Broadcast Authentication with Practically Unbounded One-way Chains

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

Abstract— **A protocol for assuring the authenticity of information broadcasted over long periods of time is proposed. The protocol is based on time synchronization and uses one-way chains constructed with the squaring function which gives the possibility to construct a one-way chain of whose length is unbounded in practice. Although the computational cost is somewhat increased, compared to the use of hash chains, these computational requirements are affordable for the addressed scenario. In brief, 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 the integer factorization problem since the squaring function is used in the construction of the one-way chain. A failure mode analysis of the protocol is done; this is an aspect of novelty and applies to other protocols based on time synchronization as well. Also, a formal proof on the security of the protocol is sketched.**

Index Terms — **authentication, broadcast, one-way chain, protocol.**

I. Introduction

It is commonly acknowledged that authentication is one of the most important security objectives. Although authentication comes at a lower price compared to other security objectives, since for example message authentication codes are cheap cryptographic primitives compared to encryption function (required to assure confidentiality) or digital signatures (required to assure non-repudiation), real world scenarios can not be solved by the straightforward use of these primitives. A good example is a broadcast scenario where multiple entities receive the same messages from a sender. The problem that occurs is the fact that message authentication codes require secret shared keys, and therefore a potential sender will need to share a distinct secret key to each receiver, and more, compute and send a distinct message authentication code for each receiver, even if the message is the same.

Fortunately a good solution emerged for this problem, the use of authentication protocols based on one-way chains and time synchronization proposed by Perrig et al. [20]. One-way chains are arrays generated by the successive composition of a one-way function. Usually in practice, for computational efficiency, a hash function is used for this purpose. Such protocols prove to be a versatile solution that comes at reduced computational costs and offer security properties that are close to the schemes that use expensive public key operations.

In this paper we extend one of our previous proposals of broadcast authentication protocol based on time synchronization and quadratic residue chains which has the advantage that can be used for long periods of time since the chain is unbounded for practical use. More concrete, the same loose time synchronization as in the proposal of Perrig et al. [20] is used but our proposal differs at the communication participants and more relevant at the construction of the chain. The advantages of this proposal are: first it requires minimal interaction between senders and receivers, being efficient especially when there are many receivers, and secondly, it can be used for broadcast over long periods of time since the one-way chain that we use has an unbounded length in practice. Relevant extensions to our previous proposal from [11] consist in: a detailed analysis of lengths of periods for the chain that we use, a sketch on a formal proof of security, a complete description of the protocol with details on the parameters setup and the short discussion on the presence of the failure modes is extended.

The paper is organized as follows. Section 2 inspects some related work. Section 3 holds the general setting of our scenario while section 4 describes time synchronization issues and section 5 presents the construction of the one-way chain. In section 6 the complete description of the protocol is given. Section 7 is a discussion on failure modes and section 8 gives a formal proof of security for the proposed protocol. Section 9 holds the conclusion of our paper.

II. Related Work

The history of one-way chains begins with the work of Lamport [14] which proposed the use of elements from a one-way chain as one-time passwords in order to authenticate a user to a remote system. Later, this proposal was used in the S-Key system by Haller [12]. However this systems is not secure [16], and the limitation in using Lamport's scheme in a real-world scenario is obvious: since it provides unilateral authentication, an adversary impersonating the real system can receive and store passwords that are not yet

used for subsequent impersonation of the user (this is known as the pre-play attack).

The great success of one-way chains begins with the work of Perrig et al. which used them to assure information authenticity [17], [18], [19], [20]. Message Authentication Codes (MAC) are the cryptographic primitive that is used for this purpose, but MAC codes come with the disadvantage that they require a secret shared key between the sender and each receiver. Using elements from a one-way chain as keys for MAC codes is a good solution to remove this disadvantage. Briefly, this advantage can be explained as follows: the MAC is secure if the key of the MAC is disclosed only after the MAC is received, and since each element of a one-way chain serves as a commitment for the following element of the chain, after the disclosure of the key, a new MAC can be computed with the forthcoming element and so on.

