An Holistic Approach to Public/Private–Key Based Security in Locator/Identifier–Split Architectures

Oliver Hanka
Technische Universität München
Institute of Communication Networks
80333 Munich, Germany
oliver.hanka@tum.de

Wolfgang Fritz
Leibniz Supercomputing Centre
85748 Garching, Germany
wolfgang.fritz@lrz.de

***Abstract*—Network security has become an essential business requirement over the past few years. As this demand will increase even more in the future, researchers agree that security must be a key element for any novel Next Generation Internet architecture. Contrary to today's add–on approach to security, the mechanisms must be anchored in the overall architecture and should be a major concern already during the design phase. In this article we present an approach based on the private/public–key principle for almost any locator/identifier–split architecture. We suggest to extend the mapping system to also serve as public–key infrastructure and recommend to use smart cards for the client side key management.**

***Keywords*-Public–Key Infrastructure, Locator/Identifier–Split, Smart Cards, Asymmetric Cryptography, Next Generation Internet, HiiMap**

I. Introduction

Today's Internet architecture faces some well-known limitations and many ongoing research activities exist to define the so called *Next Generation Internet* (NGI). For example, there is currently only one address representing different aspects—the IP address stands both for the particular host we want to contact and for the topological location, where it can be reached. That's why many clean–slate approaches towards an NGI architecture favor a so called locator/identifier–split [1][2][3][4][5][6]. Furthermore, as the Internet evolved during the years, many new aspects had to be considered, i.e., how to communicate in a secure way? Many small and different solutions have been applied to the architecture to answer this question. Rather than using these numerous add-ons, it is agreed that security needs to be an integral part in future concepts, providing an holistic approach to guarantee secure communication. This is because the Internet has transformed from a communication means to transfer files and messages between some few nodes to the basis of today's economy with billions of participants.

Like Moskowitz et al. [4], for example, many have suggested linking the identifier with a public–key in some way. This has the benefit that each communication partner can be authenticated based on the public/private–key principle by Diffie et al. [7]. Additionally, it can be used to exchange a symmetric secret for stream encryption.

Moskowitz et al. suggest hashing the public–key and using it as the identifier of that node. This, however, raises some problems as described in [2]. For example, it it easy to find a random private–key, public–key, identifier triple and furthermore, the public–key can not be exchanged while keeping the identifier. Therefore, we propose a *loose coupling* between the public–key and the identifier [2]. For that loose coupling, the relation between a certain key and an identifier is stored in the mapping system.

In this article we introduce a way to use a mapping system of an locator/identifier–split architecture as a public–key infrastructure. One very important aspect of the private/public–key principle is the integrity of the public–key. Therefore, we focus on the retrieval of the public–key from the mapping system and discuss how the user can verify the integrity of it. Additionally, we describe the key management on the client side supported by smart cards. We detail the initial bootstrap process and discuss the mechanisms for an encrypted communication. We will also consider client devices with low computational power, like sensors, as they already play an important role today that will even more increase in the future.

The remainder of this article is structured as follows: Section II discusses related work like the Host Identity Protocol, some key features of UMTS/GSM regarding security and basics of a public–key infrastructure. In Section III, we give a brief overview of HiiMap which we use as an example architecture throughout the rest of this article. Afterwards in Section IV we detail our approach to integrate the public–key infrastructure into the mapping system. The smart card based client key management is outline in Section V. Before we conclude our work in Section VII, we will evaluate the concept in Section VI.

II. Related Work

In the following, we will first give a brief introduction of the so called locator/identifier–split principle and discuss one example architecture *(Host Identity Protocol (HIP))* that

is based on it. Afterwards, some key functionalities of the *Universal Mobile Telecommunications System*'s (UMTS') security mechanisms are described, as some approaches are similar to the ones used in our concept (see Section V). Finally, we outline the key concepts of today's *Public Key Infrastructure* (PKI).

A. Locator/identifier-split

In today's Internet, the IP–address represents in fact two different meanings: First, it is of course an identifier of the particular node we want to contact (*who?*). Secondly, it also answers the question how this node can be reached (*where?*). Many *Next Generation Internet* (NGI) approaches propose to use a so called locator/identifier–split, in order to answer only one question at a time. In these concepts, there is the *identifier*, that stands for the endpoint we want to contact. In contrast, the *locator* answers the question how this can be done. Therefore, we have two different addresses, one for each meaning. By providing these, the locator/identifier–split solves several issues concerning mobility, routing table growth and scalability. Additionally, it leaves some space to integrate security mechanisms into the network layer.

B. Host Identity Protocol

HIP [4] uses the *Host Identity Tag* (HIT) as identifier. The HIT either represents the 128 bit long public key or—in case of greater key length—the hash of it. In that way, any node can verify the public key of its peer by only knowing the identifier. Therefore, no PKI is required. During HIP base exchange, the public keys and a secret are exchanged [8].

As already stated in the introduction, this approach has a major security vulnerability. An attacker could start to generate many private/public–key pairs and hash the public key into a HIT. In a next step, he could query the mapping system and check whether the HIT is already reserved. In case he finds an already reserved HIT, the attacker holds a valid private key to that HIT. This does not enable an attacker to find a specific private key for a certain HIT, but allows for random attacks and could become interesting for Botnets, for example.

C. UMTS/GSM

The *Global System for Mobile Communications* (GSM) has some major drawbacks as described in [9][10]. As it was not initially designed for Internet purposes, it faces many new challenges. Most of them are eliminated in pure UMTS environments [11][12]. Nevertheless, some problems still exist when roaming from UMTS to GSM and vice versa is supported [13].

The UMTS architecture also uses a smart card based principle for authenticating its clients—the so called *Authentication and Key Agreement* (AKA) [13]. In addition to authentication, it provides data encryption (with cipher key CK) as well as integrity protection (with integrity key IK). Similar to parts of our concept, UMTS is therefore able to prove the integrity of received messages. In contrary, UMTS security functions only protect the last *security command mode message* used in AKA [13] and all subsequent ones, whereas we are able to provide integrity protection and encryption from the beginning. Mechanisms of delivering and activating smart cards (sending card, *Personal Identification Number* (PIN) and *Personal Unblocking Key* (PUK) per mail and activating it with the appropriate PIN) is also realized analogously in UMTS systems.

