VERSITA

Annales UMCS Informatica AI XI, 3 (2011) 41–56
DOI: 10.2478/v10065-011-0019-2

Annales UMCS
Informatica
Lublin-Polonia
Sectio AI
http://www.annales.umcs.lublin.pl/

Secure time information in the internet key exchange protocol

Paweł Szałachowski[1*], Zbigniew Kotulski[1,2†]

[1] *Faculty of Electronics and Information Technology, Warsaw University of Technology, Nowowiejska 15/19, 00-665 Warsaw, Poland*
[2] *Institute of Fundamental Technological Research, Polish Academy of Sciences, Pawinskiego 5 b, 02-106 Warsaw, Poland*

Abstract

Many network services and protocols can work correctly only when freshness of messages sent between participants is assured and when the protocol parties' internal clocks are adjusted. In this paper we present a novel, secure and fast procedure which can be used to ensure data freshness and clock synchronization between two communicating parties. Next, we show how this solution can be used in other cryptographic protocols. As an example of application we apply our approach to the Internet Key Exchange (IKE) protocol family.

1. Introduction

Data freshness guarantees protection from variants of the replay attack, it is very important and desired in network communication. We distinguish two types of freshness: weak and strong. Weak freshness provides only partial ordering of messages. This type does not supply any other kind of time information, e.g., a delay. However, the strong freshness provides total messages

[*] *E-mail address: p.szalachowski@stud.elka.pw.edu.pl*
[†] *E-mail address: zkotulsk@tele.pw.edu.pl*

ordering and delay information, so this type of freshness can be obtained in time synchronization protocols.

The scheme presented in this paper addresses the freshness issue and it has ability to synchronize time. It is very light (sending only one short message is required) and based on cryptographic hash functions, which are fast constructs. Our proposal can be applied in many existing communication protocols, where small modifications can result in significant advantages. We show them for a popular IKE protocol. Our method is generic for the IKE protocol family, but as an example we focus on a concrete element, which is ISAKMP[**1**]. This is a sub-protocol of IKE responsible for key agreement. It provides framework for authentication and key exchange. Our extension enables, except for the standard secret key agreement by two parties, synchronization of their clocks in a cryptographically secure way.

This paper is organized as follows. In Section 2 we present the state of the art and in Section 3 we shortly describe the IKE and ISAKMP underlying a role of the special structures for their functionality. In Section 4 we introduce our time refreshment scheme and a proposition of its implementation. The security and performance analysis of the approach are presented in Section 5, while Section 6 describes the future work and conclusions.

2. State of the art

Precise time is necessary in many areas of our everyday life. Besides scientific and engineering applications like synchronous measurements, all legal and financial transactions, transport, business and other social activities with distributed resources demand reliable and accurate time. IEEE provides standard for precise clock synchronization in [**2**]. Barak in [**3**] describes an efficient and fault-tolerant clock synchronization method. For applications which require higher trust level (e.g. electronic documents), paper [**4**] introduces a reliable clock synchronization method. This is especially important for network communication. The most widespread time synchronization protocol is NTP (The Network Time Protocol) [**5**], however, there are many different solutions for specific network environments [**6, 7**]. For example, paper [**8**] presents a scheme of synchronization of a time-of-day clock in nodes of a local area network. In paper [**9**] time synchronization solution for high latency acoustic networks is introduced. Paper [**10**] presents a time synchronization approach for large decentralized systems. Another example, which is WSN, is a very hostile environment for communication protocols. It operates in an open medium and nodes of the network are hardware-constraint. In such a case there are many opportunities to attack network services. The time synchronization service is

also prone to the attacks in this environment. Vulnerabilities of this service in the sensor networks are presented in [**11**]. Therefore, these networks especially need secure and very efficient solutions, such as [**12, 13, 14**]. The surveys of time synchronization schemes in WSNs are presented in [**15, 16**].