The most successful proposal based on this principle is the Timed Efficient Stream Loss-tolerant Authentication (TESLA) protocol proposed by Perrig et al. [19]. Several variants are proposed, all of them relying 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 a 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 [17]. Also, some related proposals based on similar principles can be found in [15].

Different proposals of authentication protocols in which elements of a one-way chain are used as keys for MAC codes are in [3], [9], [10] - here an authentic confirmation, which is also an element from a one-way chain, is used instead of time synchronization. Also, probably the first protocol based on this principle was the Guy Fawkes protocol from [1].

III. General Setting for the Protocol

A. 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 [20]). 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 [19] 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.

B. Registration of a sender on the registration server

The objective of sender S is to establish his 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 when 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 4). The registration procedure is also suggested in figure 1.

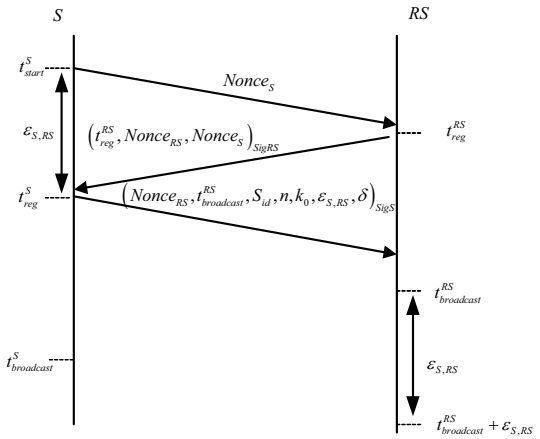

Figure 1. Registration procedure

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:

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

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:

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

We will call this time value the Minimal Disclosure Time (*MDT*) for the i^{th} key, *MDT* is of particular interest since 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 $MDT^{RS}(i+1)$.

C. 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 4). This procedure is also suggested in figure 2.

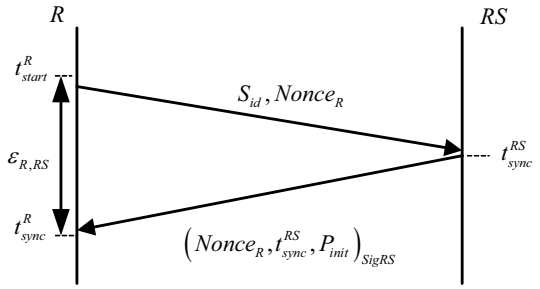

Figure 2. Synchronization procedure

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

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

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

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:

$$MaxTV^{R,RS}\left(t_i^R\right) < MDT^{RS}\left(i+1\right) \quad (7)$$

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

IV. TIME SYNCHRONIZATION ISSUES

A. 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 $MaxTV^{S,RS}\left(DisT^S\left(i\right)\right)$. 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:

$$MaxTV^{S,RS}\left(DisT^S\left(i\right)\right)+\varepsilon_{R,RS}+\Delta_R < MDT^{RS}\left(i+1\right)$$
$$\Rightarrow \varepsilon_{S,RS}+\varepsilon_{R,RS}+\Delta_R < \delta \quad (8)$$

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 [19] 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 [19] for details.

B. 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 protocol 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 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 synchronization procedure); also, the best thing that the sender can do is to ensure that packets are not released too soon by applying an adjustment computed in the same way, however in some situations packets may be released too late causing receivers to drop them.

V. CONSTRUCTION OF A KEY CHAIN THAT IS UNBOUNDED IN PRACTICE

A. General construction of one-way chains

As already stated, protocols based on one-way chains offer good computational advantages while preserving security properties that are close to protocols based on expensive public key operations. Basically, in order to construct a one-way chain, a one-way function is required. This is a non-restrictive condition since all cryptographic primitives, such as hash functions, encryption functions or digital signatures, behave as one-way functions. As a trivial example, one may use an encryption function and set $f\left(x\right)=E_x\left(0\right)$ then successively compute $f\left(r\right), f^2\left(r\right), f^3\left(r\right)$ and so on, for some random value r, in order to generate a one-way chain.