D. Public Key Infrastructure

Today, the exchange of public–keys is done via a public key infrastructure e.g., defined by the ITU-T standard X.509 [14][15]. All approaches have in common that a particular user or node publishes its public key on a key server of some sort from which it can be downloaded by other peers. After that, encrypted and signed messages can be exchanged. This, of course, requires that each participating node has to trust the key server. If a key pair ever gets lost, it can be revoked by including it in the so–called *Certificate Revocation List* (CRL), where all invalid keys and certificates are kept [15].

In [16] Ellison et al. argue that today's public key infrastructure based on *certificate authorities* (CA) imposes ten major risks. They describe, for example, the problematic trust background of self-proclaimed authorities and discuss the weakest link issue of the CA structure. Furthermore they raise the question how the certificate holder identifies himself against the CA. They state that several procedures do exist and that there are no consistencies over all CAs.

III. HiiMap Architecture

The HiiMap Next Generation Internet architecture [2] is based on the locator/identifier–split principle and provides a two–tier hierarchical mapping system. In the following, we will give a brief overview of the mapping system, as we will use HiiMap as example architecture.

In HiiMap, the mapping system is divided into so called regions as illustrated in Figure 1. Each region is responsible for all its nodes and has to provide the mapping for them. The region remains responsible, even if the node temporarily roams to another region. To identify a *responsible region* (RR) for a node, an additional 8 bit *regional prefix* (RP) to the identifier is provided. Within the HiiMap architecture, the identifier is assigned for life time and not subject to change as long as the owner doesn't request a new one. The regional prefix, however, is allowed to change whenever a node permanently migrates to another region. In HiiMap the identifier is called *unique identifier* (UID). The identifier and regional prefix is depicted in Figure 2.

Whenever a node wants to contact another node, it needs to query the RR of that node for the actual locator (which is called *local temporary address* or *LTA* in HiiMap). Therefore, it needs to know the regional prefix for that

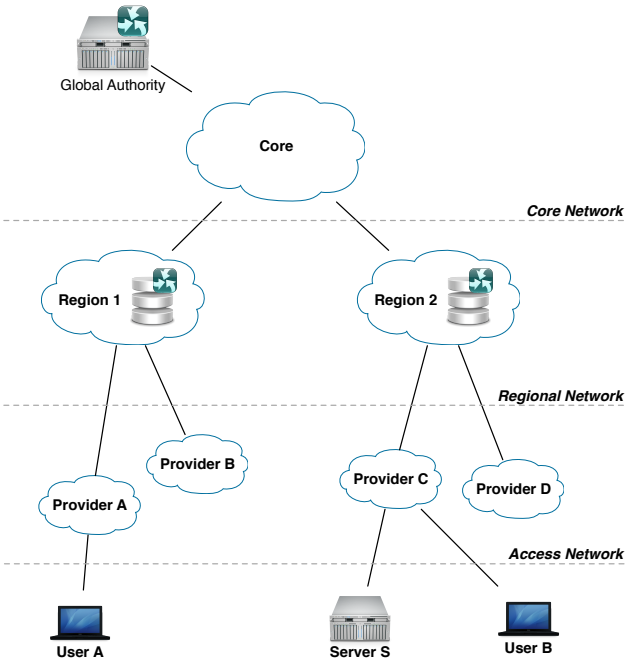

Figure 1. Example HiiMap topology

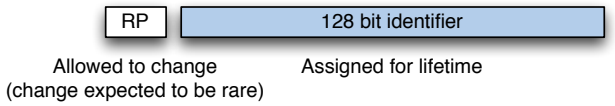

Figure 2. Identifier with regional prefix

region. In case the regional prefix hasn't been cached from previous communications, there are two ways to obtain it. The first possibility is together with the identifier itself. In case the identifier was learned from a link on a website or by means of a domain name system, the regional prefix can be provided along with the identifier. The second and fail proof possibility is to query the global authority. The global authority (GA) holds all $\langle RP, identifier \rangle$ tuples and can be queried, in case the regional prefix can't be learned by any other means.

As mentioned earlier, the mapping system is partitioned into multiple regions. For HiiMap, we propose to base the partitioning on countries, whereby each country forms its own region. This concept has two important advantages. Firstly, most countries show a relatively stable state. It rarely occurs that a country institutes or vanishes. This means there is a very seldom need to adjust the regional prefix. The second benefit of a partitioning based on countries is the common legal system. Each country has its own laws and ways of law enforcement. Therefore, a region based on more than one country has to deal with different political and legal systems. Smaller countries with similar laws, of course, can form a single region to lower the administrative overhead. In this way it is easier to build trust relationships between providers and handle infringements of contracts by the local law enforcement. Furthermore, we propose that the mapping system in each region is operated by a non–profit organization.

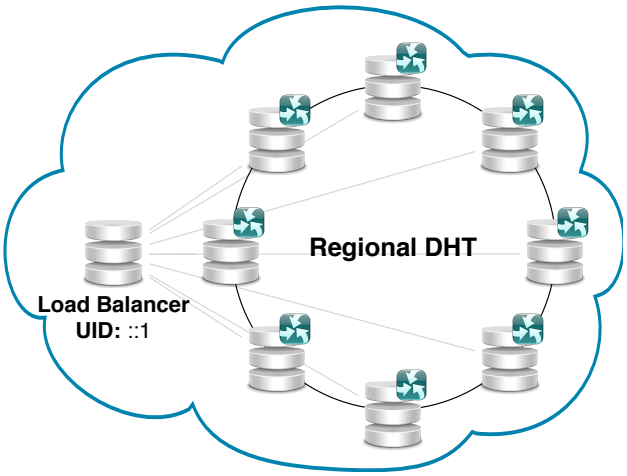

Figure 3. Mapping system of one region

Figure 3 illustrates the mapping architecture within one region. To be able to cope with the huge load of the mapping system, one–hop distributed hash tables (DHT) are used. In that way, it is no problem to meet the storage capacity requirements and the servers within the DHT are able to handle frequent locator updates and mapping requests. To provide a well–known address and fairly distribute the request load over the DHT, a load balancer is used. The address for each region is the same (e.g. region number::1) and clients do not need any additional information to access the mapping service of any region.