Another crucial issue connected with time is freshness. In practical solutions [**17**] timestamps, counter values and pseudo-random numbers are used as freshness identifiers. In the case of strong freshness, every time when a synchronization message is being sent, the sender must disclose his time or another value which he uses to ensure freshness. It is often undesirable in networks with open medium (e.g. Wireless Sensor Networks, WSNs) or in dynamic networks, like Internet. For example, an attacker knowing time can compromise a pseudorandom number generator (if the time value has been used as a seed which is a frequent practice). Another case is, e.g. IP Timestamp in Linux implementations. An attacker knows when a computer was restarted last time, so he knows if the restart occurred after some critical system update. Freshness is so important that many cryptographic protocols require assurance of this property. A precise definition of freshness and examples of attacks against it can be found in [**18**] where also complexity of checking freshness for many different scenarios is presented. Corin in [**19**] develops and analyses a model for security protocols that takes time into account. He considers two aspects of the problem: an influence of time on messages flow (e.g. timeouts, retransmissions) and time information within protocol messages (e.g., a timestamp). Next method for analyzing security protocols with time aspects is presented in [**20**]. This paper analyses real-time properties of security protocols by the Strand Space-based approach.

3. Protocols and structures

IKE[**21, 22**] is one of the IPsec protocol suites. It is designed to set up a Security Association (SA). SA is a policy and key material used to protect information. So one goal of the protocol is secure key agreement, which is a very topical and important task for network communications. IKE introduces abstraction for key agreement (ISAKMP) and its implementation (Oakley protocol[**23**]). One of the protocol goals is a key established between two parties communicating through an insecure channel. It is based on the Diffie-Hellman[**24**] key agreement protocol but it has some additional advantages. Its main features are presented below:

- it allows the two parties to negotiate the methods of: encryption, key derivation and authentication,

- it allows the two parties to agree a shared secret without resource demanding public key encryption,
- it has several options for the key computation,
- the parties can derive a new key from an existing one in a few ways, with the aid of the Diffie-Hellman protocol or without,
- the protocol offers strong authentication methods for the parties' identities,
- before authentication, two parties do not have to compute the exponentiations shared, so it is efficient,
- the authentication checks the results of exponentiations assigned to the identities of the parties,
- it provides a mechanism which helps avoiding Denial of Service (DoS) attacks. This will be presented in detail in the next subsection,
- additionally, the parties can define their own or select the existing mathematical structures for the Diffie-Hellman protocol,
- the protocol allows two parties to use features that are best suited to their needs and capabilities.

We see that protocol is very flexible and scalable. The main and mandatory cryptographic attributes of negotiations are: encryption algorithm, hash algorithm, authentication method and information about a group over which to do the Diffie-Hellman calculations. Each implementation of IKE must provide the following values for these attributes: DES [**25**] in the CBC mode (with weak and semi-weak), MD5 [**26**] and SHA [**27**], authentication via pre-shared keys and default group for the Diffie-Hellman algorithm (prime and generator defined in [**21**]).

Now we focus on ISAKMP. It provides framework for authentication and key exchange but does not define them.

In Fig. 1 the ISAKMP architecture is described. As security protocols we can use, for example, TLS/SSL [**28, 29**] or IPSEC-ESP/AH [**30**]. In the case of TLS/SSL relationship will be changed (to Socket Layer). The header message of ISAKMP is shown in Fig. 2.

The first and second fields are Anti-Clogging Tokens (described more detail in the next Section). Next payload indicates the type of the first payload in the message. Major and Minor Version fields are to represent the version of ISAKMP protocol which is used. Exchange Type states the type of exchange being used. The flags field is for specific options of ISAKMP, it includes Authentication bit, Encryption bit and other exchange-related options. The message identifier identifies the protocol state (in the second phase of negotiations) and the last field (Length) describes the length of total message.

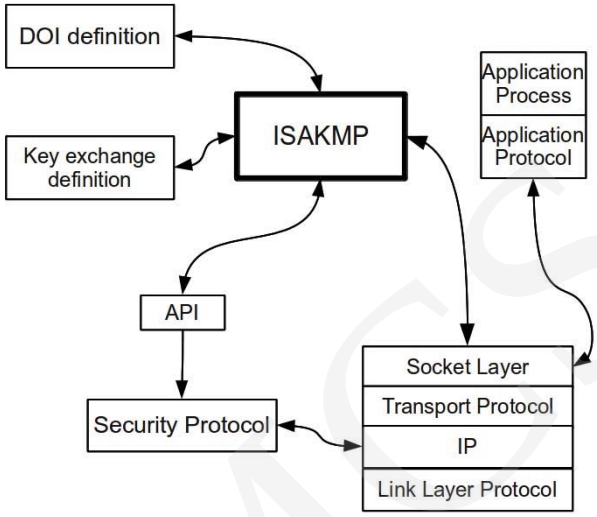

Fig. 1. ISAKMP architecture.