Because of their computational efficiency, since they are the cheapest cryptographic primitives, hash functions are the best choice for this purpose. The only limitation that occurs in using hash functions is that the length of the chain is fixed and the computational complexity depends linearly on the length of the chain. In order to improve on the computational requirements for the traversal of hash chains several time-memory tradeoffs

were proposed [7], [8], [24]. The basic principle on which these optimizations are based is to compute the hash chain and store some values of the chain for faster subsequent re-computation.

The solution proposed in the following section removes the disadvantage of fixed length, by providing a one-way chain that is unbounded in practice, and also makes it possible to compute values at any given index without requiring additional storage. However these advantages come with an increase in the computational complexity for the one way function that we use, but finally the increase in computational requirements is still affordable for the targeted scenarios.

B. The use of the squaring function

Function $f(x) = x^{\varepsilon} \bmod n$, where usually $n = p \cdot q$ is the product of two large primes p, q, is commonly used in public key cryptography. The most well known proposals for using this function are the RSA and Rabin cryptosystems [21], [22] which use different values for the exponent ε.

Later, the particular case of $f(x) = x^2 \bmod n$, i.e. the squaring function, was proposed by Blum et al. [5], [6] in order to construct a pseudorandom number generator that has its security equivalent to the integer factorization problem. Several facts that are established in [6] about this function will serve to our proposal as well.

Even more recently, the squaring function was proposed for creating time-lock puzzles [23]. These are cryptographic constructions that can be used for "sending information into future" and are close related to cryptographic puzzles used to prevent denial of services attacks. The time-lock puzzles from [23] have the tremendous advantage that they can be solved only after a predetermined amount of time, without giving a potential solver the ability to parallelize the solving process due to the intrinsic sequential nature of the repeated squaring process.

Both the proposals from [6], [23] exploit the same property of the function that we will use in our protocol. Namely, we use the fact that while working over groups of integers, exponents can be reduced modulo the order of the group. Therefore, the result of the successive composition of the squaring function can be efficiently computed as $f^{\eta}(x) = x^{2^{\eta}} \bmod n = \; = x^{2^{\eta} \bmod \phi(n)} \bmod n$, here $\phi(n)$ is the Euler totient function which can be computed if and only if the factorization of n is known.

We can use the same property for creating one-way chains of unbounded length. Since the value of $f^{\eta}(x) = x^{2^{\eta} \bmod \phi(n)} \bmod n$ can be efficiently computed by first computing the exponent $e = 2^{\eta} \bmod \phi(n)$ and then computing $f^{\eta}(x) = x^{e} \bmod n$ a one-way chain of unbounded length can be computed in this way. We underline that 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 that can be used to perform modular exponentiation.

Therefore, for the proposed protocol we will define each session key, which is an element of the one-way chain generated by the squaring function, i.e. a chain of quadratic residues, as follows:

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

Here x_0 is a random value chosen by the sender. More, the elements of the one-way chain can be computed in a time memory trade as suggested in [10] 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 $f^{\eta-i}(x)$ with 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.

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 time of broadcast.

C. The length of period for the $x^2 \bmod n$ sequence

As stated in [23] looking for perfection in number theory may be considered overkill in this context. It is natural to expect that by choosing random values for x_0 and the modulus n we will obtain a sequence with a large period that will not lead to the loss of security. Therefore, the use of random values should be safe in practice and the analysis from this section can be skipped. However, for the completeness of the results a solution for choosing number that will not defile the expectancies is preferable. In [6] a good analysis of the length of period is presented, here we give a brief approach, which is related to the one from [6], for choosing numbers that offer good security properties.