IV. Public Key Infrastructure

In this section, we describe the integration of the public key infrastructure into the HiiMap mapping system.

A. PKI and the mapping

In today's Internet, the public–key infrastructure is separated from all network services. This means that additional resources for the PKI must be provided despite all the network elements already in place for other functionality (e.g. DNS server). Contrary to this, we propose to integrate the PKI into the mapping system for the HiiMap architecture. This has the benefit that resources can be shared between functionalities and maintenance can be kept significantly lower compared to operating separate services.

Each mapping entry consist of the identifier as the primary key and a set of locators by which the node currently can be reached (see section III). Further, a timestamp of the last

update and a flag indicating whether the location update was cryptographically signed by the node or not is stored (we will come back to this issue later on).

To combine the PKI with the mapping system, only the public–key of each node must be additionally stored for each mapping entry. This means, that no additional protocol or infrastructure must be provided for querying and storing the public keys. Because the public–key is a very static value and not expected to change frequently, the additional burden for the mapping system is limited and the public–key databases can be optimized for frequent read requests—contrary to frequent read and write requests for the locators.

In comparison to today's public–key infrastructure, it is also not necessary to provide additional lists for key revocation. This is implicitly realized by the loose identifier - public–key binding. Whenever a public–key for a certain identifier changes, the old public–key implicitly becomes invalid.

B. Trusting the mapping

By storing the public–key at only one location (region) in the mapping system, however, the user heavily depends on the trustworthiness of that particular location. In case the mapping service provider collaborates with an attacker, it could send a wrong or manipulated public–key to the client. Therefore, any security functionality based on the public/private–key principle would be rendered useless. Even worse, the client considers the connection to be secure while in fact talking directly to the attacker.

Therefore, we propose to distribute several copies of the public–key to various independent locations (regions) in the mapping system. Figure 4 illustrates the basic principal.

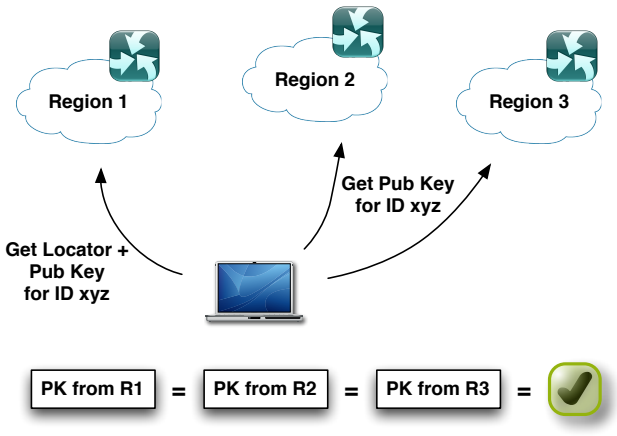

Figure 4. The public key is stored at multiple regions

The client first queries the responsible region (RR) of the identifier it wants to communicate with. As response, the RR replies with the locator and public–key stored for that identifier. In a next step, the client queries additional regions for the public–key. We will explain which regions to query in the next section. After receiving all requested public–keys, the client compares these. In case they do match, it is very likely that the public–key is the correct one. Contrary, if they differ, the client can either stop the communication setup or decide, which is the correct key based on the majority principle.

There is one special case, however. If the public–key from the RR differs from the other ones, then the retrieved locator must be considered incorrect as well. This is because having identified the RR as accessory or even the attacker itself, it is very likely that the locator has been modified as well and is now pointing directly towards the attacker.

A solution to this problem would be to also replicate the locator over several other regions. This, however, is not a good idea performance wise. The locator is the entry in the mapping system, which will be updated and changed frequently. In case several regions hold a copy of it, these changes have to be carried out to all of them. The public–key on the other hand is expected to change very rarely and thus causing very little update traffic.

C. Determining the storage location

Having copies of the public–key distributed over several locations in the mapping system, one questions remains: In which way does the client learn about the storage location of the additional copies.

Storing the list of the additional locations at the RR does not solve the problem. In case the RR is the attacker, it can simply manipulate this list as well and distribute the malicious key to collaborating regions. Therefore, the client must learn the information about the storage locations in another way.

For the following proposal, we assume that less than 256 regions exist and the key is distributed to two additional regions. After having received the locator and public key from the RR, the client hashes the identifier to an 16 bit value. The 16 bit value is split into two halves (8 bit each). Each 8 bit value represents the storage location of one of the public key copies. We will call it key storage address space (KSA) from now on. Since the 8 bit address space for the regions is not completely full, a mapping directive is required. This mapping directive can be downloaded from the global authority. It is sufficient to do this very seldom, as the directive is expected to change very rarely. For each value in the KSA, the mapping directive specifies a region, where the key is stored. This means, that a single region can be responsible for several KSA values. In that way, the load can be fairly distributed over all regions depending on their size. Figure 5 illustrates the process.

Should the hashing and mapping to regions result in two copies of a key being stored at the same region, the first copy is stored at this region and the second copy at the region with the next higher region number.

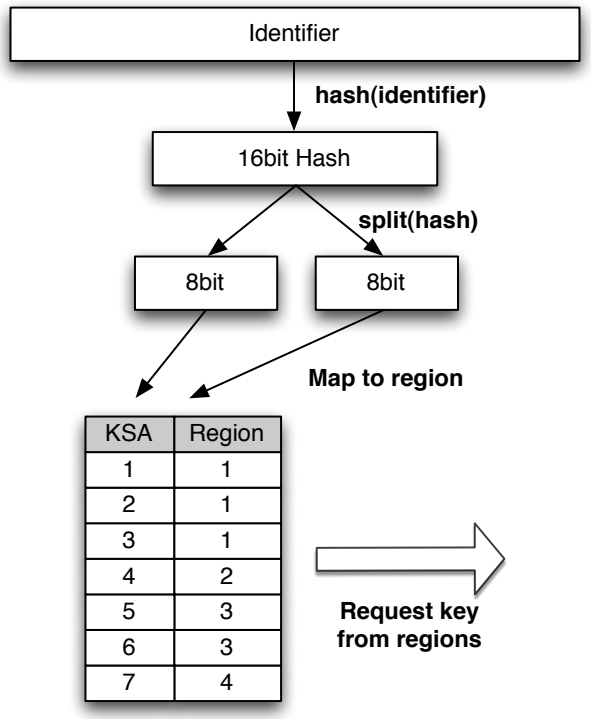

Figure 5. Retrieving the additional key storage locations

V. User Key Management

The user key management in our concept is based on cryptographic smart cards (herein after referred to as *smart card*). For the sake of simplicity, we assume that the smart card can't be compromised on a physical level and that it is protected by the well–known PIN/PUK mechanism. As of today, this mechanism is considered to be secure enough to protect any smart card from unauthorized access.

