One-time identification with low memory

Serge Vaudenay

Laboratoire d'Informatique de l'Ecole Normale Supérieure
(research group affiliated with the CNRS)
45 rue d'Ulm – 75230 Paris cedex 05 – France

Abstract

In this paper, we describe a practical cryptosystem based on an authentication tree. It is an extension of Ralph Merkle's results [1]. The security of the system depends on assumptions on a *one-way function*. This cryptosystem allows to make interactive proofs of knowledge using keys only once. Thus, it can be used to prove one's identity or to sign some message. A very efficient implementation can be done on a low-cost smart card. This cryptosystem is thus an alternative to the *zero-knowledge* algorithms.

1 Introduction

The notion of *authentication tree* has been defined in [2]. It allows to authenticate some values taken from a directory. If we have a list of values, we can assign these values to the leaves of a tree. Using a *collision-free hash function*, we can propagate the values from the leaves to the root of the tree. Then, if we assume the value of the root is authenticated, we can authenticate any value of the tree by producing the values in the *authentication path* (see figure 1). It is just the values assigned to the brothers of the ancestors of the given node.

In [1], authentication trees are used to achieve a signature scheme. It is based on the Diffie-Lamport scheme [3][4]. This scheme is a one-time signature scheme. It

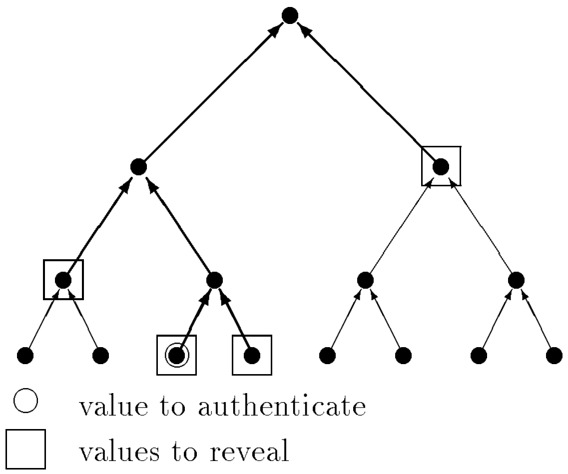

Figure 1: Authentication binary tree

needs a set of keys to sign a message. As it is a one-time signature scheme, it needs as many sets of keys as we have messages to sign. That is the reason why we need an authentication tree to authenticate all the keys.

To achieve such a cryptosystem, we can define an authentication tree that is made with three levels from the top to the bottom :

- in the first-level tree, each leaf corresponds to a user, so we can authenticate every users;
- for a given user, we have a second-level tree in which each leaf corresponds to a set of keys to be used in only one transaction;
- for a set of keys, we have a third-level tree to authenticate each key.

In the third level tree, we can compute all the values quickly with the hash function. In the first level tree, we can store permanently the values of the only authentication path we need. However, in the second level tree, we have to compute successively all the authentication paths.

We wish to sign N messages by using at most M value registers between each transaction. We let T^c be the time needed to compute each authentication path, and C be the number of values of the the authentication path. We can compute without memory, so $M = O(1)$, $T^c = O(N)$ and $C = \log_2 N$. We can keep all the values in memory, so $M = O(N)$, $T^c = O(1)$ and $C = \log_2 N$.

In [1], a path regeneration algorithm was proposed to get :

$$M \sim \frac{1}{2}(\log_2 N)^2,\ T^c_{exp} = O(\log N) \text{ and } C = \log_2 N$$

In this paper, we obtain a family of algorithms whose complexity vary from these figures to :

$$M \sim \frac{1}{2}\left(\frac{\log N}{\log\log N}\right)^2,\ T^c_{exp} = O\left(\frac{\log N}{\log\log N}\right) \text{ and } C_{exp} \sim \frac{(\log N)^2}{2(\log\log N)^3}$$

Thus, we can reduce the memory requirement by a factor $(\log_2 \log N)^2$. However, this is an asymptotic evaluation. In practice, if we need $N \simeq 2000$, we may reduce the memory requirement by a factor of 2.2 which is already quite significant in several applications.

This scheme can be used in other settings than signature. In this paper, an interactive proof of knowledge scheme is proposed. It is not a zero-knowledge scheme : it is a one-time proof of knowledge scheme. We can use this scheme with an arbitrary security parameter with the path regeneration algorithm to obtain a practical cryptosystem.

