# Zero Common-Knowledge Authentication for Pervasive Networks

André Weimerskirch[1] and Dirk Westhoff[2]

[1] Communication Security Group, Ruhr-University Bochum, Germany
weika@crypto.rub.de
[2] NEC Europe Ltd. - Network Laboratories, Heidelberg, Germany
dirk.westhoff@ccrle.nec.de

**Abstract.** Ad-hoc networks and even more intrinsic pervasive networks face huge security lacks. In the most general case entities need to build up a well-defined security association without any pre-established secret or common security infrastructure. Under these circumstances it turns out that without unrealistic assumptions authentication of previously unknown parties is not achievable. However, for a wide spectrum of scenarios much weaker authentication forms are reasonable, e.g., for routing protocols and other protocols aiming to intensify cooperation. Like in real world when foreign subjects meet for the very first time, inferring the opposites identity is impossible. Nevertheless even from this zero common-knowledge status some minor level of trust establishment is possible for both scenarios, in real world and on a technical level. In this paper we will present a very light-weight still provably secure authentication protocol not aiming at inferring the involved entities' identities but re-recognizing foreign communication partners whenever necessary. We do not make any assumptions to the scenario, and we also have no requirements for the devices' abilities. For the technical realization we propose extremely efficient security primitives applicable for nearly all types of restricted devices. Our solution is more efficient than a public-key operation by some orders of magnitude.

**Keywords:** Authentication, Pervasive Networks, Ad-hoc Networks, Key-Chains, Public-Key, Symmetric Ciphers

## 1 Introduction

Wireless ad-hoc networks and pervasive networks face huge security lacks. In the most general case entities need to be able to establish well-defined security associations without any pre-established secret or common security infrastructure. Unlike in military or closed networks where there is a single logical and commonly agreed trust authority we cannot assume such a situation in the general case. For example, assume two entities that have a certificate issued by two different authorities. In an ad-hoc network with entities belonging to different administrative domains there might be no pre-defined association between these

M. Matsui and R. Zuccherato (Eds.): SAC 2003, LNCS 3006, pp. 73–87, 2004.

two authorities, or the nodes might not be able to reach the authorities. In both cases entities cannot easily establish a trust relationship.

In this paper we follow a fully infrastructure-less approach of establishing trust relationships in pervasive networks that are highly dynamic. We define that a network is of *pure* kind (or just pure) if there exist neither central services nor does a fixed infrastructure exist, and if there are no pre-established knowledge between two entities in a general manner. We do not assume that the network has a special topology, specifically it might be a wireless multi-hop network that is exposed to several attacks such as a man-in-the-middle attack. We define a *weak device* to be a simple device without any further assumptions. Such assumptions would be tamper resistance, or a unique identification number. We assume the most general case in meaning that there is no common trusted third party (TTP), and the ad-hoc network is indeed pure.

We define our security objective, which we call *zero common-knowledge authentication (ZCK)* as follows: $A$ is able to authenticate $B$ in a zero common-knowledge fashion if $A$ is able to identify again the authority that runs $B$, i.e., $B$ is able to convince $A$ that both had some relationship in the past. We also say that $A$ *recognizes* $B$, or that $B$ *authenticates* to $A$. For example, if $B$ was forwarding a data-packet for $A$, then later on $A$ is able to recognize $B$ to forward another packet, probably in exchange for another service. Now it is clear what we want to achieve: ZCK authentication in a pure pervasive network consisting of weak devices where there are no pre-established secrets, i.e., we do not use any assumptions like tamper resistant devices, devices that are able to perform expensive public-key operations, special access structures for deploying a distributed PKI, and so on. We believe that such an approach is more practical and realistic, yet still sufficient. Certainly, we are not able to determine a user's identity as it is necessary for financial transactions. However, in many cases like the financial one the problem of authentication does not differ to the problem in traditional networks and can be solved with schemes similar to the ones that are used in today's Internet. Clearly, this involves further assumptions or infrastructure. Our approach focuses on problems that are inherent to pure pervasive and ad-hoc networks. For example, our solution is suited to cooperation and motivation based schemes [7][3][9] as well as secure routing methods. These schemes require a large number of authentication steps for which ZCK authentication is strong enough. Our scheme also makes sense for all kinds of client-server and peer-to-peer relations, respectively.

A limited authentication based on re-recognition is reasonable. We cannot achieve as much as other authentication schemes, but we can keep our promises without any assumptions. Especially for pure ad-hoc and pervasive networks this is a realistic approach. Furthermore, in a pure network we believe that we cannot promise anything more than our objective. We formulate this as follows:

*Claim.* Under the assumption that there exists no public-key scheme that works without any pre-established knowledge[1] or without a common TTP, recognition is the best we can achieve in a pure network.