As with the previous section, the approach is not bound to a specific architecture. We again, however, will use HiiMap as an example architecture to illustrate the functionality. Any other concept, which provides a *single point of trust* (SPT) can be used as underlying architecture. As with HiiMap, the SPT should be independent—politically and organizationally—to reflect all participants' needs and interests in the same way. It will take responsibility considering key management issues and authentication of particular nodes later on. SPT, however, doesn't mean that it has to be one physical component with respect to reliability. Furthermore, management tasks can be delegated to subsequent authorities. In HiiMap the global authority (GA) acts like a SPT and can delegate tasks to the regional authorities (RA).

In this section we will discuss the topics of authentication methods, initial bootstrap, key revocation and how devices with low computational power can participate.

A. Peer communication

The authentication and communication concept differentiates between the used hardware components (notebook, PDA, mobile phone, etc.) and access authorization, which is handled by the above mentioned smart card. Vendors only have to provide an interface for this card in each of their products. The assembly of these smart cards is done by the *single point of trust* (SPT) respectively by another party authorized and trusted by the SPT. They contain a master key pair, the card-ID and, of course, the identifier address. In HiiMap, this identifier is called UID (see section III). Before the SPT can send the cards to authorized providers, it has to save every public–key stored on them and the other entries already mentioned in Section IV-A. The SPT furthermore saves the appropriate card-ID and if the particular card is already in use or not. Therefore, it always possesses all relevant information. We have to remark that authorized providers of course can keep a certain amount of smart cards in stock so that they do not have to request every single card each time they get a new customer.

Every time somebody buys a new device, he chooses a provider. If he is in possession of such a smart card already, he can either sign up for a new one (and meanwhile use the existing card) or use the old one in the new device. It is also possible to change the provider with every card request. This modularity is an important advantage of smart cards in comparison to fixed security modules as they provide much more flexibility and do not involve manufacturers in the network management process (assignment of identifiers, etc).

If the user requests a new card, the provider then sends smart card and PIN/PUK to the user, for example by mail. If the user requests such a card for the first time, the trader informs the particular provider directly at buying time to minimize downtime. At the same time, the provider tells the authorities (the SPT or its delegates) about the selling of this card. They can then update their databases and know that the particular card is in use from now on. Thus, the authorities know which cards are in use and which aren't at any point in time. This makes it difficult for possible attackers to use non–assigned card-IDs. Assigned IDs cannot be compromised, because the attacker cannot prove the possession of the private key, as we will see later on.

After the user has received the smart card, he can sign on to the device by inserting the card and typing the correct PIN. The security mechanisms can then be enabled and the device is able to authorize itself to the network.

B. Bootstrap

The procedure of joining a network for the first time is called *bootstrap*. If the particular user is not yet known to the network and other users, there is no possibility to prove his identity in general. In most cases of security mechanisms, other peers have to trust this user once. After keys have been

exchanged between participants, they can later on check the identity with the corresponding key pairs. As this is a great drawback (possible attackers could replace the keys with their own), we will present a solution to this problem.

As mentioned above, the authentication procedure uses a smart card for key storing and cryptographic functions. The SPT (and its delegates—also called "authorities" in the following) is in possession of all public keys stored on these cards and can connect them to the respective IDs (see section V-A). This is the essential point of the bootstrap mechanism. Imagine a node i joining the network where the associated user has already enabled the smart card by entering the correct PIN. As shown in Figure 6, the node first has to send a *location update request* to the responsible authorities. It contains the card ID of the smart card used, so that the network can check whether or not the card is allowed to participate in the network's functionalities. This message is already signed with its private key to prove integrity. Therefore, the whole communication is integrity protected from the beginning.

The authority can then lookup the node's public key. If the public key is not yet known to the authority, it has to request it from the SPT. After that, the authority computes a common secret K_{ir} using node i's public and its own private key, similar to the Diffie-Hellman–procedure [7]. This secret K_{ir} can also be computed by node i in the same way (i also gets the public key of authority R from the SPT). Therefore, the common secret K_{ir} never has to be exchanged between the two peers, which eliminates the danger of being compromised. Furthermore, it is only used once to encrypt data (part of message 2 in Figure 6). With this message, the authority chooses a random session key K_p and a rule to generate a modified common key K_{ir}^*. K_{ir}^* can be calculated, for example, by shifting K_{ir}, computing the product K_{ir} XOR itself or other methods. The authority can then answer the location update request by sending this message containing the rules for generating K_{ir}^* and the security functions the authority is capable of (message 2 in Figure 6). This information is encrypted using the random session key K_p. K_p is encrypted with the common secret K_{ir} and sent inline in the packet (message 2 in Figure 6), based on the principle used in SKIP [17]. Besides, K_{ir} is only used once to encrypt data. All other packets use the modified version, which again minimizes the risk of compromising K_{ir} itself. The header information in this and all other subsequent packets are sent in plain text. This reduces complexity for network nodes, firewalls and so on. A possible attacker possibly acquires part of the payload by resolving the security functions by sending an own location update request. However, this is not enough information to decrypt the key. Afterwards, the authority sends another packet to node i containing a random number n_A, which is again encrypted using K_p (and K_p with the *modified* common secret K_{ir}^*), see message 3. Node i can extract the chosen session key K_p by decrypting it with the common secret K_{ir} of message 2 and then the security functions and rules for generating K_{ir}^* with K_p. After that, node i can compute the modified common secret and therefore decrypt the random number n_A of message 3. Node i