It is obvious that we are concerned with two things, first the order of x_0 in Z_n, denoted by $ord_n(x_0)$, and second, the order of 2 in $Z_{\phi(n)}$, denoted by $ord_{\phi(n)}(2)$. We will use the same notation as in [6] for the length of the period, which will be denoted by π and represents the smallest number such that $x_0 = x_0^{2^{\pi}} \bmod n$, i.e.:

$$x_0 = x_0^{2^{\pi}} \bmod n, \neg\exists \pi' < \pi, x_0 = x_0^{2^{\pi'}} \bmod n \quad (10)$$

The first step, in order to a achieve a large length of period, is to obtain a high order of x_0 in Z_n. We can use a straight forward solution for this purpose, two random

numbers α and β that are generators in Z_p and Z_q respectively are chosen. Now by using the Chinese remainder theorem we compute the solution of the following system:

$$\begin{cases} x_{-1} = \alpha \bmod p \\ x_{-1} = \beta \bmod q \end{cases} \quad (11)$$

It is obvious that the order of x_{-1} in Z_n is $ord_n x_{-1} = lcm(p-1, q-1)$. Now we set $x_0 = x_{-1}^2 \bmod n$.

The second step is to obtain a high value for $ord_{\phi(n)}(2)$. The order of 2 in $Z_{\phi(n)}$ can be easily checked if the factorization of $\phi(n)$ is available. For this purpose we will use the same kind of primes as in [6], called special primes, for which it holds that $p = 2 \cdot p' + 1$, $p' = 2 \cdot p'' + 1$, here p, p', p'' are all prime numbers. Also, we make the same requirement for the modulus as in [6] and we request for n to be a special number which means that both p and q are special an also congruent to 3 mod 4 (this implies that each quadratic residue has exactly one root that is also a quadratic residue). If p and q are special primes, then the value of $\phi(\phi(n))$ is obviously $\phi(\phi(n)) = \phi((p-1)\cdot(q-1)) = \phi(4 \cdot p' \cdot q') = 8 \cdot p'' \cdot q''$. As the value of $\phi(\phi(n))$ and its factorization is known, the value of $ord_{\phi(n)}(2)$ can be easily computed. Even more, it is obvious that $ord_{\phi(n)}(2)$ is $p'' \cdot q''$ if 2 is a quadratic residue with respect to either p'' or q'' and $2 \cdot p'' \cdot q''$ otherwise; either ways, since n is an integer infeasible to factor, the value will be safe for practical use. The special primes p and q will be chosen in the same manner as suggested in [6] by choosing random numbers and using probabilistic primality testing then select the one that are special.

Finally, for the values established above, it is trivial to prove that the length of the period π will be equal to:

$$\pi = ord_{\phi(n)} 2 \quad (12)$$

This gives the desired length of period for the sequence – a length which is unbounded for practical use.

D. The verification of some key from the initialization key

We preserve our initial analysis from [11] for determining the computational time required to verify 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 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:

$$v_{keys} = \left(t^{RS} - t_{start}^{RS}\right) / t_{mul} \quad (13)$$

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:

$$r_{keys} = \left(t^{RS} - t_{broadcast}^{RS}\right) / \delta \quad (14)$$

By putting relations (13) and (14) together we get the following:

$$v_{keys} = r_{keys} \Rightarrow t^{RS} = \frac{\delta}{\delta - t_{mul}} \cdot t_{start}^{RS} - \frac{t_{mul}}{\delta - t_{mul}} \cdot t_{broadcast}^{RS} \quad (15)$$

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

$$\Delta_{recovery} = t^{RS} - t_{start}^{RS} = \frac{t_{mul}}{\delta - t_{mul}} \cdot \left(t_{start}^{RS} - t_{broadcast}^{RS}\right) \quad (16)$$

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 (16) becomes:

$$\Delta_{recovery} = \frac{t_{mul}}{\delta - t_{mul}} \cdot t_{start}^{RS} \quad (17)$$

For example after 2 years of broadcast at a disclosure period $\delta = 10$ seconds and $t_{mul} = 74 \times 10^{-6}$ seconds by using (16) the receiver needs $\Delta_{recovery} \approx 467$ seconds ≈ 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 3 can be used for this purpose by replacing k_0 with k_i and $t_{broadcast}^{RS}$ with $MDT^{RS}(i)$, new receivers may use the new initialization packet while the old initialization packet is still correct. From both (16) and (17) it can be easily seen that as δ increases the recovery time decreases.

