Broadcast Authentication Protocol with Time Synchronization and Quadratic Residues Chain

Bogdan Groza
Politehnica University of Timisoara, Romania
Faculty of Automatics and Computers
bogdan.groza@aut.upt.ro

Abstract

Assuring information authenticity is an important issue in the field of information security. A new broadcast authentication protocol is proposed. The protocol is based on time synchronization and uses chains constructed with the squaring function. The proposed solution is efficient for transmissions over long periods of time since the chains have an unbounded length. The protocol assures information authenticity at the reduced cost of almost one modular multiplication for each broadcasted packet. Time synchronization issues are discussed and the security of the protocol is equivalent to factoring since the squaring function is used. A failure mode analysis of the protocol is done; this is also an aspect of novelty and applies to other protocols based on time synchronization as well.

1. Introduction and related work

It is commonly acknowledged that authentication is one of the most important security objectives. One-way chains are arrays generated by the successive composition of a one-way function which can be efficiently used for assuring authentication. The first use of one-way chains is due to Lamport for assuring entity authentication [7], the S-Key system proposed by Haller is based on the same mechanism [6], however this systems is not secure [8].

More recently one-way chains proved to be useful in assuring information authenticity [2], [9], [10], [11], [12]. Usually information authenticity is assured by Message Authentication Codes (MAC), however MAC codes introduce a disadvantage since secret shared keys are required. The use of elements from a one-way chain as keys for MAC codes proved to be a good solution in order to remove this disadvantage. The most successful proposal for assuring information authenticity by using one-way chains is the Timed Efficient Stream Loss-tolerant Authentication (TESLA) protocol and its several variants proposed by Perrig et. al. [11], [12]. All variants of this scheme rely on loose time synchronization, which means that the receivers must have an upper bound on the time from the side of the sender. The principle is to use a key which is an element of one-way chain in order to compute a MAC and to disclose this key only in some forthcoming packet, the security condition which must be met to make this authentication secure is the following: a packet arrives safely if the receiver can unambiguously decide based on its synchronized time that the sender did not yet send the key disclosure packet. In brief the TESLA protocol offers authenticity at reduced costs without involving any shared secret between senders and receivers. For this advantage the protocol was suited even in constrained environments such as sensor networks. Different proposals of authentication protocols in which elements of a one-way chain are used as keys for MAC codes are in [2], [4], [5] - here an authentic confirmation is used instead of time synchronization. Also the CSA protocol from [2] has a timed variant T-CSA which is also based on time synchronization similar with the TESLA protocol. Probably the first authentication protocol based on a related principle is the Guy Fawkes protocol from [1].

In this paper we propose a new broadcast authentication protocol which is based on time synchronization and uses quadratic residue chains. The same loose time synchronization as in the case of the TESLA protocol is assumed but our proposal differs at the communication participants and at the construction of the chain. The advantage of this proposal is that it requires minimal interaction between senders and receivers, being efficient especially when there are many receivers, and that it can be used for broadcast over long periods of time since the squaring function is used which allows one-way chains of unbounded length. Also a new aspect is discussed: the presence of the failure modes.

Section 2 holds our proposal; details on time synchronization and cryptographic construction are given. In section 3 we analyze the introduced protocol both in terms of failure modes and security. Section 4 holds the conclusion of our paper.

2. The proposed protocol

2.1. Communication Participants

The addressed scenario assumes the existence of the following participants: a registration server and a number of senders and receivers (this possible setting has also been pointed out in [12]). Each sender establishes its initialization information on the registration server and then at some time later starts broadcasting authentic information. Additionally, if there is some clock drift between the sender and the registration server, the sender can synchronize again its time with the registration server (however this is not the main intention of our proposal). Each receiver obtains the initialization information of a particular sender from the registration server and then it can check the authenticity of the information that is broadcasted by that sender; we underline that except for receiving information that can be checked for authenticity there is no other interaction between senders and receivers. Again, to prevent clock drifts between receivers and the registration server; the receivers can synchronize their time with the registration server. As in the case of the TESLA protocol [11] only loose time synchronization is required which means that only an upper bound for the time value at the registration server is needed. The registration server does not have access to any private or secret information of senders or receivers; therefore it is not an unconditionally trusted entity. All that we request from the registration server is to be functionally secure, which means to behave honest. Its role is to provide time synchronization, to store sender's initialization information and to distribute it to receivers. We assume that this scenario can take place over a long period of time; for example a sender stores its initialization information on the registration server and then starts broadcasting for five years, in all this period there is no need for any other interaction between the sender and the registration server except for the case when the sender needs to synchronize its time with the registration server.