We cannot provide a formal proof but only intuitive arguments. If we were able to achieve more than ZCK authentication in a pure network, i.e., identify a user instead of recognizing him, then we had a public-key scheme that works without any pre-established knowledge or TTP. Today this seems to be impossible though.

Imagine two foreigners that meet in the real world. Usually, and obviously strongly related to the considered level and type of interaction these people do not doubtlessly identify each other, e.g., by showing their passports. Assuming there is nobody else available to ask about the opposite's reputation the best both can do is to establish step-by-step a trust relationship build on their bilateral, personal experiences. To do so, it is mandatory that these two people recognize again the next time they meet. In our model we do not assume geographical proximity though.

The main contribution of this paper is a new authentication scheme especially suited to pervasive networks. The scheme only requires one-way hash-functions and is suited to nearly any computationally weak device since it is more efficient than any public-key scheme by some order of magnitudes. Our scheme provides provably secure authentication against passive adversaries and secure message authentication against active adversaries, where the scheme is as sound as the underlying one-way hash function. We also present solutions for slightly different application scopes. The paper is organized as follows. Section 2 gives a description of our ZCK authentication scheme in a general framework. Section 3 introduces our new scheme. Section 4 compares the different instantiations of our general authentication scheme and rates them by comparing their complexity and by giving advise for which scope they are applicable. Finally, the last section concludes this work.

## 2 General ZCK Authentication Scheme

In this section we describe our scheme in a general kind. Consider the case where an entity $A$ wants to be able to recognize an entity $B$ after an initial contact. Obviously the communication channel they use is insecure. In this scenario, $B$ might be a service provider and $A$ the client. In the first step $B$ provides $A$ with some data that allows the later one to recognize $B$. Let $S$ be a set of secrets, and $x \in S$ be a secret. $B$ is able to prove that it knows $x$ in such a way that $A$ is not able to impersonate $B$. Furthermore $B$ is able to use $x$ in such a way that $A$ can verify that a message origins of $B$, i.e., $B$ is able to perform a message authentication by using the key $x$. Let $(m)_x$ be an authentication of a message $m$, then $A$ can verify the origin by checking $((m)_x)_{f(x)} = m$,

[1] Note that we consider a unique global and tamper resistant ID for each device as pre-established knowledge.

where $f$ is a mapping on $S$ such that the knowledge of $f(x)$ enables a prover to check if a verifier knows $x$, i.e., $f(x)$ is a public key. A simple example is a private/public-key pair of a digital signature scheme. The protocol runs as follows. The entity $B$ sends the public key $f(x)$ to $A$. Then $A$ and $B$ perform a challenge-response protocol as described in [8] such that $A$ is able to verify that $B$ knows the secret $x$. A simplified version of the protocol looks as follows:

1 : $B$ `generates` $x \in S$ `at random`
2 : $B$ `sends` $f(x)$ `to` $A$
`Repeat Steps 3 to 5 for each authentication process`
3 : $A$ `sends random` $r$ `to` $B$
4 : $B$ `sends authenticated` $(r)_x$ `to` $A$
5 : $A$ `checks if` $((r)_x)_{f(x)} = r$
~ `If 'yes',` $A$ *`accepts`*`, otherwise she` *`rejects`*

**Remarks:**

- Step 1 only needs to be performed once for each authority and/or entity. An authority might hold several devices such that each device can hold the same secret $x$ or that each device has its own secret $x_i$.
- Step 2 needs to be performed once for each communication pair $A$ and $B$.
- Steps $3-5$ have to be performed for each authentication process.
- Step 4 usually consists of a message block $r', (r, r')_x$ with random $r'$ to avoid chosen-text attacks. Since this is no threat to a ZCK scheme we omit it here.

We consider as main objective of this scheme the capability to ensure that entities are able to re-recognize another entity in order to receive a service they requested. Hence the public key $f(x)$ always has to be send together with the offered service, i.e., service and key have to be bound together to avoid that a malicious entity can inject his public key to a service that he did not offer at all. It follows that the authentication scheme we are envisioning here usually is connected to some offered service, i.e., to messages that are exchanged. Therefore we add the following steps to *authenticate messages* that replace Steps $3-5$:

`Repeat Steps 3 to 4 for each message to authenticate`
$3'$ : $B$ `sends` $(m)_x$ `to` $A$
$4'$ : $A$ `checks if` $((m)_x)_{f(x)} = m$
~ `If 'yes',` $A$ *`accepts`*`, otherwise she` *`rejects`*

Note that message integrity and also freshness, i.e. that the data is recent and not replayed, comes with the message authentication. In contrast to PKI scenarios where there is a logical central certificate directory, $A$ has to store $B$'s public key (together with $B$'s $ID$ string) to be able to recognize $B$. After $A$ deletes $B$'s public key from her memory $A$ is not able anymore to build a connection to a previous relationship with $B$. Note that in many applications a mutual authentication process is required. The above protocol can easily be extended for this case.