2 Notations

We assume we have a tree T. Each node is denoted by a word over the alphabet of the non-negative integers. The root is denoted by the empty word $\wedge$. Given a node of degree α denoted by the word a, the words $a0$, $a1$, ..., $a(\alpha-1)$ represent the ordered list of all sons of a.

We assume we have a collision-free hash function $<.>$, and an assignation ϕ which is a map from the set of T leaves to the domain of $<.>$. We can recursively define a map ϕ^* from the set of T nodes by :

$$\phi^*(a) = <\phi^*(a0), \phi^*(a1), \ldots, \phi^*(a(\alpha-1))>$$

To authenticate $\phi^*(a_0a_1\ldots a_{n-1})$, we have to give the values $\phi^*(a_0a_1\ldots a_{k-1}j)$ where $0 \leq k < n$ and $j \neq a_k$.

We assume that the function $<.>$ satisfies Damgård's property :

$$< x_1, x_2, \ldots, x_i > = << x_1, x_2, \ldots, x_{i-1} >, x_i >$$

Therefore, we only have to define the hash of two values to get the function $<.>$. This property will be useful.

A one-way function f will be used in the next section. A pseudo-random generator g will be used too. We can assume that $<.>$, g and f are a set of related cryptographic tools built using the same algorithms such that their implementation can be very efficient. We will assume that hashing two values costs as much time as computing two values of f, and that g and f have the same complexity.

3 Diffie-Lamport scheme

The Diffie-Lamport scheme [4] is a one-time signature scheme. Assume we have a message M to sign. This message is coded by s bits $b_1 b_2 \ldots b_s$. We assume that we have a set of secret keys $\{K_i; 0 < i \leq s\}$. The values $f(K_i)$ are assumed to be public.

The signature of the message M is the set of keys K_i for each i such that $b_i = 0$ [1]. Then, if keys K_i are the leaves of the authentication tree T, an authenticated signature of the message M is the set of all $f^{b_i}(K_i)$[2].

If a cheater gets an authenticated signature, he can forge a signature of a message M' coded by $b'_1 b'_2 \ldots b'_l$ if $b'_i \geq b_i$. Thus, we have to use a monotone code (i.e. a code such that for two different codewords $c_1 \ldots c_l$ and $d_1 \ldots d_l$, there exists i and j such that $c_i < d_i$ and $c_j > d_j$) to be sure that no $b'_1 b'_2 \ldots b'_s$ is a valid codeword.

A proposed code is the set of $b_1 b_2 \ldots b_{2l}$ such that $b_i \oplus b_{i+l} = 1$. Another code consists in appending the weight of the message to the codeword.

The first code proposed in [4] is the set of $b_1 b_2 \ldots b_s$ such that :

$$b_1 + b_2 + \ldots + b_s = \lfloor \frac{s}{2} \rfloor$$

A theorem of Sperner (1928) shows that this code is the shortest monotone binary code of a given cardinality. The main problem was to exhibit an efficient coding scheme, that is the reason why it was abandoned.

In [1], an improvement of Winternitz is described. It simply consists in using a larger alphabet for the code. If the codewords are coded over the alphabet of the set of all the integers from 0 to $\beta - 1$ (thus in base β), and if we assume that the values $f^{\beta-1}(K_i)$ are public, then an authenticated signature is the set of all $f^{b_i}(K_i)$. To verify the signature, we have to apply $\beta - 1 - b_i$ times f to the $f^{b_i}(K_i)$ to get the public keys.

We have to use a monotone code for the same reason. The set of the words $b_1 b_2 \ldots b_s$ such that :

$$b_1 + b_2 + \ldots + b_s = \lfloor s \frac{\beta - 1}{2} \rfloor$$

is still a good one. However, it is still difficult to exhibit an efficient coding scheme. To get rid of this difficulty, we will not use this algorithm for signatures, but for interactive proofs.

4 A one-time proof of knowledge scheme

We assume we have the three-level authentication tree T described in the introduction. At each level l, we use an assignation ϕ_l.

[1]In the original paper, it is the set of keys K_i for each i such that $b_i = 1$, but it is equivalent.

[2]We denote $f^0(K) = K$, $f^1(K) = f(K)$, $f^2(K) = f(f(K))$, ...