2.2. Registration of a sender on the registration server

The objective of sender S is to establish its initialization information on the registration server RS. This information consists in a packet $P_{init} = \left(t^{RS}_{broadcast}, S_{id}, n, k_0, \varepsilon_{S,RS}, \delta\right)_{SigS}$ signed by S. Here $t^{RS}_{broadcast}$ is the minimum time value at RS at which S starts broadcasting, below it is shown how to compute this value, S_{id} is an identifier for the sender (for example it may be some number or an IP address), n is the public modulus and k_0 is the initialization key, δ denotes the key disclosure period, $\varepsilon_{S,RS}$ is the time synchronization error computed as shown below and Sig_S denotes that the information is signed by S (as a general condition we assume that all the participants of this scenario can verify the signature of each other).

The registration procedure involves the following steps:

1. $S \rightarrow RS$: $Nonce_S$
2. $RS \rightarrow S$: $\left(Nonce_{RS}, Nonce_S, t^{RS}_{reg}\right)_{SigRS}$
3. $S \rightarrow RS$: $\left(Nonce_{RS}, t^{RS}_{broadcast}, S_{id}, n, k_0, \varepsilon_{S,RS}, \delta\right)_{SigS}$

Here $Nonce_S$ is a nonce used by S in order to ensure that the response from RS is not a replay of some old response and $Nonce_{RS}$ is a nonce used by RS to ensure that the registration information sent by S is also new, Sig_{RS} denotes that the information is signed by RS. The time delay between steps 1 and 2 is the synchronization error $\varepsilon_{S,RS}$ between S and RS, i.e. $\varepsilon_{S,RS} = t^S_{reg} - t^S_{start}$. We request for the synchronization error $\varepsilon_{S,RS}$ to be much smaller than the disclosure period δ, i.e. $\varepsilon_{S,RS} \ll \delta$, this is a natural requirement; a detailed explanation is in section 2.4. The registration procedure is also suggested in figure 1.

Now S can estimate for any time value t^S the minimum and maximum time value at RS as follows:

$$MinTV^{S,RS}\left(t^S\right) = t^S + t^{RS}_{reg} - t^S_{reg} \quad (1)$$

$$MaxTV^{S,RS}\left(t^S\right) = t^S + t^{RS}_{reg} - t^S_{reg} + \varepsilon_{S,RS} \quad (2)$$

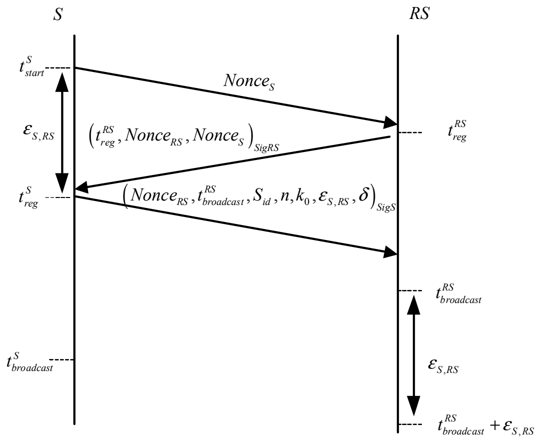
Let $t^S_{broadcast}$ be the time at which sender S starts broadcasting authentic information, now the minimum time value at RS when the time value at S is $t^S_{broadcast}$ can be easily computed as $t^{RS}_{broadcast} = MinTV^{S,RS}\left(t^S_{broadcast}\right)$. We will also define the disclosure time for the i^{th} key as:

$$DisT^S\left(i\right) = t^S_{broadcast} + \left(i-1\right)\cdot\delta \quad (3)$$

Because of the synchronization error $\varepsilon_{S,RS}$, by using relations (1) and (2) when the time value at S is $DisT^S(i)$ the time value at the registration server is somewhere in the interval $\left[MinTV^{S,RS}\left(DisT^S(i)\right), MaxTV^{S,RS}\left(DisT^S(i)\right)\right]$; since this is the time at which the i^{th} key is released we will call this interval the disclosure interval for the i^{th} key. What is important is that as long as the loose time synchronization is preserved between S and RS the i^{th} key is not released sooner than:

$$MDT^{RS}(i) = MinTV^{S,RS}\left(DisT^S(i)\right)$$
$$\Leftrightarrow MDT^{RS}(i) = t^{RS}_{broadcast} + (i-1)\cdot\delta \qquad (4)$$

We will call this time value the Minimal Disclosure Time (*MDT*) for the i^{th} key, *MDT* is of particular interest since as will be shown in the forthcoming sections the proposed protocol guarantees that packet P_i which contains a MAC computed with the $i+1^{th}$ key can not be forged sooner than $MDT^{RS}(i+1)$.