We now consider the traditional security objectives, namely authentication, confidentiality, integrity, and non-repudiation [8]. The above scheme ensures

ZCK authentication. It does not allow a key-exchange by itself, hence it cannot establish confidentiality. However, it is possible to establish integrity of messages by authenticating these messages as shown above. Obviously the scheme does not provide non-repudiation. However, for some type of scenario a weaker form of non-repudiation may also be appropriate. We define *ZCK non-repudiation* to be the service which prevents an entity to deny a commitment or action chain. In our case this means that an entity $A$ is able to prove to a third party that a number of actions or commitments were taken by the same (probably unknown) entity $B$. Obviously a scheme that provides signatures satisfies the ZCK non-repudiation objective.

The presented protocol is as secure as the underlying security scheme, i.e., to break the protocol an adversary has to construct an authenticated message $(m)_x$ for given $m$ and $f(x)$. Note that a man-in-the-middle attack is always possible, and consider what this means at an example. There is an entity $B$ that offers a service, an entity $A$ that seeks this service, and a malicious entity $M$. The entity $B$ sends $f(x_B)$ to $A$. $M$ interrupts this message, and sends $f(x_M)$ to $A$. Then $M$ satisfies the needs of $A$ by offering its services. All that $A$ is interested in was that service. She does not care who provided the service, but she wants to receive the service in the same quality again. It becomes now clear that a man-in-the-middle attack is no threat to our protocol.

Two instantiations of the ZCK authentication scheme that immediately arise are based on traditional signature schemes as well as symmetric ciphers. Authentication is done by the proof of knowledge of the secret key in a challenge and response manner. The verifier sends a challenge $r$, and the prover signs the challenge or computes a message authentication code (MAC) of $r$. As we argued before, a man-in-the-middle attack is possible but irrelevant in the sense of our intention. In the case of digital signatures the scheme ensures ZCK authentication and ZCK non-repudiation, and also message authentication. In most cases public-key operations overstrain the capabilities of weak devices. Hence for our scope of weak devices, only very efficient public-key schemes such as NTRU [6] are appropriate, while RSA and even ECC might be too slow and too resource consuming.

In the case of a symmetric cipher a secret key $s$ has to be shared a priori, i.e., the protocol requires a secret channel to exchange the shared secret. It is suited for applications where devices are geographically close, e.g., where devices can exchange the keys by infrared channel, or where there is only a single trust authority. If the key $s$ is a $t$-bit string, and an authenticated message $(m)_s$ is a $t$-bit string, then the protocol requires storage of $t/8$ bytes and the exchange of $2t/8$ bytes which are split into two messages. For a security level similar to 1024-bit RSA we choose $t = 80$. A protocol run then requires the exchange of 20 bytes in two messages, where the key has a size of 10 bytes. Clearly, a secure channel for key-exchange could also be established by a secure key-exchange such as Diffie-Hellman. Such a key-exchange has to be performed for each communication pair. Since a key-exchange demands computationally powerful devices and the

heavily changing topology of pervasive networks of mobile nodes induces many key-exchanges such a solution is not suited to weak devices.

## 3 Key-Chain Scheme

This section introduces our new protocol for ZCK authentication that only requires one-way hash functions but no expensive public-key operations. Doing so our scheme is extremely efficient and by orders of magnitudes faster than any public-key scheme. Again Bob wants to authenticate to Alice. Let $x_B^G$ be a $t$-bit string which is Bob's global private key. Let each device have a network identifier $ID$[2], in our case $ID_A$ and $ID_B$. Let $f$ be a function that maps the identity $ID$ to a bit-string, $r_{ID}$ be $t$-bit random strings, and $\oplus$ be an operation on bit strings. Then $x_0^B$ is Bob's private key for the communication with Alice which is derived from Bob's global key, Alice's identity and $r_B$ such that $x_0^B = x_B^G \oplus f(ID_A) \oplus r_B$. Likewise $x_0^A = x_A^G \oplus f(ID_B) \oplus r_A$ is Alice's private key for the communication with Bob. The private key for the communication with an entity having the identifier $ID$ can later again be derived by the stored associated string $r_{ID}$ and the global private key. Note that these keys are only applicable to the communication pair Alice and Bob, and to no one else.

