SECURITY AND COMMUNICATION NETWORKS
*Security Comm. Networks* **00**: 1–20 (2009)
Published online in Wiley InterScience
(www.interscience.wiley.com) DOI: 10.1002/sec.0000

# Generating Certification Authority Authenticated Public Keys in Ad Hoc Networks

G. Kounga[1], C. J. Mitchell[2] and T. Walter *[3]

[1] *Systems Security Lab, Hewlett-Packard Laboratories, Long Down Avenue, Stoke Gifford, Bristol, BS34 8QZ, UK*
[2] *Information Security Group, Royal Holloway, University of London, Egham, Surrey TW20 0EX, UK*
[3] *Research Planning and Promotion/Security Specialist, DOCOMO Communications Laboratories Europe GmbH, Landsberger Strasse 312, 80687 Munich, Germany*

## Summary

In an ad hoc network, nodes may face the need to generate new public keys. To be verifiably authentic, these newly generated public keys need to be certified. However, because of the absence of a permanent communication infrastructure, a certification authority (CA) that can issue certificates may not always be reachable. The downside is that secure communication channels cannot be established. Previously proposed solutions do not guarantee that identities contained in certificates are valid or, when they do, they rely on neighbors to validate user-key bindings. However, there is no guarantee that nodes that are known in advance will always be present in the network. Therefore, neighbors are not always able to verify a node's identity before certificate issuance. In this paper we define a scheme that permits nodes to generate, on-demand and independently of any third entity, public keys that can be authenticated with the aid of a unique certificate, issued by a CA at initialization. This certificate binds a valid identity to a hash code. We then extend this scheme to a solution permitting certificates to be generated, on-demand and independently of any third entity, that can be authenticated with a unique signature generated by a CA. Finally we solve the problem of updated revocation information. Copyright © 2009 John Wiley & Sons, Ltd.

KEY WORDS: Public key cryptosystems; Authentication; Security; Hash chains

## 1. Introduction

The establishment of secure communication channels between two nodes requires that they are mutually authenticated. When nodes that need to securely communicate do not know each other in advance and do not share any secret key a priori, public/private key pairs can be used provided that an infrastructure is always in place that permits nodes to be issued certificates and to be notified when a certificate has been revoked. However, this assumption does not hold in ad hoc networks.

Ad hoc networks are wireless networks that do not rely on any infrastructure. They are composed of wireless nodes that are free to move, causing frequent changes in the network topology; network partitions may even occur. This means that no connectivity to a central entity such as a certification authority (CA), a certificate revocation list (CRL) [1] repository or an online certificate status protocol (OCSP) responder [2] can be guaranteed. Ad hoc networks are also self-organized, which means that they operate with no

*Correspondence to: Thomas Walter, Research Planning and Promotion/Security Specialist, DOCOMO Communications Laboratories Europe GmbH, Landsberger Strasse 312, 80687 Munich, Germany
Email:walter@docomolab-euro.com

central administrative authority. As a result, no part of the network is dedicated to providing any specific services. Therefore, the security services provided by dedicated entities such as CAs, CRL repositories and OCSP responders must be provided by the nodes themselves.

- ***Identity verification:*** Nodes must be able to verify by themselves that an entity that requests a certificate for a given identity is the legitimate owner of that identity. This task is traditionally done by a CA. However, in an ad hoc network it is not guaranteed that a CA is always reachable. Further, there is no guarantee that only nodes are present in the ad network whose identity is known in advance, since no authority responsible for network membership is always reachable [3]. Hence, in ad hoc networks, it is almost impossible to guarantee that nodes can always generate certificates that permit the establishment of secure communication channels with the desired entity.
- ***Certificate revocation:*** Nodes must be able to determine whether a certificate has been revoked or not, i.e. whether or not a certificate still represents a valid binding between an identifier and a public key. This action should be possible even when no node is present in the network that has current information regarding the revocation status of a certificate. Without any means of determining the revocation status of a certificate, nodes in ad hoc networks can never be sure whether they are interacting with the intended entity or with an attacker that is using a certificate that was previously revoked in an unreachable part of the ad hoc network or in another network.
- ***Node compromise:*** Finally, since nodes in ad hoc networks are mobile devices, they are less physically protected than 'non-mobile devices' that can be kept in locked areas [4]. This can permit an attacker to compromise a public/private key pair stored on the node by simply gaining control over that node.

### 1.1. Contributions of the paper

In this paper we adopt an incremental approach to the definition of a solution that permits nodes in ad hoc networks to generate, on-demand, multiple public/private key pairs and certificates without relying on their neighbors and without having to generate and store in advance all the keys and certificates they may use in the future. Instead, our solution relies on a unique certificate that is issued by a CA at initialization, and that proves the authenticity of all public keys that are generated later. The initial certificate does not contain a public key. It binds a node's identity, verified by a CA in the fixed network at initialization, to a hash code value. Then, when a node is active in an ad hoc network, it generates a public key whose authenticity can be checked by verifying that it is correctly linked to the hash code contained in the certificate. Only the node that was issued a certificate by the CA at initialization is able to generate a valid public/private key pair that is correctly linked to the certified hash code. The solution relies on a novel scheme that permits the verifiable binding of multiple public/private key pairs to a single hash code, which existing solutions do not permit — see Section 5. The defined solution also solves the revocation problem existing in ad hoc networks by using public/private key pairs with a short lifetime. Finally, our solution avoids attackers accessing public/private key pairs stored on nodes with no physical protection and with no secure hardware such as tamper-resistant hardware. This is achieved by relying on interactions with users to generate key pairs.

### 1.2. Outline of the paper

The paper is organized as follows. In Section 2, we identify the problems that our solution must solve, and we define the requirements that it must fulfil. In Section 3 we use an incremental approach to the definition of our solution. A first solution is proposed in Section 3.1 that is improved in Sections 3.2 and 3.3. Finally in Section 3.4, a solution is defined that fulfils all the requirements defined in Section 2. In Section 4, we illustrate how our solution supports the provisioning of security services. We then extend our solution to permit certificates to be generated whose authenticity can be checked with a unique signature generated in advance by a CA. We discuss related work in Section 5, evaluate the performance of our scheme in Section 6 and conclude the paper in Section 7.

## 2. Problem statement

### 2.1. Proposed Approach

Considering the properties of ad hoc networks as described in the previous section, our goal is to define a scheme that permits nodes to generate, on-demand, a series of public/private key pairs whose authenticity can be checked using a single certificate, issued in advance by a CA — throughout this paper we refer to this problem as (*).

To achieve the above, identity verification of nodes is done once by a CA from a fixed network during an initialization phase. After successful identity verification the CA issues a certificate to the node. Subsequently, nodes do not need to interact with the CA or any third entity in order to generate or validate public keys. The latter differentiates our solution from the previously proposed ones that run the identity-proving process in the fixed network as well. But, in these previous solutions nodes must access a third party, that can send updated revocation status information, in order to be able to validate public keys. However, as already discussed, such an entity is not always reachable. To handle revocation, our solution employs key pairs that are only valid for a short period of time. As is detailed in the main part of the paper, only the legitimate owner of a certificate is able to generate a valid key pair that is correctly bound to that certificate during its validity period. Because key pairs become automatically invalid after a short time (the shorter, the better), there is no need to distribute revocation information. Finally, nodes do not store any secret value, such as a passphrase or private key. Instead, private keys are generated after users have entered their passphrase, and the passphrase is immediately erased. The private key is also immediately erased after it has been used. This approach protects all devices against node compromise, even with no tamper-resistant hardware.

## 2.2. Issues to solve

Of course, if the key pairs are all pre-generated, and all the public keys are contained in the single certificate, then problem (*) would be solved. However, this would mean that the certificate could become very large, and would also impose a serious storage overhead on nodes. Moreover, in such a scheme, when a node is compromised, all the pre-generated public/private keys are readily usable. Thus, our goal is to generate a single certificate of regular size that proves the authenticity of a sequence of public keys.

One partial solution to this problem (due to Weimerskirch and Westhoff [5]) exploits the properties of one-way hash chains. The hash codes in a one-way hash chain are uniquely bound to a single 'final' hash code by a cryptographic (one-way) hash function. When this 'final' hash code is signed by a CA, hash codes in the chain are uniquely bound to that signature. In the Weimerskirch-Westhoff scheme, hash codes in the hash chain are treated as secret keys. Only the node that was issued the certificate can generate a key that is correctly linked to the hash code and signature. This solution allows nodes in ad hoc networks to prove the authenticity of public keys by disclosing keys from their one-way hash chains. However, this approach is subject to replay attacks — see Section 5.

If one-way hash chains were composed of public/private key pairs, and if use of a public key involved proving knowledge of the corresponding private key, then an attacker would not be able to build a successful attack by replaying a public key. Nodes would be able to prove the authenticity of their generated public key by proving that it is correctly linked to the hash code contained in their certificate. However, as they are currently defined, one-way hash chains do not permit the verifiable binding of public/private key pairs to a single hash code. We propose a scheme that overcomes this problem and that permits the generation of one-way hash chains of public/private keys which have similar properties to 'traditional' one-way hash chains [5, 6, 7]. These public/private key pairs are suitable for providing security services such as confidentiality, authentication, integrity, etc.

## 2.3. Requirements

The one-way hash chain scheme to be defined must fulfil the following requirements:

- It must uniquely link a series of public/private key pairs using a cryptographic hash function.
- These key pairs must be suitable for use with a discrete logarithm-based public key cryptosystem.
- It must be computationally infeasible to find key pairs that precede others in the chain.
- It must be possible to verify that a public key is linked to the last generated element of the chain by applying a cryptographic hash function the right number of times to a disclosed value.

## 3. Generating a one-way hash chain of public/private key pairs

In Section 3.1 we first propose a very simple scheme designed to meet the requirements defined in Section 2.3. This scheme is incrementally improved in Sections 3.2 and 3.3. Finally, we define a hash chain scheme that fulfils all the defined requirements and does not possess the limitations of the previously described schemes.

## 3.1. A very simple scheme

### *3.1.1. Specification*

We start by proposing a very simple scheme inspired by the original work of Merkle on hash-trees [8]. Prior to use of the scheme, certain system parameters need to be agreed by all parties, as follows.