E. Reducing the computational overhead by the use of delayed key chains

In order to reduce the computational overhead induced by the verification of a key from a key received long time before the use of delayed key chains can be a solution. This means that instead of using a single one-way chain we can use two chains with distinct disclosure delays. Namely, besides the one-way chain that we use for the authentic broadcast, another chain with a larger disclosure period can be used in order to authenticate the current key from the chain used for the authentic broadcast.

We note that this solution reduces the computational overhead at the cost of introducing a delay equal to the disclosure period of the second chain (since the client must now wait for the disclosure of the key from the second chain, and also verify this key, which can be done indeed faster). Also this solution can be generalized for more than two chains. The idea of using multiple one-way chains was also used in a related context in [15].

VI. THE PROPOSED PROTOCOL

A. Protocol description

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

The following initialization stage is required for senders and receivers:

- **Sender:** Establish the number of communication sessions η. In principle, 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. However, the proposed protocol is intended for long time periods, or even uncertain, therefore, due to the construction of the chain any value for η can be chosen. As an example, one may choose $\eta = 2^{128}$ which will result in a chain that will never exhaust in practice. Choose two large special primes p, q according to the specifications from section V, for this purpose random values will be chosen until special primes are found, this is going to happen in polynomial time as suggested in [6]. Compute x_0 by choosing two random generators from Z_p and Z_q as shown in section 5. 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 4 for details on the value of ξ).
- **Receiver**: Use the time synchronization procedure from section 4 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 k_{i+1}. $KD(k_{i+1})$ 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.

B. 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) 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 [26]. 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 properties of the function that are used in [6], [23] are also useful for the proposed scenario.

TABLE I.
TIME REQUIRED FOR MODULAR MULTIPLICATION AND EXPONENTIATION (TIME IS EXPRESSED IN SECONDS, EXPONENTIATION IS DONE FOR 1024 BIT MODULE AND EXPONENT)

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 II.
HASH FUNCTIONS AND MAC (TIME IS EXPRESSED IN SECONDS)

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

VII. FAILURE MODES AND SECURITY ANALYSIS

A. About fail safe and fail danger failures in real world applications

Real world applications may fail in more than one way, therefore we say that they have more than one failure mode [4]. A basic distinction can be made between two opposite kind of failures:

- Fail danger failure– is a failure in which the system moves into a dangerous condition which harms other systems that are connected to it.

- Fail safe failure – is a failure in which the system moves to a safe condition without harming other systems that are connected to it.

To make things clearer we consider a trivial example from the real world - the case of a boiler. A boiler is a closed vessel in which some fluid, usually water, is heated under pressure. If the pressure gets to high levels the boiler may explode potentially causing catastrophes. In order to prevent this, a safety valve is present. The safety valve is a security device; the role of the safety valve is to release pressure in order to prevent the potential explosion of the boiler. Now consider that the safety valve is implemented on some electronic circuit, such as a transistor or a diode. Obviously the circuit inside the safety valve may fail too. Let us assume that the electronic circuit may fail in open circuit, which causes the element output to rise to a high value, or may fail in short circuit which causes the element output fall to a low value. Now if we consider that a high output will cause the valve to stuck close and a low output will cause the valve to stuck open we get: a) a fail danger failure when the valve remains closed since obviously the boiler may explode b) a fail safe failure for the case when the valve remains opened since obviously there will not be the desired pressure inside the boiler.

As the notions of fail safe and fail danger failures are clear we proceed in the next section to analyze their presence in cryptographic protocols based on time synchronization.

B. Fail safe and fail danger failures in cryptographic protocols based on time synchronization

Security tends to have a black and white image; a cryptographic protocol can or cannot be broken. 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: $RT_i^{RS} \in \left[MinTV^{S,RS}\left(t_{broadcast}^{S}\right), MDT^{S,RS}(i) - \Delta_R - \varepsilon_{R,RS} \right]$.

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 4 and 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.