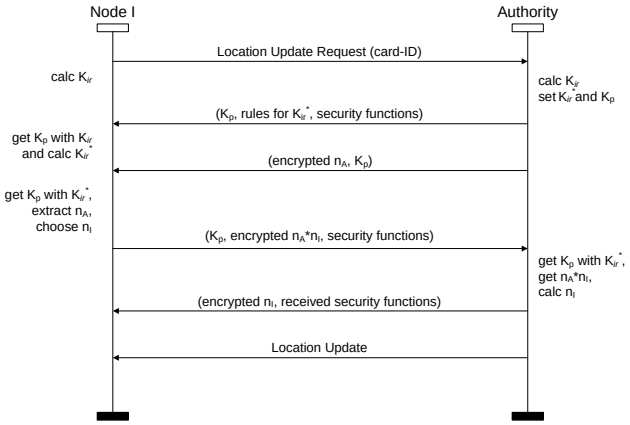

Figure 6. Bootstrap Message Flow Chart

then also chooses a random number n_I and calculates the product $n_A * n_I$. This product is sent back to the authority together with i's security functions. Both are again encrypted using a random session key K_p, which can be the same as above or vary depending on whether the key is valid packet- or session-wide. K_p again is encrypted using the modified common secret K_{ir}^* (message 4). The authority can extract the particular information in the same way node i did before and therefore calculate the chosen random number n_I. Afterwards, the authority can select the strongest algorithms for creating new session keys K_p supported by i and R. By sending back this random number in message 5 of Figure 6 both parties can be sure that the other part is in possession of the right (modified) keys. Additionally, the received security functions of node i are sent back to prove they have not been manipulated. A modification of all those messages would also mean that the signatures become invalid as every message is not only encrypted, but also signed to prove integrity. In a last step, the location update request is accepted by the authority, which results in publishing node i's assigned locator address in the mapping system, so that other nodes can resolve it from then on. The user or node is then allowed to upload his own key pair for further use, which has to be validated with the old key pair again. Thereby, the peer has flexibility to use own algorithms for creating the keys and the possibility to influence the parameters, such as the key length. If keys are changed later on, every party can signalize the wish to update it with a special *key_update* message. It contains the new key validated with the old one. Bootstrap is completed and node i can go on communicating with other peers.

Erroneous messages are ignored by the system and the

user has to resend them. To avoid denial of service attacks by exploiting this, the system only allows a maximum number of requests and responses at a time, i.e., five requests within ten seconds. After that, the system will not accept any more messages of the particular node in a certain time. At any time, the user has to be informed if encryption is disabled. This can be done by the operating system, for example.

C. Dealing with network components

Cases may occur, where users are located behind a firewall, proxy or similar network entities and are not directly reachable. These non–end–to–end cases are considered as well. A typical connection establishment for those cases looks like the following:

First of all, the requesting peer contacts the firewall by looking up the appropriate UID associated with a human readable address as known from the *Domain Name System* (DNS). The firewall redirects the connection request to the particular node, i.e., according to load balancing rules and also informs the requesting peer (see message 2 in Figure 7). Of course, UIDs have to be looked up by the firewall, too. We call them *RSLVreq (resolve request)* and *RSLVresp (resolve response)*, but left them out in Figure 7 for the sake of simplicity. After that, both nodes can request the

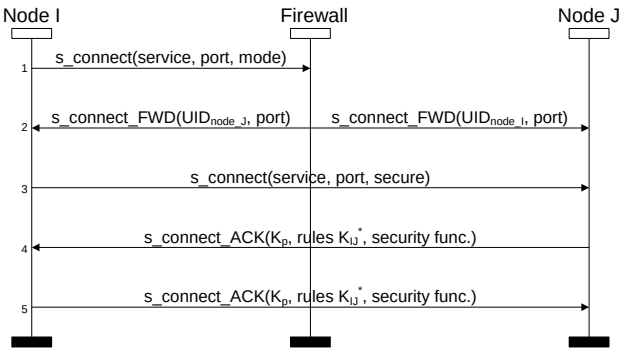

Figure 7. Passing firewalls (simplified)

needed information held by the SPT and start connecting to each other (see messages 3 to 5 in Figure 7, where some additional parameters for encryption and defining the desired service are negotiated—detailed explanation of single parameters see Section V-B). The firewall itself is able to route the packets correctly as it stores the connection data like in NAT–gateways. By using clear headers instead of encrypted ones, every network node and therefore the firewall is able to process all needed data. Both users can decide about the connection mode on their own at any time, e.g., encrypted or plain. Similar to the bootstrap process (see Section V-B), keys can be updated anytime by sending a *key_update* request.

D. Disabling authentication

In some cases it may be necessary to connect even devices without smart cards, as they are difficult to reach physically, e.g., sensors, satellites, etc. Moreover, most of the available sensor data is not crucial, so that there is no drawback to operate them in plain text communication mode. Additionally, not every single sensor needs to be connected to the Internet, e.g., in cars it is sufficient if the board computer is connected. Nevertheless, cases may occur in which those devices have to be integrated without the chance of attaching the smart card to them. The procedure then is as follows: First of all, we assume a legal owner of this device, let us say, a company operating a sensor. This owner requests a smart card for the sensor in the described manner. After that, he securely keeps the card somewhere and implements the particular UID into the sensor's firmware and also his own UID. Concurrently, the owner connects to the network using the sensor's smart card and the appropriate PIN. After the encrypted location update is completed successfully, he then tells the network or alternatively the authorities that the UID he is connecting from will disable authentication mode in the future. This is stored in the database entry called *mode of last location update* (see Section VI-B). The network then and only then permits plain text location updates from the particular UID. Thus, disabling security functions is possible, but only on explicit inquiry. If the UID ever wants to return to secure mode, this again has to be done with the appropriate smart card and PIN and can therefore only be realized after secure location updates. After that, the owner keeps the card and PIN secret again. In this way it can always be guaranteed that only the sensor itself can disable encryption. To summarize, the procedure is depicted in Figure 8.

