Tatra Mt. Math. Publ. **29** (2004), 49–55

TATRA MOUNTAINS
Mathematical Publications

ONE-TIME PASSWORD TREE

ANDRZEJ KOZIOŁ

ABSTRACT. We consider the problem of one-time password scheme in systems with limited resources (memory, computation time, number of recordings to memory) like smart cards. We show that using the tree of passwords, recordings to memory can be omitted (with exception of the counter). The number of authorizations is exponential in relation to maximal computation time.

1. Introduction

Smart cards have a small amount of EEPROM memory (data in this memory do not vanish after unplugging power). The memory, because of physical construction, after some number of overwritings becomes unfeasible for further use. The processor on such a card is very slow comparing to desktop computers.

Authorization with a card takes place as follows. The card is inserted into the terminal, which supplies it with electricity. Smart card sends to the terminal the identification number. Next it makes computations and sends the calculated password to the terminal, which verifies if password is correct. It may be necessary to send some other data to assure security.

Authorization with a smart card can not last too long. The most time consuming calculation considered in this paper is computation of hash procedure. Therefore, time limit is assumed to be the number of hash function calculations. The time of other calculations on the card as well as time of calculations in the terminal is neglected, because they are not the bottleneck of the algorithm. The time of sending data between the card and the terminal is also neglected.

The following notation has been used in this paper:

h – good hash function, (i.e.: a function $h\colon x \mapsto h(x)$, where $h(x)$ is easily computable, but when given $h(x)$, it is computationally infeasible to derive x);

n – the number of hash procedure calculations within time bound;

s – the secret stored on the card.

2000 Mathematics Subject Classification: 94A60, 11T71.
Keywords: one-time, password true, smart card.

1.1 Lamport one-time password scheme

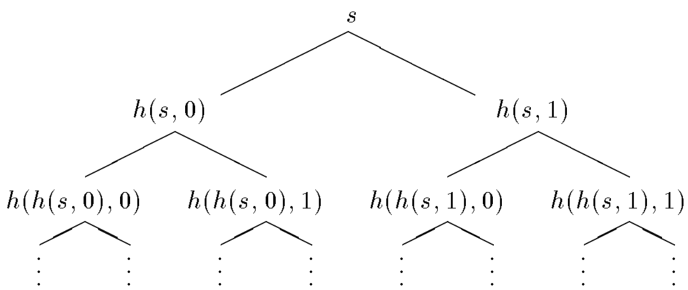
In [1], L a m p o r t proposed a login protocol based on an iterative use of a *one-way function.* This construction was exploited in many cryptographic applications, including the S/KEY one-time password system [8], electronic micropayment schemes [3, 11], generation of server-supported digital signatures [4] or with a bounded life-span [5], and also one-time signature schemes [6, 7, 9, 10].

Lamport's one-time password scheme can be briefly described as follows. Let $h^n(a) \equiv \underbrace{h\big(h(\ldots h(a)\ldots)\big)}_{n \text{ times}}$. From an initial value S_0, a user computes $S_i = h^i(S_0)$ for $i = 1, \ldots, n$. The final value S_n is given, in secure manner, to the remote system (i.e., the verifier). The first time the user wants to login, he will be asked to deliver S_{n-1} as the password. The remote system then checks whether $S_n \stackrel{?}{=} h(S_{n-1})$.

If it does, then the access is granted to the user, and the remote system updates the password database by storing S_{n-1}. This is a one-time password. The next time the user will login, he has to send S_{n-2} as the password, and so on, until S_0 is used as the password. The main advantage of this method over the fixed password solution is that replay attacks are not possible.

2. Password tree

Let $h(a,b)$ be a hash function h which takes a concatenation of a and b as an argument. Instead of calculating $h(s)$ as in standard Lamport scheme, computation $h(s,b)$ can be made, where $b \in \{0,1\}$. This means, that after adding one bit to the secret s, two new passwords are given, instead of one. Repeating interactively, the tree of passwords can be calculated.