Also, we underline that establishing fail safe and fail danger failures for a system is relevant when the system becomes part of a larger system and the condition of the entire system needs to be established. In this context techniques such as Fault Tree Analysis (FTA) or Failure Mode and Effect Analysis (FMEA) can be used [4].

VIII. A SKETCH ON A FORMAL PROOF OF SECURITY

Proofs of security for related protocols can be found in [3], [20]. In our previous paper [11], for simplicity, only an informal argument on the security of the protocol was given. The security of the protocol can be proven to be equivalent to the integer factorization problem in the Random Oracle Model (ROM) [2]. Informally, it is obvious that in order to break the protocol an attacker must 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 $f\left(k_{att}\right) = k_i$ 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.

For the completeness of the result we now give a sketch on a formal proof for the security of the protocol. We underline that we assume that there are no significant clock drifts between the participants and the synchronization error is inside the prescribed margins (i.e. situation a) from section 7, which implies that the keys are released in the correct time interval), also we assume that the registration and initialization values are correct.

In what follows, we will use the notation $M, MAC_{KD(k_{-1})}\left(M\right), k$ where $k = f\left(k_{-1}\right)$. Obviously any packet sent by the broadcast protocol fits to this notation, in fact we renounced only to the index i. Also, the proof of security is given for an adaptive chosen ciphertext adversary against the indistinguishability of a MAC given a random key encapsulated with function f and some random value, i.e. IND-CCA2. The first approach that used indistinguishability to prove the security of such a protocol was the proof of security for the TESLA scheme in [20]. However, we take an approach that is more close to the proofs of security for the Key Encapsulation Mechanism from [2], [25]. Such a proof gives in fact a solid argument for security in a less constrained environment. Since both the MAC and the key derivation function behave as pseudorandom functions, we assume that $MAC_{KD(k_{-1})}\left(M\right)$ behaves as a random function as well and we use a random oracle to simulate this function. Now we assume the existence of an IND-CCA2 adversary against the cryptosystem $M, MAC_{KD(k_{-1})}\left(M\right), k$ and the following theorem establishes the equivalence between breaking this scheme and the integer factorization problem:

Theorem 8.1. If there exists an adversary A that can distinguish between $M, MAC_{KD(k_{-1})}\left(M\right), k$ and M, r, k with advantage ε by making q_D queries to a MAC oracle $\mathcal{O}$ that when given M, k returns $MAC_{KD(k_{-1})}\left(M\right)$, then the adversary can be used to factor integers with advantage $\varepsilon' > \frac{1}{2}\left(\varepsilon - \frac{q_D}{n}\right)$.

We sketch a proof for theorem 8.1. The MAC oracle $\mathcal{O}$ is simulated as follows: a *MACList* is preserved which is initially empty, if the attacker queries $\mathcal{O}$ with M, k_{-1} to obtain $MAC_{KD(k_{-1})}\left(M\right)$ then $k = k_{-1}^{\,2} \bmod n$ is computed and a new random value r is generated then the values M, k, r, k_{-1} are stored in the *MACList*, if the attacker queries $\mathcal{O}$ with M, k to obtain $MAC_{KD(k_{-1})}\left(M\right)$ then *MACList* is inspected and if M, k are found the corresponding values are returned, else a new random value r is generated and the triple M, k, r is stored in the *MACList*. If $\mathcal{O}$ is ever queried with M, k_{-1} such that $k = k_{-1}^{\,2} \bmod n$ and M, k are in the *MACList* then the corresponding value r from the *MACList* is returned. We underline that the MAC oracle $\mathcal{O}$ works in a similar fashion to the decryption oracle for the RSA-KEM in [25] to which we refer the reader for a more detailed proof.

