). It would probably be better to utilise an unsecured routing protocol rather than direct communication when running the key management protocol.

The threshold parameters t and n are fixed since network formation, and the same nodes comprise the DPKG for the whole lifetime of the system. If enough of these nodes are disconnected, PKG services become unavailable. Mechanisms for dynamically changing n (and possibly t) during the network lifetime is preferable.

The DPKG nodes should not issue the private key corresponding to the same identity twice. But how can we ensure that all disjoint sets of $t+1$ DPKG nodes do not issue the private key corresponding to the same identity?

Khalili et al. do not describe how the nodes participating in network formation, and generate the threshold and IBC system parameters, authenticate each other. Without authentication, one malicious node can pretend to be $t+1$ nodes to obtain the IBC master-key. It seems like the authors have overlooked this issue.

Lastly, during network formation, some mechanism is needed to ensure that t corrupted nodes do not stop the honest nodes from agreeing on and generating system security parameters. Khalili et al. require $t < n/3$ during network formation. There are indeed information-theoretic[37] results proving that any function can be computed in this scenario [16], but pairwise secure channels are a prerequisite, and generic solutions are too inefficient in practice.

Summary Khalili et al. stress to make few assumptions in their protocol, which results in a very generic protocol with no guarantees regarding security or efficiency. The main idea is to combine IBC and threshold cryptography, which indeed has a great potential in a MANET. However, as the authors agree, security needs to be defined and proved in terms of a model. To achieve this, more concreteness and assumptions are needed.

[37]The information-theoretic model is a model for communication. It assumes that all pairs of nodes have secure (authenticated and confidential) channels, even against an adversary with unbounded computing power.

6 A Distributed Private Key Generator Framework

6.1 Introduction

The Private Key Generator (PKG) plays an important role in an IBC system as it is trusted to store and use the master-key of the system. As described in Section 3.2.2, an IBC system consists of four algorithms, where the PKG is involved in two of these: *Setup* and *Extract*.

However, in a MANET, keeping a central PKG available and properly secured at all times is challenging, at best. Motivated by our work with threshold cryptography in Chapter 4, we will consider how multiple nodes can share the authority of the PKG in a MANET. When the PKG authority is shared, we will refer to it as a *Distributed Private Key Generator* (DPKG). A DPKG consists of two protocols corresponding to the *Setup* and *Extract* algorithms, respectively. The first creates a secret sharing of the IBC system's master-key, while the latter is a function sharing of *Extract*. A node having a share of the system's master-key will be referred to as a *DPKG node*.

As we noted in Chapter 5, many proposals for distribution of a CA or PKG exist. Some of the issues addressed in the proposals overlap, while others do not. For example, every proposal include a discussion on how a joining node obtains its private key corresponding to the system parameters. However, the issue of changing the set of DPKG nodes is not always addressed. Additionally, we argue that the meaning of security for a DPKG should be uniform. For these reasons, a generic characterisation of a DPKG is useful. We will consider this issue in the context of a MANET, and introduce a few concepts in the following to structure our work.

A major contribution in our following work is the definition of security for a Distributed Private Key Generator. As far as we know, this has not been addressed earlier. The definition may be used in any IBC system where threshold cryptography is applied to distribute the PKG authority. We use the work on a secure threshold signature scheme by Gennaro et al. [31] as a basis for our definition, as we observe the similar characteristics of a Distributed Private Key Generator and a threshold signature scheme.

In Chapter 7, we present a concrete DPKG, and prove its security in the definition we state in this Chapter. Chapter 7 will thus serve as a proof of concept for the work we do in Chapter 6.

6.2 Network formation

In the most general scenario, a set of nodes within communication range decide to form a network. We will refer to these nodes as the *founder nodes*, and let m be the number of them. We will consider what needs to be done before a DPKG is created using distributed key generation.

As usual, we choose a positive integer t and assume that at least $m - t$ nodes behave according to our instructions, while at most t nodes are corrupted and may act in any way. Note that it is very desirable to assume $t < m/2$, because this will ensure that there are always $t + 1$ honest nodes available, which implies that the secret is reconstructible and a shared function may be computed. One might think that threshold cryptography can be applied immediately by the founder nodes, but this is not the case because pairwise secure channels are a prerequisite[38]. Without authentication, a corrupted node may pretend to be more than one node. In this way, the adversary might effectively be in control of $t + 1$ nodes, while the honest nodes will believe that there are *more* than m founder nodes after this attack[39]. We will thus need an authentication mechanism that allows pairwise node authentication. This issue will be considered in Section 6.8.1. Note that authentic channels can be made secure by using the Diffie-Hellman key exchange.

6.3 Distributed Setup

During distributed *Setup*, we assume secure channels between all pairs of nodes, as constructed in the network formation phase. Recall that in an IBC system, the purpose of *Setup* is to generate system parameters. Note that most of the system parameters (e.g. hash functions, pairings, etc.) might be preestablished as universal standards because they are hard to generate, without compromising security. To generate the master-key in a distributed fashion, we can use a distributed key generation protocol (see Section 4.6). The protocol run to generate system parameters will be denoted DS. We say that DS *completes successfully* if it generates system parameters that have the same probability distribution as the parameters generated by *Setup*.

After DS is run, a MANET certificate (as described below) may be created, which can be presented to potential joining nodes.

[38]For example, the combiner needs to obtain shares (secret sharing) or partial results (function sharing) to compute a secret value. If the adversary can read the messages sent, the secret information will be compromised.

[39]To guarantee that the adversary never obtains $t+1$ shares, we might define $t = m-1$, assuming at least one honest node. However, this is undesirable since we want $t < m/2$.

6.4 MANET Certificate

Before nodes are able to carry out any cryptography in the network, they will need to obtain the IBC system parameters. A node may also want to be assured that the network is authentic, i.e. not set up by some adversary, before joining. Depending on the scenario, this might be achieved by having some entity it trusts sign the system parameters. For example, if the joining node already has an external public key of one of the founder nodes, it may regard the system parameters authentic by receiving a signature over them, with respect to this public key. In addition, some network-wide policies might be in place, such as the valid format of an identity, the maximum number of nodes, threshold parameters, etc. All this information should be available to a node before it joins the network. We collect this information and require it signed by the PKG authority to protect its authenticity, and refer to it as the MANET *certificate*, denoted MCert. It can be signed with the authority of the PKG as discussed in Section 6.9. Without going too deep into detail, Table 6 gives entries that seem natural to include in the MCert. The entries that are not self-explanatory, will be discussed later.

IBC system parameters
Signatures by external entities
MCert version
master-key share names and commitments (x_i, c_i)
Threshold parameters t and n
Revoked identities
DPKG signature prefix policy
Self-signature by PKG authority

Table 6: Natural entries in the MANET certificate.

Also note that any MANET with a (possibly distributed) authority will need an MCert to distribute the public parameters, but the MCert may well be preestablished before network formation. This merely shows that an MCert is a generic construction applicable in any MANET with an authority.

Dynamic MCert Note that the MCert may change during the network lifetime. For example, the master-key share names and commitments[40] may change if the set of DPKG nodes changes (see Section 6.7), and new identities might get revoked. We therefore include a version number in the MCert,

[40]We can use *commitments* to secret shares to check if a share-function is correctly evaluated (e.g. when receiving the response from a distributed *Extract*).

which will be incremented with every change. The signatures by the external entities are only computed over the system parameters, to avoid the need to recompute them. But this is sufficient to gain trust in the whole MCert[41].

6.5 Distributed Extract

Suppose we have established IBC system parameters and a corresponding MCert. The next issue to consider is how a joining node J with identity ID_J obtains its private key. Recall that in an IBC system, it is the *Extract* algorithm that generates private keys. When the master-key is secret shared, *Extract* is a distributed algorithm, also known as a protocol, run by DPKG nodes. We denote it DE.

Each DPKG node participating in DE is given ID_J and has its secret share of the master-key as private input. In addition, all DPKG nodes and J know the IBC public system parameters. This might be obtained from the MCert. We assume that DPKG nodes have pairwise authenticated and confidential channels to J. Note that we do not consider how access control is carried out by the DPKG nodes here, but assume that J is authorised to obtain the IBC private key corresponding to the identity ID_J.

The protocol DE consists of two phases. In the first phase, the DPKG nodes generate shares of the private key corresponding to ID_J and send them securely to J. Secondly, J constructs its private key from these shares. We say that DE *completes successfully* if for all honest J and identities ID_J, J always obtains *Extract* corresponding to ID_J.

6.6 DPKG security definition

Assume we have protocols DS for distributed *Setup* and DE for distributed *Extract*, as previously discussed. The system parameters will be created by running DS once. After this, DE can be run arbitrary many times to generate private keys. A DPKG consists of the protocols (DS, DE).