Thus, storing on the card only one secret s, the exponential number of passwords can be computed. The maximum number of hash function calculations is n, hence the maximum depth of the tree is $n+1$ (the card must not calculate leaf values, because they are already known to terminal).

The authorization goes bottom-up. The terminal stores leaf values, then can authorize passwords in higher nodes. This is an obvious disadvantage, because terminal has to store the exponential number of values for each user.

Note. The number of possible passwords is 2^w (where w is the size of a password in bits). The number of passwords in the tree is $2^h - 1$, where h is the depth of the tree. Using *brute-force* attack (i.e.: the computation hash procedure for every possible secret of a given length) this scheme can be broken, hence w should be much greater than h to overcome this problem.

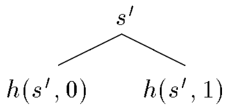
3. Economizing terminal memory

Let $\widehat{h}^n(a,c) \equiv \underbrace{h\big(h(\ldots h}_{n \text{ times}}(a,c),\ldots,c),c\big)$,
where $c \in \{0,1\}$.

Lemma 3.1. *One memory cell in the terminal is enough to authorize the password tree with the maximum depth of $n-1$.*

Proof. (The induction of tree structure).

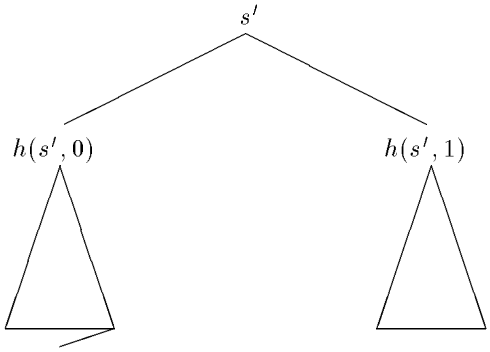
Step 1. The depth of the tree $= 1$.



Only one of the leaf values is needed to authorize s', e.g., the left leaf $h(s',0)$. After a successful authorization the terminal records s' in place of old value.

Step 2. The depth of the tree $= j+2$ ($j < n-2$).

Let s' be the secret calculated from s after making an i hash procedure computations.



$k = h\big(h^j\,(h(s',0),1),0\big)$

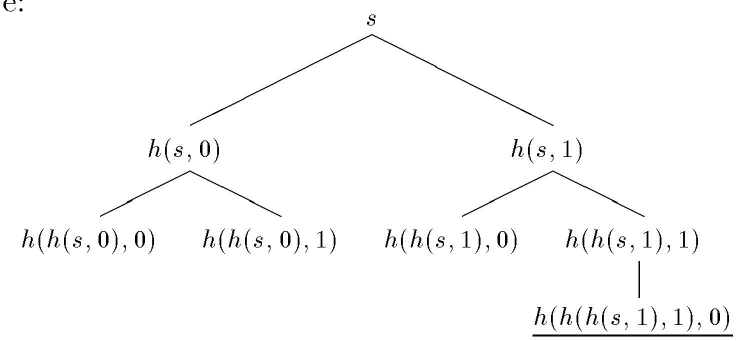
The right subtree is authorized using induction thesis. After the authorization of the root of the right subtree, the card sends $k = h\big(\widehat{h}^j\big(h(s',0),1\big),0\big)$. The terminal replaces the old value which was used for authorization with k. Now, using induction thesis the left subtree can be authorized. After authorizing both subtrees we have the situation described in Step 1.

In order to calculate s' from s in Step 2, the card has to make i computations of the hash function. In addition there were two h calculations of roots of left and right subtrees, and finally one with computing k. As the maximum number of h calculations can not be greater then n, therefore $j+i+3 \leq n$. Thus, the depth of the tree $(j+i+2)$ can not be greater than $n-1$. $\square$