- $\mathcal{G} = (G,*)$ is a finite cyclic group of order $q$ (for some large $q$), $g \in G$ is a generator of $\mathcal{G}$, and we assume that computing discrete logarithms in $\mathcal{G}$ with respect to $g$ is computationally infeasible. For example, $\mathcal{G}$ might be a large multiplicative subgroup of $Z_p^*$ for some large prime $p$, where $q$ is a large prime dividing $p-1$; alternatively $\mathcal{G}$ could be the group of points on an elliptic curve (usually written additively).
- $h$ is a cryptographic hash function mapping arbitrary length binary strings to strings of a fixed length $\ell$ (where a typical value for $\ell$ might be 224).
- $f$ is a cryptographic (one way) hash function mapping the set $\{0,1,\ldots,q-1\}$ onto itself. In practice, $f$ could be derived from $h$.
- $m \geq 1$ is a positive integer that determines the number of key pairs available to a node.

When the scheme is initialized, a node $A$ must first choose a secret $s \in \{0,1,\ldots,q-1\}$. $A$ generates a total of $m$ private keys, $K_i$, $0 \leq i < m$, as

$$K_i = f(s+i)$$

where $s+i$ is computed modulo $q$. The corresponding public key for $K_i$ is simply $g^{K_i}$ (where here, as throughout, we use multiplicative notation as a shorthand for the group operation). The proving node $A$, i.e. the node that is to be authenticated, then generates the check value $v$ as

$$v = h(h(g^{K_0})||h(g^{K_1})||\cdots||h(g^{K_{m-1}}))$$

where $g^{K_i}$ is converted to a bit string (by some means) prior to applying $h$, and here, as throughout, $||$ represents the concatenation operation.

Finally, $v$ is included in a certificate signed by some CA. Before proceeding, note that initializing the scheme potentially involves a significant amount of computation, especially if $m$ is large. This might be a major obstacle for very limited devices. However, we observe that these computations could be carried out for $A$ at the time of distribution of a device, e.g. by the same CA that generates the certificate containing $v$.

$A$ now stores the certificate, and also the values $h(g^{K_i})$, $0 \leq i < m$. $A$ also securely maintains the secret $s$. This can be based on a password or pass-phrase that is not stored by $A$, but is instead entered into $A$ whenever a private key $K_i$ needs to be generated, for some $i$.

During time interval $T_i$, $0 \leq i < m$, $A$ can use private key $K_i = f(s+i)$ and the corresponding public key $g^{K_i}$. To enable the verifying node $B$, i.e. the node that authenticates $A$, to verify this public key, the proving node $A$ sends it the values:

- $g^{K_i}$,
- $h(g^{K_j})$, $0 \leq j < m, j \neq i$, and
- the certificate containing $v$.

$B$ computes $h(g^{K_i})$, and combines it with the other hash values supplied by $A$ to compute:

$$v^* = h(h(g^{K_0})||h(g^{K_1})||\cdots||h(g^{K_{m-1}})).$$

and finally checks that

$$v = v^*.$$

### *3.1.2. Properties*

This scheme has the following desirable properties:

- $A$ only needs to retain a single secret value, $s$.
- The certificate only needs to contains a single $\ell$-bit hash code, $v$.
- Knowledge of one public key $g^{K_i}$ (and the information required to verify it) does not reveal any information about the other public keys.
- Knowledge of one private key $K_i$ does not reveal any information about the other private keys.
- Finally, although $A$ needs to have access to the values:

$$h(g^{K_0}), h(g^{K_1}), \ldots, h(g^{K_{m-1}}),$$

these do not need to be kept securely, since they are public values. Even if they are corrupted, there is no threat to the security of the system, (just loss of service).

### 3.1.3. Limitations

We note certain disadvantages of the simple scheme. In particular we note that, whenever a proving node wishes to allow a verifying node to have access to a verified public key, it is necessary to send $m-1$ hash codes from $A$ to $B$. For large $m$ this could be a significant communications overhead. We therefore propose, in the next section, a scheme which operates in a very similar way to the current scheme, except that the communications requirement is significantly reduced.

## 3.2. An improved simple scheme

### 3.2.1. Specification

In this scheme the system parameters are precisely as before, as is the method of generating private and public keys. The only difference is in the method used to generate the check-value $v$. For simplicity of presentation we suppose $m = 2^r$ for some integer $r$. We then compute $v$ using a Merkle hash-tree [8].

That is, we compute a binary tree of hash codes as follows.

- Let $H_{0,i} = h(g^{K_i}), \quad 0 \leq i \leq 2^r - 1 (= m - 1)$.
- For every $k$, $1 \leq k \leq r$, let:

$$H_{k,i} \quad = \quad h(H_{k-1,2i} || H_{k-1,2i+1})$$

  for $0 \leq i < 2^{r-k}$.
- Finally, let

$$v \quad = \quad H_{r,0}.$$

$A$ then stores: (a) a certificate containing $v$ (as before), and (b) the entire tree of hash codes $H_{i,j}$ (of which there are $2^r + 2^{r-1} + \cdots + 2^0 = 2^{r+1} - 1 = 2m - 1$). To enable $B$ to verify any one public key, $A$ sends the appropriate set of $r$ hash codes (together with the certificate containing $v$).

### 3.2.2. Properties

The modified scheme now reduces the communications cost from $m$ hash codes to $\log_2(m)$ hash codes. The only disadvantage is that it doubles the storage requirement for $A$.

### 3.2.3. Limitations

The schemes we have so far described have one common problem, namely the requirement to store $m$ (or nearly $2m$) $\ell$-bit hash codes. If $m$ is large, say $m = 2^{20}$, and $\ell = 256$, then this would require the proving node to store around 30 Megabytes of information. This could be a major disadvantage if the proving node has limited storage. We therefore define, in the next section, a scheme that does not require the proving node to store $m$ (or nearly $2m$) $\ell$-bit hash codes.

## 3.3. An exponentiation-based scheme

### 3.3.1. Specification

The system parameters are the same as in the previous two schemes. However, the key pairs and the check-value are derived differently, and one additional public parameter, which could be a system parameter, is required (if it is not a system parameter it could be included in $A$'s certificate). This is a value $t \in \{2, 3, \ldots, q-1\}$ with the property that $t$ has large multiplicative order modulo $q$. This could, for example, be arranged by choosing $q$ prime, and letting $t$ be a primitive element modulo $q$. There are also implementation advantages in choosing a small $t$, e.g. $t = 2$.

When the scheme is initialized $A$ chooses a secret $s \in \{0, 1, \ldots, q-1\}$. $A$ then generates the check value $v$ as

$$v \quad = \quad h(g^{st^m})$$

and $v$ is included in a certificate signed by some CA.

Subsequently, in time interval $T_i$, $0 \leq i < m$, $A$ uses as private key $st^{m-1-i}$, and as corresponding public key $g^{st^{m-1-i}}$. To enable a verifying node to obtain a verified copy of the public key, $A$ sends $B$:

- $g^{st^{m-1-i}}$, and
- the certificate containing $v$.

$B$ is assumed to know $i$ (because it is time-dependent) and can then compute

$$v^* = (g^{st^{m-1-i}})^{t^{i+1}} = g^{st^m}.$$

Finally, $B$ checks that $v = h(v^*)$.

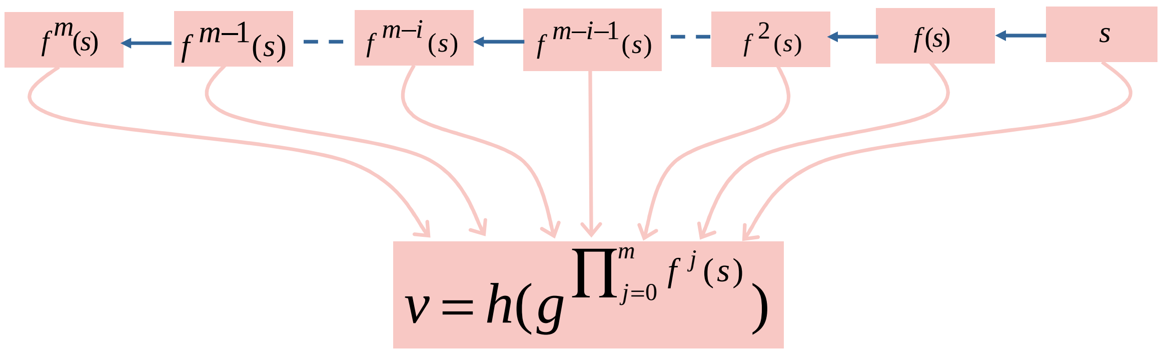


Fig. 1. Generation of the check value $v$

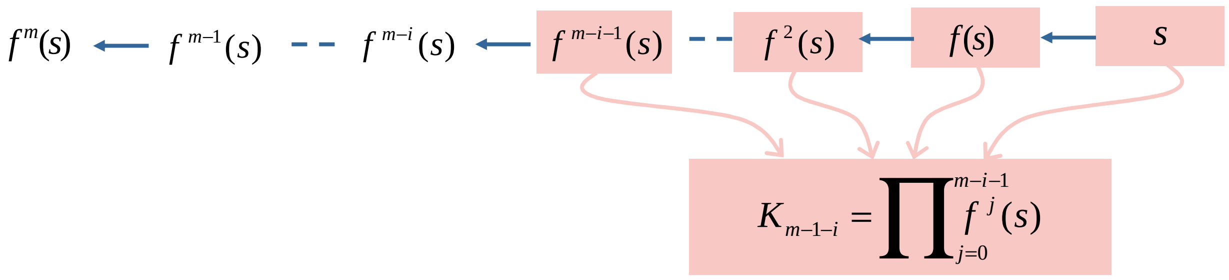


Fig. 2. Generation of the private key $K_{m-1-i}$

#### 3.3.2. Properties

This scheme has the following desirable properties:

- $A$ only needs to retain a single secret value, $s$.
- The certificate only needs to contains a single $\ell$-bit hash code, $v$.
- Knowledge of one public key $g^{st^{m-1-i}}$ does not reveal any information about future public keys.
- $A$ only needs to send to $B$ a public key and a certificate.

#### 3.3.3. Limitations

It is important to observe that some desirable properties have been lost. First, the public/private key pairs are not linked by a cryptographic hash function. Second, knowledge of one private key is sufficient to determine all the other private keys. The scheme we consider in the next section avoids this problem.

### 3.4. A hash chain scheme

#### 3.4.1. Specification

The system parameters are the same as in the previous scheme. However, the key pairs and the check-value are derived differently.