We define a hash chain, which is also known as Lamport's hash chain [8], as $h(x_i) = x_{i+1}$ with $x_0$ being the anchor and $h$ being an unkeyed one-way hash function that has a security of $t$-bits, i.e., which maps bit-strings to $t$ bits. Our scheme is based on such hash chains with $x_0^A$ and $x_0^B$ being the anchors. Let $x_{n_A}^A$ and $x_{n_B}^B$ be the final elements of the hash chains, respectively. We call these the public keys of Alice and Bob. We can use a key-value of the chain to generate an authenticated message by a MAC (keyed hash function). Let $(m)_x$ be the MAC of a message $m$ by the key $x$. The core idea of our protocol is as follows: First exchange a value $f(x)$ which the receiver will tie together with some experience. Then prove knowledge of the pre-image of $f(x)$, i.e. $x$, in order to authenticate by establishing a relationship to $f(x)$ and the past experience. Since we want to be able to repeat the authentication step arbitrary many times we propose the use of a key-chain based on a one-way hash function. Our protocol works as follows:

[2] For the protocol this is an identifier to recognize another entity again. We do neither assume that the $ID$ cannot be tampered with nor that $ID$s are unique. $A$ just has to be able to map $B$ to some name $ID_B$.

1 : $A$ sends her public key $x^A_{n_A}$ to $B$, who stores $(x^A_{n_A}, r_B, n_B, 1)$
2 : $B$ sends his public key $x^B_{n_B}$ to $A$, who stores $(x^B_{n_B}, r_A, n_A, 1)$
Repeat Steps 3 to 9 for each authentication process
3 : Assume $A$ stores $(x^B_i, r_A, j, u)$ and $B$ stores $(x^A_j, r_B, i, v)$
4 : (+) $A$ sends authenticated messages $(m)_{x^A_{j-u-1}}$ to $B$
5 : (+) $B$ sends authenticated messages $(m)_{x^B_{i-v}}$ to $A$
6 : $A$ opens her key by sending $x^A_{j-1}$ to $B$.
7 : $B$ checks if $h(x^A_{j-1}) = x^A_j$
8 : For $k = 1$ to $k' \leftarrow \max\{u, v\}$ repeat Steps 8.1 to 8.5
8.1 : $B$ opens his key by sending $x^B_{i-k}$ to $A$
8.2 : $A$ checks if $h(x^B_{i-k}) = x^B_{i-k+1}$
8.3 : $A$ opens her key by sending $x^A_{j-k-1}$ to $B$
8.4 : $B$ checks if $h(x^A_{j-k-1}) = x^A_{j-k}$
8.5 : If any check fails, or the loop is interrupted, $A$ and $B$ stop execution. Then $A$ stores $(x^B_i, r_A, j, \max\{u, k+1\})$ and $B$ stores $(x^A_j, r_B, i, \max\{v, k+1\})$.
9 : $A$ and $B$ store the new values $(x^B_{i-k'}, r_A, j-k'-1, 1)$ and $(x^A_{j-k'-1}, r_B, i-k', 1)$

**Remarks:**

- Steps $1-2$ ensure the exchange of public-keys which is done only once per pair $A$ and $B$.
- Steps $3-9$ are done for each authentication process.
- For each authentication process we assume that $A$ stores $B$'s key $x^B_i$ and that $B$ stores $A$'s key $x^A_j$ (Step 3). Furthermore $A$ and $B$ store information to compute the subsequent keys, i.e., they store the random value $r_A$ and $r_B$ to compute the chain anchor from the global key $x^G_A$ and $x^G_B$, respectively. Furthermore they store $i$ and $j$ to obtain the chain value from the anchor. In the initialization, i.e., after Steps 1-2, $A$ and $B$ store tuples $(x^B_{n_B}, r_A, n_A, 1)$ and $(x^A_{n_A}, r_B, n_B, 1)$. These are replaced later by updated tuples.
- Steps 4 and 5 provide messages authentication and are optional. The messages are guaranteed to be fresh.
- The messages sent in Steps $4-5$ can be read but are not authenticated at this moment. It has to be ensured that all messages are received, e.g., by the underlying routing protocol or by further steps in the authentication protocol. Messages can be checked after the keys were opened.
- Before Step 6 is performed it has to be ensured that $A$ and $B$ send the last messages in Steps 4-5 (if any). $A$ might send a message to $B$ expressing that she finished sending messages, and $B$ sends after receiving such a message to $A$. To reduce the overhead of additional messages a flag-bit in the packet header might be used to indicate a final message. Note that this information has to be protected by a MAC as well to prevent forgery, e.g., it might be part of the final message $(m_f)_{x^A_{j-u-1}}$ and $(m_f)_{x^B_{i-v}}$, respectively.

