One-Way Cross-Trees and Their Applications

Marc Joye[1] and Sung-Ming Yen[2]

[1] Gemplus Card International, Card Security Group
Parc d'Activités de Gémenos, B.P. 100, 13881 Gémenos Cedex, France
marc.joye@gemplus.com
http://www.gemplus.com/smart/
http://www.geocities.com/MarcJoye/
[2] Laboratory of Cryptography and Information Security (LCIS)
Dept of Computer Science and Information Engineering
National Central University, Chung-Li, Taiwan 320, R.O.C.
yensm@csie.ncu.edu.tw
http://www.csie.ncu.edu.tw/~yensm/

Abstract. This paper considers the problem of efficiently generating a sequence of secrets with the special property that the knowledge of one or several secrets does not help an adversary to find the other ones. This is achieved through *one-way cross-trees*, which may be seen as a multidimensional extension of the so-called *one-way chains*. In a dual way, some applications require the release of one or several secrets; one-way cross-trees allow to minimize the amount of data necessary to recover those secrets and only those ones.

1 Introduction

In [10], Lamport proposed a login protocol based on the iterative use of a *one-way function*. This elegant construction was exploited in many cryptographic applications, including the S/KEY one-time password system [8], electronic micropayment schemes [2,16], generation of sever-supported digital signatures [3] or with bounded life-span [5], and also one-time signature schemes [7,9,13,14].

Lamport's one-time password scheme can briefly be described as follows. From an initial value S_1 and a one-way function h, a user computes $S_i = h(S_{i-1})$ for $i = 2, \ldots, n$. The final value S_n is given, in a 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 password. The remote system then checks whether $S_n \stackrel{?}{=} h(S_{n-1})$. If yes, then 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 password, the remote system then checks whether $S_{n-2} \stackrel{?}{=} h(S_{n-1})$, ... and so on until S_1 is used as password. The main advantage of this method over the fixed password solution is that replay attacks are no longer possible.

Roughly speaking, one-way cross-trees generalize the idea behind Lamport's scheme through the use of several one-way functions and several initial values.

D. Naccache and P. Paillier (Eds.): PKC 2002, LNCS 2274, pp. 346–356, 2002.
© Springer-Verlag Berlin Heidelberg 2002

The resulting constructions naturally introduce a *pyramidal hierarchy* between the secrets (passwords) and therefore provide a simple means to allow *controllable* delegation. Lamport's scheme also presents some hierarchy but only vertical; for example, given the secret S_r anyone can compute $S_{r+1}, S_{r+2}, \ldots, S_n$. Consequently, if for some reason or other, secret S_r has been disclosed, then the security of secrets $S_{r+1}, \ldots,$ is compromised. One-way cross-trees are more flexible, the delegation is *fully* parameterizable. Such desirable property is useful for key escrow systems. It is so possible to construct a system wherein the release of some secret keys only enables to recover the messages encrypted under those keys and not all the past and future communications.

The rest of this paper is organized as follows. In Section 2, we formally define one-way cross-trees. We also derive some useful properties. Section 3 shows how one-way cross-trees allow to efficiently generate and release secrets. Some applications are then presented in Section 4. Finally, we conclude in Section 5.

2 One-Way Cross-Trees

We begin by formalizing the necessary definitions.

Definition 1. *A function $h : x \mapsto h(x)$ is said* one-way *if when given x, $h(x)$ is easily computable; but when given $h(x)$, it is computationally infeasible to derive x.*

Notation. When a function h is iteratively applied r times to an argument x, we will use the notation $h^r(x)$, that is $h^r(x) = \underbrace{h(h(\cdots(h}_{r \text{ times}}(x))\cdots))$.

$$h^0(x) = x \longrightarrow h^1(x) \longrightarrow h^2(x) \longrightarrow \cdots \longrightarrow h^n(x)$$

Fig. 1. One-way chain.