When the scheme is initialized, $A$ chooses a secret $s \in \{0, 1, \ldots, q-1\}$. As shown in Figure 1, $A$ then generates the check value $v$ as:

$$v \quad = \quad h(g^{\prod_{j=0}^{m} f^j(s)}). \tag{1}$$

The value $v$ is included in a certificate signed by a CA.

Subsequently, as shown in Figure 2, in time interval $T_i$, $0 \leq i < m$, $A$ uses as private key $K_{m-1-i}$, generated as follows:

$$K_{m-1-i} \quad = \quad \prod_{j=0}^{m-i-1} f^j(s) \tag{2}$$

and as corresponding public key $g^{K_{m-1-i}}$.

To enable $B$ to obtain a verified copy of the public key $g^{K_w}$, $A$ sends to $B$:

- $g^{K_w}$,
- $f^{w+1}(s)$, and
- the certificate containing $v$.

$B$ is assumed to know $w$ (because it is time-dependent) and can then compute

$$f^j(s), \quad w+1 \leq j \leq m.$$

Using these values and, as shown in Figure 3, $B$ then computes $v^*$:

$$\begin{aligned} v^* &= (g^{K_w})^{\prod_{j=w+1}^{m} f^j(s)} \\ &= g^{\prod_{j=0}^{w} f^j(s) \prod_{j=w+1}^{m} f^j(s)} \\ &= g^{\prod_{j=0}^{m} f^j(s)}. \end{aligned} \tag{3}$$

Finally, $B$ checks that $v = h(v^*)$.

### 3.4.2. Properties

This scheme has the following desirable properties:

- $A$ only needs to retain a single secret value, $s$.
- The certificate only needs to contains a single $\ell$-bit hash code, $v$.
- Knowledge of one private key $K_i$ does not on its own reveal any information about the other private keys.
- Knowledge of one public key $g^{K_w}$ does not reveal any information about future public keys.
- $A$ only needs to send to $B$ a public key and a certificate.

Finally note that certain optimizations in the public key verification process are possible if the verifying node caches a trusted copy of a past public key of the proving node.

Note that all except the exponentiation-based scheme comply with the requirements defined in Section 2.3 — see Table II. However, the hash chain scheme combines advantages of all the other schemes, without having their shortcomings — see Tables I and III. Therefore, this scheme is the preferred solution and is used in the remainder of the paper. Its inherent computational complexity is studied in Section 6.

## 4. Use case

We now present a use case for the hash chain solution of the previous section. We show how nodes can use the scheme to generate public keys that can be authenticated with a unique certificate issued by a CA. We also propose a solution that permits the generation of certificates whose authenticity can be verified by checking a signature that was generated by the CA at network initialization.

### 4.1. Registering in the fixed network

As explained in [9], before issuing certificates for use in providing entity authentication or key establishment, a CA must first run an identity-proving process during which the identity of the certificate requester is verified. In ad hoc networks, we cannot assume that an administration authority will be present that can run the identity-proving process. Hence, in our solution, certificates are issued by a CA situated in the fixed network — see Figure 4(a). This CA may, for example, be the network provider which a principal registers with in order to have his communications conveyed through a fixed network. The registration process at a network provider often requires principals to present valid paper credentials. These credentials can be used by the network provider to authenticate a principal prior to generating a certificate. The detailed authentication process is as follows.

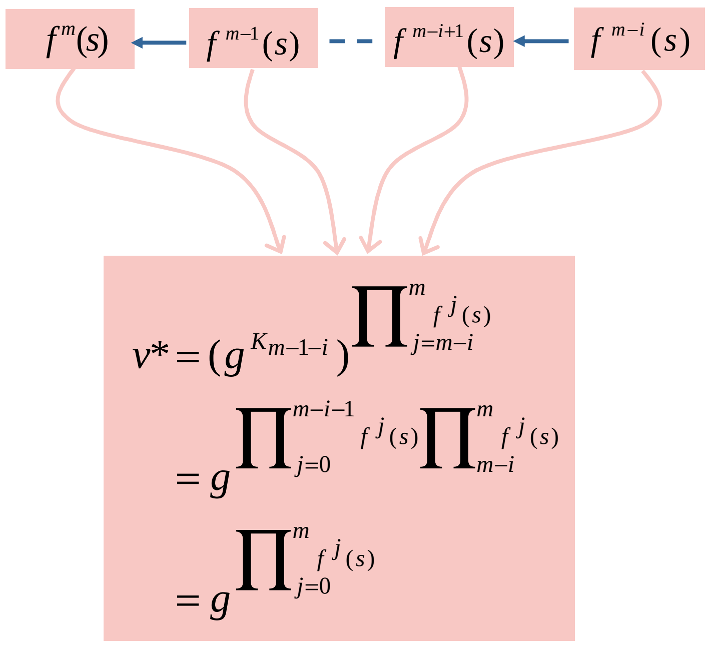
When a node $A$ is in the fixed network, it contacts a CA in order to synchronize its clock, and obtain a reliable copy of the system parameters, as defined in Section 3.1, and $K_{CA}$, the CA's public key. The CA sends the same parameters to all the entities that request them. When $A$ receives these parameters, it chooses a secret key $s$. Note that $s$ is not stored by $A$; instead, it is generated from a strong passphrase. Then $A$ generates the check value $v$, as defined in (1).

$A$ then sends its identity $ID_A$, the check value $v$ and the hash code $f^{m+1}(s)$ to the CA, in order to obtain a certificate that binds these values. The CA next verifies that $A$ really owns $ID_A$ and that no certificate has been issued that already contains $v$ and $f^{m+1}(s)$. If all the verifications succeed, then CA sends to $A$ the certificate $Cert_A$:

$$[ID_A,\ v,\ f^{m+1}(s),\ t_0,\ L,\ nb\_keys]_{K_{CA}^{-1}} \tag{4}$$

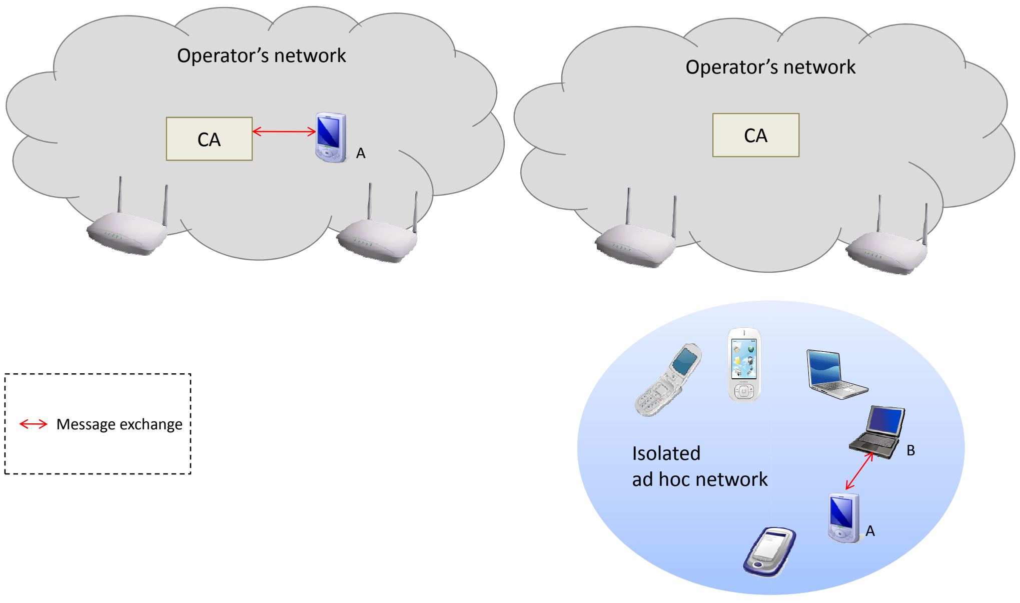
where $t_0$ is the issue time of the certificate, $L$ is an integer that may be specified by $A$ in the certificate request or by the CA, $nb\_keys$ are as defined in Section 4.4 below, and $[\ldots]_{K_{CA}^{-1}}$ denotes a signature generated by the CA using its private key $K_{CA}^{-1}$. Once $A$ has received its certificate it does not need to access the CA again.

$A$ divides time into intervals of equal length $L$. Each interval is assigned a key that is generated as defined



$f^m(s)$
$f^{m-1}(s)$
$f^{m-i+1}(s)$
$f^{m-i}(s)$
$v^* = (g^{K_{m-1-i}})^{\prod_{j=m-i}^{m} f^j(s)}$
$= g^{\prod_{j=0}^{m-i-1} f^j(s) \prod_{m-i}^{m} f^j(s)}$
$= g^{\prod_{j=0}^{m} f^j(s)}$

Fig. 3. Generation of $v^*$



(a) Certificate issue during registration in the fixed network
(b) Entity authentication in the ad hoc network

Fig. 4. From registration to entity authentication

in (2). The mapping between keys and time intervals is shown diagrammatically in Figure 5.

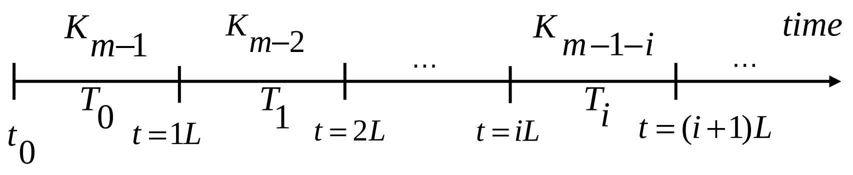


Fig. 5. Mapping between private keys and time intervals

## 4.2. Establishing secure communication channels in ad hoc networks

During time interval $T_i$, $0 \leq i < m$, i.e. between times $t = iL$ and $t = (i+1)L$, $A$ can be authenticated by $B$ and both can establish an authenticated session key with each other using the protocol we now describe. $A$ first sends $B$:

- $g^{K_w}$,
- $f^{w+1}(s)$, and
- $Cert_A$.