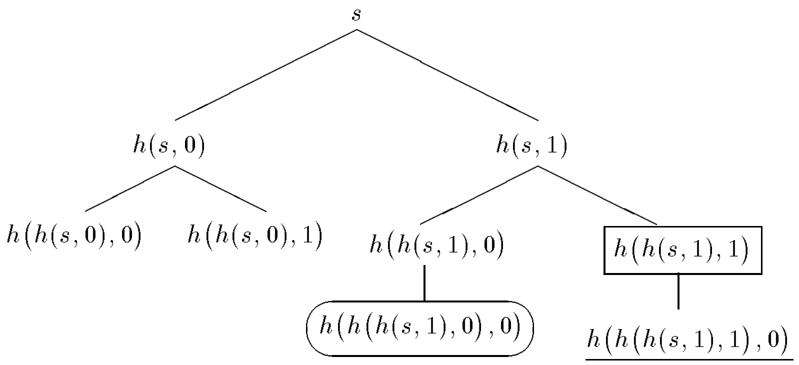
3.1 Example: $n = 4$

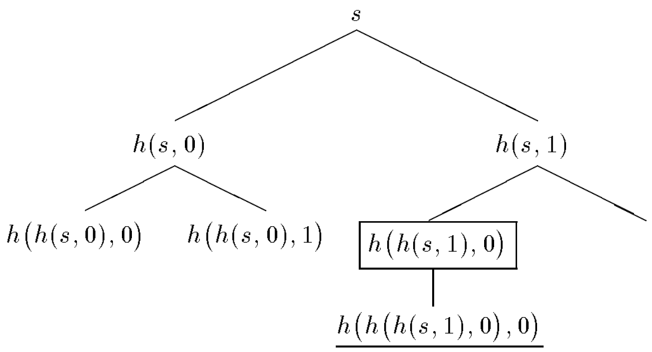
Values known to the terminal are underlined; values which are sent to the terminal for the purpose of authorization are marked by $\boxed{\text{box}}$; additional values which are send to the terminal in case of changing a subtree are marked by (ovalbox); used vallues are cut off from the tree.

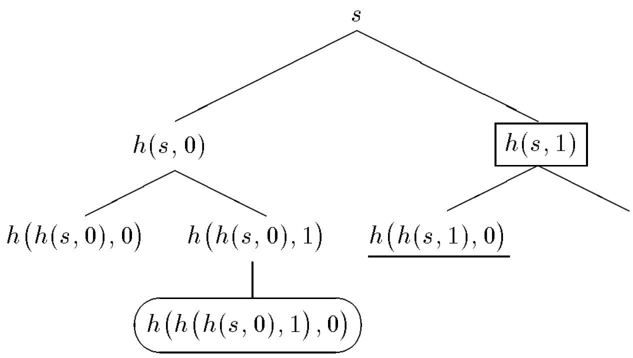
Starting state:



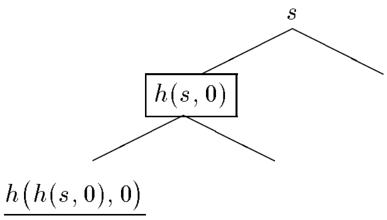
Next, following authorizations are made:

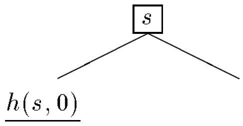






Left subtree is authorized in the same way as right. After this, the following state appears. Two last passwords can be send to the terminal:





4. Security

This scheme is not immune to *Man in the Middle* attack. If M controls data flow between the card and the terminal, when the card sends k, M can replace it with another value $k' = h\big(\widehat{h}^j(\widetilde{s}, 1), 0\big)$, where $\widetilde{s}$ is a number known to M. After this, M gets control over following authorizations.

4.1 Shared secret stamp

Let the card and the terminal share secret x. Every critical information S is sent with the stamp $h(S, x)$. Using the stamp and x the terminal can check if s is correct. If M wants to put his value in place of s, he should send the correct stamp, but he does not know x, therefore he can not.

