Using the Discrete Squaring Function in the Delayed Message Authentication Protocol

Bogdan Groza, Toma-Leonida Dragomir, Dorina Petrica
Politehnica University of Timisoara, Faculty of Automatics and Computers
Bd. Vasile Parvan nr. 2, 300223 Timisoara, Romania,
Email: bogdan.groza@aut.upt.ro

Abstract

The Delayed Message Authentication Protocol (DeMA protocol) is a protocol that can be used in order to exchange authentic information between two entities by using a one-way chain (i.e. an array of elements generated by a one-way function) on each entity's side. The use of the discrete power function in the DeMA protocol offers the advantage that the length of the chains does not influence the computational time. In this paper we are concerned with the particular case of the discrete squaring function which offers more computational advantages since the elements of the one-way chains may be efficiently computed in a time-memory trade. The complete description of the DeMA protocol in the case of the discrete squaring function is given and also some experimental results are presented which help on understanding the computational performance of the protocol. By using this implementation of the DeMA protocol message authentication can be assured at the cost of almost one modular multiplication for each exchanged message.

1. Introduction

The use of one-way chains was initially proposed for assuring entity authentication [8] and a commonly known example of application that implements this technique is the S-Key system by Haller [7]. There are also a number of user authentication schemes based on one-time passwords which have distant relation to this technique [4], [9]. A one-way chain $f^0(x), f^1(x), f^2(x),..., f^i(x)$ is an array generated by the successive composition of a one-way function f. Here x is a secret value and $f^i(x)$ denotes the composition of f with itself for i times $f^i(x) = f\left(f^{i-1}(x)\right)$; of course, $f^0(x) = x$. We will define the length of this chain as the number of function compositions necessary to obtain the highest order element, therefore the length of the previous chain is i, also note that in fact the chain contains $i+1$ elements.

A first approach to construct a one-way chain is to use hash functions which are easy to compute. However, by using such function the length of the chain is fixed and when the chain is exhausted it requires re-initialization. An alternative solution is the use of functions from public-key encryption such as the discrete power function. By using such functions the one-way chain can have an unbounded length - but these functions require much more computational power [1], [2], [6].

In present, one-way chains are also used in some applications to assure information authenticity. An example is the electronic payment scheme proposed by Rivest and Shamir [12]. More recently one-way chains were proposed to be used to assure authenticity in constrained environments such as sensor networks [10]. Other proposals for the use of one-way chains in assuring information authenticity are in [3], [11].

The Delayed Message Authentication Protocol was proposed in [5] to assure authenticity in the exchange of information between two entities by using a one-way chain on each entity's side. In this paper we propose and investigate the use of the discrete squaring function for generating the one-way chains necessary in the DeMA protocol. Although this function is more computational intensive than some of the simplest one-way functions (such as hash functions) it has the advantage that the one-way chain can have an unbounded length. Also the elements of the one-way chain may be efficiently computed in a time-memory trade which significantly increases the computational performance.

In section 2 the Delayed Message Authentication Protocol is described and the complete description of the protocol for the case of the discrete squaring function is given. In section 3 the theoretical

performance of the protocol is discussed and practical results are analyzed. Section 4 holds the conclusions of this paper.

2. The Delayed Message Authentication Protocol with the Discrete Squaring Function

2.1. Description of the DeMA protocol

To introduce our proposal we briefly describe the Delayed Message Authentication Protocol from [5].

The DeMA Protocol can be used to exchange authentic information between two entities A and B by using a one way chain on each side.

The protocol consists in a variable number of communication sessions and each session consists in exactly two rounds. Each session provides the necessary information to prove the authenticity of the message from the previous session and in particular each round is the confirmation of the previous round. Before the first communication session, entity A randomly chooses x_A and B randomly chooses x_B, both these values will be kept secret and will be used to generate the session keys. The session keys (which are elements of the one-way chains) for A and B in session k are defined by the relations:

$$\sigma_A(k) = f^{\eta-k}(x_A),\ k = 0,1,...,\eta \qquad (1)$$

$$\rho_B(k) = f^{\eta-k}(x_B),\ k = 0,1,...,\eta \qquad (2)$$

Here η is an integer fixed by common agreement between the entities which denotes the maximum number of sessions.