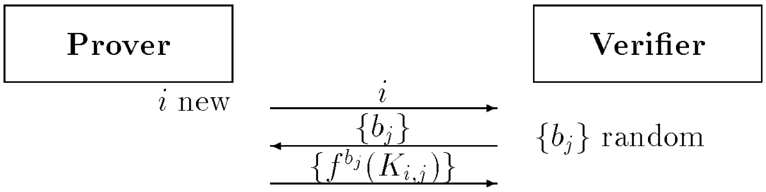

Figure 2: Identification scheme

At the first level, each leaf u corresponds to a user with identity Id and with a second-level tree $T(u)$. We define :

$$\phi_1(u) = < Id, \phi_2^*(\text{root of } T(u)) >$$

to authenticate both the identity of the user and the value assigned to the root of his tree.

At the second level, each leaf a_i corresponds to the ith interactive proof, and a third-level tree T_i. We define :

$$\phi_2(a_i) = \phi_3^*(\text{root of } T_i)$$

At the third level, the tree T_i is a simple one. It consists in a root a_i with s sons $b_{i,j}$ which are the leaves. Each leaf $b_{i,j}$ corresponds to a public key. We define :

$$\phi_3(b_{i,j}) = f^{\beta-1}(K_{i,j})$$

Where β is the one defined in the previous section. Thus, we have :

$$\phi_2(a_i) = < f^{\beta-1}(K_{i,1}), f^{\beta-1}(K_{i,2}), \ldots, f^{\beta-1}(K_{i,s}) >$$

If the user has some secret key K, we may use :

$$K_{i,j} = g(K, i, j)$$

To prove that the user has the identity Id, he has to prove that he knows a set of keys $\{K_{i,j}; 0 < j \leq s\}$ for some i. The prover first chooses i. Then, the verifier chooses s digits b_j in base β such that $b_1 + b_2 + \ldots + b_s = \lfloor s\frac{\beta-1}{2} \rfloor$ Finaly, the prover reveals $f^{b_j}(K_{i,j})$ for all j. The protocol is illustrated in figure 2.

With this protocol, the prover proves his knowledge of the $K_{i,j}$, which is assumed to prove his knowledge of K. We can use this protocol to achieve a signature scheme with the usual method : we simulate the verifier by a pseudorandom generator fed with i and the message to sign. Usually, the security required by the signature is greater than the security required by an interactive protocol. So, in order to sign a message, we will use several i to increase the security of the scheme.

β	s	$\beta s + 2s - 1$	sécurité
2	16	63	12.870
3	11	54	25.653
4	9	53	30.276
5	8	55	38.165
6	7	55	24.017
8	6	59	18.152
12	5	69	12.435
25	4	107	10.425

Figure 3: Possible couples (β, s)

In fact, each i corresponds to a security unit. A user has a fixed number of security units in the beginning, and he can use them as he likes for cryptographic operations with the security required. A proof of identity *costs* one unit, and a signature *costs* several units.

If the prover uses each i once, the knowledge of a cheater is, for each i, a set of $f^{b_j}(K_{i,j})$ where $b_1 + b_2 + \ldots + b_s = \lfloor s\frac{\beta-1}{2}\rfloor$. Then, if a cheater chooses an index i, the probability that he will be able to answer to the challenge of some verifier is the inverse of the number of possible list of b_j which is thus the security parameter of the scheme.

To get a security of $1,000,000$, we can choose $\beta = 2$ and $s = 23$. However, it might not be practical for some applications. A security of $10,000$ is enough for transactions in shops, where the prover is physically there. To sign with a security of 10^{12}, three units are enough. The table of figure 3 shows each combination of (β, s) which has a security greater than $10,000$. The time required to compute the hash value of all $f^{\beta-1}(K_{i,j})$ from K and i is $t = \beta s + 2s - 1$ units with our assumptions. So, t is exactly the complexity of ϕ_2. Thus, we have to minimize t for a given security. Then, we should choose $\beta = 4$ and $s = 9$. We get a security of $30,276 \simeq 2^{15}$.

We also notice that each i, which corresponds to a *cost* of security, can represents a money unit (a *coin*). To give a coin to someone, we can give the $f^{b_j}(K_{i,j})$. It will be a proof that we gave the coin. If a cheater gave the same coin to several people, the proofs would give several combinations of b_j with a probability which depends on the security parameter. These combinations of b_j would prove that the cheater gave more information about the $K_{i,j}$ than he was allowed. It would be a proof that he cheated. Thus, we can imagine to achieve an electronic cash system based on this cryptosystem.

