# Enhancing the Oakley key agreement protocol with secure time information

Pawel Szalachowski and Zbigniew Kotulski

Institute of Telecommunications, the Faculty of Electronics and Information Technology, Warsaw University of Technology, Warsaw, Poland
Email: p.szalachowski@stud.elka.pw.edu.pl, zkotulsk@tele.pw.edu.pl

***Abstract*—Message freshness and time synchronization are nowadays essential services in secure communication. Many network protocols can work correctly only when freshness of messages sent between participants is assured and when internal clocks protocol's parties 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 cryptographic protocols. As an example we apply our approach to the Oakley key determination protocol providing it with time synchronization without any additional communication overhead.**

***Index Terms*—freshness, security protocols, time synchronization, Oakley protocol, cryptographic protocols**

## I. Introduction

Freshness is the security property of data which is very important and desired in network communication. This property guarantees protection from variants of the replay attack. 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, strong freshness provides total messages 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 it is 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 key agreement protocol which is Oakley. Our extension of the Oakley protocol enables, except of standard secret key agreement by two parties, additionally synchronization of their clocks in a cryptographically secure way.

This work is supported by the National Science Center (NCN), under Grant with decision's number DEC-2011/01/N/ST7/02995

This paper is organized as follows. In Section II we present the related work and in Section III we shortly describe the Oakley protocol underlying a role of cookies for its functionality. In Section IV we introduce our time refreshment scheme and its implementation in the Oakley protocol. The analysis of security and performance of the approach presented is in Section V, while Sections VI and VII describe the applications of the new protocol and conclusions.

## II. Related work

In practical solutions [1] timestamps, counter values and pseudo-random numbers are used as freshness identifiers. In 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. in 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, what is a frequent practice). Another case is, e.g., IP Timestamp in Linux implementations. An attacker knows when a computer had been restarted last time, so he knows if the restart occurred after some critical system's 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 [2] where also complexity of checking freshness for many different scenarios is presented. Corin in [3] 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's messages (e.g., a timestamp). Next method for analyzing the security protocols with time aspects is presented in [4]. This paper analyses real-time properties of security protocols by a Strand Space-based approach.

Another crucial issue connected with time is time synchronization. Precise time is necessary in many

areas of our every day 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 [5]. It is especially important for applications which require the highest trust level (e.g. electronic documents). Barak in [6] describes an efficient and fault-tolerant clock synchronization method. This is especially important for network communication. The most widespread time synchronization protocol is NTP (The Network Time Protocol) [7], however, there are many different solutions for specific network environments [8], [9]. For example, the paper [10] presents a scheme of synchronization of a time-of-day clock in nodes of a local area network. In the paper [11] time synchronization solution for high latency acoustic networks is introduced. The paper [12] presents a time synchronization approach for large decentralized systems. Another example, which is the 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 sensor networks are presented in [13]. Therefore, these networks especially need secure and very efficient solutions, such as [14]–[16]. Surveys on time synchronization schemes in the WSNs are presented in [17], [18].

Protocol which connects freshness with time synchronization, but without actual time disclosure, would be very interesting and helpful in many applications.

## III. The Oakley protocol

Secure key agreement is a very actual and important task for network communications. The Oakley key determination protocol [19] is a generic key agreement protocol. It is widespread in Internet communication because it is often included in the IPsec protocol (more precisely, in the ISAKMP [20]). The goal of Oakley is establishment of a secret key between two parties communicating through an insecure channel. It is based on the Diffie-Hellman key agreement protocol but it has some additional advantages. The Oakley protocol is scalable and secure. Its main features are presented below:

- 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;
- Oakley allows 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;
- several options for the key computation are available;
- the parties can derive a new key from an existing one in a few ways, with aid of the Diffie-Hellman protocol or without;
- Oakley uses cookies to provide a mechanism which helps avoiding Denial of Service (DoS) attacks. This will be present it in details 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;

As we can see from the above, the Oakley protocol is very powerful and flexible. However, in spite of that it fulfills its usual duties, it may be enhanced with additional functions. Since, as many other popular cryptographic protocols, it omits strong freshness or time synchronization service, it can be extended with these security services.

The Oakley protocol defines two parties: Initiator and Responder. This is similar to the Client-Server architecture in messages exchange services. However, in Oakley the parties provide equal contents in the key negotiated. 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, Oakley includes several permanent elements. One of those obligatory elements is a cookie, which will be discussed now as an essential part of our freshness scheme.