- The loop in Step 8 avoids an attack, where a malicious entity starts an authentication process, interrupts it after he gained a key, and uses that key to the other benign entity. The idea is that $A$ assumes after a faulty execution, that an attacker uses the gained key to obtain another key from $B$. Hence $A$ does not accept a single key anymore but requires knowledge of the next two keys, and so on. $B$ acts in the same way.
- After the public keys are exchanged, $u, v = 1$. Also after a successful protocol run, $u$ and $v$ are reset to 1. Hence in a normal protocol run the loop is only executed once, i.e., $k' = 1$.
- If $A$ opens a key which $B$ does not receive due to network faults, $A$ can send the key again without endangering the security of the scheme until $B$ gets the key, or until $A$ halts the execution of the protocol due to a time-out error. The same holds for keys opened by $B$.
- $A$ and $B$ do not need to know $k' = \max\{u, v\}$ before the execution of the loop. The entities just continue running the loop until both signalize their satisfaction.
- The scheme uses mutual authentication to avoid that a key of Bob is stolen by a malicious entity that pretends to be Alice, and later on used by the malicious entity to impersonate Alice when authenticating to Bob. Note that all complexities for our scheme obtained below are for mutual authentication whereas the previous scheme instantiations only ensured mutual authentication at additional cost.
- $A$ and $B$ always have to be in the same role, i.e., for a pair $A$ and $B$, the same entity always has to act as $A$. This limitation can be removed at the cost of a further message by requiring that $A$ and $B$ open at least two keys. The following steps which replace Step 9 implement such a case:

$9'$ : $B$ `opens his key by sending` $x^B_{i-k'-1}$ `to` $A$
$10'$ : $A$ `checks if` $h(x^B_{i-k'-1}) = x^B_{i-k'}$
~ `If the check fails, or the execution is interrupted,`
~ $A$ `stores` $(x^B_i, r_A, j, \max\{u, k'+1\})$
$11'$ : $A$ `and` $B$ `store the new values` $(x^B_{i-k'-1}, r_A, j-k'-1, 1)$
~ `and` $(x^A_{j-k'-1}, r_B, i-k'-1, 1)$

- A man-in-the-middle attack at execution time is possible as it is for most other authentication schemes including the public-key scheme.
- The man-in-the-middle attack is not possible to forge authenticated messages since the messages are already exchanged at the time when the authentication starts.
- The scheme is not resistant to denial-of-service attacks (as is neither of the other schemes). A malicious entity can try to exhaust the keys of the key chain by starting an authentication process with $A$, gaining $A$'s next key, using it against $B$ to obtain $B$'s key, using it again for $A$, and so on. An implementation can take care of this attack by forcing an exponentially increasing time interval between two authentication processes.

- The security of the scheme is as sound as the security of the underlying one-way hash-function and the underlying message authentication code. A formal proof of all above mentioned security properties can be found in the Appendix.
- The public key is only send once. Usually, the loop is only executed once, thus an authentication process requires three messages (Steps 6, 8.1, and 8.3), each of them a $t$-bit string
- To compute an arbitrary element of a hash chain, $A$ only needs to know $B$'s $ID$ and the random seed $r_A$ used for $B$.
- Assume that $A$ and $B$ use hash chains of length $n$. To compute $x^A_{i-1}$ and $x^B_{j-1}$ in Steps 4 and 5 from the global private key, this requires on average $n/2$ applications of the hash function $h$. Since $A$ needs twice as many keys as $B$ for an authentication, he might want to use chains that are twice as long.
- For each communication pair of $A$ with any $B$, she needs to store a tuple $(x^B_i, r_A, j, u)$, i.e., the public key. Assuming that we use short key-chains the indices $j$ and $u$ can be expressed in 2 bytes. Then $A$ requires storage of $(2t+32)/8$ bytes. Assuming that $A$ stores data of $p$ communication parters, she needs storage of $p(2t+32)/8$ bytes.

Considering a hash function that maps strings to $t$ bits, the probability to find a collision for any pair of strings (collision resistance) is $2^{-t/2}$ whereas the probability to find a collision to a given message (target collision resistance) is $2^{-t}$. We believe that target collision resistance suffices our demands. It is widely believed that computing a collision to a given message in a one-way hash function with $t = 80$ is as hard as factoring an RSA modulus of 1024 bits. Hence to establish a security level similar to 1024-bit RSA we assume $t = 80$. Let the size of the average key-chain be $n = 100$ which should meet the demands of most short-lived pervasive networks. As we said before authentication in a pervasive network is usually connected to some service that is offered or requested, i.e., authentication is bound to messages that are exchanged. Altogether the scheme requires on average 50 applications of the hash function, and three message exchanges each of 10 bytes. It furthermore requires 24 bytes of storage on the prover's and verifier's side which are far less than an RSA public key and about the same size of a 160-bit ECC public key (assuming each entity uses the same curve parameters). Note that in our scheme the prover and the verifier have to store the other entity's public key. Since most relationships in a pervasive network are mutual we do not believe this to be a disadvantage but a feature. A hash function has very low running time. For example, the SHA-1 implementation in [1] obtains hashing speeds of 48.7 Mbit/s whereas an RSA verification (which is the most efficient signature verification scheme) runs in roughly 1 ms [5]. Thus an application of a hash function is faster than an RSA verification by almost a factor of $10^6$, and the repeated application of a hash function nearly is for free for 50 or even more iterations. If a device has on average contacts to $p = 50$ communication parters, there is storage needed of 1200 bytes.