S
RS
t^S_{start}
$Nonce_S$
$\varepsilon_{S,RS}$
$\left(t^{RS}_{reg}, Nonce_{RS}, Nonce_S\right)_{SigRS}$
t^{RS}_{reg}
t^S_{reg}
$\left(Nonce_{RS}, t^{RS}_{broadcast}, S_{id}, n, k_0, \varepsilon_{S,RS}, \delta\right)_{SigS}$
$t^{RS}_{broadcast}$
$t^S_{broadcast}$
$\varepsilon_{S,RS}$
$t^{RS}_{broadcast} + \varepsilon_{S,RS}$

Figure 1. Registration procedure

2.3. Synchronization of a receiver with the registration server

The objective of the synchronization of a receiver R with the registration server RS is to obtain the initialization information of a particular sender S and to achieve loose time synchronization with the registration server, i.e. establish an upper bound on the time value at RS. This will make possible for R to check the authenticity of the information that is broadcasted by S.

The synchronization procedure involves the following steps:

1. $R \rightarrow RS$: $S_{id}, Nonce_R$
2. $RS \rightarrow R$: $\left(Nonce_R, t^{RS}_{sync}, P_{init}\right)_{SigRS}$

Here $Nonce_R$ is a nonce used by R in order to ensure that the response from RS is not a replay of some old response and S_{id} is the identifier of the particular sender from which R wants to receive authentic information. The time delay between steps 1 and 2 is the synchronization error $\varepsilon_{R,RS}$ between R and RS, i.e. $\varepsilon_{R,RS} = t^R_{sync} - t^R_{start}$. We will assume that $\varepsilon_{R,RS} + \varepsilon_{S,RS} << \delta$ and if this condition does not hold the synchronization procedure must be repeated; an explanation for this is given in section 2.4. This procedure is also suggested in figure 2.

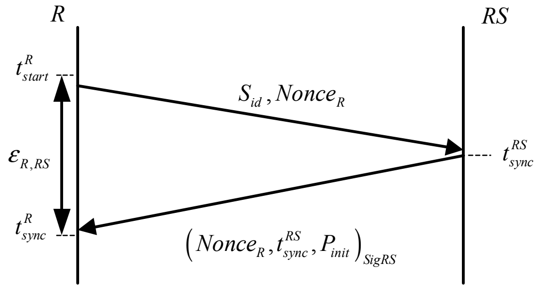


Figure 2. Synchronization procedure

After this synchronization R can also estimate at any time t^R the minimum and maximum value for the time value at RS:

$$MinTV^{R,RS}\left(t^R\right) = t^R + t^{RS}_{sync} - t^R_{sync} \qquad (5)$$
$$MaxTV^{R,RS}\left(t^R\right) = t^R + t^{RS}_{sync} - t^R_{sync} + \varepsilon_{R,RS} \qquad (6)$$

Now R can use the maximum time value at RS in order to decide if packet P_i received at time t^R_i which contains a MAC computed with the $i+1^{th}$ key is secure, i.e. the key used for the computation of the MAC was not already released. This can be verified by checking that:

$$MaxTV^{R,RS}\left(t^R_i\right) < MDT^{RS}(i+1) \qquad (7)$$

To prevent significant clock drifts the synchronization procedure can be periodically repeated by R.

2.4. The influence of the synchronization error on security

Because of the synchronization error $\varepsilon_{S,RS}$ between the sender and the registration server, key k_i is disclosed in the worst case when the time value at the registration server is $MaxTV^{S,RS}\left(DisT^S(i)\right)$. In this case a receiver having synchronization error $\varepsilon_{R,RS}$ with the registration server knows that the time value at the registration sever is at most $MaxTV^{S,RS}\left(DisT^{S}\left(i\right)\right)+\varepsilon_{R,RS}$. Also we must take into account the network delay for a particular receiver R, i.e. the time needed for a packet to travel through the network from S to R. Of course this time may vary for different packets but for our purpose it is sufficient to have an average value. Let this delay be Δ_R, now in order for the security condition to be verified when the packet arrives, we need:

$$MaxTV^{S,RS}\left(DisT^{S}\left(i\right)\right)+\varepsilon_{R,RS}+\Delta_R < MDT^{RS}\left(i+1\right)$$

$$\Rightarrow \varepsilon_{S,RS}+\varepsilon_{R,RS}+\Delta_R < \delta \qquad (8)$$