### *Cookies*

The Oakley protocol is protected against some sort of DoS attacks. This is realized by anti-clogging tokens called cookies. The cookies 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 its execution they exchange the cookies to ensure that they are legitimate and they are interested in the protocol's execution. For both parties the cookies act as participants' identifiers and they rely on source addresses.

Another duty of the cookies is keys naming. In [19] the cookie of Initiator is denoted by $CKY\text{-}I$ and analogously, the cookie of Responder is $CKY\text{-}R$. A concatenation of these identifiers gives the key's

identifier $KEYID$. $KEYID$ is associated with the key shared by the parties and it is very important in further keying material usage and regeneration.

The Oakley standard defines several ways of executions of the protocol. The short one, called aggressive, needs only three messages exchanges, see [19]. Among other parameters that are sent in the protocol's steps, cookies are included in almost each message.

Cookies are present always in the Oakley's initial messages (sometimes they are null values), so calculation of their values should be quick and easy. Usually they are 64-bit pseudo-random numbers. They are connected with remote IP addresses of the protocol's users and they must be unique over some period of time, for each such an IP address. Because of these requirements, many implementations use the cryptographic hash functions for cookies generation.

In the Oakley protocol, the cookies are connected with time and freshness, but they do not provide any direct information about time. In our approach presented in the next section the cookies will be modified in such a way that they will retain their uniqueness property, but they will be additionally equipped with a fresh time information. As a consequence, the whole key agreement protocol will be enhanced with a new service.

## IV. Time refreshment protocol

Absolute time synchronization in a large network is almost impossible to achieve. In 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:

- $\|$ is the concatenation of two blocks of bits [1]
- $t_R$ is the actual Responder's (or Server's) time, assumed to be the reference time;
- $n$ is the time tolerance parameter;
- $f_n(.)$ is a function that converts each of the values $\hat{x} \in \{t_R - n, ..., t_R, ..., t_R + n\}$ to one value $\hat{f}_n = f_n(\hat{x})$ and which satisfies the following condition:

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

  for all integer $k$ such that $|k| > n$;
- $H(.)$ is a cryptographic hash function or a MAC scheme;
- $K$ is the secret key shared by Initiator and Responder;
- $IP_I$ is the IP address of Initiator (known to the both parties).

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

$$H_R = H(K\|IP_I\|n\|f_n(t_R)), \tag{2}$$

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

$$H(K\|IP_I\|n\|f_n(t_I)) \neq H_R, \tag{3}$$

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 a 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's parameter, it is not necessary to merge $K$ with the messages before the algorithm's execution.

*Protocol details*

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 the property given in (1) is satisfied. The auxiliary function $\hat{f}_n(.)$ can be defined as:

$$\hat{f}_n(x) = \begin{cases} \dfrac{x - r}{p}, & r \leq n \\ \dfrac{x + p - r}{p}, & r > n \end{cases} \tag{4}$$

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, as it is presented in Fig. 1. As we can see, the reference time must lie in the middle of the time range. The function $\hat{f}_n(.)$ is piece-wise constant. So, to pass time $t_R$ with tolerance $n$, we must shift the arguments of the function $\hat{f}_n(.)$ 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

$$f_n(t_R) = \hat{f}_n(t_R - \hat{o}) \tag{6}$$

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

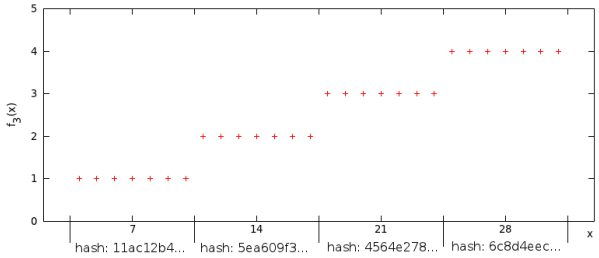


Fig. 1: Intervals of constant values of the output of function $f_n(.)$

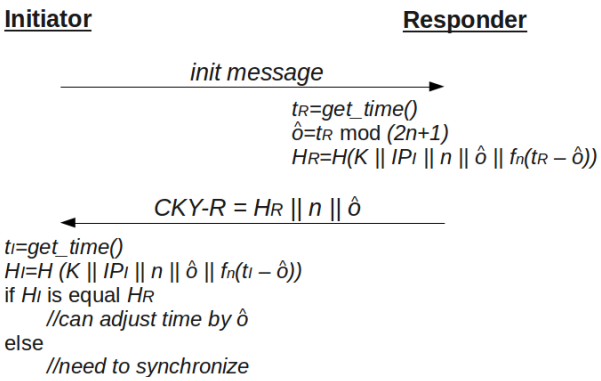


Fig. 2: Initial messages of the Oakley protocol with the time refreshment scheme included