When it receives these values, $B$ first verifies the CA's signature on $Cert_A$. If it is valid, $B$ determines $i$ from its local time, the value of $L$ in the certificate, and the issue time of $Cert_A$. $B$ then verifies that $f^{w+1}(s)$ belongs to the same one-way hash chain as $f^{m+1}(s)$. If so, $B$ generates $v^*$, as described in Section 3.4 by (3). Finally, $B$ checks whether $v = h(v^*)$. If the verifications are successful, $B$ knows that $g^{K_w}$ was generated by the entity that was issued the certificate $Cert_A$ by the CA. $B$ now sends its currently valid public key $g^{K_y}$ to $A$. $A$ can verify the validity of $B$'s key using the same method as just described. After $A$ has validated $g^{K_y}$, $A$ and $B$ will share the following authenticated Diffie-Hellman session key $SK$:

$$\begin{aligned} SK &= (g^{K_w})^{K_y} \\ &= (g^{K_y})^{K_w} \\ &= g^{K_w K_y}. \end{aligned}$$

$SK$ can be used by $A$ and $B$ to derive shared secret session keys for mutual authentication, confidentiality, integrity, etc. The message exchange used to establish $SK$ is summarized in Figure 6. $A$ does not store $SK$ and $K_w$ but generates them when they have to be used and erases them after they have been used. Only $g^{K_y}$ is stored by $A$ during time interval $T_i$. When $T_i$ is elapsed $g^{K_y}$ becomes invalid and can be erased by $A$.

The scheme can also be used with an asymmetric cryptosystem to support public key encryption and digital signature. $A$ and $B$ may negotiate the

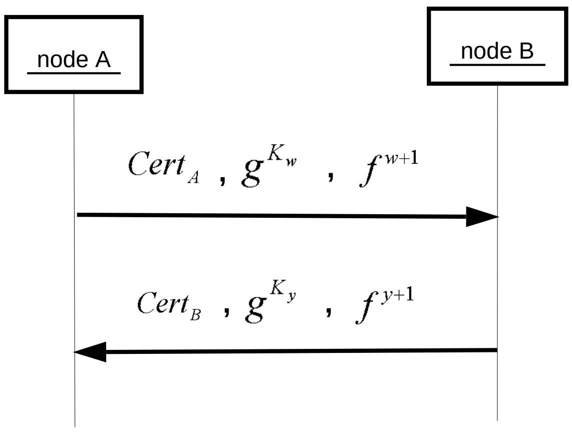

node A
node B
$Cert_A$ , $g^{K_w}$ , $f^{w+1}$
$Cert_B$ , $g^{K_y}$ , $f^{y+1}$

Fig. 6. Messages exchanged by $A$ and $B$ to establish the authenticated session key $SK$

cryptosystem to be used when they exchange their public values.

The information $g^{K_w}$ and $f^{w+1}(s)$ released by $A$ during the time interval $T_i$ does not permit an attacker to establish an authenticated session key with $B$ or to impersonate $A$. To do so, the attacker would need to discover $K_w$ from $g^{K_w}$ during the interval $T_i$ (of length $L$). However, this is computationally infeasible [10]. Moreover, since only $A$, that knows the correct passphrase, can generate $s$, only $A$ can establish a valid session key $SK$ with $B$ during time interval $T_i$.

## 4.3. Generating certificates in ad hoc networks

The scheme defined in Section 3.4 requires nodes to generate new key pairs during each time interval. This may not be convenient when, for instance, a secure exchange is expected to last longer than one time interval. To deal with that case, we now define a solution that permits nodes to generate public key certificates with a very short lifetime that, nevertheless, last longer than one time interval. This solution is designed for situations where a node $A$ knows or is able to evaluate how long a secure exchange with a node $B$ will last. Therefore, node

$A$ is able to specify in its certificate an expiry time that restricts the validity of that certificates to the suitable time period. $A$ can also reduce the use of that certificate to a specific context by, for instance, adding in the certificate a field containing $B$'s identity. That way, $B$ automatically considers that $A$'s certificate is invalid as soon as the secure interaction is terminated. It also makes the certificate invalid when an attacker tries to use it at node other than $B$. The authenticity of these certificates can be verified by checking that they are correctly linked to a signature generated by the CA at initialization. Our solution works as follows.

During time interval $T_i$, $A$ generates a public/private key pair $Pub_A/Priv_A$, and also generates the certificate $Cert2_A$ for $Pub_A$, with the following content:

- $\left[ID_A,\ v,\ f^{m+1}(s),\ t_0,\ L,\ nb\_keys\right]_{K_{CA}^{-1}}$, i.e. the certificate $Cert_A$ that was issued by the CA at initialization;
- $Pub_A$;
- *$B$'s identity*
- *issue time*;
- *expiry time*;
- A signature generated on the previous fields using the private key $K_{m-1-i}$ for this time interval.

The private key $K_{m-1-i}$ used to sign $Cert2_A$ is generated as defined in (2). After signing the certificate, $A$ deletes $K_{m-1-i}$ and does not generate it again. This prevents any attacker from being able to generate a valid certificate.

Then, to be authenticated by $B$, $A$ sends $Cert2_A$ along with $g^{K_w}$ and $f^{w+1}$ to $B$. If the certificates contains his identity, $B$ then verifies that the *expiry time* has not passed and that $\left[ID_A,\ v,\ f^{m+1}(s),\ t_0,\ L,\ nb\_keys\right]$ was signed by the CA. If so, $B$ uses $t_0$, $L$ and *issue time* to determine the current time interval at $A$. $B$ then checks the authenticity of $g^{K_w}$ as described in Section 3.4 using knowledge of the time interval and $f^{w+1}$. If the verification succeeds, then $B$ verifies the signature generated by $A$ on $Cert2_A$. Finally, if all the verifications succeed, $B$ knows that $Cert2_A$ was generated by the node that was issued the certificate $\left[ID_A,\ v,\ f^{m+1}(s),\ t_0,\ L,\ nb\_keys\right]_{K_{CA}^{-1}}$ by the CA. $B$ then considers $Pub_A$ as valid. The certificate $Cert2_A$ must be validated in $T_i$ and can then be used until the expiry time of the certificate has passed.

In the case where $A$ stores $Pub_A/Priv_A$, an attacker may get the knowledge of this public/private key pair. However, since $s$ is not stored on the device, the attacker is not able to generate a new valid certificate for the key pair. Then, if $A$ has already terminated an interaction with $B$, $B$ considers $Cert2_A$ as invalid. Therefore, the attacker cannot impersonate $A$. However, if $A$ has not yet terminated an interaction with $B$, $B$ still considers $Cert2_A$ as valid and the attacker can impersonate $A$ at $B$ until the expiry time of the certificate $Cert2_A$ has passed. In this latter case, $A$ can need to inform $B$ that $Cert2_A$ must be considered as invalid. To achieve this $A$ can send a revocation statement signed with $Priv_A$. However, as previously discussed, the frequently-changing topology of an ad hoc network makes that $B$ may not receive that revocation statement. An alternative approach requires $A$ not to store $Priv_A$, but to generate it when needed. As for the key $K_{m-1-i}$, $Priv_A$ should be erased immediately after it is used. As a result, an attacker is not able to learn $Priv_A$. Hence, an attacker is not able to impersonate $A$ by proving ownership of the public/private key pair $Pub_A/Priv_A$ to $B$.

It is important to note that the revocation discussion does not concern the private key $K_{m-1-i}$ that is never stored and that is only used once to sign the certificate $Cert2_A$. It is also important to notice that if the key pair $Pub_A/Priv_A$ is stored, our solution significantly reduces the number of nodes that need to receive a revocation statement — compared to existing solutions. It also restricts the need for revocation to the short period of time when the certificate $Cert2_A$ is valid.

## 4.4. Dealing with time synchronization problems

In the above schemes, $A$, $B$ and the CA must have synchronized clocks. This is achieved as part of the initialization phase, when $A$ and $B$ are connected to the fixed network. However, subsequently their clocks will run independently of the CA and the fixed network. Clock drift between $A$ and $B$'s clocks will then mean that, when $A$ and $B$ are in the ad hoc network, their clocks will no longer be precisely synchronized. This can make that $A$'s current time interval at $B$ is different from $A$'s current time interval. The previous implies that the value of $i$, computed by $B$ in order to verify that the disclosed public key is properly linked to $v$, differs from the correct value of $i$ that indeed permits to verify this linkage. Therefore, the authentication of $A$ by $B$ always fails in this case.

In order to be authenticated by $B$ in this context, a solution is proposed that requires $A$ to choose a

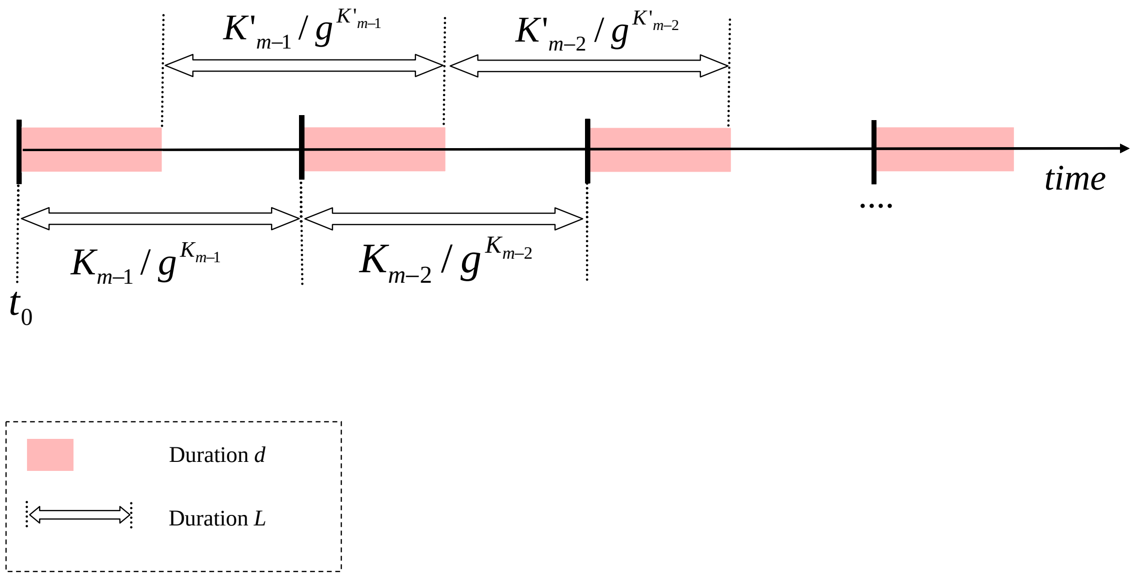


Fig. 7. Taking into account the maximum authorized clock drift $d$ in the mapping between keys and time intervals

parameter $d$ to represent the maximum disparity that it tolerates between its clock and $B$'s clock. Then, during each time interval, $A$ can disclose two public keys whose validity period starting time is separated by $d$, as represented in Figure 7. Each of the disclosed public keys has a validity time $L$. Therefore, the proposed solution allows $B$ to use a valid public key during a time period $(L + d)$, i.e. during a time period increased by $d$ compared to the case where only one public key is disclosed by $A$. The Figure 8 illustrates how the proposed solution makes it possible for $B$ to authenticate $A$ as long as the clock drift between their clock is bigger than 0 and less than or equal to $d$. In that figure, when $A$ is in the first time interval, it sends to $B$ the public key $g^{K_{m-1}}$, whose validity period goes from time $t = 0$ to time $t = 1L$, and the public key $g^{K'_{m-1}}$, whose validity period goes from time $t = d$ to time $t = L + d$. Therefore, the proposed solution guarantees that at least one of the disclosed public keys allows $B$ to authenticate $A$ and to establish an authenticated session key with $A$ in each time interval. If more than one key must be disclosed during a time interval, then the CA can specify it in the certificate using the parameter $nb\_keys$.