When defining security for a DPKG (DS, DE), we will follow the lines of Gennaro et al. [31] as they define security for a threshold signature scheme. The reason for this is that we observe the similar characteristics of a threshold signature scheme and a DPKG. There are two protocols in both scenarios, and the first is a distributed key generation protocol (see Section 4.6). In a threshold signature scheme, the second protocol is a shared signature function, that on input a message M, a public key, and a secret shared private

[41] These signatures create trust in the system parameters. Trusting the system parameters means trusting the PKG authority. All entries can then be trusted, since they are signed by the PKG authority.

key outputs the signature σ of M. This protocol corresponds to distributed *Extract* by inputing an identity ID instead of M and obtaining a private key d_{ID} instead of σ. The only difference in the protocols is that d_{ID} should be privately output to the owner of the identity ID, while the signature σ may be publicly known. We do not need to worry about a threshold signature scheme in our discussion, but merely note that we follow the lines in a security definition of such a scheme during our work.

In the following, we assume a *static adversary*, i.e. the adversary must choose which nodes to corrupt prior to execution of any protocol, and he cannot change the set of corrupted nodes. We require a secure `DPKG` to be robust and free of information leakage. We precisely define these notions next, starting with the former.

With robustness, we mean that a `DPKG` generates correct output, even when an active adversary controlling up to t nodes is participating in the protocol.

Definition 6.1. A Distributed Private Key Generator (DS, DE) is *robust* if for all probabilistic polynomial time adversaries $\mathcal{A}$ that are allowed to corrupt up to t nodes, both protocols DS and DE complete successfully.

Next, let us consider what it means for a protocol to not leak information. First note that an adversary controlling t nodes can see the internal state of these nodes and all communication to and from them, which we will call the adversary's view of the protocol. But we want to show that this does not give the adversary any useful information. In order to achieve this, we use the notion of *simulatable adversary view* [56, 6]. This means that given the input to and output from the protocol, the adversary's view of the protocol can be simulated by an algorithm called the simulator. The rationale behind this is that an adversary seeing all information in corrupted nodes, together with public input and output, during an execution of the protocol, might just as well run the simulator to generate this information on his own as participating in the protocol. Thus, the simulator shows that the adversary's view of the protocol does not give him any extra information over the public output from the protocol. We say that the protocol is simulatable if we can create a simulator for it no matter how the adversary $\mathcal{A}$ behaves.

Definition 6.2. Let $\mathcal{A}$ be a probabilistic polynomial-time adversary who corrupts up to t nodes. A Distributed Private Key Generator (DS, DE) is *simulatable* if there exists a probabilistic polynomial-time simulator SIM that on input `IBC` public parameters P, an identity ID and corresponding private key d_{ID}, can simulate the view of $\mathcal{A}$ on an execution of DS outputting P and a following execution of DE outputting d_{ID}.

By that SIM can simulate the view of $\mathcal{A}$, we mean that the probability distribution of the output from SIM is in polynomial time indistinguishable from the distribution of the adversary's view of DS and DE. Another way to say this is that the probability distributions from the protocol and simulator are computationally indistinguishable. As mentioned earlier, we require a secure DPKG to be both robust and simulatable.

Definition 6.3. A Distributed Private Key Generator (DS, DE) is a *secure Distributed Private Key Generator* if it is robust and simulatable.

Note that this definition of security only ensures that the protocols do not leak *more* information than the *Setup* and *Extract* algorithms they are based upon. This is however very desirable. In an IBC system, *Setup* and *Extract* are always used in conjunction with signature or encryption algorithms. The IBC system is then usually proved secure in some security model. Using distributed versions of the *Setup* and *Extract* algorithms that is secure in our definition thus results in an IBC system that is secure in the same security model. Our definition thus ensures *composability*.

6.7 Dynamic set of DPKG nodes

As we noted when discussing some proposals for distributed authorities in MANETs in Chapter 5, it may be useful to dynamically adjust the set of DPKG nodes. For example, if the network initially consists of five nodes, but 50 more nodes join, more nodes should perhaps become DPKG nodes in order to keep PKG services efficiently available to the whole network. In addition, a mechanism for revoking the DPKG privileges from a node is useful if a node is known to be compromised. Thus, we may want to make any node a DPKG node, which we will refer to as *DPKG promotion*, and revoke the DPKG privileges, which we refer to as *DPKG degradation*.

Dynamic threshold Related to the previous discussion is also how to change the threshold parameter t. If the number of DPKG nodes n increases due to DPKG promotion, one might think that capturing any $t + 1$ of these becomes easier, and thus perhaps t should be increased as well. On the other hand, if n is decreasing (e.g. due to failing nodes), it could be useful to decrease t to avoid that $n < t + 1$, in which case the secret is clearly lost.

6.8 Authentication requirements

During our discussions on network formation, distributed *Extract*, and DPKG promotion, we have assumed existence of authentic channels[42]. Because MANETs have a variety of applications, we can not determine authentication levels that fits all MANETs — this is rather the subject of a security policy designed for a specific MANET. But from our previous discussions, we can point out the *minimum* authentication requirements. Note that if nodes meet physically, all authentication issues can be solved.

6.8.1 Network formation

The authentication carried out during network formation should give pairwise authentic channels between the founder nodes. But the requirement on the authentication is, in a sense, weak. Namely, a node should be authenticated if it has not *previously* been authenticated, i.e. a node should not be authenticated twice. This prevents an adversary controlling t nodes to effectively control more than t nodes.

One way to achieve this is to assume that every node has exactly one public key certificate, and define node authentication as certificate verification. Another, perhaps less useful, way is to assume only a passive adversary, i.e. an adversary that never sends messages. An authentication request message can always be accepted in this scenario. We will consider an approach in-between these two in Chapter 8.

6.8.2 Distributed Extract

During distributed *Extract*, the joining node J is assumed to have authentic channels to all DPKG nodes. The minimum authentication requirement here is that it is the *same* node J that request the private key corresponding to ID_J to all DPKG nodes. If this was not the case, an adversary could, once he received a request for the private key corresponding to ID_J, request the same private key from any other DPKG node. If just one request succeeds and he controls t DPKG nodes, he can obtain $t + 1$ *Extract* share-function evaluations, which allows him to compute the private key. The adversary could also potentially request the private key at a later point in time.

Note that we do not consider who should be granted the private key corresponding to an identity, but leave this as a security policy issue. For

[42]Recall that a pairwise authentic channel can be made confidential by using the Diffie-Hellman key exchange. Thus, a pairwise secure channel is available if a pairwise authentic channel is.

example, the adversary may well be granted the private key for any identity he wishes, as long as no other node has successfully obtained the same private key first.

Also note that, if only nodes with sufficient credentials, such as possession of a private key corresponding to a certificate, should be given the private key corresponding to an identity, it is enough that $t + 1$ DPKG nodes accept the credentials. Therefore, every DPKG node does not always need to store all the accepted credentials, which may save some storage. It also allows DPKG nodes to act more independently, since they do not all need to trust the same entities.

6.8.3 DPKG promotion

The minimum authentication requirement for DPKG promotion is almost the same as the one for network formation; a node should not have more than one share of the master-key. It should thus not be authenticated twice and receive distinct shares. But receiving the same share multiple times is allowed.

However, unlike during network formation, the node might have obtained a private key through distributed *Extract*, which may render authentication easier here. In addition, we can enforce a more restrictive authentication policy since it is generally not necessary that all MANET nodes are DPKG nodes, and if more than t malicious nodes become DPKG nodes, the PKG is compromised.

6.9 DPKG signature scheme from IBE

It is very useful to have a mechanism for allowing $t + 1$ DPKG nodes to sign some information as the PKG authority. This allows the creation of revocation lists, for example. But which key(s) should be used to create and verify the signature, and how are private and public parts of the key(s) distributed? Only $t + 1$ DPKG nodes should be able to sign, while any node should be able to check a signature. This seems like a difficult task.

However, the DPKG nodes have a function sharing of the *Extract* algorithm, as discussed in Section 6.5. In any IBE system, *Extract* may actually be used as a signature function, while *Encrypt* and *Decrypt* constitute the verification algorithm. This is, according to Boneh and Franklin [12], observed by Moni Naor. The method is shown in Figure 17. The security of the scheme rests on the security of the underlying IBE system; forging a signature on a message M is equivalent to generating a private key for $ID = M$ (without knowing the master-key) in the underlying IBE scheme. This can be stated more precisely in terms of security models [12].

- **Setup**
  The public key is the IBE system parameters.
  The private key is the corresponding master-key.

- **Sign**
  Given a message $M \in \{0,1\}^*$ and the master-key, run *Extract* on input $ID = M$.
  The private key corresponding to the identity $ID = M$, as returned by *Extract*, is the signature σ on M.

- **Verify**
  Given the IBE system parameters, a message M and a corresponding signature σ, pick a random element R from the IBE plaintext space.
  Run *Encrypt* on R, using $ID = M$, to obtain the ciphertext C.
  Run *Decrypt* on C, using σ as the private key, to obtain R'.
  Accept the signature σ on M if and only if $R = R'$.

Figure 17: Converting any IBE system into a signature scheme.