In session 0 the entities make known to each other the values of $\sigma_A(0)$ and respectively $\rho_B(0)$ which will be later used to verify the authenticity of the new session keys – this will be done in a secure manner to guarantee the authenticity of this information. This initialization step is essential for the security of the proposed protocol, both entities must be strictly assured that the values $\sigma_A(0)$ and $\rho_B(0)$ were generated by A and respectively B and these values are part of a chain that was not used before and is intended for the communication between A and B. This can be achieved by exchanging four packages between the entities as follows:

$$A \to B:\ \left\{A, B, N_A, Sig_A(A, B, N_A)\right\}$$

$$B \to A:\ \left\{B, A, N_B, Sig_A(A, B, N_A, N_B)\right\}$$

$$A \to B:\left\{A, \sigma_A(0), Sig_A(A, B, N_A, N_B, \sigma_A(0))\right\}$$

$$B \to A:\ \left\{B, \sigma_B(0), Sig_A(A, B, N_A, N_B, \sigma_B(0))\right\}$$

Here Sig_A and Sig_B denotes a digital signature and N_A, N_B are two nonce to ensure the uniqueness of this communication.

Entity A starts the $k^{th}, k \geq 1$ communication session by sending to B a package with the structure $P_{A,k} = \left\{M_{A,k}, MAC\left(M_{A,k}, \sigma_A(k+1)\right), \sigma_A(k)\right\}$. The meaning of the notations is the following: $M_{A,k}$ denotes the message from session k, MAC is a message authentication code computed on $M_{A,k}$ with the key $\sigma_A(k+1)$ (which will be disclosed in session $k+1$) and $\sigma_A(k)$ is the current session key. Upon receiving the package from session k, entity B must verify that the session key $\sigma_A(k)$ is correct by checking that $f(\sigma_A(k)) = \sigma_A(k-1)$, where $\sigma_A(k-1)$ is the key disclosed in session $k-1$. If the session key proves to be correct then the package $P_{A,k}$ is memorized and the authenticity of the message $M_{A,k-1}$ from session $k-1$ can now be verified with the disclosed key $\sigma_A(k)$ by checking the message authentication code $MAC(M_{A,k-1}, \sigma_A(k))$.

Entity B will confirm the arrival of a correct session key by computing and sending a package $P_{B,k} = \left\{M_{B,k}, MAC\left(M_{B,k}, \rho_B(k+1)\right), \rho_B(k)\right\}$, the significance of the notations is similar to the package sent by A. The next communication session $k+1$ will be started by A only if the received value of $\rho_B(k)$ is correct and this can be easily verified by checking that $f(\rho_B(k)) = \rho_B(k-1)$. Both entities A and B will have to store only the last authentic session key that was received.

For the k^{th} communication session the two rounds will be as follows:

Session $k, 1 \leq k \leq \eta$

Round 1 $A \to B$:

$$P_{A,k} = \left\{M_{A,k}, MAC\left(M_{A,k}, \sigma_A(k+1)\right), \sigma_A(k)\right\}$$

Round 2 $B \to A$:

$$P_{B,k} = \left\{M_{B,k}, MAC\left(M_{B,k}, \rho_B(k+1)\right), \rho_B(k)\right\}$$

It is very important to note that each round will play the role of a confirmation for the previous round, for example round 1 of session k is the confirmation of round 2 from session $k-1$ while round 2 from

session k is the confirmation of round 1 from session k, and such confirmation will be sent only if the session key from the confirmed round was correct.

In the DeMA protocol only one of the entities, in our case A, will have the ability to start and stop the protocol in some session. The protocol can be stopped in any round of any communication session by A, obviously the authenticity of the message from that session will be proved only when the protocol starts again and the next session key is disclosed. When A decides to restart the conversation it will send a new package with a new session key only if the package from the previous round was confirmed, otherwise it has to resend the package from the previous round. The same policy should be applied in the case of accidental stops of the protocol which may be caused by some communication failures or by an intruder. For example we will suppose that the conversation was stopped at some session k. If A has received the correct $\rho_B(k)$, this means that round 2 was successfully completed, the protocol can be restarted later by computing and sending the package for session $k+1$. Otherwise, if the value $\rho_B(k)$ was not correctly received and round 2 was not successfully completed the protocol will be restarted by resending the package from session k until a correct $\rho_B(k)$ is received. These rules must be strictly followed, since if A computes and sends the package for session $k+1$ without having the confirmation key $\rho_B(k)$ from session k, in the case that the package from session k was not received by B this package can now be forged by an attacker who will now be in possession of the session key $\sigma_A(k+1)$. The same rule must be strictly followed by B which must send the confirmation $\rho_B(k)$ in the second round of session k only if $\sigma_A(k)$ proves to be correct.