Compared to the case where $A$ discloses one public key in each time interval, the proposed solution increases by a factor $nb\_keys$ the number of computations that $A$ has to perform in order to generate the public keys that it discloses to $B$ in each time interval $L$. Therefore, the choice for the value $nb\_keys$ must be a trade-off between computational costs and the practicability of taking into account large clock drifts. If the use of multiple keys in each time interval do impacts of the computational cost of the solution at $A$, it does not increases the computational costs at $B$, as $B$ only needs to validate one public key, among the two disclosed one, whose validity period covers $A$'s current time at $B$. It may be suitable to add, in the message that $A$ sends to $B$, in order to be authenticated, some fields making it possible for $B$ to easily identify the validity period covered by each of the disclosed public keys.

**Remark** The computational cost of the key generation and validation solution is discussed in Section 6.

The security of the schemes given in Section 3 increases when $L$ becomes shorter. However, the shorter $L$ is, the more precisely nodes need to be synchronized. Thus the choice for the value of $L$ must be a trade-off between these two influences. Certain other constraints also apply to the value of $d$:

- If $d$ is greater than $L$, then the solution defined previously does not guarantee that an authenticated session key can always be

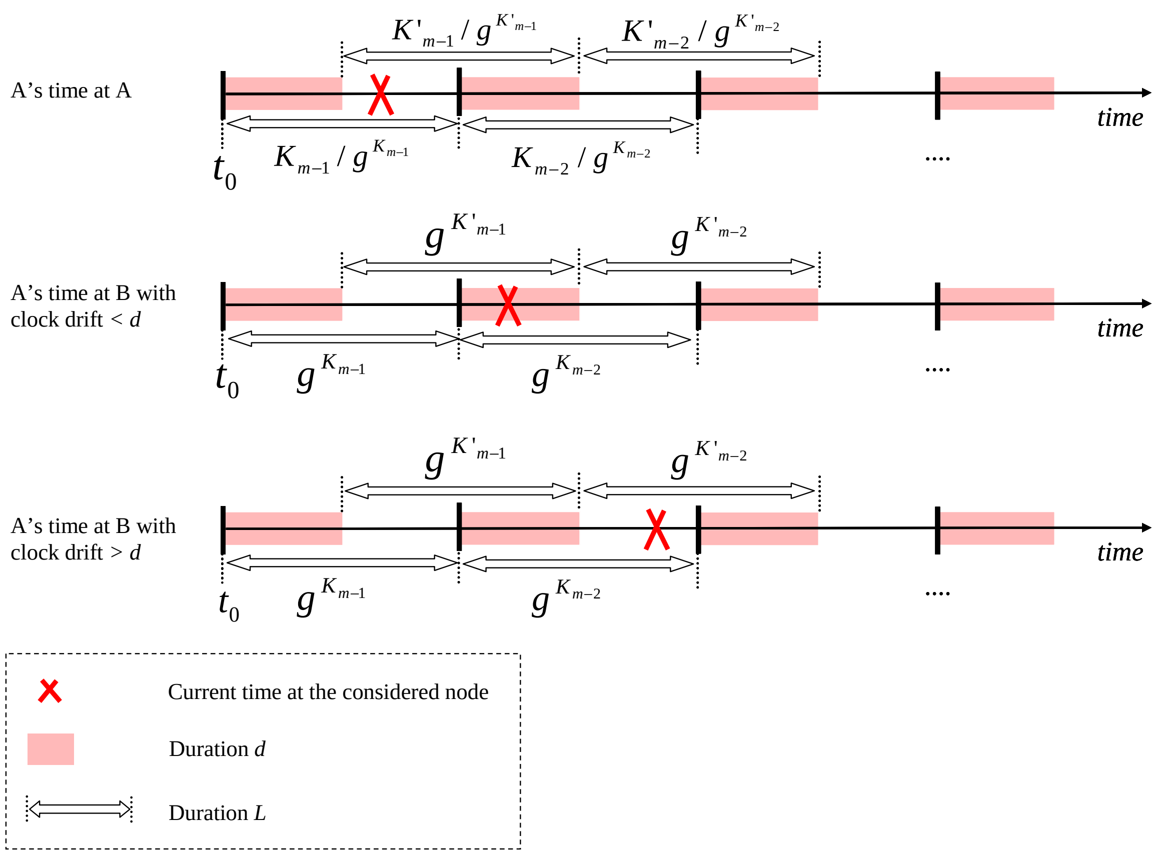

A's time at A
A's time at B with clock drift < d
A's time at B with clock drift > d
$K'_{m-1} / g^{K'_{m-1}}$
$K'_{m-2} / g^{K'_{m-2}}$
$K_{m-1} / g^{K_{m-1}}$
$K_{m-2} / g^{K_{m-2}}$
$g^{K'_{m-1}}$
$g^{K'_{m-2}}$
$g^{K_{m-1}}$
$g^{K_{m-2}}$
$t_0$
time
....
Current time at the considered node
Duration d
Duration L

Fig. 8. Effects of clock drift between $A$ and $B$

established between $A$ and $B$, if a large disparity exists between their clocks. This, because the validity period of the two public keys disclosed by $A$ can have validity period that do not necessarily cover $A$'s current time at $B$, as represented in Figure 9;

- If $d$ is smaller than $L$, then the solution defined previously permits the establishment of an authenticated session key between $A$ and $B$ even when a disparity exists between their clocks. However, if $d$ is too small, then $A$ may not be able to establish an authenticated session key with all nodes with which it wants to exchange confidential messages.

Therefore, $d$ must be smaller than $L$ and must be chosen by $A$ in such a way that $d$ is as close as possible to the average existing difference between its clock and the clocks of other nodes in the network.

## 5. Related work

We structure the related work section following the key issue identified in the introduction, i.e. identity verification, certificate revocation and node compromise. Before going into detail we elaborate on the related work in the area of hash chains and hash chain based authentication.

### 5.1. Hash chains

One-way hash chains were introduced by Lamport [6]. The initial application was to prevent eavesdroppers from impersonating a client by replaying intercepted passwords to a server. A one-way hash chain is

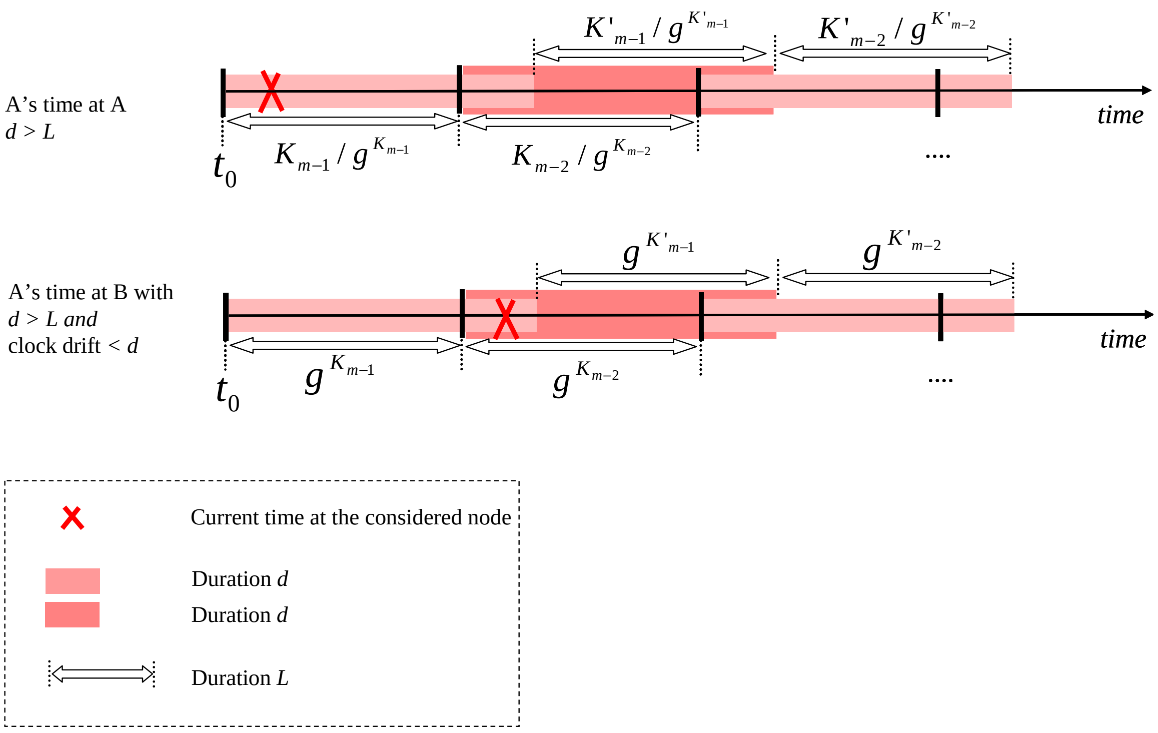


Fig. 9. Effects of clock drift between $A$ and $B$ when $d > L$

generated by a client by applying a cryptographic hash function $f$ repeatedly to its secret $s$. Each element in the chain is a different password that the client can use at each new authentication. The passwords are linked to $s$ and to each other by:

$$\begin{aligned} K_i &= f^i(s), \\ K_{i+1} &= f(K_i), \\ K_j &= f^{j-i}(K_i). \end{aligned} \tag{5}$$

where $0 \leq i \leq m$, $1 \leq j \leq m$, $i < j$, and $f^j(K_i)$ means that the cryptographic hash function $f$ is applied $j$ times to $K_i$. For the $i$th authentication to the server the client uses the password $K_{m-i}$. Basically, our hash chain scheme builds on this scheme although the elements of the hash chain are different and it does not share the following drawback. The server authenticates the client by verifying that $K_{m-i}$ is linked to $K_{m-i+1}$, i.e. the password used at the previous authentication, by the relation defined in (5). Since cryptographic hash functions are computationally hard to invert, an attacker is not able to generate a valid password from a compromised one. However, as it is well known (see, for example, [11]) this scheme is subject to server impersonation attacks. On the other hand, the solution proposed in this paper is not subject to these attacks because the client never discloses to the server a key that alone permits the server to impersonate the client towards another third party.