Distinguishing identities from messages A problem that arises if we allow the system parameters to be used as both an IBE system and a signature scheme is to distinguish an identity from a message. For example, suppose $t + 1$ DPKG nodes is convinced that "Eve" is compromised and thus create a signature σ on the message $M =$ "Bad Eve" [43]. Now, any node obtaining σ may claim to have the *identity* "Bad Eve" as σ is the identity's private key.

To counter this, we can create a policy stating that messages are always prefixed with "Sign:" before they are signed. This policy is then incorporated together with the system parameters into the MCert, as described in Section 6.4. If we apply this, the message signed by the DPKG nodes in our example above would be $M =$ "Sign:Bad Eve" instead. DPKG nodes will never issue private keys for identities starting with "Sign:", so every node having the MCert would know that M is not an identity. Of course, any other method for recognising a message over an identity is possible, as long as it is known by all nodes (e.g. stated in the MCert).

We have thus successfully constructed a DPKG signature scheme without introducing any new keys or algorithms into the system. The security of the scheme rests on the security of the distributed *Extract*. Furthermore, note that the signatures can be very short, about 160 bits if the IBE system

[43] This is indeed a strange revocation list, but the point should become clear.

described in Section 3.2.3 is used.

6.10 Summary

We have discussed important aspects of establishing and maintaining a Distributed Private Key Generator in a network, and identified a need for three protocols to achieve this. The input assumptions and result of these protocols are summarised in Table 7.

Protocol	Input assumption	Result
Network formation	A set of founder nodes	Pairwise secure channels
Distributed Setup	Pairwise secure channels	IBC system parameters (possibly incorporated into a MCert) with secret shared master-key
Distributed Extract	IBC system parameters with secret shared master-key, secure channels between joiner J with identity ID_J and DPKG nodes	J obtains the private key corresponding to ID_J

Table 7: Protocols for DPKG operation.

Our most important contribution is the definition of security for the distributed *Setup* and distributed *Extract* protocols. In addition, we saw that support for DPKG promotion and degradation may be useful in a MANET, and noted that a DPKG signature scheme can be created without any extra cost of key management. We also suggested the notion of a MANET certificate to present the IBC system parameters to potential joining nodes.

We will realise a Distributed Private Key Generator in Chapter 7, showing that implementations for the protocols exist. We also prove that the construction is secure with respect to our definition.

7 A Concrete Distributed Private Key Generator

7.1 Introduction

In this Chapter, we will use the framework discussed in Chapter 6 to create a concrete DPKG which is applicable in a MANET. We base ourself on the IBC schemes by Boneh and Franklin (see Section 3.2.3) and Cha and Cheon (see Section 3.2.4), which gives us both identity-based encryption and signatures. We will refer to these schemes as the BF-IBE and CC-IBS scheme, respectively. The purpose of our work then, is to securely distribute the *Setup* and *Extract* algorithms. We assume $\mathbb{G}_1$=$\mathbb{G}_2$ such that the *Setup* and *Extract* algorithms of the schemes become equivalent, except for the hash functions required. When we refer to system parameters in the following, we mean the parameters as defined in the *Setup* algorithms of these schemes.

The distributed PKG that we present was sketched in a paper by Boneh and Franklin [12]. However, to the best of our knowledge, no security analysis of this system has been presented. We show that the security properties of the base schemes BF-IBE and CC-IBS are preserved when the distributed version of the PKG is used.

7.2 Network formation

We use a non-standard way of authenticating founder nodes, as described in Chapter 8. To this end, we assume that the honest nodes have about the same computational capacity. In addition, we assume that the total computational power of the adversary does not exceed the total computational power of the honest nodes.

The authentication mechanism we discuss in Chapter 8 may then be used to create authenticated channels between the founder nodes. Confidentiality is created by running the Diffie-Hellman key exchange pairwise between all authenticated nodes. The result is secure channels between the founder nodes, as desired. If the assumptions on the adversary hold, less than half of the endpoints are controlled by the adversary. We let n be the total number of successfully authenticated nodes.

7.3 Distributed Setup

Recall the *Setup* algorithms of the BF-IBE system in Figure 11, and the CC-IBS system in Figure 12. We assume that the parameters $\{r, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3, e, P\}$ and the hash functions either are universal standards accepted by all nodes

or can be agreed upon after network formation[44]. This assumption can be justified since it is not necessary to choose random groups or hash functions when establishing system parameters, as long as the selected parameters do not contain particular weaknesses. For example, National Institute of Standards and Technology [51] has composed a list of recommended elliptic curves which is widely used. Choosing specific parameters also gives optimisation opportunities.

The only missing system parameter is thus $P_{pub} = sP$. However, the master-key s is to be secret shared between the DPKG nodes. To this end, we use the distributed key generation protocol by Gennaro et al. [32], which will implement the DS protocol from Section 6.3. We name it DS_I. The input to the protocol is $\{r, P, t, n\}$ [45], where t and n are the desired threshold parameters. After the completion of the key generation protocol, each node has privately obtained a Shamir secret share $s_i \in \mathbb{Z}_r$ of the master-key $s \in \mathbb{Z}_r$ (unknown to all nodes), while the corresponding commitment $c_i = s_i P \in \mathbb{G}_1$ is known by all nodes. The protocol also outputs the last system parameter $P_{pub} = sP \in \mathbb{G}_1$ to all nodes.

As a last step, one might incorporate the public parameters and share commitments into a MANET certificate and sign it using the technique discussed in Section 7.7. This can be done by any $t + 1$ honest nodes. Founder nodes can also sign the certificate with any external keys they may have.

7.4 Distributed Extract

For an identity ID, *Extract* corresponding to ID is in BF-IBE and CC-IBS defined as $s\mathcal{H}_1(ID)$. Recall from Section 6.5 that the corresponding distributed *Extract* protocol consists of two phases. Firstly, DPKG nodes use algorithms $\mathcal{E}_i$ to generate shares of a private key. Secondly, the joining node uses two algorithms; $\mathcal{V}$ verifies that a share-function evaluation is correct, while $\mathcal{R}$ computes the private key given enough correct share-function evaluations. We describe these three algorithms in detail.

1. $\mathcal{E}_i$: Given $i \in [1, n]$, the hash function $\mathcal{H}_1$ from the system parameters and the master-key share s_i, for an identity ID, we define

$$\mathcal{E}_i(ID) = s_i \mathcal{H}_1(ID)$$

[44] Each node can possibly have a set of preestablished parameters that it accepts. Any non-empty intersection of accepted parameters between the nodes may be used.

[45] Note that P is a generator of $\mathbb{G}_1$ since it has prime order. In the protocol description, $q = r$ and $g = P$.

2. $\mathcal{V}$: Let e be the pairing, $\mathcal{H}_1$ the hash function and P the point from the system parameters. Let $i \in [1, n]$. For an identity ID, suppose we are given a share commitment c_i and a value v_i, we check if $v_i \in \mathbb{G}_2$ and
$$e(P, v_i) = e(c_i, \mathcal{H}_1(ID))$$
and define $\mathcal{V}$ to output 1 if both checks hold, and 0 otherwise. Proposition 7.1 shows that $\mathcal{V}$ outputs 1 if and only if $v_i = \mathcal{E}_i(ID)$ [46].

3. $\mathcal{R}$: For an identity ID, without loss of generality, assume we are given $t+1$ share-function evaluations $\mathcal{E}_1(ID), \mathcal{E}_2(ID), \ldots, \mathcal{E}_{t+1}(ID)$. Recall the recombination vector $(\delta_1(0), \delta_2(0), \ldots, \delta_{t+1}(0))$, as defined at the end of Section 4.2.1. We compute
$$\sum_{i=1}^{t+1} \delta_i(0)\mathcal{E}_i(ID) = \sum_{i=1}^{t+1} \delta_i(0) s_i \mathcal{H}_1(ID) = s\mathcal{H}_1(ID) = d_{ID}$$
and define d_{ID} to be the output of $\mathcal{R}$. This is seen to be *Extract* corresponding to ID as defined in BF-IBE and CC-IBS.

Proposition 7.1. *Let $e, \mathcal{H}_1, P, i, c_i$ and v_i be as in the description of $\mathcal{V}$ above. Then $v_i = \mathcal{E}_i(ID)$ if and only if $e(P, v_i) = e(c_i, \mathcal{H}_1(ID))$.*

Proof. 1. Assume that $v_i = \mathcal{E}_i(ID)$. Then
$$e(P, v_i) = e(P, \mathcal{E}_i(ID)) = e(P, s_i\mathcal{H}_1(ID)) = e(s_iP, \mathcal{H}_1(ID)) = e(c_i, \mathcal{H}_1(ID))$$

2. Conversely, assume $e(P, v_i) = e(c_i, \mathcal{H}_1(ID))$. Recall that by definition of BF-IBE and CC-IBS, $P \neq O$. We first take the case $\mathcal{H}_1(ID) = O$. We have $e(c_i, \mathcal{H}_1(ID)) = e(c_i, O) = 1 = e(P, v_i)$. Proposition 3.6 on page 27 implies $v_i = O$. We also have $\mathcal{E}_i(ID) = s_i\mathcal{H}_1(ID) = s_iO = O$, so $v_i = \mathcal{E}_i(ID)$ in this case.