and then the hash value of the concatenation of all the parameters required:

$$H_R = H(K\|IP_I\|n\|\hat{o}\|f_n(t_R)). \quad (7)$$

In this scheme, we additionally used the parameter $\hat{o}$, defined above. It is crucial for achievement property (1) of the function $f_n(.)$. 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.

*Implementation in the Oakley protocol*

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

$$CKY - R = H_R\|n\|\hat{o}. \quad (8)$$

In common scenarios our solution may be used as is presented in Fig. 2. The *init message* is generic for Oakley. Responder delivers a cookie with other Oakley's fields. The cookie consists of the hash, tolerance and offset values. Initiator produces another hash value from his local time (see Remark 1) and then checks if the hashes are equal. If they are, then his time is in a time range of Responder, so Initiator can adjust his own clock to Responder's time using the offset value. Otherwise, Initiator needs to synchronize his time by an external service (see Remark 2 and Remark 3 below). Note that Initiator knows his own IP address, but the tolerance and offset parameters are delivered in the cookie. Remaining steps of execution of the Oakley protocol are specific for a given scenario.

**Remark 1.** In our approach we treat transfer delays as negligible values. If these delays are significant then Initiator must take them into consideration, so the time of Initiator obtained by the system function $get\_time(.)$, before using a cryptographic hash function, should be adjusted according to the delays.

**Remark 2.** Our scheme also allows Initiator to make an attempt of the "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. [21]).

**Remark 3.** We rely on the Oakley protocol in ensuring integrity of the protocol's messages. So, when Oakley fails, the Initiator must not apply any changes in configuration of his system. We describe this aspect in Section V.

*Execution of the enhanced protocol*

Presentation of a complete Oakley's execution needs description of several additional elements of the protocol. Concatenation operation $\|$ and the hash function $H(.)$ already have been introduced. The rest of notation follows the Oakley's specification [19]:

- $ID(I)$, $ID(R)$ denote the identities of Initiator and Responder;
- $E_{K_I}(.)$ and $E_{K_R}(.)$ are the encryption using public key of Initiator and Responder;
- $EHAO$ is a list of encryption/hash/authentication methods choices offered;
- $EHAS$ is a set of three items selected from the list $EHAO$ (they are the methods for encryption/hash/authentication accepted);
- $T = OK\_KEYX$ represents a type of the message;
- $NIDP$ means that the identities are not encrypted;
- $N_I$ and $N_R$ are the nonces supplied by Initiator and Responder;
- $K_{IR} = H(N_I\|N_R)$;
- $sK_I$ and $sK_R$ are values which are included into the nonce fields of Initiator and Responder;
- $R'$ is the identity of Responder in the form of a plaintext;
- $KEYID = CKY\text{-}I\|CKY\text{-}R$ is the name of a secret key negotiated;

- $sKEYID$ denotes the keying material for the key called $KEYID$.

An example of the so-called aggressive version of the Oakley protocol with private identities and without Diffie-Hellman protocol is now considered [19]. This scenario is noteworthy if we need simplicity and high performance of the protocol. However, the perfect forward secrecy for the session key is not achieved in this mode. Exchange of messages in this version of the protocol is presented in Fig. 3. Let us now

| $I$ | *message* | $R$ |
|---|---|---|
| $\rightarrow$ | $CKY\text{-}I$, 0, $T$, 0, 0, $EHAO$, $NIDP$, $ID(R')$, $E_{K_{R'}}(ID(I)\|ID(R)\|sK_I)$, $N_I$ | $\rightarrow$ |
| $\leftarrow$ | $CKY\text{-}R$, $CKY\text{-}I$, $T$, 0, 0, $EHAS$, $NIDP$, $E_{K_I}(ID(R)\|ID(I)\|sK_R)$, $Nr$, $H(K_{IR}\|ID(R)\|ID(I)\|N_R\|N_I\|EHAS)$ | $\leftarrow$ |
| $\rightarrow$ | $CKY\text{-}I$, $CKY\text{-}R$, $T$, $EHAS$, $NIDP$, $H(K_{IR}\|ID(I)\|ID(R)\|N_I\|N_R\|EHAS)$ | $\rightarrow$ |