Consequently, an attacker is not able to force plain text communication. If the owner decides to sell the device, he simply has to distribute the particular card to the new owner, whereon he can handle communication modes on his own. Requesting a new smart card for the sensor (with same ID, of course) is also possible. This mechanism is another great advantage of modular chips in comparison to fixed ones as they easily enable such devices to join the network. Every time such a sensor connects to the network, it sends an unencrypted location update request to the responsible authorities. The mode of the connection (secure or plain) is indicated by special fields in the message's header. The authorities can then check, whether plain text location updates are enabled for the particular node or not. After that, the location update without authentication is granted and the sensor can participate in the network's functionalities. We have to remark that such devices only get limited access to resources as they have not been identified securely. Furthermore, they have to keep plain text communication enabled for security reasons (otherwise one could easily

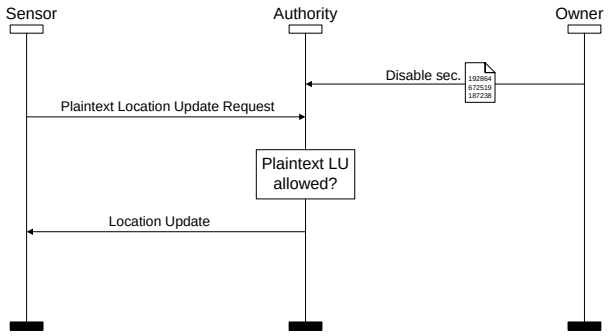

Figure 8. Location update with disabled security

upgrade to ”secure” mode and get access to critical content).

Every time a user requests the public key of another user from the authorities, they also inform him about the last kind of location update (secure or insecure). He can then decide on his own whether he wants to continue connecting to the desired node or not. This offers many possibilities for user-defined security policies, as it is possible, e.g., to grant access to e-business-products only to securely authorized nodes, whereas insecurely authorized nodes only get access to information material.

The concept of providing plain text authentication can also be seen as fallback solution in case the whole system becomes compromised in the future. Then it is sufficient to switch all participants to plain text mode and provide an overlay network that is responsible for security issues.

E. Key Revocation

The high modularity in this concept using smart cards also implies a drawback: the card can easily be lost or even stolen. This can be mitigated as the card itself is not usable without the appropriate PIN assigned to the authorized user. In our opinion, the chance for a possible attacker to get the valid PIN in only three guesses is very small and thus negligible. Besides, after that the card is disabled until the correct PUK (*personal unblocking key*) is entered. The PUK again may not be entered incorrectly more than ten times. This behavior is known from today's concepts and is assumed to be secure enough. The smart card itself—as in today's ones—has to be protected against physical and chemical manipulation such as side channel attacks, power analysis, etching and so on (details see [18]). Baring these things in mind, the physical theft of a smart card implies no great security risk. Of course, there has to be some kind of approach the user has to follow if his card is stolen or lost:

The user has to report the loss or theft immediately to his responsible provider, i.e. by phone, who then disables the card by denying the particular card-ID from joining the network in the future. This can be done by the authorities or delegated to the providers. The disabled card-IDs are stored in order to detect future connections of the card and eventually having the chance of locating it. The SPT then also has to be informed as it now is in charge of producing a new card containing the old identifier (as the identifier shall not change), a new key pair and, of course, a new card-ID. This step may take some time, as the card is not already produced but has to be created on inquiry. After that, the normal procedure takes place again: The SPT and the responsible authorities get the public key and replace the old one with it. The card/PIN pair is delivered to the provider who then ships it to the desired customer. After that, the user can proceed as normal.

The provider has to make sure that only the actual owner of the card is allowed to report the loss or theft so that an attack on disabling all cards by simply calling all providers is not possible. This can be done by requesting some additional information of the caller, for example street, postal code, birth date etc. Even if the attacker is in possession of this information, the risk of such an attack is highly improbable as this causes much effort for the attacker and can not be automated in an easy way.

Following this procedure, the device or the user is able to request a new key pair without loosing his assigned identifier. Other nodes in the network will probably not even notice the change as they request the public key from the authorities and do not cache the old public key for an unlimited time.

VI. Evaluation

The security mechanism proposed in this paper is very flexible and neither bound to a specific algorithm nor architecture. Only some already mentioned pre-requirements must be met. For the analysis of the mechanism, however, we need to assume some protocols and algorithms, which are likely to be used. Please note that the mechanism can also be applied to different proposals and is not limited to the ones discussed in this section.

A. Algorithm

As already mentioned, we choose HiiMap as the underlying locator/identifier–split architecture where the *Global Authority* (GA) acts as single point of trust. The GA can delegate management and maintenance tasks to *Regional Authorities* (RA), which are responsible for their region respectively. Thus, without any security capabilities enabled, the system already has to store these entries (UID, LTA and Region Prefix).

The public/private–key principle requires an asymmetric cryptographic algorithm. The probably best known one is *RSA* by Rivest et al. [19]. RSA is widely used in electronic commerce protocols, and is believed to be secure given sufficiently long keys. While a key length of 1024 bit is still assumed to be secure enough, a length of 2048 bit is recommended (as of early 2010). A downside of the

RSA algorithm, however, is its large memory footprint and requirement towards the computational power of a device.

Another asymmetric algorithm, the *ECC* by Koblitz et al. [20], can be found on some low power units like smart cards and sensors. This is because ECC is less computational power and memory consuming compared to RSA [21]. Also the required key length is almost one magnitude smaller while maintaining the same level of security. An 160 bit ECC key, for example, is believed to be equal in terms of cryptographic strength compared to a 1024 bit RSA key.

To provide a better overview and point out the flexibility once again, we will calculate the resource requirements for both, RSA and ECC. As you will see, replacing the algorithm will result in a huge decrease of requirements. In case of ECC, we choose a key length of 160 bit. We will first do the calculation with the RSA algorithm and then present the respective values for ECC.