2.2. The discrete squaring function

If symmetric primitives, such as hash-functions, are used in the construction of the one-way chains for the DeMA protocol, after the chains are exhausted, re-initialization can became a security problem [5]. By using the discrete power functions this problem is completely removed while such chains can have extreme lengths without influencing the computational time [6]. This is because the value of $f^{\eta}(x)$ can be easily computed by reducing the exponents modulo $\phi(n)$, also without requiring η successive compositions of the function. A particular case of the discrete power function is the discrete squaring function:

$$f(x) = x^2 \bmod n \tag{3}$$

for which it holds that:

$$f^{\eta}(x) = x^{2^{\eta} \bmod \phi(n)} \bmod n \tag{4}$$

Here n is a large composite integer which is infeasible to factor, ε is an integer exponent and $\phi(n)$ is the Euler totient function which can be computed only if the factorization of n is known. By using the function given in relation (3) the one-way chain becomes a chain of quadratic residues in Z_n. The great advantage in using this function is that the elements of the one-way chain can be computed in a time memory trade as suggested in [6] and the computational time is significantly reduced. 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. This can greatly increase the computational performance if the discrete squaring function is used in the DeMA protocol.

2.3. The DeMA protocol with the discrete squaring function

Both the entities will use a modulus which is infeasible to factor, let n_A and respectively n_B denote the modulus, and a random value x_A and x_B respectively. Therefore, by using function (3) the session keys are defined by the following relations:

$$\sigma_A(k) = f^{\eta-k}(x_A) = x_A^{2^{\eta-k} \bmod \phi(n_A)} \bmod n_A \tag{5}$$

$$\rho_B(k) = f^{\eta-k}(x_B) = x_B^{2^{\eta-k} \bmod \phi(n_B)} \bmod n_B \tag{6}$$

The following is the complete description of the protocol, it is important to mention that the computation of the session keys induced in step 2 of

the protocol actions can be efficiently performed with the time memory trade solution.

Key Setup.

a) A constant number η which represent an upper bound for the number of communication sessions is fixed by common agreement (since the discrete power function is used, the upper bound can be chosen as large as needed without influencing the computational performance).

b) Entity A randomly chooses two large primes p_A, q_A and a random value x_A then computes $n_A = p_A \cdot q_A$, $\phi(n_A) = (p_A - 1)\cdot(q_A - 1)$ and $\sigma_A(0)$.

c) Entity B randomly chooses two large primes p_B, q_B and a random value x_B then computes $n_B = p_B \cdot q_B$, $\phi(n_B) = (p_B - 1)\cdot(q_B - 1)$ and $\rho_B(0)$.

d) Entities A and B inform each other, in a secure manner to guarantee the authenticity of this information, of the values of $\sigma_A(0)$, n_A and $\rho_B(0)$, n_B respectively; all the other values (the two primes and the value of the Euler Totient function) will be kept secret on each side since disclosing them will lead to the lose of security.

Protocol Messages. For session $k = 1,...,\eta$

Round 1 $A \to B$:

$$P_{A,k} = \left\{M_{A,k}, MAC\left(M_{A,k}, \sigma_A(k+1)\right), \sigma_A(k)\right\}$$

Round 2 $B \to A$:

$$P_{B,k} = \left\{M_{B,k}, MAC\left(M_{B,k}, \rho_B(k+1)\right), \rho_B(k)\right\}$$

Protocol Actions.

The protocol actions may be described in the following 6 steps:

1. Increment the session counter $k = k + 1$.
2. Compute the package for session k (this requires the computation of one new session key and one MAC – the package structure is given above).
3. Send package for session k to the other entity.
4. Receive a new package from the other entity.
5. Check if the received package contains a new authentic session key – this is easy to check by verifying that $f(\sigma(k)) = \sigma(k-1)$ for A's key and $f(\rho(k)) = \rho(k-1)$ for B's key. If the session key is not authentic than return to step 4 else proceed to step 6.
6. Use the new authentic session key to test the authenticity of the message from the previous session.

Both entities A and B start the protocol with $k = 0$. The order in which the steps are performed is different for A and B, this is because A starts the protocol. Therefore the order in which A performs the steps is 1, 2, 3, 4, 5, 6 while for B is 4, 5, 6, 1, 2, 3 and so on for the completion of each session, the protocol run for session k is also suggested in figure 1. Because of this, A starts the protocol in step 1 while B starts the protocol in step 4 and after the completion of the η sessions A ends in step 6 while B ends in step 3.