Fig. 3: Execution of the enhanced version of the Oakley protocol

focus on cookies. The first message contains only $CKY\text{-}I$ (0 in a message denotes a null value). Next, Responder produces $CKY\text{-}R$ in a way presented in Fig. 2 and sends it within the second message. When Initiator receives it, he is able to check if his time is synchronized (with a tolerance $n$). If it is, then the time value can be adjusted by $\hat{o}$ to the reference time. Otherwise, in case of a desynchronized clock, Initiator can detect this and adjust time using some external time-synchronization protocol, see e.g. [7]–[10].

Sometimes, during communication it is a necessity to obtain a new key from the old one. It should be realized in an easy and fast way. Oakley provides us such a mechanism, called the quick mode. Usually this operation is performed periodically so it is an excellent opportunity to synchronize time. However, in the standard protocol there is no field (like a cookie) to carry secure time information. Therefore we must add this short field to some existing message. The modified quick mode with a secure time information is presented in Fig. 4. After quick mode execution the parties have the new key computed as $H(sKEYID\|N_I\|N_R)$ and, moreover, Initiator is able to synchronize his clock (in the way presented in the previous example). The cookie of Responder is constructed in Eq. (7). Additionally, to authenticate the cookie, the value of $CKY\text{-}R$ is concatenated with other inputs of the hash function. It is realized in the second message. The quick mode is very fast, it requires only sending three messages and the calculations are also very simple.

| $I$ | *message* | $R$ |
|---|---|---|
| $\rightarrow$ | $KEYID$, $N_I$, $H(sKEYID\|N_I)$ | $\rightarrow$ |
| $\leftarrow$ | $KEYID, N_R$, $H(sKEYID\|1\|N_R\|N_I\|CKY\text{-}R)$, $CKY\text{-}R$ | $\leftarrow$ |
| $\rightarrow$ | $KEYID$, 0, $H(sKEYID\|0\|N_I\|N_R)$ | $\rightarrow$ |

Fig. 4: The Oakley's quick mode with a secure time information

## V. Analysis

A cookie is a quite short message. We want to include 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 cookie 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 approach presented 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 general attack against hashes is the birthday attack, which enables finding collisions [22] with the lowest computational effort. For the hash function $H(.)$ we define the collision as finding two messages $m_1$ and $m_2$ such that:

$$m_1 \neq m_2 \wedge H(m_1) = H(m_2). \tag{9}$$

TABLE I: Average number of hash calculations to obtain a collision with a given probability versus length of given parameters.

| Prob. | $n_l = 1$, $o_l = 2$, $h_l = 61$ | $n_l = 2$, $o_l = 3$, $h_l = 59$ | $n_l = 5$, $o_l = 6$, $h_l = 53$ | $n_l = 7$, $o_l = 8$, $h_l = 49$ | $n_l = 9$, $o_l = 10$, $h_l = 45$ | $n_l = 15$, $o_l = 16$, $h_l = 33$ |
|---|---|---|---|---|---|---|
| 25% | $1.15\times10^9$ | $5.76\times10^8$ | $7.20\times10^7$ | $1.80\times10^7$ | $4.50\times10^6$ | $7.03\times10^4$ |
| 50% | $1.79\times10^9$ | $8.94\times10^8$ | $1.12\times10^8$ | $2.79\times10^7$ | $6.98\times10^6$ | $1.09\times10^5$ |
| 75% | $2.53\times10^9$ | $1.26\times10^9$ | $1.58\times10^8$ | $3.95\times10^7$ | $9.88\times10^6$ | $1.54\times10^5$ |

Of course, the occurrence of collisions is very undesirable. In our case a collision is not very interesting for an attacker, it rather can mislead Initiator. Having unsynchronized time, he could obtain the same hash, so he could deduce that his clock is synchronized. In Table I we present how many (in average) hashes one must compute to obtain the collision with a given probability. This estimation depends on the parameters' lengths in a cookie message. In 8 octets we must place the values of: tolerance, offset and hash.