B. Analyses

To calculate the overall overhead and storage requirements, we have to make some assumptions and declarations. First of all, we have to choose an average key length for the *public–key*. The initial key pairs stored on the smart cards are all of the same size, but users can later change them and influence these parameters. As in today's *Trusted Platform Module* (TPM) [22], which we consider to be safe enough, RSA keys of 2048 bit are used, we will assume an average *public–key length* of 2048 bit for the RSA study. In fact, this is the entry which consumes most of the needed storage in comparison to the rest. Increasing or decreasing this value will have a strong impact on the overall storage capacities. The next entry is the smart card's *actual allowed ID*. With this ID, the network can verify if the particular node is allowed to join the network or if this card has been reported as stolen or lost. The card-ID is only valid within the UID range, therefore it is sufficient to reserve 32 bits for it. This value is high enough to avoid guessing the next valid card ID by attackers, as well as leaving sufficient space to replace the card several times every day. The last entry in Table I *(list of disabled card IDs)* is the opposite: it holds all card IDs that have been disabled and are no longer allowed to connect. In case of such an ID joining the network, the authorities can trace the request and thereby locate the missing card. Depending on the number n of disabled smart cards, the list may increase.

Last but not least, there are two entries of four bit each: On the one hand, the field *UID assigned*, which specifies if the particular UID is already in use by a node or not, and on the other hand, the entry *mode of last location update*. This field gives information about whether the last location update was encrypted or in plain text. It is sent with every public key request so that the peer can decide on its own whether or not it wants to continue connecting. Both fields could have also been realized with only one bit, but by reserving three additional bits we get enough flexibility to adapt future challenges, e.g., the connection modes can be split up in more detail.

entry	length
UID	128 bit
LTA	128 bit
RP	8 bit
public key	2048 bit
valid card ID	32 bit
mode of last location update	4 bit
UID assigned	4 bit
list of disabled card IDs	$n*$ 32 bit
Sum	$2352 + n * 32$ bit

Table I
ENTRIES TO BE STORED (LIKE IN HIIMAP [2]) – RSA

A summarizing overview of all necessary entries can be found in Table I. Based on them we want to present a typical example to estimate the needed storage capacities in a future NGI system. Therefore, we choose ten as the number of smart cards per human being on earth (currently about $6.7 * 10^9$). We assume an average invalid card–ID count of 20 per smart card. Putting all these parameters (see Table II) together, we get a total requirement of $10*6.7*10^9*(2352+20*32)$ bit $= 2.00464 * 10^{14}$ bit $=$ **25.05 Terabyte**. Even in today's architectures, this value is no major challenge. Taking the computing power available in 10 to 15 years into account, we are not talking about a huge burden compared to the security benefits we are gaining. If the load can be delegated to subsequent authorities like the RAs in HiiMap, the burden is distributed over several nodes. This also applies to bandwidth and other metrics so that the SPT itself ideally is not involved in handling authorization requests unless the delegates (RAs in HiiMap) can not resolve them on their own. Thus, our concept demands no great resources and therefore is suited for use in any kind of locator/identifier–split architecture.

parameter	value
n	20
cards per human being	10
human beings on earth	$6.7 * 10^9$

Table II
PARAMETERS USED (LIKE IN HIIMAP [2])

Having outlined in detail the requirements for an implementation with RSA, we will now shortly present the respective values in case of using ECC. Most of the values presented in Table I do not change as they do not depend on a specific algorithm. What in fact does change, is the public–key, of course. This decreases from 2048 bit (RSA) to 160 bit (ECC). So the overall sum decreases, too (see Table III). The values presented in Table II do not change

entry	length
UID	128 bit
LTA	128 bit
RP	8 bit
public key	160 bit
valid card ID	32 bit
mode of last location update	4 bit
UID assigned	4 bit
list of disabled card IDs	$n*$ 32 bit
Sum	$464 + n * 32$ bit

Table III
ENTRIES TO BE STORED (LIKE IN HIIMAP [2]) – ECC

at all. If we calculate the sum again, we get an overall storage requirement of $10 * 6.7 * 10^9 * (464 + 20 * 32)$ bit $= 1.10 * 10^{13}$ bit = **1.38 Terabyte**. So the storage capacity can be decreased by an order of more than ten.

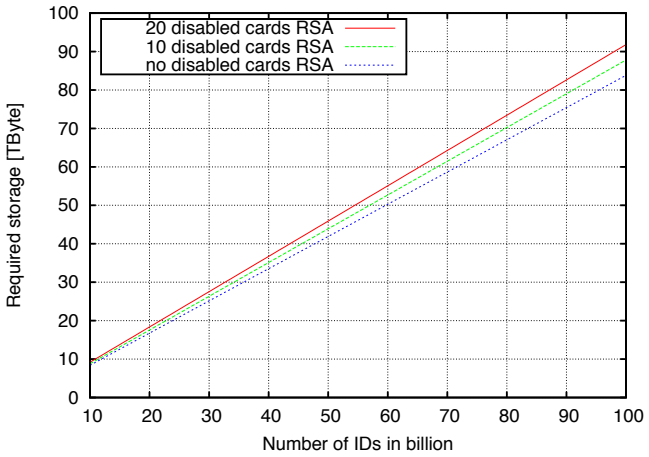

Figure 9. Storage requirement RSA

As already pointed out in section IV-A, we need to store the public key not only at one, but at multiple locations. For the overall storage calculation we have to reflect this additional requirement as well. The total storage varies depending on the number of used UIDs and the number of disabled card IDs per UID. Figure 9 shows the storage requirement using 2048 bit RSA keys and Figure 10 the requirements with 160 bit ECC.

Concluding our computations, the concept does not consume a huge amount of resources. No matter which algorithm we choose, we are not facing a huge burden to the architecture—even in today's view. Nevertheless, by decreasing the biggest factor, the public–key length, we can decrease storage requirements drastically.