   Now consider the case $\mathcal{H}_1(ID) \neq O$. Since $\mathcal{H}_1(ID) \neq O$, and $\mathbb{G}_2$ has prime order, $\langle \mathcal{H}_1(ID) \rangle = \mathbb{G}_2$. Thus $\exists k \in \mathbb{Z}$ such that $v_i = k\mathcal{H}_1(ID)$. Since $P \neq O$ and $\mathcal{H}_1(ID) \neq O$, Proposition 3.6 implies $e(P, \mathcal{H}_1(ID)) \neq 1$. As we noted at the end of Section 3.1.3, $e(P, k\mathcal{H}_1(ID)) = e(s_iP, \mathcal{H}_1(ID))$ if and only if $k \equiv s_i \pmod{r}$. Thus, $v_i = k\mathcal{H}_1(ID) = s_i\mathcal{H}_1(ID) = \mathcal{E}_i(ID)$.

□

Using these three algorithms, we describe the distributed *Extract* protocol in Figure 18. This is the implementation of the DE protocol from Section 6.5, which we will name DE_I.

[46] A subtle issue here is that we need an efficient algorithm for testing membership of $\mathbb{G}_2$. We assume that this can be achieved, but note that it might be a problem in practice.

INPUT:	An identity ID_J, public IBC and threshold parameters. DPKG nodes have their shares. J has commitments $(c_1, c_2, \ldots, c_n)$ to the shares.
OUTPUT:	J obtains the output of *Extract* corresponding to ID_J

For $i \in [1, n]$, DPKG node i does the following

1. Compute $\mathcal{E}_i(ID_J)$.
2. Use a pairwise secure channel to send $\mathcal{E}_i(ID_J)$ to J.

J does the following

1. For each value v_i received, verify it with $\mathcal{V}$, and assume without loss of generality that $v_1, v_2, \ldots, v_{t+1}$ pass verification.
2. Run $\mathcal{R}$ on $v_1, v_2, \ldots, v_{t+1}$ to obtain d_{ID}.
3. Return d_{ID}.

Figure 18: Distributed Extract algorithm DE_I.

7.5 Proof of security

We now prove that a combination of the DS_I and DE_I protocols described above form a secure Distributed Private Key Generator in the sense of Definition 6.3. Our proof will follow the lines of a proof of security for the threshold BLS signature scheme by Boldyreva [10, Theorem 4.2]. The rationale for this is that this particular threshold signature scheme uses protocols identical to (DS_I, DE_I). Thus, the only difference between our scenario and the one considered by Boldyreva is interpretation; we consider the output from DE_I as a private key, while Boldyreva consider it as a digital signature. As a consequence of this, the definitions of security differ slightly also [10, Definition 4.1].

Recall from Section 3.1.3 that a group $\mathbb{G}$ is a Gap Diffie-Hellman group if the CDH problem is hard, while the DDH problem is easy in $\mathbb{G}$.

Theorem 7.2. *If $\mathbb{G}_1$ from the system parameters is a Gap Diffie-Hellman group and the adversary corrupts no more than $t < n/2$ DPKG nodes, (DS_I,DE_I) is a secure Distributed Private Key Generator.*

Proof. Recall that the robustness condition of Definition 6.3 means both DS_I and DE_I complete successfully. For DS_I, Gennaro et al. [32, Theorem 1] assume the discrete logarithm problem is hard in $\mathbb{G}_1$, and prove that the master-key s generated is uniformly random in $\mathbb{Z}_r^*$ and all subsets of $t+1$ shares s_i define the same s corresponding to the unique public key $sP = P_{pub}$. This also holds in the presence of an active adversary corrupting no more than $t < n/2$ nodes. This implies that the parameters generated by DS_I have

the same distribution as *Setup* in BF-IBE and CC-IBS, and thus DS_I completes successfully.

Next, we consider robustness with respect to DE_I. In Proposition 7.1, we show that $\mathcal{V}$ can always distinguish correct share-function evaluations from other values. We furthermore know that $\mathcal{R}$ always constructs *Extract* corresponding to an identity given $t+1$ correct evaluations. Since there are at least $t+1$ honest nodes, DE_I will complete successfully.

Now we will show that (DS_I,DE_I) is simulatable. A simulator for a threshold version of the BLS signature scheme is constructed by Boldyreva [10, Theorem 4.2]. This threshold signature scheme uses exactly the DS_I and DE_I protocols from our DPKG, but interprets the output from DE_I as a signature rather than a private key. Thus, the simulator due to Boldyreva shows that (DS_I, DE_I) is simulatable when it is interpreted as a DPKG also. □

Boneh and Franklin [12] show that their IBE system is secure against an adaptive chosen ciphertext attack, assuming the hardness of the BDH problem. The IBS system of Cha and Cheon [15] is secure against existential forgery under an adaptive chosen message attack, assuming the CDH problem is hard. As we noted in Section 6.6, these results still hold when the PKG is distributed with (DS_I,DE_I).

7.6 Dynamic set of DPKG nodes

We will sketch a protocol that performs DPKG promotion. However, note that the security definition and proof do not take this protocol into account. Any security implications should be examined before using it as a part of the DPKG system.

DPKG promotion entails issuing a new share s_{n+1} of the master-key s. We require that the node which should receive s_{n+1} has obtained a private key from DE_I, and assume a mechanism to verify that the node does not already have a share of the master-key, other than possibly s_{n+1}. This is needed by the minimum authentication requirement, as discussed in Section 6.8.3.

Recall the notation used when discussing Shamir's secret sharing in Section 4.2.1. We can create a new share with name x_{n+1} if we have $t+1$ (distinct) evaluations $s_i\delta_i(x_{n+1})$, by computing $s_{n+1} = \sum_{i=1}^{t+1} s_i\delta_i(x_{n+1})$. This works, but allows the entity that computes s_{n+1} to find all s_i used (and thus also s) since the $\delta_i(x_{n+1})$ are publicly known. One way to prevent the s_i's from leaking is to use techniques of proactive secret sharing; add a random value to each share, a random sharing of zero (see Section 4.5).

A drawback is that proactive secret sharing introduces much communica-

tion overhead[47]. But more importantly, we do not cover this DPKG promotion protocol in our security definition and proof. Thus, we can not, without further investigation, give any guarantees for the security of the system when the DPKG promotion protocol is used.

Now consider DPKG degradation. It cannot "take back" a secret share, but it can revoke identities and keys (see Section 3.2.6) and invalidate a share by using proactive secret sharing (see Section 4.5).

Dynamic threshold There are many proposed schemes for achieving a dynamic threshold; Steinfeld et al. [64] have written a section with related work which gives an overview. Some schemes assume pairwise secure channels between the shareholders, and these schemes typically enjoy better efficiency due to this strong assumption. Since the DPKG nodes do have pairwise secure channels available, it is natural to choose such a scheme.

In particular, the share redistribution scheme due to Desmedt and Jajodia [23] is well suited for our purposes. The idea is to have each shareholder act as a dealer to create a secret sharing of its existing share, using the desired threshold parameters, and securely send the shares to the new shareholders. A receiver then use the shares it obtained to compute its new share. The resulting sharing can have any agreed-upon threshold parameters. However, a corrupted existing shareholder should be stopped from destroying the secret by dealing bad shares to the new shareholders. To this end, verifiable secret sharing can be employed (see Section 4.6).

7.7 DPKG signatures

In Section 6.9, we noted how a DPKG signature scheme can be created with an IBE system (e.g. BF-IBE) as basis. However, the threshold BLS signature scheme of Boldyreva [10] is constructed with the exact same system parameters as we are using. This scheme enjoys a security proof and the verification algorithm is more efficient than the one in Section 6.9. We can also relax our security assumptions. More precisely, we do not need to assume hardness of the BDH problem, as required by BF-IBE, but we only need the CDH assumption[48].

Furthermore, a signature share-function evaluation can be viewed as a signature by the DPKG node that possesses the secret share needed to compute

[47] An extended version of Shamir secret sharing using bivariate polynomials allows very efficient distributed share generation [22]. However, to generate this type of secret shares the DS_I protocol is replaced, and thus also a new proof of security is needed.

[48] Recall that if we can solve the CDH problem, we can also solve the BDH problem. The converse is however not known to be true.

the share-function. Thus DPKG nodes can also sign messages individually.

7.8 Summary

For an IBC system, a DPKG is created by distributing the *Setup* and *Extract* algorithms, which essentially creates a function sharing of them (see Section 4.3). The *Setup* and *Extract* algorithms of the BF-IBE and CC-IBS systems are identical, and we have described how these algorithms can be distributed. Our main result in this Chapter is a proof of security for this concrete DPKG.

The distributed versions of *Setup* and *Extract* are quite efficient and may probably be used in a MANET. But before using our concrete DPKG in a MANET, there are still important issues that must be solved. The most prominent is authentication (see Section 6.8). We suggest how authentication can be done during network formation in Chapter 8, but we also need authentication for distributed *Extract*. If we want to dynamically adjust the threshold parameters, a third authentication mechanism is also required, but perhaps the other two can be used to construct it. Note that authentication can easily be solved if we assume preestablished trust, e.g. a PKI, or physical meetings, but we try to avoid such assumptions since they can not easily be justified in a MANET.