And therefore relation (8) needs to be satisfied in order for the receiver to obtain authentic packets at a delay Δ_R and synchronization errors $\varepsilon_{S,RS}$, $\varepsilon_{R,RS}$; this is why we have requested for the synchronization errors to be much smaller than the key disclosure period. If δ is chosen by the sender too small to satisfy (8) then the receiver will get only packets that must be dropped since the security condition (7) does not hold. In order to overcome this, an improvement that was proposed by Perrig et. al. in the case of the TESLA protocol [12] can be also used here: the key which is used to compute the MAC from packet P_i can be disclosed only in some later packet $P_{i+\tau}$ instead of packet P_{i+1}, see [12] for details.

2.5. Synchronization between a sender and the registration server

Assuring that clock drifts between the sender and the registration server are negligible is critical for the security of the communication. In order for the security of the scheme to hold, the sender must ensure that at any time t^S the time value at the registration server side is between the minimum and maximum values given in (1), (2), this is required in order to disclose the keys in the correct disclosure intervals. If the sender suspects that clock drifts between its clock and the registration server clock are not negligible then there are two possible solutions to overcome this. The first solution is for the sender to repeat the registration procedure and to commit new initialization information on the registration server (this means to restart the entire protocol), however this solution is inefficient for receivers that have already obtained the registration information of the sender. The second solution is for the sender to re-synchronize its time with the registration server. At time t^S the sender can estimate that the time value at the registration server is between $MinTV^{S,RS}\left(t^S\right)$ and $MaxTV^{S,RS}\left(t^S\right)$ by using (1), (2). In order to achieve a new synchronization the sender has to follow the synchronization procedure described in section 2.3. Now the sender plays the role of a receiver and after completing the synchronization procedure it can estimate that the time value at the registration server is between $MinTV^{R,RS}\left(t^R\right)$ and $MaxTV^{R,RS}\left(t^R\right)$ by using (5), (6). We suppose that $\varepsilon_{R,RS}\le\varepsilon_{S,RS}$, this is needed in order for the new synchronization to be more accurate than the previous one. Now the sender can compute a time adjustment $\xi = MinTV^{R,RS}\left(t^R\right)-MinTV^{S,RS}\left(t^S\right)$ (here $t^S=t^R$ since the sender plays the role of the receiver) and use this adjustment by broadcasting packets at time $t^S_{broadcast}+\left(i-1\right)\cdot\delta+\xi$ instead of $t^S_{broadcast}+\left(i-1\right)\cdot\delta$. The case of $\varepsilon_{R,RS}>\varepsilon_{S,RS}$ should be avoided since in some situations the sender cannot be certain that its estimation is or not wrong compared to the new estimation (after following receiver's registration procedure); also, the best thing that the sender can do is to ensure that packets are not released too soon by applying the same adjustment methodology, however in some situations packets may be released too late causing receivers to drop them.

2.6. Construction of the key chain

The discrete power function, i.e. $f\left(x\right)=x^{\varepsilon} \bmod n$ is commonly used in public key cryptography, some of the most prominent public key encryption schemes are based on this function, for example RSA [13]. Recently the squaring function, i.e. $f\left(x\right)=x^{2} \bmod n$, was proposed for creating time-lock puzzles [14] (a cryptographic puzzle that can be solved only after a predetermined amount of time). The construction from [14] is based on the property of the squaring function that exponents can be reduced modulo the order of the group and therefore $f^{\eta}\left(x\right)=x^{2^{\eta}} \bmod n = = x^{2^{\eta} \bmod \phi(n)} \bmod n$. The same property can be exploited in our construction for creating one-way chains of unbounded length. Since the value of $f^{\eta}\left(x\right)=x^{2^{\eta} \bmod \phi(n)} \bmod n$ can be easily computed by first computing $e=2^{\eta} \bmod \phi\left(n\right)$ and then computing $k_0 = x_0^{\,e} \bmod n$ a one-way chain of unbounded length can be computed in this way (the computational complexity for computing a one-way chain based on a hash function depends linearly on the length of the chain while by the use of this function it depends only logarithmically on the length of the chain – because of the repeated square and multiply algorithm). Therefore for the proposed protocol we will define each session key as follows:

$$k_i = x_0^{2^{\eta-i} \bmod \phi(n)} \bmod n, i = 0,..\eta \qquad (9)$$

Here n is a large composite integer which is infeasible to factor and $\phi(n)$ is the Euler totient function which can be computed only if the factorization of n is known and x_0 is a random value chosen by the sender. By using this function the one-way chain becomes a chain of quadratic residues in Z_n. More, the elements of the one-way chain can be computed in a time memory trade as suggested in [5] and the computational time is significantly reduced to almost one modular multiplication. The time-memory trade is based on the fact that it is possible to compute the value of $f^{\eta-i}(x)$ in only one modular multiplication if the value of $f^{\eta-i-1}(x)$ is known; indeed $f^{\eta-i}(x) = f^{\eta-i-1}(x) \cdot f^{\eta-i-1}(x)$. Because of this, the chain of η elements can be split into smaller chains of λ elements. Instead of performing one modular exponentiation for every element of the chain a smaller chain of λ elements can be computed with only one modular exponentiation followed by $\lambda - 1$ modular multiplications.

High order of 2 in $Z_{\phi(n)}$ is indeed necessary for the security of the protocol, as pointed out in [13] one may choose carefully a modulus for this purpose, but still random values should give the desired values with overwhelming probability. So the use of random values should be safe in practice.

Although this function is more computational intensive than a hash functions and its output is larger it has the advantage that the chains can have extreme lengths without influencing the computational time and therefore the chain can be used for a long period of broadcast.

2.7. The verification of some key from the initialization key

Each new key k_i must be checked for authenticity by the receiver, this can be easily done if the receiver already has the a previous authentic key k_l by checking that $k_i = f^{i-l}(k_l)$. If a significant amount of time has passed from the moment when the sender has start broadcasting and a recent authentic key is not available for the receiver (in the worst case the receiver has only the initialization key received from the registration server, i.e. k_0), since each key has to be computed from the previous one with one modular multiplication, the verification of the current session key may require a significant number of modular multiplications. We want now to establish how fast the receiver can synchronize its key with the sender, i.e. verifying the session key for the current time interval based on the initialization key k_0. If the receiver starts computations at time t_{start}^{RS} (we suppose that it has received key k_i of the current time interval and without loosing generality we take the time value at RS as reference) then at some later time t^{RS} the number of verified keys is:

$$v_{keys} = \left(t^{RS} - t_{start}^{RS}\right) / t_{mul} \qquad (10)$$

Here t_{mul} is the computational time required for a modular multiplication on the receiver side. In order to synchronize its key with the current session key we need that the number of verified keys to be equal to the number of keys released by the sender which is:

$$r_{keys} = \left(t^{RS} - t_{broadcast}^{RS}\right) / \delta \qquad (11)$$

By putting relations (10) and (11) together we get the following:

$$v_{keys} = r_{keys} \Rightarrow t^{RS} = \frac{\delta}{\delta - t_{mul}} \cdot t_{start}^{RS} - \frac{t_{mul}}{\delta - t_{mul}} \cdot t_{broadcast}^{RS} \qquad (12)$$

Therefore the time after which the current session key is verified is:

$$\Delta_{recovery} = t^{RS} - t_{start}^{RS} = \frac{t_{mul}}{\delta - t_{mul}} \cdot \left(t_{start}^{RS} - t_{broadcast}^{RS}\right) \qquad (13)$$

This further simplifies if we consider that the broadcast starts at $t_{broadcast}^{RS} = 0$ and then t_{start}^{RS} is in fact the time elapsed after broadcast starts, under these circumstances relation (13) becomes:

$$\Delta_{recovery} = \frac{t_{mul}}{\delta - t_{mul}} \cdot t_{start}^{RS} \qquad (14)$$

For example after 2 years of broadcast at a disclosure period $\delta = 10$ seconds and $t_{mul} = 74 \times 10^{-6}$ seconds by using (14) the receiver needs $\Delta_{recovery} \approx$ 467 seconds $\approx$ 8 minutes. After this computation the receiver will need to recover every new session key with only one modular multiplication, i.e. $t_{mul} = 74 \times 10^{-6}$ seconds. This is not a great amount of time after 2 years of broadcast in order to synchronize the key chain of the receiver with that of the sender. However, if this could be a problem, as an improvement on this, the sender can refresh from time to time its initialization information from the registration server by renewing the initialization key k_0 with some recently disclosed key k_i, the registration procedure from section 2.2 can be used for this purpose by replacing k_0 with k_i and $t^{RS}_{broadcast}$ with $MDT^{RS}(i)$, new receivers may use the new initialization packet while the old initialization packet is still correct. From both (13) and (14) it can be easily seen that as δ increases the recovery time decreases.