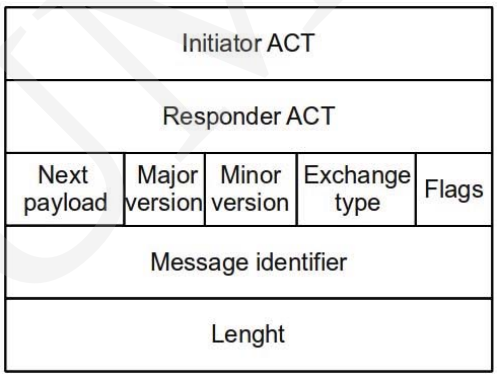

Fig. 2. ISAKMP header.

As many other popular cryptographic protocols, IKE omits strong freshness or time synchronization service. We will extend it with these security services. The IKE protocol defines two parties: Initiator and Responder. This is similar to the Client-Server architecture in messages exchange services. As mentioned before the protocol offers many scenarios of establishing a new secure communication channel; its versions depend on participants' preferences and capabilities. In spite of that the messages exchanged are different in the protocol's versions, IKE includes several permanent elements. One of those obligatory elements is an Anti-Clogging Token. It will be discussed now as an essential part of our freshness scheme.

3.1. Anti-Clogging Tokens

Key agreement protocols should be protected against a sort of DoS attacks. In ISAKMP it is accomplished by Anti-Clogging Tokens (ACTs), also called cookies. This method is generic for the IKE protocol family. The ACTs are exchanged between the parties in each version of the protocol as messages' headers. Since large integer exponentiation is computationally the most expensive step of the protocol, before the parties start their execution they exchange the ACTs to ensure that they are legitimate and interested in the protocol's execution. For both parties the ACTs act as participants' identifiers and they rely on source addresses.

The creation of ACTs must satisfy the following requirements:

- ACT must depend on specific parties (the goal is to prevent the attacker from obtaining ACT using real IP address and UDP port),
- no one other than the issuing entity is able to generate ACT that will be accepted by that entity,
- the process of ACT generation must be fast.

Because of requirements, ACTs are often generated by fast hash function. The input consists of the IP source and destination address, the UDP source and destination ports and a locally generated secret random value.

Another duty of the ACTs is keys naming. We denote the ACT of Initiator by *ACT-I* and analogously, the ACT of Responder is *ACT-R*. ISAKMP uses the concatenation of these values to name the keys. The IKE standard defines several ways of executions of the protocol. The short one, called aggressive, needs only three messages exchanges. Among other parameters that are sent in the protocol's steps, ACT is included in almost each header.

Finally, the ACTs are connected with time and freshness, but they do not provide any direct information about time. In our approach the ACTs are modified in such a way that they retain their unique property, but they are additionally equipped with fresh time information. As a consequence, the whole key agreement protocol is enhanced with a new service.

4. Time refreshment protocol

In the protocols which deploy using synchronized time in parties' internal clocks, there is always acceptable tolerance of a local time from a reference time. Therefore in our protocol we take into consideration a tolerance parameter. For security of time synchronization let us assume that the parties of the protocol share a common secret.

In further considerations we will use the following notation:

- $prf(key, .)$ is the keyed pseudo-random function (cryptographic hash function or MAC scheme),
- K is the secret key shared by Initiator and Responder (it can be used only for time refresh),
- IP_R is the IP address of Responder,
- IP_I is the IP address of Initiator,
- P_R is the UDP port of Responder,
- P_I is the UDP port of Initiator,
- $\|$ is the concatenation of two blocks of bits [‡],
- t_R is the actual Responder's, assumed to be the reference time,
- n is the time tolerance parameter,
- $f_n(.)$ is a function that converts each of the values $\hat{x} \in \{x-n, ..., x, ..., x+n\}$ to one value $\hat{f}_n = f_n(\hat{x})$ and which satisfies the following condition:

$$f_n(x) \neq f_n(x+k) \tag{1}$$

for all integer k such that $|k| > n$,

The protocol starts with sending *init message* from Initiator to Responder. After receiving *init message*, Responder (which keeps the reference time) computes only:

$$H_R = prf(K, IP_I \| IP_R \| P_I \| P_R \| n \| f_n(t_R)), \tag{2}$$

concatenates it with the time tolerance parameter n and sends the result to Initiator as a reply. In the next step, Initiator reads his actual time (t_I). Succeeding, if

$$prf(K, IP_I \| IP_R \| P_I \| P_R \| n \| f_n(t_I)) \neq H_R, \tag{3}$$

then Initiator decides that his clock is desynchronized (his time is outside of the set $\{x-n, ..., x, ..., x+n\}$). When the hashes are equal, he decides that his clock and the Responder's clock are synchronized with precision defined by the parameter n. To come to a decision only one hash computation is required for each party. In the above calculations we assumed that the parties of the protocol shared the common secret K. To authenticate the messages they merged the secret K with the other parameters being sent. However, since in many MAC schemes a secret parameter K is used as the algorithm parameter, it is not necessary to merge K with the messages before the algorithm execution.