The number of hash function iterations can be reduced at the cost of storage, or by using shorter key-chains. To store the elements of a chain we can store each element of the chain, or only store the anchor and compute the elements on demand. A storage efficient mechanism was proposed in [4] which only requires $\log n$ storage and $\log n$ computation to access an element in a chain of $n$ elements. For instance, if $n = 100$ we can reduce the number of average hash-function iterations to 7 at the cost of 7 more storage entries.

To safe memory and to avoid extensive computations short key-chains are used. We now show how to establish a new key-chain without breaking the trust relationship. Let $x_i$ be the values of an old key-chain that is nearly used up, and $x'_i$ be the values of a new key-chain. These key-chains use different anchors $x_0$ and $x'_0$ which are derived of the same global private key $x^G$ and different random seeds, i.e., $x_0 = x^G \oplus f(ID) \oplus r$ and $x'_0 = x^G \oplus f(ID) \oplus r'$ with $r \neq r'$. We consider $x_1^A$, $x_0^A$, and ${x'}_{n_A}^A$ as the involved elements, where the first two keys are the last keys of the old chain and the last one is the first key of the new chain. Let $r_A$ and $r'_A$ be the random seeds of the old and new key-chain. The following protocol presents a way for Alice to introduce a new key-chain to Bob.

1 : Assume $A$ stores $(x_i^B, r_A, j)$ and $B$ stores $(x_1^A, r_B, i)$
2 : $A$ sends $B$ the new chain start value $(x_{n_A}^A)_{x_0^A}$
3 : Do Steps 6 to 8 as in the previous protocol
4 : Finally, if all checks were successfully, $A$ stores
~ $(x_{i-k'}^B, r'_A, n_A, 1)$ and $B$ stores $({x'}_n^A, r_B, i-k', 1)$.

**Remarks:**

- To avoid a man-in-the-middle attack as for the previous protocol we introduce Step 3.
- As before, keys always have to be checked for correctness, e.g., after Step 5.
- It is wise not to use the anchor $x_0$ in the protocol. It is better to keep some elements before initializing a new key-chain to avoid a complete loss of the trust relation due to network errors or malicious interceptions.

## 4 Comparison of the Schemes

In the previous sections we presented three instantiations of the ZCK authentication scheme. Table 1 gives an overview over all three instantiations. The first rows describe the complexity whereas the following rows describe features. Note that the values for the public-key scheme depend on the used scheme. Where applicable, a 'x' means that the scheme provides a feature, a '-' means that it does not, and a 'o' means that the scheme can provide the feature with modifications (at higher computational cost). The symmetric-key scheme is the most efficient one but it requires geographical proximity for a key exchange, or a single trust authority such as in military scenarios or such as for application in a private home. A secure key-exchange could be established by means of public-key schemes. At this point the symmetric-key scheme integrates in the public-key scheme.

**Table 1.** Overview of ZCK authentication schemes

| | Key-Chain | Public-Key | Sym. Key |
|---|---|---|---|
| public-key size (bytes) | 24 | ‡ | 10† |
| exchanged messages | 3 | 2 | 2 |
| exchanged bytes | 30 | ‡ | 20 |
| computational effort | 0 | 2 PK Op. | 0 |
| ZCK authentication | x | x | x♯ |
| message authentication § | x | x | x♯ |
| ZCK non-repudiation | - | x | - |
| key-exchange | - | x | - |
| signature | - | x | - |
| mutual authentication | x | o | o |

† size of the shared key
‡ depends on the used public-key scheme
♯ requires secure channel for key-exchange
§ includes message integrity and freshness

That one requires computationally demanding operations that are significantly more expensive than all other schemes but provides ZCK non-repudiation as well as a key-exchange. Our new scheme nearly has the same complexity as the symmetric scheme without its shortcomings. It is not capable of providing ZCK non-repudiation and a key-exchange, but comes with a mutual authentication for free. It is extremely efficient in providing ZCK authentication and message authentication in the view of computational as well as network resources.

Finally we want to give a remark about key distribution. A core benefit of a public-key infrastructure (PKI) compared to some infrastructure using symmetric keys is the key distribution. In a PKI each user has a public/private key pair that enables an entity to communicate with each other entity, whereas a symmetric key infrastructure (SKI) requires a shared key between any entity pair. Thus for $n$ entities a PKI requires $n$ key pairs whereas a SKI requires up to $n(n-1)/2$ shared keys if any entity wants to be able to communicate with any other entity. In our scheme, there are even $n(n-1)$ key-chains, i.e. keys, required if any pair of entities established a relationship. Thus one could conclude that our key-chain scheme has this disadvantage compared to a public-key scheme. However, in our scope of ZCK authentication the facts are different. Clearly there are no central directories of public-keys in a pure pervasive network as we are envisioning it. Furthermore, each user stores foreign entities' keys and binds them to some common past or experience. If an entity does a broadcast of its public key the receivers will not take notice of that key since they are not able to establish a relationship to that key. Each key has to be hand over bound to a service as we said before. Hence in our case, the complexity of a full distribution of all keys is always $n(n-1)$, both in the symmetric and asymmetric case.