In addition, more efficient schemes for dynamically adjusting the threshold parameters is desirable. The most useful aspect is probably DPKG promotion, so creating an efficient mechanism for this is perhaps most important. Furthermore, if DPKG promotion is allowed, we should include the desired security properties of DPKG promotion in the security definition and prove that they hold. In Section 9.4, we give a more comprehensive overview of the issues we should consider before this DPKG may be implemented in a MANET.

We discuss the DPKG in Chapter 6 and Chapter 7 in the context of a MANET. However, the interest for creating a DPKG is not limited to MANETs, and thus our work should be applicable in other contexts as well. Furthermore, our concrete DPKG can be used in all IBC schemes that use the same *Setup* and *Extract* algorithms as BF-IBE and CC-IBS. This includes the signcryption scheme we discussed in Section 3.2.5.

8 Autonomous Authentication During Network Formation

8.1 Introduction

Recall that we need an authentication mechanism during network formation, as argued in Section 6.2. A popular approach is to assume preestablished trust between the founder nodes. But this directly contradicts a main property of a MANET, namely self-configuration. For instance, if preestablished trust is proven through public key certificates, self-configuration is limited to nodes that share the public key of a `CA`. Another assumption used is that nodes physically meet to authenticate each other during network formation [22]. This is clearly not a desirable assumption, because human interaction is needed.

On the other hand, schemes that do not assume any preestablished trust or secure channels give no security guarantees. For example, the authors of the key distribution protocol we discussed in Section 5.1 do not even point out that authentication during network formation is an issue. Since unauthenticated founder nodes can create a trusted authority, the adversary may easily gain control of the authority's secret key by receiving enough shares of it.

Therefore, there are currently two approaches to network formation:

1. **Secure**
   A trusted authority is established by means of threshold cryptography, but self-configuration is limited.

2. **Self-configured**
   The network is fully self-configured, but no security assumption can be made on a trusted authority.

We believe that there is a lot of room for solutions in between these two. In the following, we will explore a possible approach to authentication that is fully self-configuring but still has some arguably useful security properties for network formation. We aim for the minimum authentication requirement discussed in Section 6.8.1; a node should be authenticated if it has not previously been authenticated. Furthermore, authentic channels are created between all pairs of founder nodes.

8.2 The problem

We want to create a robust trusted authority by utilising threshold cryptography. Recall that in order to use threshold cryptography efficiently, we need that less than half of the participants are corrupted (see Section 6.2). As our solution should, like a MANET, be fully self-configuring, we can not assume any existing secure channels. But as pointed out in Section 6.2, we can only use threshold cryptography after pairwise authentication of founder nodes; t adversarial nodes can pretend to be more than t nodes and thus compromise the trusted authority. A node that claims multiple identities is often said to carry out a *Sybil attack* [24], and the additional identities are in the following referred to as Sybil identities[49].

Therefore, our task is to limit the number of corrupted and Sybil identities to less than half of the total number of identities. After network formation, the founder nodes will have pairwise authentic channels available.

Notation Let $\mathcal{S}$ be the set of founder nodes, with $\#\mathcal{S} = m$. Of the founder nodes, let $\mathcal{H}$ be the set of honest nodes and $\mathcal{C}$ the set of corrupted nodes, with $\#\mathcal{C} = t < m$. Thus $\mathcal{S} = \mathcal{H} \cup C$. We denote the set of Sybil identities $\mathcal{D}$. We let t_{ID} denote the number of identities controlled by the adversary, thus $t_{ID} = \#(\mathcal{C} \cup D)$. Finally, let $m_{ID} = m + \#\mathcal{D}$, the total number of identities.

With our new notation, our goal can be seen as ensuring $t_{ID} < m_{ID}/2$.

8.3 Resource testing

Resource testing is proposed as a way to disclose Sybil identities [24], and it has also been considered in the context of sensor networks [52]. In resource testing, it is assumed that each physical node is limited in some resource, e.g. computation, storage or communication. If some entity shows that it has sufficient amounts of a given resource, it is assumed to be a physical entity. Newsome et al. [52] argue that testing for computation, storage and communication resources is unsuitable for detecting Sybil identities in sensor networks. This is because the sensor nodes are limited in these resources compared to other computing devices.

We do not claim that the method we sketch solves all authentication problems, but it might be practical in some MANETs. We are also in a slightly different scenario than the one considered by Newsome et al. [52]. Most importantly, we do not need to disclose all Sybil identities, but we want to limit the number of corrupted nodes and Sybil identities in total.

[49]When a corrupted node presents multiple identities, an arbitrary one is is the node's identity, while the rest are Sybil identities.

As resource to test, we use computation, since it seems probable that processing power is less limited than communication capacity and memory in a MANET node[50].

We assume the existence of a function f which takes arbitrary bit strings as input and maps them randomly to the elements in the codomain. Furthermore, we assume that an evaluation of f takes about the same number of steps on any input, and call the number of steps needed to evaluate f once a *work unit*, denoted U_W [51]. In addition, we define a *time unit* U_T, specified in e.g. seconds. We call the computational power needed to calculate one work unit in one time unit a *processing unit*, denoted U_P.

The method we explore will require one processing unit for each successful authentication. The result is a limit on the number of times the adversary is authenticated, since we assume he is computationally bounded.

8.4 Threshold on adversarial identities

Recall that our goal is to achieve $t_{ID} < m_{ID}/2$. We will briefly consider assumptions we can make on the corrupted nodes that result in this threshold holding.

Suppose less than half of the nodes are corrupted, $t < m/2$, and that the nodes have one processing unit U_P each. Then $t_{ID} = t$ and $m_{ID} = m$, so $t_{ID} < m_{ID}/2$, as desired.

On the other hand, of we assume $t < m/3$, we can allow that the corrupted nodes have two U_P each, while the honest nodes have only one U_P each. The adversary then controls $2t$ U_P in total, so $t_{ID} = 2t$ and $\#\mathcal{D} = t$. This means that $m_{ID} = m + t$. Since $t < m/3$, $4t < m + t$, so $2t_{ID} < m_{ID}$ and $t_{ID} < m_{ID}/2$, as desired.

The logic behind these results is of course that the more processing power the adversary has available, the more identities can he authenticate. And we want more than half of the authenticated identities to be controlled by honest nodes. The point of doing these exercises is merely to see that the assumptions we must make may be realistic in some scenarios.

8.5 The scheme

We sketch a protocol that authenticates each identity that has one processing unit. We assume existence of a broadcast channel, which may be realised in

[50] Hegland et al. [35] suggest that communication capacity is likely to be a more limited resource than computational power in a MANET.

[51] The function f may be implemented by iterating a hash-function e.g. 100 000 times on the input string.

For $i \in [1, m_{ID}]$, identity i distributes its public key
1. Generate a signing key pair (pk_i, sk_i) (e.g. from the BLS signature scheme of Section 3.2.1).
2. Broadcast pk_i.

For $i \in [1, m_{ID}]$, identity i challenges the other identities
1. Choose at random $r \in_R \mathbb{Z}$.
2. Broadcast r.
3. Wait in two time units, $2U_T$, to receive responses.
4. Reject all pk_j where responses have not been received within $2U_T$ or the response is wrong (all of them, if there are multiple responses for a j), according to the description below. This also applies when other identities generate challenges r.

For $i \in [1, m_{ID}]$, identity i responds to challenges
1. Receive r.
2. Compute $a_i = f(pk_i|r)$.
3. Broadcast (i, a_i).

Figure 19: Authenticating founder nodes by testing processing capability.

a MANET if founder nodes are in direct communication range during network formation.

Regarding the notation used, f and U_T are described in Section 8.3. The protocol is described in Figure 19.

The idea of the protocol is that the identities owning the public keys not rejected (by honest nodes) when the protocol finishes have a processing unit available. These public keys are then used to construct pairwise authentic channels, which is the goal with network formation. Note that we accept responses arriving within two time units instead of one because transmission and other processing than f takes nonzero time.

A potential problem with the scheme is that the bandwidth may become the limiting factor during challenges and responses instead of the processing power. The problem severity depends on the codomain size of f, and it can be alleviated by increasing U_W and U_T (see Section 8.3).

Correctness We will briefly argue why the protocol accomplishes the desired result. Firstly, a honest node can compute $f(pk_i|r)$ in a timely manner for any pk_i and r, because we assume that an evaluation of f takes about the same amount of steps independent of the input (in Section 8.3). The other honest nodes will see the correct responses and thus accept pk_i.

Secondly, the challenges r from honest nodes will be random, which the adversary cannot precompute responses to. Furthermore, since the responses depend on a public key, knowing the response a_i corresponding to a public key pk_i does not help to find the response corresponding to a different public key pk_j. Thus, a processing unit is needed for each correct response corresponding to a public key.