5 The path regeneration algorithm

In this section, T will denote the second-level tree. We assume that it is a well balanced tree of degree α and of depth n such that the number of value to store is almost a constant. ϕ will denote the assignation of T. Thus, a leaf corresponds to a transaction. The algorithm we will describe now shows how to visit the T computing the values of the authentication path.

The authentication of the value of a leaf $a = a_0 a_1 \dots a_{n-1}$ consists in revealing the values $\phi^*(a_0 a_1 \dots a_{k-1} j)$ for all $0 \leq k < n$ and $j \neq a_k$. Because of Damgård's property, for each k we can remove the set of the values $\phi^*(a_0 a_1 \dots a_{k-1} j)$ for $j < a_k$ and keep their hash value instead. Let us define :

$$\phi_l^*(a_0 \dots a_{k-1} a_k) =< \phi^*(a_0 \dots a_{k-1} 0), \dots, \phi^*(a_0 \dots a_{k-1}(a_k - 1)) >$$

This is the hashed value of the left brothers of $a_0 \dots a_{k-1} a_k$. The authentication now consists in revealing the values $\phi^*(a_0 a_1 \dots a_{k-1} j)$ for all $0 \leq k < n$ and $j > a_k$ and the values $\phi_l^*(a_0 \dots a_{k-1} a_k)$ for all $0 \leq k < n$ and $a_k \neq 0$.

Let us define the following :

$$\begin{array}{rcll} M_{k,j} & = & \phi^*(S(a_0 a_1 \dots a_{k-1}) j) & \text{if } j < a_k \\ M_{k,j} & = & \phi^*(a_0 a_1 \dots a_{k-1} j) & \text{if } j \geq a_k \\ A_k & = & \phi_l^*(a_0 a_1 \dots a_k) & \text{if } a_k \neq 0 \\ T_{k,i} & = & \phi_l^*(S(a_0 \dots a_{i-1}) a_i \dots a_k) & \text{if } a_k \neq 0 \end{array}$$

for $0 \leq k < n$ and $0 < i < k$. The meaning of these formulae will be explained below. $S(x)$ denotes the successor of x in base α. Thus, it denotes the node "immediately on the right" of x. Those values are the ones stored in memory during the visit of the tree.

We define the function $\mathrm{Acc}(x, c, y)$ by :

$$\mathrm{Acc}(x, c, y) = \begin{cases} y & \text{if } c = 0 \\ < x, y > & \text{if } c \neq 0 \end{cases}$$

This function is used to regenerate the stored values A_k and $T_{k,i}$, which are accumulators. An accumulator is used to compute the hashed value of a node by hashing the values from the leftmost son to the rightmost son. In order to accumulate the value of a new son, we use the function Acc. To compute $\phi^*(a_0 \dots a_{k-1} a_k)$, we need $n - 1 - k$ accumulators, one of every tree's floor.

We can show that the algorithm described on figure 4 is a path regeneration algorithm — i.e. it refreshes all the stored values before incrementing a. In the beginning of the life of the authentication tree, we assume that we have the values to authenticate the leaf $00 \dots 0$. So, we can initialize the stored values in step 2. In step 3, we do some operations on the leaf a (the one-time interactive proof scheme for instance) and we

1. $a \longleftarrow 00\ldots0$;
2. initialize $M_{k,j}$ for $0 \leq k < n$ and $0 \leq j < \alpha$;
3. visit a, authenticate $\phi(a)$;
4. if $a_k = \alpha - 1$ for all k then end;
5. $p \longleftarrow \mathrm{Max}\{k; a_k \neq \alpha - 1\}$;
6. $A_p \longleftarrow \mathrm{Acc}(A_p, a_p, M_{p,a_p})$;
7. $M_{n-1,a_{n-1}} \longleftarrow \phi(S(a_0 \ldots a_{n-2})a_{n-1})$;
8. for all $0 < i < n-1$:
   $T_{n-1,i} \longleftarrow \mathrm{Acc}(T_{n-1,i}, a_{n-1}, \phi(S(a_0 \ldots a_{i-1})a_i \ldots a_{n-1}))$;
9. for $k = n-2$ downto $k = p$:
   (a) $M_{k,a_k} \longleftarrow T_{k+1,k}$;
   (b) for all $0 < i < k$: $T_{k,i} \longleftarrow \mathrm{Acc}(T_{k,i}, a_k, T_{k+1,i})$;
10. $a \longleftarrow S(a)$;
11. go to 3.