Assuming the adversary outputs $\tilde{b}$ the advantage of the adversary is defined as $\varepsilon = \left| \Pr\left[b = \tilde{b} \right] - \frac{1}{2} \right|$. Let *Awins* denote the event that A correctly guesses the hidden bit, let *AskChall* denote the event that A asks for the challenge M, k in its find stage and *AskMAC* the event that A asks for M, k_{-1} with $k = k_{-1}^{\,2} \bmod n$. Let $\overline{Bad}$ denote the absence of *AskChall* and *AskMAC*;

since in this situation all that the adversaries knows is independent from the challenge we have $\Pr\left[Awins \cap \overline{Bad}\right] = \frac{1}{2}$. Since $\Pr[Awins] = \frac{1}{2} + \varepsilon$, and $\Pr[Awins] = \Pr\left[Awins \cap \overline{Bad}\right] + \Pr[Awins \cap Bad]$, it follows that $\frac{1}{2} + \varepsilon \le \frac{1}{2} + \frac{q_D}{n} + \Pr[AskMAC]$. Since the event $AskMAC$ implies that the square root of k is disclosed (and computing square roots is equivalent to factoring), we get the factoring advantage $\varepsilon' \ge \frac{1}{2}\left(\varepsilon - \frac{q_D}{n}\right)$.

IX. 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 large integers. Also an analysis of the failure modes 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 periods of time.

Acknowledgment

This work was supported in part by national research grant MEDC-CNCSIS TD-122/2007.

References

[1] R. Anderson, F. Bergadano, B. Crispo, J.H. Lee, C. Manifavas, R. Needham, "A New Family of Authentication Protocols", *ACM Operating Systems Review*, pp. 9-20, 1998.

[2] M. Bellare and P. Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols", *ACM Conference on Computer and Communications Security*, pp. 62–73, 1993.

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

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

[5] L. Blum, M. Blum and M. Shub, "Comparison of Two Pseudo-Random Number Generators", *Advances in Cryptology Proceedings of Crypto 82*, pp. 61-78, 1982.

[6] L. Blum, M. Blum and M. Shub, "A Simple Unpredictable Pseudo-Random Number Generator", *SIAM Journal on Computing*, Volume 15 , Issue 2 , pp. 364 – 383, 1986.

[7] D. Coppersmith and M. Jakobsson, "Almost Optimal Hash Sequence Traversal", *Proceedings of the Fifth International Conference on Financial Cryptography*, pp. 102-119, 2002.

[8] M. Fischlin, "Fast Verification of Hash Chains", *The Cryptographers Track at the RSA Conference*, pp. 339-352, 2004.

[9] 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*, IEEE Comp. Soc., pp. 82-87, 2006.

[10] 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*, IEEE Comp. Soc., 2006.

[11] B. Groza, "Broadcast authentication protocol with time synchronization and quadratic residues chains", *Second International Conference on Availability, Reliability and Security*, pp. 550-557, IEEE Comp. Soc., 2007.

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

[13] M. Jakobsson, "Fractal hash sequence representation and traversal", *Proceedings of IEEE International Symposium on Information Theory*, ISIT'02, pp. 437–444, 2002.

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

[15] D. Liu, P. Ning, "Efficient Distribution of Key Chain Commitments for Broadcast Authentication in Distributed Sensor Networks", *Proceedings of the 10th Annual Network and Distributed System Security Symposium*, 2002.

[16] C.J. Mitchell and L. Chen, "Comments on the S/KEY User Authentication Scheme", *ACM Operating Systems Review*, pp. 12-16, 1996.

[17] 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*, pp. 521-534, 2001.

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

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

[20] 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*, pp. 56-73, 2000.

[21] M. Rabin, "Digitalized signatures and public key functions as intractable as factorization", MIT/LCS/TR-212, MIT Laboratory for Computer Science, 1979.

[22] R. Rivest, A. Shamir, L. Adleman, „A method for obtaining digital signatures and public-key cryptosystems", *Communications of the ACM*, pp. 120-126, 1978.

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

[24] Y. Sella, "On the Computation-Storage Trade-offs of - Hash Chain Traversal", *Proceedings of the Seventh International Conference on Financial Cryptography*, pp. 270-285, 2003.

[25] V. Shoup, "A proposal for an ISO standard for public key encryption", Input for Committee, 2001.

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