The TESLA broadcast authentication protocol [7] relies on the previous scheme to authenticate sources of broadcast packets. However, the solution does not provide the means to bootstrap entity authentication. This must be provided by a regular data authentication system at session setup [7, 12]. But, as discussed later in this section, regular data authentication systems require a trusted party - e.g. a TTP or an entity known in advance - to be always reachable in order to allow the establishment of shared secret keys or the verification of certificates' validity. However, the foregoing cannot be guaranteed in ad hoc networks. Our solution does not have this limitation as it does not require nodes to interact with a third party in order to be able to authenticate and establish shared secret keys.

A solution to this bootstrapping problem is proposed in [5], where a node has a number of certificates issued by a CA. Each certificate binds a node's identity to the commitment (the most recently generated key) of its one-way hash chain. More specifically, $A$ chooses a secret key $x_0$ that is the anchor of the chain. Given a secure cryptographic hash function $h$, a system parameter $m \geq 1$, and $x_i = h^i(x_0)$ $(i \geq 0)$, then the last generated hash key $x_{2m}$ from $A$'s chain is contained in $A$'s certificate. This certificate also contains the issue time and a random seed. $A$ divides time into intervals of equal length $L$, where $L$ is public, and assigns two keys of its one-way hash chain to each interval. During interval $T_i$ $(0 \leq i < m)$, between times $t = iL$ and $t = (i+1)L$, $A$ uses keys $x_{2m-2i-1}$ and $x_{2m-2i-2}$[†] from its chain to enable $B$ to authenticate $A$. $B$ can verify the correctness of these two values by first verifying the certificate to get a trusted copy of $x_{2m}$, and then iterating the cryptographic hash function the appropriate number[‡] of times on the disclosed keys and comparing the result with the value $x_{2m}$. However, once $x_{2m-2i-1}$ and $x_{2m-2i-2}$ are disclosed, an attacker can replay them to an entity $C$ to pretend it is $A$. To reduce this risk, $B$ must ask $A$ to use a certificate with a specific seed value. As a consequence, an attacker can only replay previously disclosed keys if it is asked by a verifying entity to present a certificate containing a seed already requested during the current time period or during the previous one. The scheme has the following interesting properties:

- Nodes only need to hold a single secret ($x_0$);
- Only a single public value needs to be included in the certificate;
- Knowledge of the keys released in previous time intervals does not reveal any information about keys used in later time intervals;

However, since keys must be disclosed to perform entity authentication, they cannot be used to establish a secure communications channel. This is a shortcoming that is overcome by our scheme as in our solution keys (from the hash chain) are not directly disclosed but instead used to generate public keys that are disclosed to perform entity authentication.

When they are used in ad hoc networks, existing hash chain based authentication solutions all have the limitations discussed in the foregoing. To the best of our knowledge, our solution is the only one that has been proposed and that does not have these limitations.

## 5.2. Identity verification

The goal of a certificate is to prove to a verifying principal that the binding between an identity and a key is authentic. It is achieved with the signature that a TTP or an introducer generates on a certificate if it contains a valid "identity-key" binding. By successfully verifying the validity of this signature, a verifying principal gains confidence that the certificate is authentic. To permit the authenticity of certificates to be verified in ad hoc networks three approaches have been proposed. Tounsi et al. [13], Sanzgiri et al. [14] and Weimerskirch and Westhoff [5] propose solutions that rely on certificates issued in fixed networks. This permits certificates to be signed by some CAs that cannot be accessed in ad hoc networks. However, it does not permit nodes to be issued new certificates when they are in ad hoc networks, although this can be necessary. This is for instance the case when anonymity has to be provided [15]. This is an issue that is addressed by our solution as we discussed in Section 4.3.

In order to permit certificates to be issued in ad hoc networks, a proposed approach has been to adapt PKI. Zhou and Haas [16], for instance, define a solution that exploits threshold cryptography [17, 18] and that proposes to distribute the issue of certificates to $n$ special nodes, called servers. However, even if there are more servers in the network than the number to contact in order to obtain a full signature, the dynamics of nodes do not guarantee that enough servers are always reachable. Luo et al. [19] as well as Kong et al. [20] solve this problem by distributing the CA's services to all nodes in the network without relying on an online TTP. However, as will be seen further below, these solutions have other limitations.

An alternative approach is to adapt PGP since PGP does not rely on a TTP to issue certificates. This is the approach used by Capkun et al. in [21] and Li et al. in [22]. In [21], for instance, it is considered that nodes know their neighbors and always issue them certificates with valid user-key binding. All nodes that have signed a certificate are assumed to be honest and are trusted in the same way. However, there is no guarantee that nodes that are known in advance are always present in ad hoc networks, nor that nodes always behave properly. PGP does not rely on such an assumption. Known principals are not all trusted to

[†]These are sent with the certificate containing $x_{2m}$.
[‡]The number of times the cryptographic hash function has to be applied to the disclosed values is time dependent. Indeed, since all entities are synchronized with the CA that issued their certificates, that number can be determined from the certificate issue time, the current local time, the number of keys disclosed in each time interval, and $L$.

properly validate “identity-key” bindings. Trust levels are used that differentiate fully trusted introducers from less trusted introducers. Furthermore, a trust depth is specified that defines whether the trust given to a principal can be propagated to the principals that this principal trusts. All these issues are avoided by our proposed solution.

Some solutions such as [16, 23, 24] do not explain how identities are validated at registration. Our solution does not require trust in any node in the ad hoc network. Without any validation, it cannot be guaranteed that a secure communication channels is established with the intended node when using a non-validated certificate or non-validated long term secret key. Luo et al. tackle this problem in [19] by relying on human perception. However, if human perception is used in ad hoc networks, then only nodes operated by individuals that know each other are able to authenticate. Similarly, Stajano [25] proposes a solution that relies on a physical and electronic contact to transfer the bits of the secret key from one node to another one. The imprinting has to be performed before entering the ad hoc network. Securely transmitting the bits of a secret key does not permit binding of a secret to a valid principal’s identity.

Luo et al. alternatively propose in [19] to rely on biometrics. However, the use of biometrics to validate identities also introduces concerns as during the enrollment process biometric characteristic has to be collected as a template [26]. The biometric templates of all principals that will have to be authenticated must be stored on the node. This imposes storage requirements on the node that are avoided by the proposed solution.

Deng et al. [27] and Bohio and Miri [28] propose the use of identity based encryption (IBE). IBE was first introduced by Shamir in [29] but the first efficient IBE scheme was defined by Boneh and Franklin in [30]. It is a public key cryptosystem where public keys are arbitrary strings, which represent their owners’ identities. The initial motivation for IBE was to eliminate the need for directories and certificates [31]. However, the private key generator (PKG), like the CA in a PKI, is a central entity whose access cannot be guaranteed in ad hoc networks. Therefore, the issue of identity based private keys in ad hoc networks present the same limitations as the issue of certificates in ad hoc networks.

The above analysis shows that solutions previously proposed to validate identities and to issue certificates in ad hoc networks introduce issues that remain unsolved but which are addressed by our proposed solution.

## 5.3. Certificate revocation

Some solutions such as [16, 32] do not define any method for managing revocation. Three approaches have been proposed in the literature to manage revocation and the distribution of revocation information in ad hoc networks. In this section we discuss these approaches. Time based revocation as allowed by IBE reduces the need for revocation. However, it does not necessarily make it disappear.

For example, if the validity period is rather long, e.g. a year, then the emitter may need to revoke its public key before the end of the year. In that case, the need for revocation is the same as that when normal certificates are used. But when the validity period is very short, then the need for revocation can be significantly reduced as in the solution discussed in this paper.

Another approach, used in [13, 14, 20, 21, 22], relies on the emission of revocation statements, the storage of these statements by receiving nodes into CRLs and the exchange of these CRLs between nodes to spread the list of revoked certificates in the network. However, possible partitions in the network can prevent nodes from updating their CRLs.

Some solutions rely on advice from other nodes to evaluate the validity of a certificate. The solution proposed in [19], for example, uses the monitoring of neighbors to detect misbehaving nodes whose certificates need to be revoked. This approach relies on the interpretation that nodes have of other nodes’ behavior. Therefore, this approach can lead to situations where falsely detected nodes are isolated from the network.

In [33, 34] an approach is used that is based on a trust evaluation mechanism which permits a node from a given cluster — a group of nodes — to authenticate a node from another cluster. In that approach, a certificate is deemed valid as soon as there is a majority of nodes that have sent positive recommendations for it. But since no solution is provided to permit nodes to verify the revocation status of a certificate, those nodes cannot know whether a certificate has been revoked or not. Therefore, the fact that a majority of nodes vouch for a certificate cannot prove its validity.

In summary, none of the approaches identified in this section and proposed in the literature guarantees that nodes in ad hoc networks never use certificates that have been revoked. Our approach addresses the revocation implicitly as certificates and key pairs are short-lived. A node stops using a public key until a new key pair becomes available.

### 5.4. Node compromise

Note that of the above mentioned schemes, only that of Raya and Hubaux [15] takes into account the lack of physical protection of mobile devices by explicitly protecting the public/private key pairs using tamper-resistant hardware. However, as previously noted, such hardware is not available in every mobile device. We deal with this problem as the secret used to generate the hash chain is never stored on the mobile device but is erased after it is being used.

## 6. Performance evaluation

### 6.1. Evaluation parameters

In order to evaluate the performances of our solution, that relies on the hash chain scheme, we implemented the hash chain scheme in Java and tested it on a PC as well as a mobile phone. The PC had the following system parameters:

- A 3.40 GHz Pentium 4 CPU;
- 1 GB RAM;
- Windows XP operating system.