8.6 Summary

We summarise our discussion with the main advantages and disadvantages of the sketched method for authenticating founder nodes.

The most important advantage is that the authentication method is fully self-configuring; no preestablished trust is required for the founder nodes. In addition, the nodes do not need any special physical characteristics (e.g. a GPS receiver, clock, etc.); everything may be implemented in software.

On the negative side, MANET nodes have limited, possibly varying, processing power. This may make it hard, maybe impossible in some cases, to find a function f that satisfies the requirements.

But we believe that in order to achieve self-configuring authentication, we must consider alternatives to traditional authentication methods. However, it is interesting to note that the method we discussed here actually has similarities with a `PKI`. In a `PKI`, the `CA` creates a binding between a node and its public key. In our method, a processing unit creates the same binding.

9 Discussion

9.1 Summary

The special characteristics of a MANET result in frequent changes in the network topology. In order for routing to work in such a network, the routing protocol must take these characteristics into account. The OLSR routing protocol is specifically designed for MANETs [19], and we learnt how routing in MANETs can be done when we studied it in Section 2.2.

However, routing protocols for MANETs are vulnerable to a range of attacks, mainly because a MANET is based on wireless communication without any reliable infrastructure. Therefore, we examined different protocols that alleviate such attacks, by making them impossible or merely detecting them (i.e. intrusion detection systems), in Section 2.3. Through this study, we saw that a method for cryptographic key distribution and support for message authentication, e.g. from digital signatures, are needed. At the same time, the overhead induced by cryptographic mechanisms, especially on the bandwidth, must be very small for them to be applicable in a MANET.

Elliptic curves are known to allow cryptographic mechanisms with short key length. In recent years, elliptic curves have also received a lot of interest because pairings may be defined on them. Pairings give the opportunity to define groups where the Computational Diffie-Hellman problem is hard while the Decision Diffie-Hellman problem is easy, which results in cryptographic mechanisms that are even more efficient. In a MANET these efficiency gains are attractive, and we thus explored theoretical aspects of pairings in Section 3.1 and cryptographic mechanisms using them in Section 3.2. We saw that elliptic curve cryptography and pairings have a very positive impact on the efficiency of the protocols used to secure MANET routing protocols. This is supported by the construction of secure signature schemes with signature lengths of merely 160 bits, and the elimination of pubic key certificates through identity-based cryptography. Both these properties reduce bandwidth overhead significantly, and also lightens the computational requirements.

But even with efficient security mechanisms in place, we still need a method to distribute cryptographic keys before using them. In identity-based cryptography, it is the Private Key Generator (PKG) that issues keys corresponding to identities. In order to support the self-configuring property of a MANET, the PKG must be online to issue keys to joining nodes. To protect the master-key of the PKG, but at the same time keep it available, we studied the theory of threshold cryptography, including secret sharing and function sharing, in Chapter 4.

With all this theory as background, we considered the possibilities to distribute the tasks of the PKG in Chapter 6 and Chapter 7. The PKG is involved in the *Setup* and *Extract* algorithms of an IBC system, so we focused on the distribution of these. But in addition, we discussed how signatures may be generated using a secret sharing of the master-key. This may be used to sign a message as the PKG authority, which is useful for instance when creating revocation lists. But we did not find any definition of security for distributed versions of *Setup* and *Extract*, so we defined security, based on work by Gennaro et al. [31].

Through our work, we saw that self-configuring authentication, especially nodes claiming multiple identities, is a major challenge when distributing the authority of a PKG in a MANET. We sketched a method for authenticating founder nodes that supports self-configuration in Chapter 8. Self-configuring authentication prior to issuing private keys through distributed *Extract* is a challenge that we have not considered a solution to. Note that all authentication can be solved by traditional methods like a PKI, shared keys, and physical meetings, but in our opinion, these methods are not sufficiently self-configuring for a MANET.

9.2 Comparison with a Distributed CA

To illustrate the main advantages and disadvantages of the key management system we discussed in Chapter 7, we now give a rough qualitative comparison of it and another key management system. It is most natural to compare it with a system that uses an online distributed Certificate Authority (CA). Firstly, both systems utilise asymmetric cryptography, which may have a big impact on the characteristics, as discussed in Chapter 5. Secondly, both systems have an online authority. And lastly, since the systems use threshold cryptography, properties derived from threshold cryptography will not affect the comparison either.

During the following discussion, we will sometimes need a more concrete description of the system we are comparing with, other than that is has a distributed CA. When this is the case, we will assume RSA signatures, and the system of Kong et al. [43] when even more concreteness is required.

9.2.1 Protocol descriptions

The DPKG system we discussed in Chapter 7 consists of two protocols; distributed *Setup* (denoted DS) and distributed *Extract* (denoted DE). In addition, we outlined the possibility for a third protocol that implements DPKG promotion in Section 7.6. In generic terms, these protocols carry out system

key generation, key generation for a node, and share generation of the system key, respectively. Similar algorithms are needed in a key management system that is based on a distributed CA.

Distributed key generation schemes for RSA do exist [11, 34], but Kong et al. assume a centralised entity that generate the system key and initialise founder nodes with shares of it. Generating a key for a node is a distributed signature scheme, where the node's public key is signed. Nodes with a share of the system key send share-function evaluations to the node that should receive the certificate. Receiving these, the node runs an algorithm to combine the evaluations into a public key certificate. To generate a new share of the system key in the system of Kong et al., the protocol we sketched in Section 7.6 is used, but the shares are considered modulo the RSA-modulus instead of a prime.

9.2.2 Authentication assumptions

As we described in Section 6.8, we need authentication before running the DS and DE protocols, and before DPKG promotion. This also applies for the similar protocols used in a distributed CA. We now compare how authentication is done in the two systems.

Before running the DS protocol from Chapter 7, authentication of the founder nodes is done during network formation as described in Chapter 8. Here, the processing power of the adversary and the number of adversarial nodes is assumed to be limited. Kong et al. assume a centralised entity that can authenticate and initialise nodes.

We have not specified how a node is authenticated before receiving its private key from the DE protocol of Chapter 7. However, any traditional method like physical authentication, a PKI or shared predistributed keys may be used. In the system of Kong et al., the founder nodes are initialised with a public key certificate by the centralised entity that initialises the nodes. Furthermore, it is suggested that nodes joining at a later point in time are authenticated physically.

Before DPKG promotion, we assume a method authenticating that the node has not already received a share of the master-key is run. Again, any traditional method for authentication suffice here. Kong et al. use the certificate that a node has to authenticate it, before it receives a share of the system key.

9.2.3 Security guarantees

Any attack from the adversary may mainly have two undesirable effects. Firstly, the adversary may obtain important information. Secondly, he may cause denial of service by sending corrupted messages to the other nodes, or not send messages at all.

The system we discussed in Chapter 7 is secure against both these attack types, since it satisfies the conditions in Definition 6.3.

However, Kong et al. have not stated a model and proof of security. In fact, it can be shown that if the adversary has a certificate in their system, he can cause failure when a node combines both signatures or shares [50]. The system thus lacks robustness in both the certificate and share generation protocols[52]. But in addition, the system has a more important security problem that allows the leakage of the system secret key [39].

There exists however a secure distributed signature scheme for RSA [33] that may be used instead of the version of Kong et al. But at least, this illustrates the importance of a model and proof of security when designing a key management system.

9.2.4 Efficiency

We first compare the protocols used for system key generation, key generation for a node and share generation, in terms of bandwidth and processing requirements.

Kong et al. do not use a protocol to generate the system key, but assume this is done centrally. However, as we have mentioned earlier, there exist distributed key generation schemes for RSA [11, 34], but Narasimha et al. [50] argue that schemes for distributed generation of RSA keys are too inefficient for a MANET. The DS protocol of Chapter 7 also requires much communication, but we argue it is practical if there are not too many founder nodes.

To generate the key for a node, a request message and at least $t+1$ reply messages is required in both systems. However, the reply messages in the system of Kong et al. include a RSA signature share of about 1000 bits, while the reply messages in the DE protocol contain about 160 bits (an element of $\mathbb{G}_1$).

The protocols for system key share generation are almost identical in the two systems, so there is no important difference in the efficiency there.

[52] Interestingly, in our system, it is the $\mathcal{V}$ algorithm described in Section 7.4 that ensures robustness of the DE protocol. It is made possible by the existence of pairings, since pairings allow the DDH problem to be solved.

As we discovered in Section 2.3, a signature scheme is the most important mechanism used by MANET security protocols. Therefore, the signature length will play an important role for bandwidth overhead. The scheme by Cha and Cheon, which may be used in conjunction with the system of Chapter 7, has signature lengths of about 320 bits. RSA signatures are at least 1000 bits, but they can be very quickly verified if a low RSA-exponent is used.

However, the most differentiating factor for efficiency between the two systems is probably certificate distribution. In the system of Chapter 7, no public key certificates exist, while in the system of Kong et al., every node has a certificate of at least 2000 bits (1000 bits for the public key and 1000 bits for the `CA` signature). Constructing an efficient system that ensures that every node has the certificates it needs is clearly a non-trivial issue. In addition, considering the size of all the certificates, we see that certificate distribution will consume considerable bandwidth.