[‡]To protect against concatenation flaws one should assume fixed sizes of the blocks concatenated.

4.1. Details of scheme

An important element of the construction presented is the function $f_n(.)$. It should convert any of $x-n, ..., x, ..., x+n$ to one value in such a way that property given in (1) is satisfied. The function $f_n(.)$ which satisfies condition (1) can be defined as:

$$f_n(x) = \begin{cases} \frac{x}{p}, & r = 0 \\ \frac{x-r}{p}, & r \leq n \\ \frac{x+p-r}{p}, & r > n \end{cases}, \tag{4}$$

where $p = 2n+1$ and $r = x \mod p$.

The purpose of using $f_n(.)$ is to obtain the same value of hash (equal to the hash of the value in the middle) for all arguments of a given time range. It is easy to see, that the reference time must lie in the middle of the time range. The function $f_n(.)$ is a piece-wise constant. So, to pass time t_R with tolerance n, we must shift the arguments of the function $f_n(.)$, in such a way that the reference time will lie in the middle of the time range. In order to achieve this we define the offset parameter $\hat{o}$ as:

$$\hat{o} = t_R \mod p. \tag{5}$$

To obtain the time freshness information we first compute $f_n(t_R - \hat{o})$ and then the hash value of the concatenation of all the parameters required:

$$H_R = prf(K, IP_I \| IP_R \| P_I \| P_R \| n \| \hat{o} \| f_n(t_R - \hat{o})). \tag{6}$$

In this scheme, we additionally used the parameter $\hat{o}$, defined above. It is crucial for the function $f_n(.)$ proper work. This parameter depends on the actual time (it is obtained dynamically), so to achieve its integrity we concatenate it with other parameters and hash them together. Of course, the receiver of the hash does not know the $\hat{o}$ value, so we must sent it separately. Moreover, if we have not agreed the tolerance value, we must sent it also with the hash value.

4.2. Implementation in IKE

In our proposal hash, tolerance and offset parameters are included into the ACT of the Responder. We presented that structure above. Thus, the ACT of the Responder looks like this:

$$ACT - R = H_R \| n \| \hat{o}. \tag{7}$$

The processes of ACTs generation and verification can be realized as follows:

In Fig. 3 the first version of time refreshment protocol is presented. The *init message* is generic ISAKMP packet. The Responder delivers the ACT

Version I:

Initiator:
send ($initmsg$);

Responder:
t_R=get_time();$\hat{o} = t_R \mod (2n+1)$;
$H_R = prf(K, IP_I \| IP_R \| P_I \| P_R \| n \| \hat{o} \| f_n(t_R - \hat{o}))$;
$ACT\text{-}R$=$H_R \| n \| \hat{o}$;
send ($ACT\text{-}R$);

Initiator:
parse ($ACT\text{-}R$); t_I=get_time();
$H_I = prf(K, IP_I \| IP_R \| P_I \| P_R \| n \| \hat{o} \| f_n(t_I - \hat{o}))$;
if H_I is equal H_R;
  adjust time by $\hat{o}$;
else
  need to synchronize

Fig. 3. First versions of time refreshment scheme.

with other ISAKMP's fields. The ACT consists of hash, tolerance and offset values. The initiator produces another hash value from his local time and then checks if the hashes are equal. If they are, then his time is in the time range of Responder, so Initiator can adjust his own clock to the Responder's time using the offset value. Otherwise, Initiator needs to synchronize his time by an external service. Note that Initiator knows his IP address and port, but the tolerance and offset parameters are delivered in the ACT.

The second version (Fig. 4) is for the case when Initiator sends his time information. He uses the pseudo-random keyed function to encode his time. Next that information is sent within unused field ($ACT\text{-}R$ in $initmsg$). Responder checks if the time is synchronized. If it is Responder adds 1 at the beginning of his ACT, otherwise Responder adds 0. After that message Initiator knows if his time is synchronized. Unfortunately, if time is synchronized, Initiator can not adjust it. This is because Responder does not pass the offset.