The mobile phone had the following configuration:

- A 130 DMIPS ARM7 CPU;
- 80 MB memory;
- Symbian operating System 7.0S platform with J2ME Personal Profile.

The experimentation aimed at identifying whether our solution can be used for authentication and signature generation and verification.

We measured the maximum time needed by the devices to:

- Generate the private key $K_{m-1}$ when $m$ increases;
- Generate the public key $g^{K_{m-1}}$ from $K_{m-1}$ when $m$ increases;
- Generate a digital signature with $K_{m-1}$ when $m$ increases;
- Verify a digital signature with $g^{K_{m-1}}$ when $m$ increases.

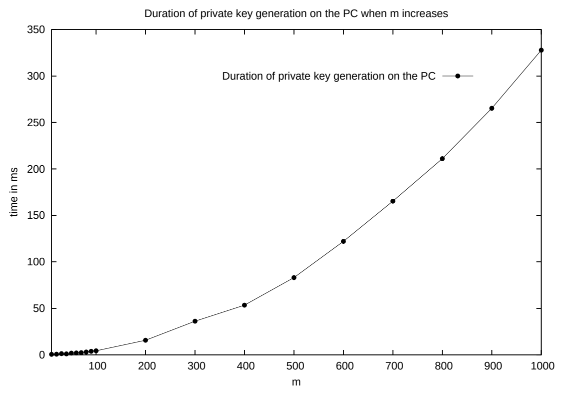


Fig. 10. Duration of private key generation on the PC

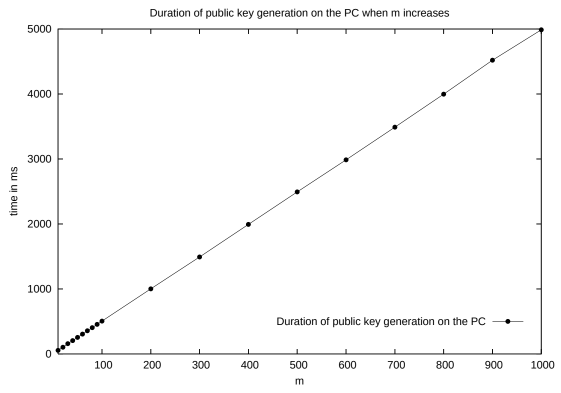


Fig. 11. Duration of public key generation on the PC

We have not measured the time needed to verify a public key because this can be deduced from the time needed to generate a public key.

The parameters used for our experimentation are given in Table IV[§].

### 6.2. Results and comments

#### *6.2.1. Duration of public and private key generation*

[§]To simplify the evaluation, we have chosen the same hash function for $f$ and $h$

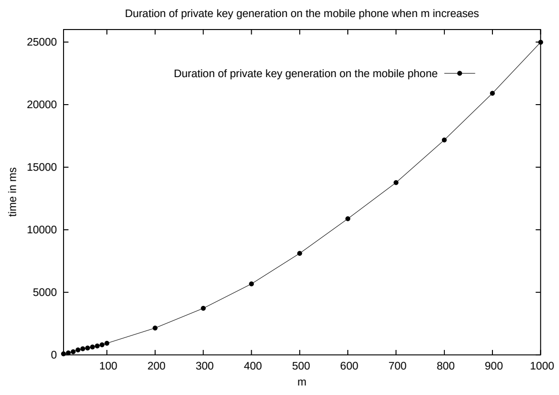


Fig. 12. Duration of private key generation on the mobile phone

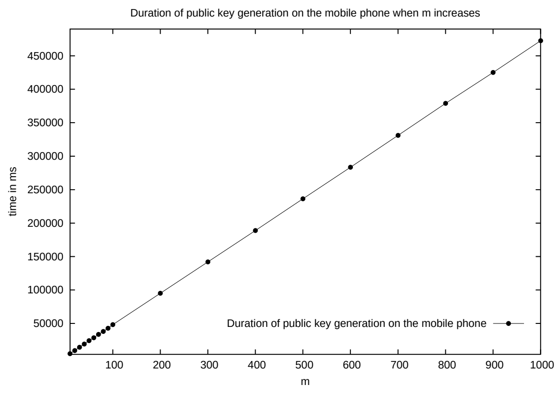


Fig. 13. Duration of public key generation on the mobile phone

The generation of the private key is faster than the generation of the public key (see Figures 10 – 13 for the respective figures for the PC and the mobile phone). This is explained by the fact that the multiplication operation is less expensive than the exponentiation operation.

With much more CPU and memory resources than the mobile phone, the time needed by the PC to generate keys goes:

- From 0.5 ms to 327 ms for the private key;
- From 55 ms to 5 s for the public key.

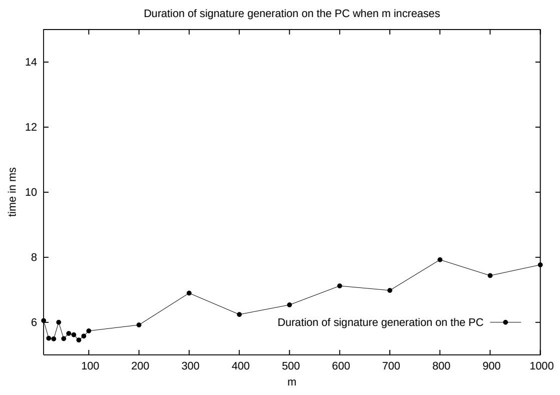


Fig. 14. Duration of signature generation on the PC

while the time needed by the mobile phone to generate keys goes:

- From 78 ms to 25 s for the private key;
- From 4.9 s to 472.6 s for the public key.

These results show that our solution can be used on any PC. They also show that our solution can be used on a mobile phone provided a suitable value is chosen for $m$. $m$ must be defined based on the lifetime of the certificate issued by the CA and $L$. If, for instance, the CA issues an initial certificate that has a lifetime of one year and if node security policies require new key pairs to be used every week, then defining $m = 52$ is sufficient.

Our results show that for $m = 60$, the time needed by the mobile phone to generate the keys are:

- 547 ms for the private key;
- 28.7 s for the public key.

Public key generation on the mobile phone is quite long (i.e. 28.7 s). However, the newest generation of mobile phones are equipped with CPUs offering much higher performance; for instance, the ARM11 MPCore CPU [35] delivers up to 2600 DMIPS, i.e. 8 times more powerful than the mobile phone we used for our evaluation. This allows us to conclude that key generation and verification will be much faster with newer mobile phones.

## 6.3. Signature generation and verification

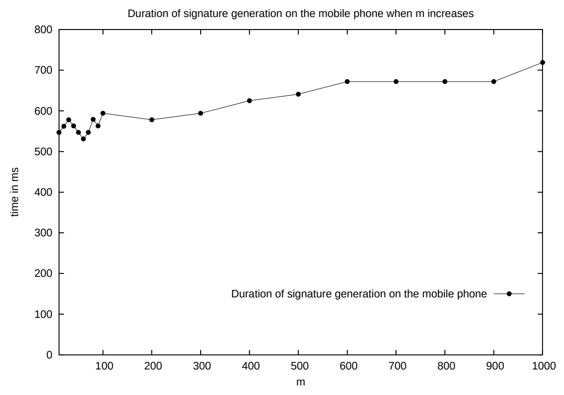


Fig. 15. Duration of signature generation on the mobile phone

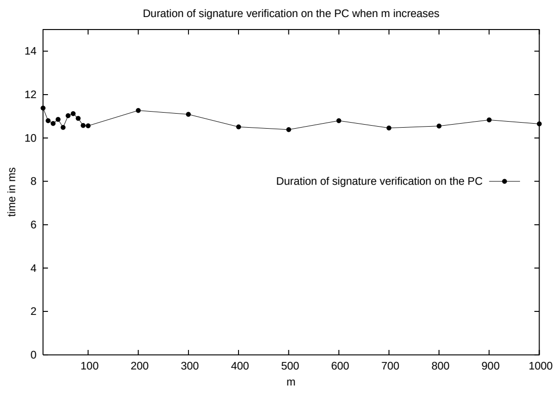


Fig. 16. Duration of the signature verification on the PC

The time needed for signature generation and verification is important, since nodes may need to generate a digital signature to be authenticated or may need to verify a digital signature to authenticate other nodes. As shown in Figures 14 and 15, the time needed for signature generation increases slowly when $m$ increases. It takes less than 9 ms on the PC while it takes less than 750 ms on the mobile phone. The time needed for signature verification is almost stable as $m$ increases. It takes less than 12 ms on the PC, while it takes less than 2s on the mobile phone. These results show that key pairs generated with our solution can be used for signature generation and verification on a PC as well as on a mobile phone.

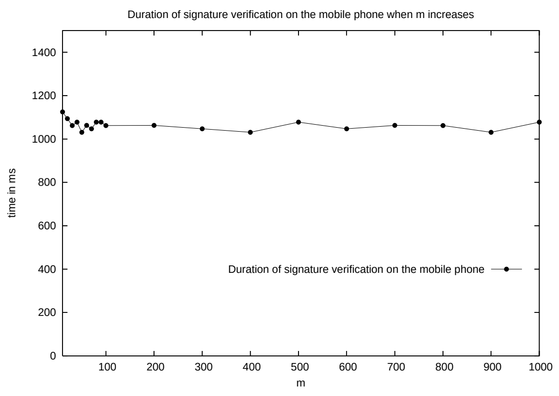


Fig. 17. Duration of the signature verification on the mobile phone

## 7. Summary and Conclusions

In an ad hoc networks, nodes may need to generate new public keys. At the same time, because of the absence of a permanent communication infrastructure, a CA that can issue certificates for these public keys may not be reachable. This can prevent the establishment of secure communication channels. Previously proposed solutions do not guarantee that identities contained in certificates are valid or, when they do, they either require that nodes store many certificates in advance or rely on neighbors to validate user-key bindings. However, the first approach may require a large storage space to be available, while, in the second approach, neighbors are not always able to verify a node's identity before certificate issue.