Hegland et al. [36] have simulated the impact of the overhead due to increased message lengths, using the OLSR protocol. They conclude that not more than a few hundred bits should be added to each message, on the average.

9.2.5 Summary

We summarise our comparison with the important similarities and differences of the two systems, starting with the similarities.

Issues related to authentication are identical in the two systems, and they can consequently be solved with the same methods. The key to the degree of self-configuration in a system lies in authentication. In addition, we have seen that a model and proof of security is critical, independent of the system[53].

Efficiency, especially bandwidth consumption, is the main differentiator. The most important point here is that the system of Chapter 7 avoids the difficult problem of public key certificate distribution, since it is identity-based.

9.3 Conclusion

Key management systems for MANETs have been compared in Chapter 5 and Section 9.2. It is found that threshold cryptography combined with identity-based cryptography results in some of the most promising systems. The main reasons for this are that threshold cryptography make the systems

[53]The security model can in fact be very similar in a system based on a distributed `CA` and a distributed `PKG`, as seen by the close resemblance of the security definition for a threshold signature scheme of Gennaro et al. [31] and a secure `DPKG` in Definition 6.3.

robust, while by using identity-based cryptography, one avoids the difficult challenge of public key distribution in a MANET. A common problem for all key management systems is to authenticate nodes when no preestablished trust exists. An important result of this thesis is that the authority of the PKG in some of the most prevalent IBC systems can be distributed while the security properties are preserved.

9.4 Further Work

During our work on distribution of the PKG authority in Chapter 6, we were unable to find earlier publications that define security for such a system. We therefore proposed a definition, based on the security definition for a threshold signature scheme by Gennaro et al. [31]. This definition proved to be applicable as a security definition for the concrete DPKG system we discussed in Chapter 6, since we were able to show that its conditions hold in this system. But it would be interesting to examine if this definition can be used in other DPKG systems as well. For example, the IBE system of Waters [68], which does not rely on random oracles, uses different algorithms for *Setup* and *Extract* than the IBE system of Boneh and Franklin [12] that we based ourself on in Chapter 7. By first finding distributed versions of these two algorithms, one could then try to prove that the resulting system is secure with respect to our definition.

Furthermore, we saw in Chapter 6 that support for DPKG promotion, i.e. distributed generation of system key share, is useful. However, we have not taken a protocol for DPKG promotion into account in our security definition. Extending the definition to include such a protocol is thus interesting.

In Chapter 8, we sketched a method that may be used to authenticate nodes before system parameters are generated, without requiring preestablished trust between any nodes. But as discussed in Section 6.8, we also need a degree of authentication before a node obtains the private key corresponding to an identity (distributed *Extract*) and a share of the system key (DPKG promotion). If authentication methods for these purposes, that do not require preestablished trust, is created, we can construct a fully self-configuring DPKG.

Much can also be done to improve the efficiency of the protocols used to create a DPKG, especially with respect to bandwidth consumption. An efficient protocol for DPKG promotion is probably most useful. Daza et al. [22] propose such a protocol, based on bivariate Shamir secret sharing, which requires little communication. An interesting task would be to analyse if this protocol can be used in conjunction with the DPKG system we discussed in Chapter 7. In this case, a new distributed *Setup* algorithm that uses

bivariate secret sharing would need to be designed. Additionally, security for the system must be proven in an extended definition, as mentioned above. Daza et al. only consider a passive adversary when discussing security.

Of the protocols we described for distributed *Setup* and distributed *Extract* i Chapter 7, the distributed *Setup* protocol is much more intensive on communication. Even though it is probably run more seldomly, finding replacements is still interesting. Pairings have enabled a significant improvement of the efficiency of `IBC` and signature schemes. Perhaps the properties of pairings can be exploited to construct a more efficient distributed *Setup* as well.

In the protocols we discussed in Section 2.3 that protect MANET routing, a digital signature scheme is a widely used mechanism. We have seen that identity-based signature schemes avoid the bandwidth-consuming problem of certificate distribution. However, traditional signature schemes may offer shorter signatures. For example, the BLS signature scheme in Section 3.2.1 offer signatures of 160 bits, while the identity-based signature scheme due to Cha and Cheon in Section 3.2.4 requires 320 bits per signature. An identity-based signature scheme with short signatures would lead to a very efficient solution.

Lastly, an interesting task is to try to implement the `DPKG` system discussed in Chapter 7. This would serve as a real test for the applicability of it, and could possibly disclose new challenges that need to be solved. Considering the practicality of the authentication method sketched in Chapter 8 would also be interesting.

References

[1] C. Adjih, T. Clausen, A. Laouiti, P. Mühlethaler, and D. Raffo. Securing the OLSR protocol. In *IFIP Annual Mediterranean Ad Hoc Networking Workshop*, pages 25–27, 2003.

[2] M. Akane, H. Kato, Y. Morikawa, Y. Nogami, and Y. Sakemi. Integer Variable χ-Based Ate Pairing. In *Pairing 2008*, volume 5209 of *Lecture Notes in Computer Science*, pages 178–191. Springer-Verlag, 2008.

[3] T. R. Andel and A. Yasinsac. Surveying Security Analysis Techniques in MANET Routing Protocols. *IEEE Communications Surveys & Tutorials*, 9(4):70–84, 2007.

[4] P. G. Argyroudis and D. O'Mahony. Secure Routing for Mobile Ad hoc Networks. *IEEE Communications Surveys & Tutorials*, 7(3):2–21, 2005.

[5] E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid. Recommendation for Key Management — Part 1: General (Revisited). NIST Special Publication 800-57, March 2007.

[6] D. Beaver. Foundations of Secure Interactive Computing. In *Advances in Cryptology – CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 377–391. Springer-Verlag, 1992.

[7] M. Bellare and P. Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[8] I. F. Blake, G. Seroussi, and N. P. Smart, editors. *Advances in Elliptic Curve Cryptography*, volume 317 of *London Mathematical Society Lecture Note Series*. Cambridge University Press, 2005.

[9] G. R. Blakley. Safeguarding cryptographic keys. In *Proc. AFIPS 1979 National Computer Conference*, volume 48 of *AFIPS Conference proceedings*, pages 313–317. AFIPS Press, 1979.

[10] A. Boldyreva. Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme. In *International Workshop on Practice and Theory in Public Key Cryptography*, volume 2567 of *Lecture Notes in Computer Science*, pages 31–46. Springer-Verlag, 2003.

[11] D. Boneh and M. Franklin. Efficient generation of shared RSA keys. *Journal of the ACM*, 48(4):702–722, July 2001.

[12] D. Boneh and M. Franklin. Identity-Based Encryption from the Weil Pairing. In *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229. Springer-Verlag, 2001.

[13] D. Boneh, C. Gentry, and M. Hamburg. Space-Efficient Identity Based Encryption Without Pairings. In *48th Annual IEEE Symposium on Foundations of Computer Science*, pages 647–657. IEEE Computer Society, 2007.

[14] D. Boneh, B. Lynn, and H. Shacham. Short Signatures from the Weil Pairing. In Colin Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 514–532. Springer-Verlag, 2001.

[15] J. C. Cha and J. H. Cheon. An Identity-Based Signature from Gap Diffie-Hellman Groups. In Y. Desmedt, editor, *Public Key Cryptography*, volume 2567 of *Lecture Notes in Computer Science*, pages 18–30. Springer-Verlag, 2003.

[16] D. Chaum, C. Crepeau, and I. Damgård. Multiparty unconditionally secure protocols. In *Proceedings of the 20th Annual ACM Symposium on the Theory of Computing*, pages 11–19. ACM Press, May 1988.

[17] L. Chen and J. Malone-Lee. Improved Identity-Based Signcryption. In *Public Key Cryptography - PKC 2005*, volume 3386 of *Lecture Notes in Computer Science*, pages 362–379. Springer-Verlag, 2005.

[18] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults. In *26th Annual Symposium on Foundations of Computer Science*, pages 383–395. IEEE Computer Society, 1985.

[19] T. Clausen and P. Jacquet. Optimized Link State Routing Protocol (OLSR). RFC 3626 (Experimental), October 2003. http://www.ietf.org/rfc/rfc3626.txt (2008-08-24).

[20] C. Cocks. An Identity Based Encryption Scheme Based on Quadratic Residues. In *Cryptography and Coding*, volume 2260 of *Lecture Notes in Computer Science*, pages 360–363. Springer-Verlag, 2001.

[21] I. Damgård. Secret Sharing. *Unpublished manuscript*, 2006. http://www.daimi.au.dk/~ivan/SecretSharing.pdf (2008-12-19).

[22] V. Daza, P. Morillo, and C. Ràfols. On Dynamic Distribution of Private Keys over MANETs. *Electronic Notes in Theoretical Computer Science*, 171(1):33–41, 2007.

[23] Y. Desmedt and S. Jajodia. Redistributing Secret Shares to New Access Structures and Its Applications. Technical report, George Mason University, 1997.