2.8. Protocol description

The description of the protocol now easily follows from the previously described procedures.

The following **initialization stage** is required for both senders and receivers:

Sender: Establish the number of communication sessions η, the value of η can be computed as $\eta = T/\delta$ where T represents the duration of the entire transmission and δ represents the duration of the disclosure interval. For example for 5 years of broadcast with a key disclosed every 10 seconds we have $\eta = 5\cdot 365\cdot 24\cdot 60\cdot 60\cdot 10^{-1} = 15768000 \simeq 16\times 10^6$. Choose two large primes p, q, such that it is infeasible to factor $p\cdot q$ and a random value x. Then compute $n = p\cdot q$, $\phi(n) = (p-1)\cdot(q-1)$, $k_0 = x_0^{2^\eta \bmod \phi(n)} \bmod n$ (this can be done efficiently by first computing $e = 2^\eta \bmod \phi(n)$ and then computing $k_0 = x_0^{e} \bmod n$). Use the registration protocol to establish the initialization packet $P_{init} = \left(t^{RS}_{broadcast}, S_{id}, n, k_0, \varepsilon_{S,RS}, \delta\right)_{SigS}$ on the registration server. Set the time adjustment $\xi = 0$ (see section 2.5 for details on the value of ξ).

Receiver: Use the time synchronization procedure from section 2.3 in order to obtain an upper bound on the time from the registration server's side and the initialization packet for a particular sender.

The **communication stage** is now described for both senders and receivers:

Sender: At time $t^{S}_{broadcast} + (i-1)\cdot\delta + \xi$ broadcast the packet $P_i = \left\{i, M_i, MAC_{KD(k_{i+1})}(M_i), k_i\right\}, i = \overline{1,\eta}$. Here M_i denotes the broadcasted message and k_i denotes the session key which is $k_i = x_A^{2^i \bmod \phi(n)} \bmod n$ (this computation can be easily performed in a time-memory tradeoff at the reduced cost of almost one modular multiplication, see [5] for details). As a potential improvement we also note that a sender may broadcast more than one packet authenticated with key k_{i+1} until this key expires, i.e. $MDT^{RS}(i+1)$. For example it can broadcast packets with the structure $P_i = \left\{i, j, M_{i,j}, MAC_{KD(k_{i+1})}(M_{i,j}, j), k_i\right\}, j = \overline{1,r}$, here r is the number of messages authenticated with the same key. $KD(k_{i+1})$ is a key derivation process used to derive the key of the MAC from the next session key.

Receiver: If packet P_i is received on time, which means that the security condition (7) holds, then store the message and the MAC otherwise drop them. Verify the authenticity of each session key by checking that $k_i = f^{i-l}(k_l)$, here k_l is the last authentic key that was received; in the worst case if no previous authentic key is available then use the key from the initialization packet, i.e. k_0. If the key is authentic then use it to verify the authenticity of the previously received packets.

2.9. Implementation aspects

Of course the proposed protocol can be also implemented with any other one way function, for example a hash function. We considered that the use of the squaring function is more suited for the addressed scenario (a long term broadcast protocol) since chains of unbounded length can be computed with this function.

Implementing the discrete squaring function is not difficult; there are many libraries which allow working with large integers, a good example is Java BigInteger class [15]. The disadvantage in using the squaring function f is that this function is more computational intensive and the size of the keys is also larger. In order to set a more accurate point of view in tables 1 and 2 some practical results on the time requirements of the squaring function and some hash function are given.

We conclude that although this function is more intensive than a hash function it is not unaffordable; a key of 1024 bits at requirements of microseconds for computing a key should be no great concern for many environments. It is obvious that this function can be successfully used in the proposed broadcast authentication protocol and the same property of the function that is used in [14] is also useful for the proposed scenario.