In our approach we treat transfer delays as negligible values. If these delays are significant then Initiator can take them into consideration (Version I). Thus, the time of Initiator obtained by the system function $get\_time(.)$, before using a cryptographic hash function, should be adjusted according to the delays. Our

Version II:

Initiator:
t_I=get_time();$\hat{o} = t_I \mod (2n+1)$
$H_I = prf(K, IP_I \| IP_R \| P_I \| P_R \| n \| \hat{o} \| f_n(t_I - \hat{o}))$;
use $H_I \| n \| \hat{o}$ as $ACT\text{-}R$ in $initmsg$;
send ($initmsg$);

Responder:
parse ($initmsg$); t_R=get_time();
$H'_R = prf(K, IP_I \| IP_R \| P_I \| P_R \| n \| \hat{o} \| f_n(t_R - \hat{o}))$
$ACT\text{-}R$=generate_ACT();
if H'_R is equal H_R (from $initmsg$)
    $ACT\text{-}R = 1 \| ACT\text{-}R$
else
    $ACT\text{-}R = 0 \| ACT\text{-}R$
send ($ACT\text{-}R$)

Initiator:
check first bit of $ACT\text{-}R$

Fig. 4. Second version of time refreshment scheme.

scheme also allows Initiator to make an attempt of "off-line" time synchronization. Initiator may be interested in looking for the reference time by checking probable time values from different ranges (see e.g. [**31**]). We rely on the IKE protocol in ensuring integrity of the protocol's messages. So, when IKE fails, the Initiator must not apply any changes in configuration of his system. We describe this aspect in Section 5.

4.3. Variants of protocol's execution

Presentation of a complete IKE's execution needs description of several additional elements of the protocol. Concatenation $\|$ and the function $prf(.)$ have already been introduced. The rest of notation follows the IKE's specification [**21**]:

- HDR is an ISAKMP header (we described it in Fig. 2),
- HDR^* indicates the payload encryption,
- SA is a negotiation payload with one or more proposals,
- KE is the key exchange payload,
- ID_I and ID_R denote the identities of Initiator and Responder,

- N_I and N_R are the nonces supplied by Initiator and Responder,
- $HASH_I$ and $HASH_R$ are the authentication tags for Initiator and Responder

In this section we focused on Phase 1 of IKE. The goal is to establish a secure authenticated communication channel. An example of so-called Authenticated With the Pre-Shared Key (Main Mode) version of IKE with private identities and without the Diffie-Hellman protocol is now considered. This scenario is noteworthy if we need simplicity and high performance of the protocol. Exchange of messages in this version of the protocol is presented in Fig. 5.

I	*message*	R
$\longrightarrow$	HDR, SA	$\longrightarrow$
$\longleftarrow$	HDR, SA	$\longleftarrow$
$\longrightarrow$	HDR, KE, N_I	$\longrightarrow$
$\longleftarrow$	HDR, KE, N_R	$\longleftarrow$
$\longrightarrow$	HDR^*, $HASH_I$ ID_I,	$\longrightarrow$
$\longleftarrow$	HDR^*, $HASH_R$ ID_R,	$\longleftarrow$

Fig. 5. Pre-shared key authentication in the Main Mode.

Consider the version of time scheme presented below in Fig. 5. The HDR in the first message contains only $ACT\text{-}I$, $ACT\text{-}R$ is null. Next, Responder produces $ACT\text{-}R$ in the way presented in Fig. 3 and sends it within the second message. When Initiator receives it, he is able to check if his time is synchronized (exact to n). If it is, then the time value can be adjusted by $\hat{o}$ to the reference time. Otherwise, in the case of desynchronized clock, Initiator can detect this and adjust time using external protocols, see e.g. [**5, 6, 7, 8**].

IKE introduces also an aggressive mode with a pre-shared key. It is very fast, requires sending only three messages and a few hashes calculations. The aggressive mode with secure time information is presented in Fig. 6. Using that mode we now describe the second version of time refresh scheme. The HDR as ISAKMP header includes ACT of Initiator and Responder. In the first message only $ACT\text{-}I$ is used. So in $ACT\text{-}R$ we pass time information. Initiator computes hash and parameters as in Fig. 4. Responder checks $ACT\text{-}R$ of the first message. Then he computes hash for his own time. Next, Responder uses one bit in his own ACT to signal if the Initiator's time is synchronized.