## 5 Conclusions

In this work we presented a basic understanding of authentication for pervasive networks. Our approach is simplistic yet still realistic. It presents the most general case of authentication in pervasive networks which we call zero common-knowledge (ZCK) authentication. We presented three instantiations. The public-key scheme will not suffice our requirements for weak devices since it is not efficient enough. The symmetric-key scheme is most efficient but requires geographical proximity for a key exchange, or a single trust authority. Our new key-chain scheme is a general and still extremely efficient solution. It virtually needs no computation, 30 exchanged bytes in 3 messages, and 24 bytes to store a public key, and it provides mutual authentication for free. Therefore in many applications our new hash-chain based ZCK authentication scheme is very well suited to pervasive computing devices since it provides ZCK authentication and message authentication and requires nearly no computational power, and very low bandwidth and memory storage.

## References

[1] A. Bosselaers. Even faster hashing on the Pentium. *Rump session of Eurocrypt'97*, 1997. 81

[2] M. Burrows, M. Abadi, R. Needham. A Logic of Authentication. *DEC SRC Research Report 39*, revised version, 1990. 85

[3] L. Buttyán and J.-P. Hubaux. Nuglets: a Virtual Currency to Stimulate Cooperation in Self-Organized Mobile Ad Hoc Networks. *Technical Report DSC/2001/001*, Swiss Federal Institute of Technology – Lausanne, Department of Communication Systems, 2001. 74

[4] D. Coppersmith and M. Jakobsson. Almost optimal hash sequence traversal. In *Proceedings of the Fourth Conference on Financial Cryptography (FC '02)*, Springer-Verlag, 2002. 82

[5] E. De Win, S. Mister, B. Preneel and M. Wiener. On the performance of signature based on elliptic curves. In *Proceedings of ANTS III*, LNCS 1423, Springer-Verlag, 1998. 81

[6] J. Hoffstein, J. Pipher, J. H. Silverman. NTRU: A Ring-Based Public Key Cryptosystem. In *Proceedings of ANTS III*, LNCS 1423, Springer-Verlag, 1998. 77

[7] B. Lamparter, K. Paul, D. Westhoff. Charging Support for Ad Hoc Stub Networks. In *Elsevier Journal of Computer Communication, Special Issue on 'Internet Pricing and Charging: Algorithms, Technology and Applications'*, Vol. 26, Issue 13, August 2003. 74

[8] A. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*, CRC Press, 1997. 76, 78

[9] S. Zhong, J. Chen, Y. R. Yang. Sprite: A Simple, Cheat-Proof, Credit-Based System for Mobile Ad-Hoc Networks, to appear in *Proceedings of INFOCOM 2003*, 2003. 74

# A Security Proof of Key-Chain Scheme

In the following we give a security proof of our new key-chain authentication scheme. We will first analyze and evaluate the protocol as proposed by the BAN-logic [2]. Then we prove that our authentication scheme is secure against a passive eavesdropper, and that our scheme for message authentication is even secure against an active adversary.

## A.1 Evaluation with BAN-Logic

We start the analysis of our protocol with means of the BAN-logic by presenting an *idealized version* of the authentication steps. Message authentication is based on the authentication steps and comes with it. For ease of description we omit the random seeds $r_i$ without loss of generality.

$A$ `stores` $(x_i^B, j, u)$ `and` $B$ `stores` $(x_j^A, i, v)$
1 : $A \rightarrow B$: $x_{j-1}^A$
2 : `For` $k = 1$ `to` $k' \leftarrow \max\{u, v\}$ `do`
  2.1 : $B \rightarrow A$: $x_{i-k}^B$
  2.2 : $A \rightarrow B$: $x_{j-k-1}^A$
  2.3 : `If failure then`
    2.3.1 : $A$ `stores` $(x_i^B, j, \max\{u, k+1\})$
    2.3.2 : $B$ `stores` $(x_j^B, i, \max\{v, k+1\})$
  `End If`
`End For`
$A$ `stores` $(x_{i-k'}^B, j-k'-1, 1)$ `and` $B$ `stores` $(x_{j-k'-1}^A, i-k', 1)$

We first recall the basic constructs of the BAN-Logic:

- $A \models X$: $A$ *believes* $X$, or $A$ would be entitled to believe $X$. In particular, the principal $A$ may act as though $X$ is true.
- $A \mid\sim X$: $A$ *once said* $X$. The principal $A$ at some time sent a message including a statement $X$. It is not known whether the message was sent long ago or during the current run of the protocol, but it is known that $A$ believed $X$ when he sent the message.
- $\#X$: The formula $X$ is *fresh*, that is, $X$ has not been sent in a message at any time before the current run of the protocol.