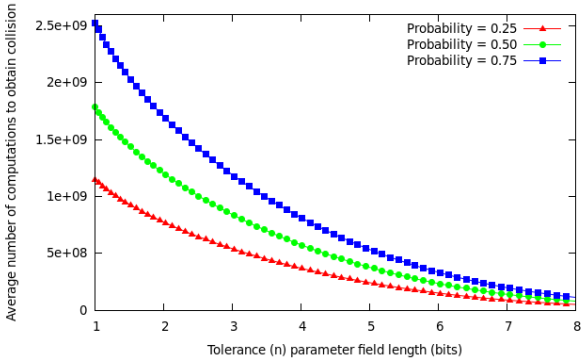


Fig. 5: Average number of hash calculations to obtain a collision with a given probability versus tolerance field length

So, in the cookie we define fields with fixed size. In Table I we denote by $n_l$, $o_l$ and $h_l$ the field lengths for the tolerance, offset and hash values. The lengths are expressed in bits. The most important for this consideration is the length of the tolerance parameter ($n_l$), because it determines the length of the offset field ($o_l$) and, as a consequence, the hash length ($h_l$). Basing on this information and on a communication profile we can evaluate the length of the field $n$ in a cookie. Additional visualization of how the tolerance field length influences the probability of collision is presented in Fig. 5.

For 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 finding 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 security parameters ($l$ is the length of $K$ and $l'$ is the length of the hash function output), but an attacker has no significant gain in collisions' calculation. The key $K$ should be at least 112-bits long for medium-term protection. The tolerance $n$ and the offset $\hat{o}$ are sent as a plaintext, 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 (7). 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 cookie, then Initiator states that his clock is desynchronized (even if it is not true). Initiator on this level cannot detect a malicious modification. But our scheme is embedded into the Oakley protocol, which provides integrity/authentication of messages, so any malicious modification would be detected in the following steps of the Oakley 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 cookie. In this case the attacker must perform the brute force attack in order to determine the values of parameters hashed. The time of a successful attack mainly depends on the value of $n$. This is because the larger size of parameter $n$ decreases the size of outputs of the function $f_n(.)$ (and, what it follows, the number of its possible values), so an attacker has to compute less hashes in average to break the protocol. The second important factor is the time resolution. The most popular size of actual time storing is 32 bits, but 16 bits and 64 bits variables are used too. Table II presents average numbers of hashes needed by an attacker to be computed in order to determine the reference time. We present it with a fixed time resolution and a fixed value of the tolerance parameter.

TABLE II: Average number of hashes needed to determine the reference time with a given time resolution and a given tolerance parameter length

| Time | $n = 2$ | $n = 6$ | $n = 15$ | $n = 40$ | $n = 100$ | $n = 250$ |
|---|---|---|---|---|---|---|
| 16bits | $8.2\times10^3$ | $2.7\times10^3$ | $1.1\times10^3$ | $4.1\times10^2$ | $1.6\times10^2$ | $6.5\times10^1$ |
| 32bits | $5.4\times10^8$ | $1.8\times10^8$ | $7.2\times10^7$ | $2.7\times10^7$ | $1.1\times10^7$ | $4.3\times10^6$ |
| 64bits | $2.3\times10^{18}$ | $7.7\times10^{17}$ | $3.1\times10^{17}$ | $1.2\times10^{17}$ | $4.6\times10^{16}$ | $1.8\times10^{16}$ |

Analyzing Table II and taking into account Table I we can adjust the parameter $n$ to achieve acceptable security level for our time refreshment protocol.

The memory, transmission and computational overheads of the scheme presented are negligible for Internet applications. Our solution uses a cryptographic hash function; generally, the most of implementations of ISAKMP use fast cryptographic hash functions (for cookies 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 Oakley protocol. For this reason, the computational overhead due to time freshness service is negligible.

## VI. Applications of the new protocol

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 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 [23], where the method described could be placed into

a *timestamp* message. Other examples of methods, which can be enhanced with time refreshment are authentication protocols like [24]–[26], where also *timestamp* fields occur.

There are many other applications where assurance of an actual time is required. One more good example could be Kerberos Protocol [27], [28], 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 the WSNs, see [29] or as freshness solution for key-agreement protocols [30].