I	*message*	R
$\longrightarrow$	HDR, SA, KE, N_I, ID_I	$\longrightarrow$
$\longleftarrow$	HDR, SA, KE, N_R, ID_R, $HASH_R$	$\longleftarrow$
$\longrightarrow$	HDR, $HASH_I$	$\longrightarrow$

Fig. 6. Pre-shared key authentication in the Aggressive Mode.

5. Security and performance

An ACT is a quite short message. Often we want to include a hash and two parameters within 64 bits. The tolerance (n) parameter can be agreed in advance, but then our scheme becomes less flexible. We save space in the ACT for a longer hash, but we can not match the time range. Fixed n may be useful in embedded systems, but we decide to consider a flexible example of the scheme.

The presented approach is closely connected with a cryptographic hash function. The hash function is a core of our solution, so this element should be chosen very carefully. It should be widely approved and secure. The best known attack against hashes is the birthday attack, which is connected with collisions. For the hash function $prf(.)$ we define the collision as finding two different messages, which produce the same tag.

The probability of collision's occurrence is estimated by $2^{-\frac{m}{2}}$, where m is the tag length. Knowing length of fields we can easily estimate how often collision will occur. Of course, the occurrence of collisions is very undesirable. In our case a collision is not very interesting for an attacker, rather it can mislead Initiator. Having unsynchronized time, he could obtain the same hash, so he could deduce that his clock is synchronized.

For the attacker with ability to eavesdrop and to tamper messages, a forgery in the time refreshment protocol is equivalent to breaking the hash function used or founding the key K. The probability of successful attack should be close to $max(\frac{1}{2^l}, \frac{1}{2^{l'}})$, where l and l' are the security parameters (l is the length of K and l' is the length of the hash function output). According to [**32**] the key K should be at least 80-bits long. The tolerance n and the offset $\hat{o}$ parameters are sent as plaintexts, so we must ensure their integrity. That is realized by concatenating these values with K, IP_I and the value of $f_n(t_R - \hat{o})$ and by calculating their hash, see (6). The verification passes when all these values are correct. However, a malicious attacker can modify all messages. So, when he decides to modify any parameter of the Responder's ACT, then Initiator states that his clock is desynchronized (even if it is not true). On this level

Initiator cannot detect a malicious modification. But our scheme is embedded into the IKE protocol, which provides integrity/authentication of messages, so any malicious modification would be detected in the following steps of the IKE protocol.

The scheme presented protects the reference time even when an attacker compromises the key K. It is because of the inherent properties of a cryptographic hash function, which produces the ACT. In this case the attacker must perform the brute force attack in order to determine the values of parameters hashed. Denote k as the bits resolution of time value. Assume that time in the group of users is random (not connected with real actual time). The probability that the attacker (with a compromised key) guesses the actual time is $\frac{q(2n+1)s}{2^k}$, where q is the number of attacker's hash computations and s is the number of known messages.

Thus the most important factor is the time resolution. The popular size of actual time storing is 32 bits, but 16 bits and 64 bits variables are used too. The time of a successful attack also depends on the value of n. This is because the larger size of parameter n decreases the size of outputs of the function $f_n(.)$ (what follows, the number of its possible values), so an attacker has to compute fewer hashes on average to break the protocol.

The memory, transmission and computational overheads of the presented scheme are negligible. Our solution uses a cryptographic hash function; generally, most of the implementations of ISAKMP use fast cryptographic hash functions (for ACT generation) too. We change only the arguments of this function, which fits (in most cases) to one block of a hash. Furthermore, modifications are introduced only in the first phase of the IKE protocol. For this reason, the computational overhead due to time freshness service is negligible.

6. Future work and conclusions

Our scheme ideally fits as a lightweight time refreshment protocol. It can be easily integrated into other existing security protocols. Its main security requirements are that the parties share the secret key K and that the protocol provides data integrity. The last requirement can be achieved by using an additional authentication code for a message. In many protocols there exist fields for timestamps, which can be used in such a case. A good example is Optimized Link State Routing Protocol (OLSR) and its secured version [**33**], where the method described could be placed into the *timestamp* message. Other examples of methods, which can be enhanced with time refreshment are authentication protocols like [**34, 35, 36**], where also the *timestamps* fields occur.