Figure 4: Path regeneration algorithm

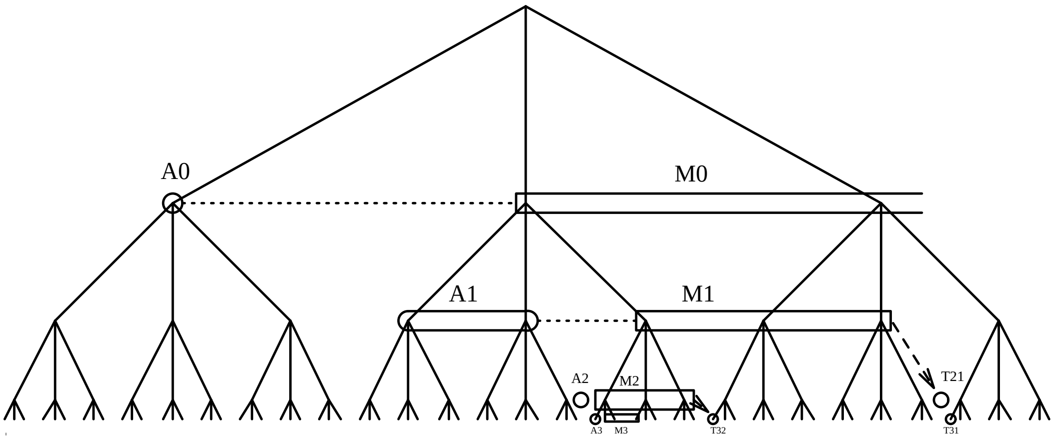

Figure 5: Example of path regeneration

authenticate $\phi(a)$. It simply consists in giving the values $M_{k,i}$ for all $0 \leq k < n$ and $i < a_k$ and the values A_k for all k such that $a_k \neq 0$. The steps 6 to 9 regenerate the stored values for the next a.

From the beginning to the end of the algorithm, $\phi^*(\wedge)$ is computed with the accumulators A_k (one accumulator per floor of the tree). After the visit of the leaf a, the leaves from $0 \ldots 0$ to a are accumulated. These accumulators are used only in the authentication of a.

For each k (at each floor of the tree), the set of $M_{k,.}$ is a window of the α nodes the leftmost of which is $a_0 \ldots a_{k-1}$ (see figure 5). If this node is not useful anymore, the window is shifted to the right (see step 7 if $k = n-1$ and step $9a$ otherwise). In order to shift the window, we have to compute the value of $a_0 \ldots a_{k-1} + \alpha$, using the accumulators $T_{.,k}$ if $k \neq n-1$.

At the step 5, we compute the index p of the highest floor where the window have to be shifted. Then, the only useful accumulator A_k is modified in step 6, the window of the lowest floor is shifted in step 7, and the other eventual window are shifted in step $9a$. At the steps 8 and $9b$, the accumulators $T_{k,i}$ are modified. The lowest ones are modified by a computation $\phi(.)$ (step 8). The other ones are modified using lower accumulators (step $9b$).

If we denote T^c the computation time, C the number of values to reveal in order to authenticate $\phi(a)$ and M the number of registers required to store all the values, we can be interested either in the average complexity (T^c_{exp} and C_{exp}) or the maximum complexity (T^c_{max} and C_{max}). With our assumptions on $< . >$, we can use the time to compute a value of f as a time unit. We denote by t the time needed to compute a value of ϕ. Then, we have :

$$\begin{array}{rcl} T^c_{exp} & \simeq & (n-1)t + 2\left(n - 1 + \frac{n-2}{\alpha-1}\right) \\ T^c_{max} & = & (n-1)t + 2\left(1 + \frac{n(n-1)}{2}\right) \\ C_{exp} & = & n\frac{\alpha+1}{2} - \frac{n}{\alpha} \\ C_{max} & = & n(\alpha-1) \\ M & = & n\alpha + \frac{n(n-1)}{2} + 1 \end{array}$$

α and n are related by the relation $N = \alpha^n$ where N is the number of leaves in the tree T.