CPU	1024 bit module	1536 bit module	2048 bit module	Exponentiation
Intel Centrino 1.6 Ghz	74x10^{-6}s	168x10^{-6} s	281x10^{-6}s	50x10^{-3}s
Athlon 64 2800+ 1.8 Ghz	62x10^{-6}s	129x10^{-6}s	223x10^{-6}s	42x10^{-3}s
Athlon 64 3800+ 2.4 Ghz	46x10^{-6}s	96x10^{-6}s	167x10^{-6}s	32x10^{-3}s

Table 1. Time required for modular multiplication and exponentiation (time is expressed in seconds, exponentiation is done for 1024 bit module and exponent)

CPU	SHA1	SHA256	SHA512	MAC (SHA1)
Intel Centrino 1.6 Ghz	2.8x10^{-6}s	8.5x10^{-6}s	27x10^{-6}s	10x10^{-6}s
Athlon 64 2800+ 1.8 Ghz	2.4x10^{-6}s	7.1x10^{-6}s	25x10^{-6}s	9.4x10^{-6}s
Athlon 64 3800+ 2.4 Ghz	1.8x10^{-6}s	5.3x10^{-6}s	18.7x10^{-6}s	6.9x10^{-6}s

Table 2. Hash functions and MAC (time is expressed in seconds)

3. Failure modes and security analysis

Security tends to have a black and white image; a cryptographic protocol can or cannot be broken. However, in real life, applications may fail in more than one way, i.e. have more than one failure mode [3, page 116]. Usually a distinction is made between two kinds of failures: fail danger failure where the system moves into a dangerous condition which harms other systems and fail safe failure where the system moves to a safe condition without harming other systems. The proposed protocol, as well as other protocols based on time synchronization (such as the TESLA protocol) may fail because of clock drifts between the sender and the registration server. However there are two distinct situations which correspond to a fail safe failure or a fail danger failure. In the first situation the clock of the sender goes slower than that of the registration server, this corresponds to a fail safe failure since the secrets are disclosed too late when the receiver assumes that the authentication key was already released and therefore packets are dropped. In the second situation the clock of the sender goes faster than that of the registration server, this corresponds to a fail danger failure since the keys are released earlier and packets can be forged by an adversary. A further analysis of the failure modes is now done. The analysis is done for a sender and a receiver with synchronization errors $\varepsilon_{S,RS}$, $\varepsilon_{R,RS}$, a network delay Δ_R (the time needed for the packet to travel from the sender to the receiver) and by RT_i^{RS} we denote the time value at RS when S releases packet P_i. Assuming potential clock drifts between the sender and the registration server, we can now distinguish between five distinct intervals according to the time value at the registration server and the time value at the sender when the keys are released:

a) **Functional Interval.** This corresponds to the case when the keys are released in the correct time interval: $RT_i^{RS} \in \left[MDT^{S,RS}(i), MDT^{S,RS}(i)+\varepsilon_{S,RS}\right]$.

b) **Potential Communication Failure.** This is the case when the keys are released outside the functional interval causing potential packet drops from the receivers (a receiver with network delay smaller than Δ_R receives packets at correct time, however the keys are not released at the correct time): $RT_i^{RS} \in \left(MDT^{S,RS}(i)+\varepsilon_{S,RS}, MDT^{S,RS}(i+1)-\Delta_R-\varepsilon_{R,RS}\right)$.

c) **Communication Failure.** Packets are released too late and are certainly dropped by the receivers with network delay higher than Δ_R: $RT_i^{RS} \in \left[MDT^{S,RS}(i+1)-\Delta_R-\varepsilon_{R,RS}, +\infty\right)$.

d) **Potential Security Failure.** Packets are released too soon, an attacker can forge packets but if the receiver has a network delay Δ_R or higher the attacker can not forge packets on time (forged packets will be dropped by the receiver since they arrive too late): $RT_i^{RS} \in \left(MDT^{S,RS}(i)-\Delta_R-\varepsilon_{R,RS}, MDT^{S,RS}(i)\right)$.

e) **Security Failure**. Packets are released too son, an attacker can forge packets for any receiver with a network delay Δ_R or smaller: $RT_i^{RS} \in \left[MinTV^{S,RS}\left(t^S_{broadcast}\right), MDT^{S,RS}(i)-\Delta_R-\varepsilon_{R,RS}\right]$.

It is important to note that situations b) and c) correspond to a fail safe failure while situations d) and e) correspond to a fail danger failure. Now it can be easily seen that the sender needs to keep its clock inside the functional interval. This is reinforced by the synchronization procedure described in section 2.5, also in the case when $\varepsilon_{S,RS} < \varepsilon_{R,RS}$ the sender can prevent a fail danger failure but however it can enter into a fail safe failure.