There are many other applications where assurance of an actual time is required. A good example could be Kerberos Protocol [**37, 38, 39**], where the time of a server and the time of clients must be synchronized. Our scheme can be applied there: when a client detects that his time is desynchronized, he should synchronize it with a trusted host (server). The server does not disclose its time, thus a potential attacker, even when the secret key is compromised, knows only that the client's time is wrong. The presented proposal of freshness may be used also in other applications, e.g. position-based nodes selecting in WSNs, see [**40**].

In this paper we proposed a protocol which realizes time synchronization and data freshness between two parties. The described approach ideally fits cryptographic or secure communication protocols. Hence we decided to use the IKE protocol as an example of its application. The IKE protocol can be enhanced in an easy way, because of ACTs contained in its messages. Moreover, such an extension gives no significant computational or memory overhead to the original protocol. The detailed implementation of one of our scheme's versions is presented based on the Oakley protocol in [**41**]. Another crucial issue is security. It strongly depends on a cryptographic hash function chosen, traffic in the network and the tolerance parameter selected. As shown above, appropriate selection of these factors can give us the expected security level. All parameters can be set according to a given key agreement scenario. We can increase the level of security by agreement of the tolerance parameter in advance (then the hash value in a ACT will be longer). The scheme should be composed into a protocol which provides integrity of messages (as the IKE protocol does), otherwise it needs additional authenticate code to protect the hash value and the protocol parameters. As presented in this paper, our solution is fast, scalable, secure and can be integrated with existing protocols in an easy way.

Acknowledgement: Research reported in this paper was partially supported by the 7FP NoE EuroNF project.

References

[1] Maughan D., Schertler M., Schneider M., Turner J., Internet Security Association and Key Management Protocol (ISAKMP), RFC 2408, November (1998).

[2] IEEE, IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems (2008).

[3] Barak B., Halevi S., Herzberg A., Naor D., Clock synchronization with faults and recoveries (extended abstract), Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing, Portland, Oregon, United States July 16-19 (2000): 133.

[4] Dias J.S., Demetrio D.B., Custodio R.F., De Rolt C.R., Reliable clock synchronization for electronic documents, in: Proceedings of III IEEE Latin American Network Management Systems (2003): 550.

[5] Mills D.L., Internet time synchronization: the Network Time Protocol, IEEE Trans. Communications COM- 39(10) (1991): 1482.

[6] Elson J., Girod L., Estrin D., Fine-grained network time synchronization using reference broadcasts, Proceedings of the 5th symposium on Operating systems design and implementation, Boston, Massachusetts, USA, December 09-11 (2002); DOI:10.1145/1060289.1060304.

[7] Maroti M., Kusy B., Simon G., Ledeczi A., The flooding time synchronization protocol, Proceedings of the 2nd international conference on Embedded networked sensor systems, Baltimore, MD, USA, November 03-05 (2004); DOI:10.1145/1031495.1031501.

[8] Johannessen S., Time synchronization in a local area network, Control Systems Magazine, IEEE 24(2) (2004): 61.

[9] Syed A. A., Heidemann J., Time Synchronization for High Latency Acoustic Networks, INFOCOM 2006. Proc. 25th IEEE International Conference on Computer Communications, April (2006): 1; DOI:10.1109/INFOCOM.2006.161.

[10] Iwanicki K., van Steen M., Voulgaris S., Gossip-Based Clock Synchronization for Large Decentralized Systems, Springer-Verlag, Berlin Heidelberg, Lecture Notes in Computer Science 3996 (2006): 28.

[11] Manzo M., Roosta T., Sastry S., Time synchronization attacks in sensor networks, Proceedings of the 3rd ACM Workshop on Security of Ad hoc and Sensor Networks, Alexandria, VA, USA, November 07 (2005): 107.

[12] Ganeriwal S., Capkun S., Han Ch-Ch., Srivastava M.B., Secure time synchronization service for sensor networks, Proceedings of the 4th ACM Workshop on Wireless Security, Cologne, Germany, September 02 (2005): 97.

[13] Li H., Chen K., Wen M., Zheng Y., A Secure Time Synchronization Protocol for Sensor Network, In: T.Washio, Z.H.Zhou, J.Z.Huang, X.Hu, J.Li, C.Xie, J.He, D.Zou, K.-C.Li, M.M.Freire (eds.) PAKDD 2007, Springer-Verlag, Berlin Heidelberg, Lecture Notes in Computer Science (LNAI) 4819 (2007): 515.