[24] J. R. Douceur. The Sybil Attack. In *First International Workshop on Peer-to-Peer Systems*, volume 2429 of *Lecture Notes in Computer Science*, pages 251–260. Springer-Verlag, 2002.

[25] N. Ferguson and B. Schneier. *Practical Cryptography*. Wiley, 2003.

[26] G. Frey and H.-G. Rück. A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves. *Mathematics of Computation*, 62(206):865–874, April 1994.

[27] G. Frey and H.-G. Rück. A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves. *Mathematics of Computation*, 62:865–874, 1994.

[28] S. D. Galbraith, K. Harrison, and D. Soldera. Implementing the Tate Pairing. In *Algorithmic Number Theory, 5th International Symposium*, volume 2369 of *Lecture Notes in Computer Science*, pages 324–337. Springer-Verlag, 2002.

[29] S. D. Galbraith, K. G. Paterson, and N. P. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121, 2008.

[30] P. S. Gemmell. An Introduction to Threshold Cryptography. *CryptoBytes*, 2(3):7–12, Winter 1997.

[31] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust Threshold DSS Signatures. In Ueli Maurer, editor, *Advances in Cryptology – EUROCRYPT '96*, volume 1070 of *Lecture Notes in Computer Science*, pages 354–371. Springer-Verlag, 1996.

[32] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure Distributed Key Generation for Discrete-Log Based Cryptosystems. In *Advances in Cryptology – EUROCRYPT '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 295–310. Springer-Verlag, 1999.

[33] R. Gennaro, T. Rabin, S. Jarecki, and H. Krawczyk. Robust and Efficient Sharing of RSA Functions. *Journal of Cryptology*, 13(2):273–300, 2000.

[34] N. Gilboa. Two Party RSA Key Generation. In *Advances in Cryptology – CRYPTO ' 99*, volume 1666 of *Lecture Notes in Computer Science*, pages 116–129. Springer-Verlag, 1999.

[35] A. M. Hegland, E. Winjum, S. F. Mjolsnes, C. Rong, O. Kure, and P. Spilling. A Survey of Key Management in Ad Hoc Networks. *IEEE Communications Surveys & Tutorials*, 8(3):48–66, 2006.

[36] A. M. Hegland, E. Winjum, P. Spilling, C. Rong, and Ø. Kure. Analysis of IBS for MANET Security in Emergency and Rescue Operations. In *AINA*, pages 155–159. IEEE Computer Society, 2006.

[37] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung. Proactive Secret Sharing, Or: How To Cope With Perpetual Leakage. In *Advances in Cryptology – CRYPTO '95*, volume 963 of *Lecture Notes in Computer Science*, pages 339–352. Springer-Verlag, 1995.

[38] Y.-C. Hu, A. Perrig, and D. B. Johnson. Ariadne: A Secure On-Demand Routing Protocol for Ad Hoc Networks. *Wireless Networks*, 11(1-2):21–38, 2005.

[39] S. Jarecki, N. Saxena, and J. H. Yi. An Attack on the Proactive RSA Signature Scheme in the URSA Ad Hoc Network Access Control Protocol. In *ACM Workshop on Security of ad hoc and Sensor Networks*, pages 1–9. ACM Press, 2004.

[40] D. Johnson, Y. Hu, and D. Maltz. The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4. RFC 4728 (Experimental), February 2007.
http://www.ietf.org/rfc/rfc4728.txt (2008-10-14).

[41] A. Joux. The Weil and Tate Pairings as Building Blocks for Public Key Cryptosystems. In *Algorithmic Number Theory, 5th International Symposium*, volume 2369 of *Lecture Notes in Computer Science*, pages 20–32. Springer-Verlag, 2002.

[42] A. Khalili, J. Katz, and W. A. Arbaugh. Toward Secure Key Distribution in Truly Ad-Hoc Networks. In *SAINT Workshops*, pages 342–346. IEEE Computer Society, 2003.

[43] J. Kong, P. Zerfos, H. Luo, S. Lu, and L. Zhang. Providing Robust and Ubiquitous Security Support for Mobile Ad Hoc Networks. In *International Conference on Network Protocols*, pages 251–260. IEEE Computer Society, 2001.

[44] J. F. Kurose and K. W. Ross. *Computer Networking: A Top-Down Approach*. Addison Wesley, 4th edition, April 2007.

[45] B. Lynn. Authenticated Identity-Based Encryption. Cryptology ePrint Archive, Report 2002/072, 2002.

[46] A. J. Menezes, T. Okamoto, and S. A. Vanstone. Reducing Elliptic Curve Logarithms to Logarithms in a Finite Field. *IEEE Transactions on Information Theory*, 39:1639–1646, 1993.

[47] J. Merwe, D. Dawoud, and S. McDonald. A Survey on Peer-to-Peer Key Management for Military Type Mobile Ad Hoc Networks. In *Military Information and Communications Syposium of South Africa*, 2005.

[48] V. S. Miller. Short Programs for functions on Curves. *Unpublished manuscript*, 1986.

[49] D. Nagaraj and B. Sury. A Quick Introduction to Algebraic Geometry and Elliptic Curves. In *Elliptic Curves, Modular Forms and Cryptography: Proceedings of the Advanced Instructional Workshop on Algebraic Number Theory*, 2003.

[50] M. Narasimha, G. Tsudik, and J. Hyun Yi. On the Utility of Distributed Cryptography in P2P and MANETs: The Case of Membership Control. In *IEEE International Conference on Network Protocols*, pages 336–345. IEEE Computer Society, 2003.

[51] National Institute of Standards and Technology. *Digital Signature Standard*. FIPS Publication 186-2, January 2000.

[52] J. Newsome, E. Shi, D. Xiaodong Song, and A. Perrig. The Sybil Attack in Sensor Networks: Analysis & Defenses. In *Proceedings of the Third International Symposium on Information Processing in Sensor Networks*, pages 259–268. ACM Press, 2004.

[53] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc On-Demand Distance Vector (AODV) Routing. RFC 3561 (Experimental), July 2003. http://www.ietf.org/rfc/rfc3561.txt (2008-10-14).

[54] A. Perrig, R. Canetti, J. D. Tygar, and D. Song. The TESLA Broadcast Authentication Protocol. *RSA CryptoBytes*, 5(2):2–13, Summer/Fall 2002.

[55] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar. SPINS: Security Protocols for Sensor Networks. In *Seventh Annual International Conference on Mobile Computing and Networks (MobiCOM 2001)*, pages 189–199, 2001.

[56] S. Goldwasser and S. Micali and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18, 1989.

[57] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-To-End Arguments in System Design. *ACM Transactions on Computer Systems*, 2(4):277–288, 1984.

[58] A. D. Santis, Y. Desmedt, Y. Frankel, and M. Yung. How to Share a Function Securely. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing (STOC'94)*, pages 522–533. ACM Press, 1994.

[59] K. Sanzgiri, B. Dahill, B. N. Levine, C. Shields, and E. M. Belding-Royer. A Secure Routing Protocol for Ad Hoc Networks. In *ICNP*, pages 78–89. IEEE Computer Society, 2002.

[60] N. Saxena, G. Tsudik, and J. H. Yi. Efficient Node Admission for Short-lived Mobile Ad Hoc Networks. In *International Conference on Network Protocols*, pages 269–278. IEEE Computer Society, 2005.

[61] A. Shamir. How to Share a Secret. *Communications of the ACM*, 22(11):612–613, November 1979.

[62] A. Shamir. Identity-Based Cryptosystems and Signature Schemes. In *Advances in Cryptology: Proceedings of CRYPTO 84*, volume 196 of *Lecture Notes in Computer Science*, pages 47–53. Springer-Verlag, August 1984.

[63] J. H. Silverman. *The Arithmetic of Elliptic Curves*, volume 106 of *Graduate Texts in Mathematics*. Springer-Verlag, 1986.

[64] R. Steinfeld, H. Wang, and J. Pieprzyk. Lattice-Based Threshold-Changeability for Standard Shamir Secret-Sharing Schemes. In *Advances in Cryptology – ASIACRYPT 2004*, volume 3329 of *Lecture Notes in Computer Science*, pages 170–186. Springer-Verlag, 2004.

[65] E. R. Verheul. Evidence that XTR is more secure than supersingular elliptic curve cryptosystems. In *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 195–210. Springer-Verlag, 2001.

[66] L. Viennot. Complexity Results on Election of Multipoint Relays in Wireless Networks. In *Report RR-3584, INRIA*, December 1998.

[67] M. Wang, L. Lamont, P. Mason, and M. Gorlatova. An Effective Intrusion Detection Approach for OLSR MANET Protocol. In *1st IEEE ICNP Workshop on Secure Network Protocols*, pages 55–60, 2005.

[68] B. Waters. Efficient Identity-Based Encryption Without Random Oracles. In *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 114–127. Springer-Verlag, 2005.

[69] L. Zhou and Z. J. Haas. Securing Ad Hoc Networks. *IEEE Network*, 13(6):24–30, 1999.