If τ denotes the time to reveal one value (the transmission time), an authentication and a path regeneration together cost $T = T^c + \tau C$. We can prove that if we have :

$$\frac{t}{\tau} < 2\log 2 - 1 \simeq 0.39$$

then the choice $\alpha = 2$ corresponds to the smallest value of T. However, the smallest possible M corresponds to the α given by the implicit equation :

$$\alpha(\log\alpha - 1) - \frac{\log N}{\log\alpha} = -\frac{1}{2}$$

obtained by derivating with respect to α the expression of M, with $N = \alpha^n$. This corresponds to :

$$\alpha \sim \frac{\log N}{(\log\log N)^2} \text{ and } n \sim \frac{\log N}{\log\log N}$$

Thus, we can choose our favorite α between $O(1)$ and $\log N/(\log\log N)^2$ to achieve the space/time tradoff we prefer. We obtain a family of algorithms whose complexity varies from :

$$M \sim \frac{1}{2}(\log_2 N)^2,\ T^c_{exp} = O(\log N) \text{ and } C = \log_2 N$$

to :

$$M \sim \frac{1}{2}\left(\frac{\log N}{\log\log N}\right)^2,\ T^c_{exp} = O\left(\frac{\log N}{\log\log N}\right) \text{ and } C_{exp} \sim \frac{(\log N)^2}{2(\log\log N)^3}$$

Figure 6 shows the several combinations (α, n) where N is larger than $2,000$. We assume $t = 53$ (this corresponds to $\beta = 4$ and $s = 9$ in section 4) and $\tau = 16$ which is about the transmission of a 16 byte value at $9,600$bps if we assume that f requires about 1ms. These are the actual parameters of a low-cost smart card in which we implemented a software one-way function.

Figure 6 shows that T_{max} is minimal for $\alpha = 5$ and $n = 5$. This corresponds to a low memory : 36 (576 bytes if we assume that a value is a 16 byte value) instead of 78 ($1,248$ bytes) with binary trees. Moreover, the program which achieves the path

α	n	N	M	C_{max}	C_{exp}	T^c_{exp}	T_{max}	T_{exp}
2	11	2048	78	11	11	568	818	744
3	7	2187	43	14	12	335	586	527
4	6	4096	40	18	14	278	585	502
5	5	3125	36	20	14	222	554	446
7	4	2401	35	24	15	166	557	406
13	3	2197	43	36	21	110	690	446
45	2	2025	92	88	46	55	1465	791
2000	1	2000	2001	1999	1000	0	31986	16000

$t = 53,\ \tau = 16$

Figure 6: Different combinations of (α, n) with $\alpha^n \geq 2000$

regeneration algorithm is very simple, and it does not require much space in ROM. The average time to compute and transmit the authenticating values is 446ms. An implementation of this algorithm with these parameters will be able to have $3,125$ security units. If we assume that a session needs 4 security units (one to authenticate, and three to sign a message), and that we have twenty sessions per month, we can use the smart card during more than three years, which is quite more than the expected life of a smart card.

With hardware one-way functions f, g and $<,>$, we could get even better results.

Conclusion

Authentication tree is a practical way to authenticate a value in a large directory. It requires a small file to read for the prover, and it requires few computations for the verifier.

The path regeneration algorithm is an efficient algorithm to visit a directory computing the values of the authentication paths. This allows to achieve one-time cryptographic algorithms : a one-time algorithm needs a one-time authenticated key. Several keys can be put in the directory of an authentication path we can visit.

The one-time interactive proof scheme can be used both to authenticate and to sign messages. It consists in revealing an information about the key we use that nobody else will be able to use (up to arbitrary security parameter). As we can prove that someone used the same key several times, because he revealed too much information, this scheme can be used to achieve electronic cash systems.

Finally, it is possible to achieve efficiently such systems with low cost smart cards.

Acknowledgment

I would like to thank GEMPLUS Card International, and particularly David M'Raihi, for having encouraged this work. I also thank Jacques Stern without whom this paper would never have been achieved.

References

[1] R. Merkle : A Certified Digital Signature, in : *Advances in Cryptology CRYPTO'89*, Springer Verlag, 1989, 218-238.

[2] R. Merkle : Protocols for Public Key Cryptosystems, in : *Proc. IEEE Symp. on Security and Privacy*, 1980, 122-133.

[3] W. Diffie, M. Hellman : New Directions in Cryptography, in : *IEEE Transaction on Information Theory*, vol. IT-22, 6, 1976, 644-654.

[4] L. Lamport : Constructing digital signatures from a one way function, technical report CSL-98, SRI Intl., 1979.