## VII. Conclusions

In this paper we proposed a protocol which realizes time synchronization and data freshness between two parties. The approach described ideally fits cryptographic or secure communication protocols. Hence we decided to use the Oakley protocol as an example of its application. The Oakley protocol can be enhanced in an easy way, because of cookies contained in its messages. Moreover, such an extension gives no significant computational or memory overhead to the original protocol. Another crucial issue is security. It strongly depends on a cryptographic hash function chosen, traffic in the network and the tolerance parameter selected. As is shown in the 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 cookie will be longer). The scheme should be composed into a protocol which provides integrity of messages (as the Oakley protocol does), otherwise it needs additional authentication code to protect the hash value and the protocol's parameters. As is presented in this paper, our solution is fast, scalable, secure and it can be integrated with existing protocols in an easy way. It can be very useful, especially for secure content distribution approaches like [30].

## References


[1] L. Gong, "Variations on the themes of message freshness and replay - or the difficulty in devising formal methods to analyze cryptographic protocols," in *In Proceedings of the Computer Security Foundations Workshop VI*. IEEE Computer Society Press, 1993, pp. 131–136.

[2] Z. Liang and R. M. Verma, "Complexity of checking freshness of cryptographic protocols," in *Proceedings of the 4th International Conference on Information Systems Security*, ser. ICISS '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 86–101. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-89862-7_6

[3] R. J. Corin, "Analysis models for security protocols," http://eprints.eemcs.utwente.nl/1307/, Enschede, Netherlands, January 2006.

[4] R. Sharp and M. R. Hansen, "Timed traces and strand spaces," in *CSR 2007*, ser. LNCS, M. V. V. Dickert and A. Voronkov, Eds., vol. 4649. Springer-Verlag, 2007, pp. 373–386. [Online]. Available: http://www2.imm.dtu.dk/pubdb/p.php?5325

[5] K. Lee and J. Eidson, "Ieee-1588 standard for a precision clock synchronization protocol for networked measurement and control systems," in *In 34 th Annual Precise Time and Time Interval (PTTI) Meeting*, 2002, pp. 98–105.

[6] B. Barak, S. Halevi, A. Herzberg, and D. Naor, "Clock synchronization with faults and recoveries (extended abstract)," in *Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing*, ser. PODC '00. New York, NY, USA: ACM, 2000, pp. 133–142. [Online]. Available: http://doi.acm.org/10.1145/343477.343534

[7] D. L. Mills, "Internet Time Synchronization: the Network Time Protocol," *IEEE Transactions on Communications*, vol. 39, pp. 1482–1493, 1991. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.20.9287

[8] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," in *Proceedings of the 5th symposium on Operating systems design and implementation*, ser. OSDI '02. New York, NY, USA: ACM, 2002, pp. 147–163. [Online]. Available: http://doi.acm.org/10.1145/1060289.1060304

[9] M. Maróti, B. Kusy, G. Simon, and A. Lédeczi, "The flooding time synchronization protocol," in *Proceedings of the 2nd international conference on Embedded networked sensor systems*, ser. SenSys '04. New York, NY, USA: ACM, 2004, pp. 39–49. [Online]. Available: http://doi.acm.org/10.1145/1031495.1031501

[10] S. Johannessen, "Time synchronization in a local area network," *Control Systems Magazine, IEEE*, vol. 24, no. 2, pp. 61–69, 2004. [Online]. Available: http://dx.doi.org/10.1109/MCS.2004.1275432

[11] A. Syed and J. Heidemann, "Time synchronization for high latency acoustic networks," in *Proceedings of the IEEE Infocom*, Barcelona, Catalunya, Spain, April 2006. [Online]. Available: http://www-scf.usc.edu/ asyed/papers/tshl.pdf

[12] K. Iwanicki, M. van Steen, and S. Voulgaris, "Gossip-based clock synchronization for large decentralized systems." in *SelfMan*, ser. Lecture Notes in Computer Science, A. Keller and J.-P. Martin-Flatin, Eds., vol. 3996. Springer, 2006, pp. 28–42. [Online]. Available: http://dblp.uni-trier.de/db/conf/selfman/selfman2006.html

[13] M. Manzo, T. Roosta, and S. Sastry, "Time synchronization attacks in sensor networks," in *Proceedings of the 3rd ACM workshop on Security of ad hoc and sensor networks*, ser. SASN '05. New York, NY, USA: ACM, 2005, pp. 107–116. [Online]. Available: http://doi.acm.org/10.1145/1102219.1102238

[14] S. Ganeriwal, S. Čapkun, C.-C. Han, and M. B. Srivastava, "Secure time synchronization service for sensor networks," in *Proceedings of the 4th ACM workshop on Wireless security*, ser. WiSe '05. New York, NY, USA: ACM, 2005, pp. 97–106. [Online]. Available: http://doi.acm.org/10.1145/1080793.1080809

[15] H. Li, Y. Zheng, M. Wen, and K. Chen, "A secure time synchronization protocol for sensor network," in *Proceedings of the 2007 international conference on Emerging technologies in knowledge discovery and data mining*, ser. PAKDD'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 515–526. [Online]. Available: http://dl.acm.org/citation.cfm?id=1780582.1780638

[16] K. Sun and P. Ning, "Tinysersync: secure and resilient time synchronization in wireless sensor networks," in *ACM Con-*

*ference on Computer and Communications Security*. ACM Press, 2006, pp. 264–277.