Assuming that there are no significant clock drifts between the participants (i.e. situation a), which implies that the keys are released in the correct time interval) in order to break the protocol the only thing that an attacker could do is to compute a packet $P_i = \left\{i, M_{att}, MAC_{KD(k_{att})}\left(M_{att}\right), k_i\right\}$ where k_{att} is the key forged by the attacker and $f\left(k_{att}\right) = k_i$. Assuming that this packet arrives on time it will prove to be authentic when packet P_{i+1} containing key k_{att} is received. If

$f(k_{att}) = k_i$ then k_{att} is the square root of k_i and since the modulus is the product of two prime numbers then there are exactly four square roots of k_i. When the sender releases the key k_{i+1} then $k_{i+1} \neq \pm k_{att} \bmod n$ with probability ½ and therefore with probability ½ the attacker can factor the modulus (it is commonly known that computing modular square roots is equivalent to factoring). This means that if an attacker manages to forge the scheme by computing the key k_{att} then it can solve the integer factorization problem, since this is infeasible to solve the security of the protocol holds.

4. Conclusions

A new broadcast authentication protocol was proposed. The solution is based on time synchronization and uses the squaring function for computing the one-way chains; this leads to the potential use for a broadcast over a long period of time since the length of the chain is unbounded. An analysis of the time synchronization issues was done and the security of the protocol is proved to be equivalent to factoring. Also an analysis of the failure mode was presented; this is an interesting aspect that applies to other protocols based on time synchronization too. This solution may be useful in assuring the authenticity of information broadcasted over public networks such as the Internet for long time periods. As future work we are interested in potential applications of the proposed solution.

Acknowledgements: This work was partially supported by national research grant MEDC-CNCSIS TD-122/2007.

5. References

[1] R. Anderson, F. Bergadano, B. Crispo, J.H. Lee, C. Manifavas, R. Needham, "A New Family of Authentication Protocols", ACM OSR, 1998.

[2] F. Bergadano, D. Cavagnino, B. Crispo, "Individual Authentication in Multiparty Communications". Computer & Security, Elsevier Science, vol. 21 n. 8, 2002, pp.719-735.

[3] Bentley, J. P., *An Introduction to Reliability and Quality Engineering*, Addison Wesley, ISBN 0201331322, 216 pages, 1998.

[4] B. Groza, "Using one-way chains to provide message authentication without shared secrets", Second International Workshop on Security, Privacy and Trust in Pervasive and Ubiquitous Computing, SecPerU 2006, IEEE, 2006.

[5] B. Groza, T.-L. Dragomir, D. Petrica, "Using the discrete squaring function in the delayed message authentication protocol", International Conference on Internet Surveillance and Protection, ICISP'06, IEEE, 2006.

[6] N. Haller, C. Metz, P. Nesser, M. Straw, "A One-Time Password System", RFC 2289, Bellcore, Kaman Sciences Corporation, Nesser and Nesser Consulting, 1998.

[7] L. Lamport, "Password Authentication with Insecure Communication", Communication of the ACM, 24, 770-772, 1981.

[8] C. J. Mitchell, "Remote user authentication using public information", 9th IMA International Conference on Cryptography and Coding, LNCS 2898, 2003, pp.360-369.

[9] A. Perrig, R. Szewczyk, V. Wen, D. Culler, J.D. Tygar, "SPINS: Security Protocols for Sensor Network", Proceedings of Seventh Annual International Conference on Mobile Computing and Networks MOBICOM, 2001.

[10] A. Perrig, , "The BiBa one-time signature and broadcast authentication protocol", Proc. of ACM Conference on Computer and Communications Security, 2001, pp.28-37.

[11] A. Perrig, R. Canetti, J. D. Tygar, D. Song, "The TESLA Broadcast Authentication Protocol", In CryptoBytes, 5:2, Summer/Fall, pp. 2-13, 2002.

[12] A. Perrig, R. Canetti, J. D. Tygar, D. Song, "Efficient Authentication and Signing of Multicast Streams Over Lossy Channels", IEEE Symposium on Security and Privacy, 2000.

[13] R. Rivest, A. Shamir, L. Adleman, „A method for obtaining digital signatures and public-key cryptosystems", Communications of the ACM, 1978.

[14] L Rivest, A. Shamir, D.A. Wagner, "Time-lock puzzles and timed-release Crypto", available at http:// theory. lcs.mit.edu/~rivest/publications.html.

[15] Java.sun.com: The Source for Java Developers, http:// java.sun.com/.