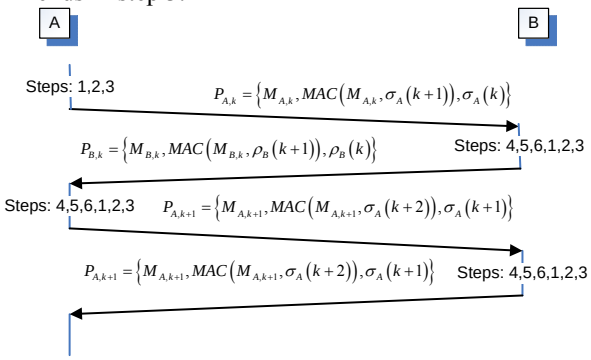

Fig 1. The DeMA authentication protocol

3. Computational Performance of The Protocol

3.1. Theoretical Performance

Since using public key primitives such as the squaring functions is more computational intensive than using symmetric primitives, we consider taking a closer look to the computational performance of the protocol. We will consider the measure of the computational time required for the completion of one round and one session of the DeMA protocol as a performance index.

The computations performed on each round of each session of the DeMA protocol consist in: computing one session key (which requires one modular exponentiation if we do not use the time-memory trade discussed in section 2), and one MAC in step 2 plus the verification of the received session key (which requires one modular multiplication) and of the MAC for the previous message in step 5 and 6. This results in the computational time given by the next relation:

$$t_{round} = t_{exp} + t_{mul} + 2 \cdot t_{MAC} \qquad (7)$$

Since computing the modular exponentiation is by far the most computational intensive operation, without

loosing the accuracy of the result we can approximate the computational time for one round as:

$$t_{round} \approx t_{exp} \quad (8)$$

The time-memory trade suggested in the previous section can greatly improve the computational performance of the protocol. If the chain of η elements is split into chains of λ elements the computational time for λ rounds is:

$$t_{\lambda} = t_{\exp} + (2\lambda - 1) \cdot t_{mul} + 2 \cdot \lambda \cdot MAC \quad (9)$$

From here, the mean computational time for one round follows directly as:

$$t'_{round} = \frac{t_{\exp}}{\lambda} + \left(2 - \frac{1}{\lambda}\right) \cdot t_{mul} + 2 \cdot t_{MAC} \quad (10)$$

In (10) we can ignore the computational time required by the MAC, since it should be about ten times smaller than a modular multiplication, and as λ becomes larger the computational time for one round can be estimated as:

$$t'_{round} \approx 2 \cdot t_{mul} \quad (11)$$

Relation (11) shows that theoretically the computational time for one round of the DeMA protocol can be reduced to the computational time of one modular squaring.

The computational time for one session of the DeMA protocol depends on the computational power from each side involved. We may define the computational time for one session as:

$$t_{session} = t_{A,round} + t_{B,round} \quad (12)$$

Here by $t_{A,round}$ and $t_{B,round}$ are the computational time for one round on A and B side respectively.

3.2. Practical Results

We have tested the performance of the DeMA protocol by running a Java implementation on two computers: a Toshiba Tecra notebook with Intel Centrino 1.6 Ghz and a desktop computer with AMD Athlon 64 2800+ at 1.8 Ghz. Both the computers were running Windows XP and had 512 MB of RAM, however this is not so important for the computational timings.

The protocol was implemented in Java NetBeans IDE 5.0 [13]. The Java environment was preferred because of the support offered for large integer's arithmetic. The computations performed over the groups of integers, were performed by using the methods contained in the *BigInteger* class: *BigInteger multiply(BigInteger val)*, *BigInteger mod(BigInteger m)*, *BigInteger modPow(BigInteger exponent, BigInteger m)*. We have computed the value of $x^2 \bmod n$ with one modular multiplication followed by one modular reduction; this method was preferred to the direct squaring with the function *modPow* which proved to be slower. The Message Authentication Codes are computed with the SHA1 hash function. The communication was assured by using a *ServerSocket* object that accepts connections on the server side and a *Socket* object on the client side.

We consider first to benchmark the computational performance of the basic cryptographic primitives in Java. In Table 1 the computational timings for the basic cryptographic primitives on both the computers are given.

In order to test the computational performance of the protocol we have connected the two computers with a TrendNet TW100 router. The computational timings with various sizes for the length of the chain and buffer are given in Table 2. The package sent between the entities in each round consisted in a message of 160 bits plus the MAC of 160 bits and the session key of 1024 bits.