[17] B. Sundararaman, U. Buy, and A. D. Kshemkalyani, "Clock synchronization for wireless sensor networks: A survey," *Ad Hoc Networks (Elsevier)*, vol. 3, pp. 281–323, 2005.

[18] A. Boukerche and D. Turgut, "Secure time synchronization protocols for wireless sensor networks," *Special Issue of IEEE Wireless Communications Magazine on Security in Wireless Mobile Ad Hoc and Sensor Networks*, vol. 14, no. 5, pp. 64–69, October 2007.

[19] H. Orman, "The oakley key determination protocol," RFC 2412, 1998. [Online]. Available: http://tools.ietf.org/html/rfc2412

[20] D. Maughan, M. Schertler, M. Schneider, and J. Turner, "Internet security association and key management protocol (isakmp)," RFC 2408, 1998. [Online]. Available: http://tools.ietf.org/html/rfc2408

[21] G. Lukas, A. Herms, and D. Mahrenholz, "Interval based off-line clock synchronization for wireless mesh networks," *SIGMETRICS Perform. Eval. Rev.*, vol. 35, pp. 10–12, December 2007. [Online]. Available: http://doi.acm.org/10.1145/1328690.1328695

[22] G. Yuval, "How to swindle rabin," *Cryptologia*, vol. 3, pp. 187–189, 1979.

[23] C. Adjih, T. Clausen, A. Laouiti, P. Mhlethaler, and D. Raffo, "Securing the olsr protocol," in *In 2nd IFIP Annual Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net 2003), Mahdia*, 2003, pp. 25–27.

[24] K.-Y. Lam and T. Beth, "Timely authentication in distributed systems," in *Computer Security - ESORICS 92, Second European Symposium on Research in Computer Security, Toulouse, France, November 23-25, 1992, Proceedings*, ser. Lecture Notes in Computer Science, Y. Deswarte, G. Eizenberg, and J.-J. Quisquater, Eds., vol. 648. Springer, 1992, pp. 293–303.

[25] V. Goyal, A. Jain, and J.-J. Quisquater, "Improvements to mitchell's remote user authentication protocol," in *ICISC*, 2005, pp. 69–80.

[26] "Logical analysis of authmac_dh: a new protocol for authentication and key distribution," *Computers & Security*, vol. 23, no. 4, pp. 290 – 299, 2004.

[27] Y. Li and J. Pang, "Extending the strand space method to verify kerberos v," in *Proceedings of the Eighth International Conference on Parallel and Distributed Computing, Applications and Technologies*, ser. PDCAT '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 437–444. [Online]. Available: http://dx.doi.org/10.1109/PDCAT.2007.43

[28] D. T. Davis and D. E. G. Jr., "Kerberos with clocks adrift: History, protocols, and implementation," *Computing Systems*, vol. 9, no. 1, pp. 29–46, 1996.

[29] P. Szalachowski, Z. Kotulski, and B. Ksiezopolski, *Secure Position-Based Selecting Scheme for WSN Communication*, ser. CCIS. Springer Berlin Heidelberg, 2011, vol. 160, pp. 386–397.

[30] P. Szalachowski and Z. Kotulski, "One-time broadcast encryption schemes in distributed sensor networks." *Special Issue of International Journal of Distributed Sensor Networks on Smart Sensor Networks: Theory and Practice*, 2012.

[31] D. Maughan, M. Schertler, M. Schneider, and J. Turner, "Internet security association and key management protocol (isakmp)," RFC 2408 (Proposed Standard), Internet Engineering Task Force, Nov. 1998, obsoleted by RFC 4306. [Online]. Available: http://www.ietf.org/rfc/rfc2408.txt

[32] P. Ashton and P. Ashton, "Algorithms for off-line clock synchronisation," 1995.