Now we can give some *assumptions* which are immediately derived from the stored values:

$$A \models (x_i^B, j, u)\ B \models (x_j^A, i, v)$$

It easily follows the *conclusions*:

$$A \models (x_{i-k'}^B, j-k'-1, 1)\ B \models (x_{j-k'-1}^A, i-k', 1) \tag{1}$$

The *objective* of our authentication scheme is to ensure that the last opened key is fresh, i.e., was not send before. Based on the security of a one-way hash

function, only the right entity could have opened this key for authentication. We denote this formally as follows:

$$A \models B \mid\sim x^B_{i-k'} \quad A \models \#x^B_{i-k'} \tag{2}$$

$$B \models A \mid\sim x^A_{j-k'-1} \; B \models \#x^A_{j-k'-1} \tag{3}$$

The proof outline below shows the idea of the proof. For easier understanding, we suppress details. First we analyze the execution of a loop. If a key is received by $A$ she can verify by applying the hash function and known key-chain values that $B$ has once sent this key.

$$\forall k \leq k'.\ A \models B \mid\sim x^B_{i-k}\ \forall k \leq k'.\ B \models A \mid\sim x^A_{j-k-1} \tag{4}$$

By construction of the loop, all key values that are larger than $u$ and $v$, respectively, have to be fresh in a general fashion, i.e., $A$ and $B$ believe that they are fresh. This is ensured by the failure condition in Step 2.3.

$$\forall k \geq v.\ A \models \#x^B_{i-k}\ \forall k \geq u.\ B \models \#x^A_{j-s-1} \tag{5}$$

Since $k' \geq u, v$, our objective (Equations (2) and (3)) easily follows from Equations (4) and (5) for the last iteration of the loop ($k = k'$). It is also clear that the conclusions (Equation (1)) immediately follow from the data seen by $A$ and $B$. If there is a failure Step 2.3 ensures that the authentication process is not successful, and that Equation (5) still holds. The proof needs only slight adjustment to cover the case where $A$ and $B$ open at least two keys as described in Section 4.

Message authentication is done with the keys $x^A_{j-u-1}$ or $x^B_{i-v}$ respectively. Assuming that the message authentication code is secure, it immediately follows that all authenticated messages were sent by $A$ or $B$, respectively, and that all messages are fresh.

### A.2 Security Proofs

As we said before, our authentication scheme is secure against a passive adversary but not an active one. However, we can show that the message authentication property of our scheme is also secure against active adversaries. More precisely, we will prove that our schemes are as sound as the underlying one-way hash function.

**Theorem 1.** *Our authentication scheme is as sound as the one-wayness of the underlying hash function against passive adversaries.*

***Proof:*** *Assume the scheme is based on a one-way hash function $h$. As we evaluated before by BAN-Logic an entity has to send a fresh key value to be authenticated. We allow an eavesdropper to listen to arbitrary many values $x_{i+k}, k \geq 1$,*

*with $x_{i+k+1} = h(x_{i+k})$. The adversary has to present the key $x_i$ to be successfully authenticated. Assume the adversary could authenticate successfully, then he could derive the key $x_i$ of all the keys $x_{i+k}$ with $k \geq 1$. Hence we could use the adversary to invert the hash function, which is a contradiction to its one-wayness.*

Finally we prove that our scheme provides provably secure message authentication. We assume that the authentication steps starts after all messages are sent, which can easily be provided by a flag in the authenticated message header. Note that at the point the final message $(m)_{x_i}$ is received it is not already authenticated but hold for authentication. However, this suffices as assumption since an adversary would have to forge the last message before the authentication steps start.

**Theorem 2.** *Our message authentication scheme is as sound, i.e. as secure against message forgery, as the underlying message authentication code against active adversaries.*

***Proof:*** *Assume that the authentication step starts after all the messages are exchanged, and that the scheme is based on a one-way hash function $h$. We allow the adversary to listen to arbitrary many messages $(m)_{x_{i+k}}$ and $x_{i+k}, k \geq 1$, with $x_{i+k+1} = h(x_{i+k})$. Now assume an adversary can forge a message, i.e., he is capable of producing a valid $(m)_{x_i}$. As before the last opened key $x_i$ has to be fresh, i.e., it was not sent before by A or B. But then we could use the adversary to produce a forged message authentication code without knowing the key. This is a contradiction to the security assumptions of the MAC.*