VII. CONCLUSION

Many security concepts for locator/identifier–split architectures bind the identifier to the public–key. Contrary to this common approach, we suggested a loose coupling between

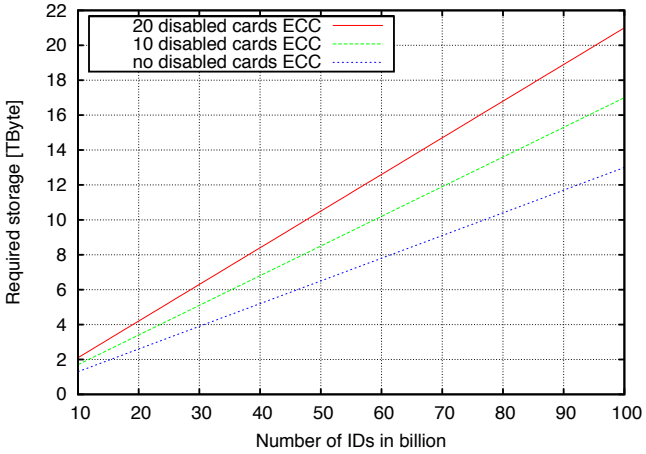

Figure 10. Storage requirement ECC

the identifier and the public–key, allowing for exchangeability of those two entities [2].

In this article, we presented a holistic approach to public key management including key distribution, key revocation and key storing. We covered the public–key infrastructure aspect by extending the mapping system to also store a public–key for each identifier. We introduced a mechanism to trustfully retrieve keys from the mapping system without being dependent on a single region of the mapping.

Furthermore, we discussed the client side key management and suggested to use smart cards to store the private and public–key. We described the initial bootstrap process, detailed the communication setup and showed how devices with very limited computational power can also participate by disabling encryption.

The concept is not bound to a specific crypto algorithm and is able to cope with varying key length. Therefore, the architecture is very flexible and open to future improvements or requirements. Although we explained the concept by using the HiiMap architecture as example, the concept can be applied to any locator/identifier–split architecture, which provides a single point of trust and a mapping, which can be divided into several administrative zones.

ACKNOWLEDGMENT

This work has been performed within the G-Lab project and was funded by the Federal Ministry of Education and Research of the Federal Republic of Germany (Project ID 01BK0807). The authors would also like to thank their colleagues at the Leibniz Supercomputing Centre of the Bavarian Academy of Sciences and Humanities (see http://www.lrz.de/) for helpful discussions and valuable comments about this paper. The authors alone are responsible for the content of the paper.

REFERENCES

[1] W. Fritz and O. Hanka, "Smart Card Based Security in Locator/Identifier-Split Architectures," *International Conference on Networking*, pp. 194–200, April 2010.

[2] O. Hanka, G. Kunzmann, C. Spleiß, J. Eberspächer, and A. Bauer, "*HiiMap: Hierarchical Internet Mapping Architecture*," *In First International Conference on Future Information Networks, Beijing, China, P.R. China*, pp. 17–24, October 2009.

[3] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica, "A data-oriented (and beyond) network architecture," in *SIGCOMM '07: Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: ACM, 2007, pp. 181–192.

[4] R. Moskowitz and P. Nikander, "Host Identity Protocol," IETF, United States, RFC 4423, May 2006.

[5] M. Menth, M. Hartmann, and M. Hoefling, "Firms: a future internet mapping system," *IEEE Journal on Selected Areas in Communications (JSAC), Special Issue on Internet Routing Scalability*, August 2010.

[6] A. Feldmann, L. Cittadini, W. Mühlbauer, R. Bush, and O. Maennel, "HAIR: Hierarchical Architecture for Internet Routing," in *ReArch09*. New York, NY, USA: ACM, December 2009.

[7] W. Diffie and M. Hellman, "New Directions in Cryptography," *IEEE Transactions on Information Theory*, vol. IT-22, no. 6, pp. 644–654, 1976.

[8] M. Komu and J. Lindqvist, "Leap-of-faith security is enough for ip mobility," in *Consumer Communications and Networking Conference, 2009. CCNC 2009. 6th IEEE*, Januar 2009, pp. 1–5.

[9] S. Siddique and M. Amir, "GSM Security Issues and Challenges," in *Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2006. SNPD 2006. Seventh ACIS International Conference on*, June 2006, pp. 413–418.

[10] M. Toorani and A. Beheshti Shirazi, "Solutions to the GSM Security Weaknesses," in *Next Generation Mobile Applications, Services and Technologies, 2008. NGMAST '08. The Second International Conference on*, September 2008, pp. 576–581.

[11] M. Khan, A. Ahmed, and A. Cheema, "Vulnerabilities of UMTS Access Domain Security Architecture," in *Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2008. SNPD '08. Ninth ACIS International Conference on*, August 2008, pp. 350–355.

[12] A. Bais, W. Penzhorn, and P. Palensky, "Evaluation of UMTS security architecture and services," in *Industrial Informatics, 2006 IEEE International Conference on*, August 2006, pp. 570–575.

[13] U. Meyer and S. Wetzel, "A man-in-the-middle attack on UMTS," in *WiSe '04: Proceedings of the 3rd ACM workshop on Wireless security*. New York, NY, USA: ACM, 2004, pp. 90–97.

[14] R. Housley, W. Ford, W. Polk, and D. Solo, "Internet x.509 public key infrastructure certificate and crl profile," IETF, United States, RFC 2459, January 1999.

[15] R. Housley, W. Polk, W. Ford, and D. Solo, "Internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile," IETF, United States, RFC 3280, April 2002.

[16] C. Ellsion and B. Schneider, "Ten risks of pki: What you're not being told about public key infrastructure," *Computer Security Journal*, vol. 16, no. 1, pp. 1–7, 2000.

[17] A. Aziz, M. Patterson, and G. Baehr, "Simple Key-Management for Internet Protocol (SKIP)," in *Internet Society: INET'95 Hypermedia Conference Proceedings*, June 1995.

[18] W. Rankl and W. Effing, *Smart Card Handbook*, 3rd ed. John Wiley & Sons, 2003.

[19] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 26, no. 1, pp. 96–99, 1983.

[20] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of Computation*, vol. 48, no. 177, pp. 203–209, 1987. [Online]. Available: http://www.jstor.org/stable/2007884

[21] N. Gura, A. Patel, A. Wander, H. Eberle, and S. C. Shantz, "Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs," in *Lecture Notes in Computer Science*. Berlin/Heidelberg, Germany: Springer, 2004, vol. 3156/2004, pp. 925–943.

[22] The Trusted Computing Group, "Trusted Platform Module (TPM) Main Specification, Version 1.2, Revision 103," http://www.trustedcomputinggroup.org/resources/tpm_main_specification,17.11.2009.