4.2 Chaining

Alternatively, the following scheme can be used. There is a sequence of vulnerable passwords $S_1, S_2, S_3, \ldots S_m$ and some random value S_0. Let the terminal after the card initialization stores S_0, S_1, $C_0 = h(S_0, S_1, S_2)$ and $C_1 = h(S_1, S_2, S_3)$. When the card sends S_2, it also computes S_3 with S_4, and sends $C_2 = h(S_2, S_3, S_4)$. The terminal checks correctness of S_2 with S_0, S_1, and C_0. Next the terminal drops S_0 with C_0, and stores S_2 as well as C_2. Next steps are followed interactively. Unfortunately, depth of the tree halves with this scheme, but sometimes it can be acceptable.

5. Counter

Apart from the root of the password tree, the smart card has to store the value of the counter in order to correctly traverse the tree. In order to represent all passwords of the tree, the size of the counter should be $\lfloor \log_2(t) + 1 \rfloor$ where t is the number of passwords in the tree. After every authorization the counter is overwritten, hence the place in the memory intended for it can be broken before all passwords would be used. In case of a memory error the counter should be moved to another part of the memory. To use all passwords of the system, the field for the counter should be of size $\lfloor \log_2(t) + 1 \rfloor * \lceil 1/e * l \rceil$, where $1/e$ is an average chance of breaking the memory in one recording (e is average lifespan of the memory, given by the producer), and l is a whole number of passwords offered by the system (if the system has only one password tree, then $l = t$).

Acknowledgment. The support of this research by the Institute of Computer Science, University of Wrocław, Poland is gratefully acknowledged. I also thank

Prof. Krzysztof Loryś and Prof. Mirosław Kutyłowski for helpful discussions and suggestions.

REFERENCES

[1] LAMPORT, L.: *Password authentication with insecure communication*, Comm. ACM **24** (1981), 770–772.

[2] MENEZES, A.—VAN OORSCHOT, P.—VANSTONE, S.: *Handbook of Applied Cryptography*, CRC Press, Singapore, 1996.

[3] ANDERSON, R.—MANIFAVAS, H.—SUTHERLAND, C.: *A practical electronic cash system*, `http://www.cl.cam.ac.uk/users/rja14/`.

[4] ASOKAN, N.—TSUDIK, G.—WAIDNER, M.: *Server-supported signatures*, in: Fourth European Symposium on Research in Computer Security (ESO-RICS'96) (E. Bertino, ed.), Lecture Notes in Comput. Sci., Vol. 1146, Springer-Verlag, Berlin, 1996, pp. 131–143.

[5] DELOSAND, P.—QUISQUATER, J. J.: *An identity-based signature scheme with bounded life-span*, in: CRYPTO '94 (Y. G. Desmedt, ed.), Lecture Notes in Comput. Sci., Vol. 839, Springer-Verlag, Berlin, 1994, pp. 83–94.

[6] EVEN, S.—GOLDREICH, O.—MICALI, S.: *On-line/off-line digital signatures*, in: CRYPTO '89 (G. Brassard, ed.), Lecture Notes in Comput, Sci., Vol. 435, Springer-Verlag, Berlin, 1990, pp. 263–275.

[7] LAMPORT, L.: *Constructing digital signatures from a one-way function*, Technical Report CSL–98, SRI International, 1979.

[8] HALLER, N. M.: *The S/KEY one-time password system*, in: Proceedings of the ISOC Symposium on Networks and Distributed Systems Security, 1994.

[9] MERKLE, R. C.: *A certified digital signature*, in: CRYPTO '89 (G. Brassard, ed.), Lecture Notes in Comput. Sci., Vol. 435, Springer-Verlag, Berlin, 1990, pp. 218–238.

[10] RABIN, M. O.: *Digitalized signatures*, *Foundations of Secure Computation*, (D. Dobkin, A. Jones, and R. Lipton, eds.), Academic Press, New York, 1978, pp. 155–168.

[11] RIVEST, R. L.—SHAMIR, A.: *Pay word and micro mint: two simple micropayment schemes*, Crypto Bytes **2** (1996), 7–11.

Received October 13, 2003

Institute of Computer Science
University of Wrocław
Wita Stwosza 1–2
PL–50-148 Wrocław
POLAND

E-mail: psionic@ks.k.pl