[14] Sun K.,Ning P., Wang C., Secure and resilient clock synchronization in wireless sensor networks, IEEE J. Selected Area Comm. 24(2) (2006): 395.

[15] Sundararaman B., Buy U., Kshemkalyan A. D., Clock synchronization for wireless sensor networks: a survey, Ad Hoc Networks 3(3) (2005): 281.

[16] Boukerche A., Turgut D., Secure time synchronization protocols for wireless sensor networks, IEEE Wireless Communications 14(5) (2007): 64.

[17] Gong L., Variations on the Themes of Message Freshness and Replay or, the Difficulty of Devising Formal Methods to Analyze Cryptographic Protocols, 6th IEEE Comp. Security Foundations Workshop (1993): 131.

[18] Liang Z., Verma R. M., Complexity of Checking Freshness of Cryptographic Protocols, In: R. Sekar and A.K. Pujari, [Eds.], Proceedings of the 4th international Conference on information Systems Security (Hyderabad, India, December 16-20, 2008), Springer-Verlag, Berlin, Heidelberg, LNCS 5352 (2008): 86.

[19] Corin R. J., Analysis models for security protocols, Ph.D. Thesis, University of Twente (2006).

[20] Sharp R., Hansen M., Timed Traces and Strand Spaces, Proceedings of the 16th Nordic Workshop on Programming Theory (2004): 96.

[21] Harkins D., Carrel D., The Internet Key Exchange (IKE), RFC 2409 (1998).
[22] Kaufman C., Internet Key Exchange (IKEv2) Protocol, RFC 4306 (2005).
[23] Orman H., The Oakley key determination protocol, RFC 2412 (1998).
[24] Diffie W., Hellman M., New Directions in Cryptography, IEEE Transactions on Information Theory IT–22(6) (1977).
[25] ANSI X3.106, American National Standard for Information Systems-Data Link Encryption, American National Standards Institute (1983).
[26] Rivest R., The MD5 Message Digest Algorithm, RFC 1321 (1992).
[27] NIST, Secure Hash Standard, FIPS 180-1, National Institue of Standards and Technology, U.S. Department of Commerce, May (1994).
[28] Frier A., Karlton P., Kocher P., The SSL 3.0 Protocol, Netscape Communications Corp., Nov 18 (1996).
[29] Dierks T., Allen C., The TLS Protocol Version 1.0, RFC 2246 (1999).
[30] Kent S., IP Encapsulating Security Payload (ESP), RFC 4303 (2005).
[31] Ashton P., Algorithms for off-line clock synchronization, Technical Report TR COSC 12/952 Department of Computer Sciences University of Canterbury, December (1995).
[32] ISO/IEC FCD 29192-1, May (2011).
[33] Adjih C., Clausen T., Jacquet P., Laouiti A., Muhlethaler P., Raffo D., Securing the OLSR protocol, In Proceedings of Med-Hoc-Net, Mahdia, Tunisia, June 25-27 (2003).
[34] Lam K., Beth T., Timely Authentication in Distributed Systems, Proceedings of the Second European Symposium on Research in Computer Security, November 23-25 (1992): 293.
[35] Goyal V., Jain A., Quisquater J.-J., Improvements to Mitchell's Remote User Authentication Protocol, ICISC (2005): 69.
[36] Aslan H. K., Logical analysis of AUTHMAC_DH: A new protocol for authentication and key distribution, Computers & Security 23(4) (2004): 290.
[37] Li Y., Pang J., Extending the Strand Space Method with Timestamps: Part I the Theory, Journal of Information Security 1(2) (2010): 45.
[38] Li Y., Pang J., Extending the Strand Space Method with Timestamps: Part II Application to Kerberos V, Journal of Information Security 1(2) (2010): 56.
[39] Davis D., Geer D., Ts'o T., Kerberos with clocks adrift: History, protocols, and implementation, In Proceedings of the 5th USENIX UNIX Security Symposium, Salt Lake City, June (1995).
[40] Szałachowski P., Kotulski Z., Księżopolski B., Secure position-based selecting scheme for WSN communication, In: A. Kwiecien, P. Gaj, and P. Stera [Eds.], CN 2011, Communications in Computer and Information Science 160 (2011).
[41] Szałachowski P., Kotulski Z., Enhancing the Oakley key agreement protocol with secure time information, (Submitted for publication) (2011).