As expected, the results from Table 2 show that the computational time is significantly improved by increasing the size of the buffer in which the pre-computed keys are stored (the size of the buffer from this implementation is the length λ of the smaller chains). In row (1) the computational time for the protocol in the case when the cryptographic primitives are enabled is given and in row (2) the computational time for the case when the cryptographic primitives are disabled is given (row (2) actually shows the communication performance with no cryptography). In rows (3) and (4) the off-line computational timings are given, these timings actually represent the time required on each side to compute one new session key and one MAC. The final conclusion that can be drawn from the experimental results is that the use of the time-memory trade (key buffer) improves the computational time significantly and for increased size of the buffers the computational time for one session with or without cryptography is close. Also the last two rows of table 2 in which the offline computational

performance is measured show that for increased sizes of the buffer the computational time is almost equal to one-modular multiplication – this is a practical result which sustain the theoretical expectations from relation (11).

CPU	MAC with SHA1	Modular Multiplication (1024 bit module)	Modular Exponentiation (1024 bit module and exponent)
Intel Centrino 1.6 Ghz	$10\times10^{-6}s$	$74\times10^{-6}s$	$50\times10^{-3}s$
AMD Athlon 64 2800+ 1.8 Ghz	$8.9\times10^{-6}s$	$60\times10^{-6}s$	$43\times10^{-3}s$

Table 1. Computational timings for the basic cryptographic primitives

	Number of Sessions / Buffer Size (λ)	$10^3/1$	$10^4/10^2$	$10^5/10^3$	$10^6/10^4$
1	Cryptography Enabled	40×10^{-3}	1.6×10^{-3}	0.8×10^{-3}	0.7×10^{-3}
2	Cryptography Disabled	0.6×10^{-3}	0.4×10^{-3}	0.4×10^{-3}	0.4×10^{-3}
3	Communication Disabled (Centrino 1.6 Ghz)	21×10^{-3}	615×10^{-6}	159×10^{-6}	113×10^{-6}
4	Communication Disabled (Athlon 64 2800+ 1.8 Ghz)	18×10^{-3}	514×10^{-6}	139×10^{-6}	96×10^{-6}

Table 2. Computational performace of the DeMA protocol with quadratic residue chains (time is measured in seconds and denotes the average time for one session $t_{session}$)

4. Conclusions

The use of the discrete squaring function for generating the one-way chains in the DeMA protocol was proposed and investigated. By using this function the computational time is not influenced by the length of the one-way chain. This function also offers the advantage that the elements of such a one-way chain may be efficiently computed in a time-memory trade at the reduced cost of almost one modular multiplication for each element. The practical results included in this paper give an estimate measure for the computational performance of DeMA protocol implemented with the discrete squaring function. The proposed solutions may be of interest in a number of applications that deal with information authenticity or user authentication over public networks such as the Internet.

5. References

[1] Bicakci, K., Baykal, N., *Infinite Length Hash Chains and Their Application*, IEEE 11th International Workshops on Enabling Technologies, WETICE, 2002.

[2] Bicakci, K., Baykal, N., "Improving the Security and Flexibility of One-Time Passwords by Signature Chains", Turkish Journal of Electrical Engineering and Computer Sciences, vol. 11, no. 3, 2003.

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

[4] Chien, Hung-Yu, Jan, Jinn-Ke, (2003). "Robust and Simple Authentication Protocol". Oxford Journal, The Computer Journal, Vol. 46, No. 2, 2003.

[5] Groza, B., (2006) Using one-way chains to provide message authentication without shared secrets, accepted at IEEE 2nd Workshop on Security, Privacy and Trust in Pervasive and Ubiquitous Computing, 2006.

[6] Groza, B., Petrica, D., Dragomir T.L., *A time-memory trade solution to generate one-time passwords using quadratic residues in Zn,* Studies in Informatics Control, ISSN 1220-1766, 2005, pp. 201- 212.

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

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

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

[10] Perrig, A., Szewczyk, R., Wen, V., Culler D., Tygar, J.D., SPINS: Security Protocols for Sensor Network, Proceedings of 7th Annual International Conference on Mobile Computing and Networks MOBICOM, 2001.

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

[12] Rivest, R., Shamir, A. Payword and Micromint: Two simple micropayment schemes. CryptoBytes, volume 2, no. 1, RSA Laboratories, 1996.

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