In this paper we have defined a solution that permits nodes in ad hoc networks to generate, on-demand, a number of public/private key pairs and certificates without relying on their neighbors and without having to generate and store in advance all the keys and certificates that they may use in the future. The solution relies on a novel scheme that permits the verifiable binding of public/private key pairs to a single hash code, which existing solutions do not permit. Our solution has the nice property that public/private key pairs change over time. A hash chain is used to generate private keys and, from them, derive public keys. The verification of public keys is simple, and only requires the proving node to disclose the currently valid public key as well as a hash code. Using these disclosed values, the verifying node is able to assess the validity of the public key by linking it to the check value included in the proving node's certificate. Moreover, as shown by our performance evaluation, the approach is feasible for computationally limited devices such as mobile phones.

The security of the hash chain scheme depends on the protection of the secret value $s$. Given that $s$ is never stored on the system, but is generated from a passphrase, biometric trait or other user-specific means when needed, then a compromised node can only be misused until the current public/private key pair expires, in the case where these private keys are stored on the system. When they are not, a compromised node can only be misused if the attacker succeeds in breaking a public key. However, because of the cryptographic properties of the basic elements used, including hash functions, hash chains and how public keys are derived, breaking either keys or hash functions is not possible (i.e. computationally infeasible).

## References

1. Housley R, Polk W, Ford W, Solo D. Internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile. RFC 3280 April 2002.
2. Myers M, Ankney R, Malpani A, Galperin S, Adams C. X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. RFC 2560 June 1999.
3. Papadimitratos P, Haas ZJ. Proceedings of the scs communication networks and distributed systems modeling and simulation conference (cnds 2002). 2002; 193–204.
4. Hubaux JP, Buttyán L, Capkun S. The quest for security in mobile ad hoc networks 2001.
5. Weimerskirch A, Westhoff D. Identity certified authentication for ad–hoc networks. *SASN '03: Proceedings of the 1st ACM workshop on Security of ad hoc and sensor networks*, ACM Press, New York, NY: New York, NY, USA, 2003; 33–40, doi: http://doi.acm.org/10.1145/986858.986864.
6. Lamport L. Password authentication with insecure communication. *Commun. ACM* 1981; **24**(11):770–772, doi:http://doi.acm.org/10.1145/358790.358797.
7. Perrig A, Canetti R, Tygar J, Song D. The TESLA Broadcast Authentication Protocol. *CryptoBytes* 2002; **5**(2):2–13.
8. Merkle RC. A Digital Signature Based on a Conventional Encryption Function. *Advances in Cryptology — CRYPTO '87*, *Lecture Notes in Computer Science*, vol. 293, Pomerance C (ed.), Springer-Verlag, Berlin, 1988; 369–378.
9. Ellison C, Schneier B. Ten Risks of PKI: What You're Not Being Told About Public Key Infrastructure. *Computer Security Journal* 2000; **16**(1):1–7. URL http://www.schneier.com/paper-pki.pdf, accessed the 26 September 2009.
10. Diffie W, Hellman ME. New Directions in Cryptography. *IEEE Transactions on Information Theory* 1976; **IT–22**(6):644–654.
11. Mitchell CJ, Chen L. Comments on the S/KEY user authentication scheme. *ACM Operating Systems Review* October 1996; **30**(4):12–16.
12. Perrig A, Song D, Canetti R, Tygar JD, Briscoe B. Timed Efficient Stream Loss-Tolerant Authentication (TESLA): Multicast Source Authentication Transform Introduction. RFC 4082 June 2005.
13. Tounsi M, Hamdi M, Boudriga N. A public key–based authentication framework for multi–hop ad hoc networks. *12th IEEE Mediterranean Electrotechnical Conference (MELECON 2004)*, vol. 2, 2004; 775–778.
14. Sanzgiri K, LaFlamme D, Dahill B, Levine B, Shields C, Belding-Royer E. Authenticated routing for ad hoc networks. *IEEE Journal on Selected Areas in Communications*, vol. 23, 2005; 598–610.
15. Raya M, Hubaux JP. The security of vehicular ad hoc networks. *SASN '05: Proceedings of the 3rd ACM workshop on Security of ad hoc and sensor networks*, ACM Press: New York, NY, USA, 2005; 11–21, doi:http://doi.acm.org/10.1145/1102219.1102223.
16. Zhou L, Haas ZJ. Securing ad hoc networks. *IEEE Network* 1999; **13**(6):24–30, doi:http://dx.doi.org/10.1109/65.806983.
17. Desmedt YG. Threshold Cryptography. *European Transactions on Telecommunications*, vol. 5, 1994; 449–457.
18. Desmedt YG, Frankel Y. Threshold Cryptosystems. *CRYPTO '89: Proceedings of the 9th Annual International Cryptology Conference on Advances in Cryptology*, Springer-Verlag: London, UK, 1990; 307–315.
19. Luo H, Zerfos P, Kong J, Lu S, Zhang L. Self-securing ad hoc wireless networks 2002; :567–574.
20. Kong J, Zerfos P, Luo H, Lu S, Zhang L. Providing Robust and Ubiquitous Security Support for Mobile Ad Hoc Networks. *ICNP '01: Proceedings of the Ninth International Conference on Network Protocols*, IEEE Computer Society: Washington, DC, USA, 2001; 251.
21. Capkun S, Buttyán L, Hubaux JP. Self-organized public-key management for mobile ad hoc networks. *IEEE Transactions on Mobile Computing* 2003; **2**(1):52–64, doi:http://dx.doi.org/10.1109/TMC.2003.1195151.
22. Li R, Li J, Kameda H, Liu P. Localized public-key management for mobile ad hoc networks. *In proceedings of the Global Telecommunications Conference 2004*, vol. 2, 2004; 1284–1289.
23. Pirzada AA, McDonald C. Kerberos assisted authentication in mobile ad-hoc networks. *ACSC '04: Proceedings of the 27th Australasian conference on Computer science*, Australian Computer Society, Inc.: Darlinghurst, Australia, Australia, 2004; 41–46.
24. Tsai YR, Wang SJ. Routing security and authentication mechanism for mobile ad hoc networks. *In proceedings of the IEEE 60th Vehicular Technology Conference*, vol. 7, 2004.
25. Stajano F. The Resurrecting Duckling – What Next? *Revised Papers from the 8th International Workshop on Security Protocols*, Springer–Verlag: London, UK, 2001; 204–214.
26. Newton EM, Woodward JD. Biometrics: A technical primer. *RAND*, 2001.
27. Deng H, Mukherjee A, Agrawal DP. Threshold and Identity-based Key Management and Authentication for Wireless Ad Hoc Networks. *ITCC '04: Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'04) Volume 2*, IEEE Computer Society: Washington, DC, USA, 2004; 107.
28. Bohio MJ, Miri A. An Authenticated Broadcasting Scheme for Wireless Ad Hoc Network. *CNSR '04: Proceedings of the Second Annual Conference on Communication Networks and Services Research (CNSR'04)*, IEEE Computer Society: Washington, DC, USA, 2004; 69–74.
29. Shamir A. Identity–based cryptosystems and signature schemes. *Proceedings of CRYPTO 84 on Advances in*

*cryptology*, Springer–Verlag New York, Inc.: New York, NY, USA, 1985; 47–53.

30. Boneh D, Franklin M. Identity–Based Encryption from the Weil pairing. *SIAM Journal of Computing*, vol. 32, 2003; 586–615.
31. Gagné M. Identity-Based Encryption: a Survey. *RSA Laboratories CryptoBytes*, vol. 6, 2003; 10–19.
32. Weimerskirch A, Thonet G. A distributed light–weight authentication model for ad–hoc networks. *ICISC '01: Proceedings of the 4th International Conference Seoul on Information Security and Cryptology*, Springer–Verlag: London, UK, 2002; 341–354.
33. Ngai E, Lyu M. Trust- and Clustering-Based Authentication Services in Mobile Ad Hoc Networks. *ICDCS Workshops*, 2004; 582–587.
34. Ngai E, Lyu M, Chin RT. An authentication service against dishonest users in mobile ad hoc networks. *Proceedings of the 2004 IEEE Aerospace Conference*, vol. 2, 2004; 1275–1285.
35. ARM11 MPCore. URL `http://www.arm.com/products/CPUs/ARM11MPCoreMultiprocessor.html`, accessed the 21 May 2008.

Table I. Advantages of the schemes

| Schemes | Advantages |
|---|---|
| Very simple scheme | • Only a single secret value needs to be retained;<br>• The certificate only needs to contains a single $\ell$-bit hash code, $v$;<br>• Knowledge of one public key does not reveal any information about the other public keys;<br>• Knowledge of one private key does not reveal any information about the other private keys. |
| Improved simple scheme | Same advantages as the very simple scheme. Compared to the very simple scheme, the communication cost is reduced from $m$ hash codes to $\log_2(m)$ hash codes. |
| Exponentiation-based scheme | • Only a single secret value needs to be retained;<br>• The certificate only needs to contains a single $\ell$-bit hash code, $v$;<br>• Knowledge of one public key does not reveal any information about future public keys;<br>• Only a public key and a certificate need to be sent to a communicating party. |
| Hash chain scheme | Same advantages as the exponentiation-based scheme. But compared to the exponentiation-based scheme, the hash chain scheme additionally meets the requirement that knowledge of one private key does not on its own enable the computation of any other private keys. |

Table II. Fulfillment of requirements

| **Property** | **Very simple scheme** | **Improved simple scheme** | **Exponentiation-based scheme** | **Hash chain scheme** |
|---|---|---|---|---|
| Single secret value | Yes | Yes | Yes | Yes |
| Linking key pairs by hash function | Yes | Yes | No | Yes |
| Discrete logarithm-based public key cryptosystem | Yes | Yes | Yes | Yes |
| Unlinked private keys | Yes | Yes | No | Yes |
| Verification of public key against hash code | Yes | Yes | Yes | Yes |

Table III. Shortcomings of the proposed schemes

| **Schemes** | **Shortcomings** |
|---|---|
| Very simple scheme | $m - 1$ hash codes must be published in order to verify the validity of one public key. |
| Improved simple scheme | Compared to the very simple scheme, it doubles the storage requirements for the entity that is authenticated. |
| Exponentiation-based scheme | • Public/private key pairs are not linked by a cryptographic hash function. • Knowledge of one private key is sufficient to determine all the other private keys. |

Table IV. Parameters used for the experimentation

| **Parameter type** | **Value** |
|---|---|
| Public key length | 1024 bits |
| Signature algorithm | DSA |
| one-way hash function $f$ | SHA-1 |
| one-way hash function $h$ | SHA-1 |
| $m$ minimum value | 10 |
| $m$ maximum value | 1000 |
| $s$ length | 160 bits |