The iterative application of a one-way function h results in the generation of a *one-way chain.* Once generated, the chain is then employed in the backward direction, exhibiting the useful property that it is computationally infeasible to derive $h^{i-1}(x)$ from $h^i(x)$. One-way cross-trees generalize this concept. Loosely speaking, in a κ-ary one-way cross-tree, there are κ possible one-way directions from a given position—a one-way chain corresponds to a unary one-way cross-tree, there is only one possible one-way direction. More precisely, we have:

Definition 2. *Let $h_1, h_2, \ldots, h_\kappa$ be one-way functions and let $(I_1, I_2, \ldots, I_\kappa)$ be a κ-tuple. A* κ-ary one-way cross-tree *(κ-*OWCT *for short) is a structure T consisting of vertices and directed edges. Each vertex is labeled with a κ-tuple of the form*

$$\boldsymbol{V}_{r_1,r_2,\ldots,r_\kappa} = \left(h_1^{r_1}(I_1), h_2^{r_2}(I_2), \ldots, h_\kappa^{r_\kappa}(I_\kappa)\right), \tag{1}$$

where $r_1, r_2, \ldots, r_\kappa \in \mathbb{N}$. *The vertex corresponding to the* κ*-tuple* $\boldsymbol{I} = (I_1, I_2, \ldots, I_\kappa)$ *is called the* root *(or generator) of* T. *Each vertex (except the root) has at least one edge directed towards it. Moreover, each vertex (including the root) has at most* κ *edges directed away from it. Given an edge directed away from a vertex* P *towards a vertex* S, *vertex* P *is called the* predecessor *of* S *and vertex* S *is called the* successor *of* P. *A vertex without successor is called a* leaf. *A* κ-OWCT *is iteratively constructed as follows. Given a vertex labeled* $\boldsymbol{V}_{r_1,r_2,\ldots,r_i,\ldots,r_\kappa}$, *its* κ *(possible) successors are given by the vertices labeled* $\boldsymbol{V}_{r_1,r_2,\ldots,r_i+1,\ldots,r_\kappa} = \big(h_1^{r_1}(I_1), h_2^{r_2}(I_2), \ldots, h_i^{r_i+1}(I_i), \ldots, h_\kappa^{r_\kappa}(I_\kappa)\big)$ *with* $i \in \{1, \ldots, \kappa\}$.

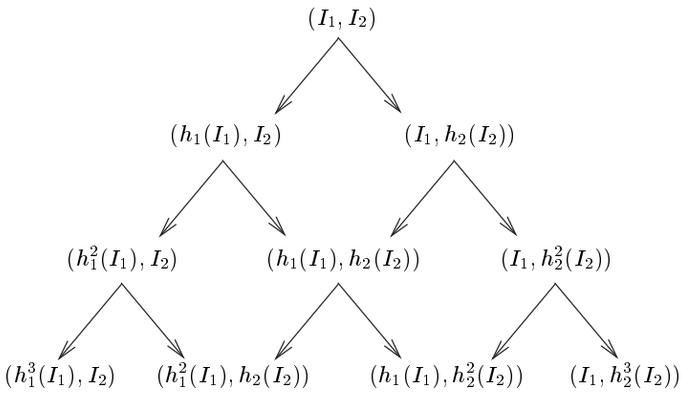


Fig. 2. Binary one-way cross-tree (2-OWCT).

Proposition 1. *Let* T *be a* κ-OWCT. *Given any* κ*-tuple* $\boldsymbol{V}_{r_1,r_2,\ldots,r_\kappa} \in T$, *it is*

(P1) computationally infeasible to find another κ*-tuple* $\boldsymbol{V}_{r'_1,r'_2,\ldots,r'_\kappa}$ *if* $r'_i < r_i$ *for some* $1 \le i \le \kappa$;
(P2) computationally easy to find another κ*-tuple* $\boldsymbol{V}_{r'_1,r'_2,\ldots,r'_\kappa}$ *if* $r'_i \ge r_i$ *for all* $1 \le i \le \kappa$.

Proof. This immediately follows from the one-wayness of T (see Eq. (1)). Finding $\boldsymbol{V}_{r'_1,r'_2,\ldots,r'_\kappa}$ for some $r'_i < j_i$ requires the inversion of the one-way function h_i, which is assumed to be computationally infeasible. □

Definition 3. *The* weight *of a* κ*-tuple* $\boldsymbol{V}_{r_1,r_2,\ldots,r_\kappa}$ *in a* κ-OWCT T *is defined as* $W(\boldsymbol{V}_{r_1,r_2,\ldots,r_\kappa}) = \sum_{i=1}^{\kappa} r_i$. *The* depth of $\boldsymbol{V}_{r_1,r_2,\ldots,r_\kappa} \in T$ *is given by*

$$\Delta(\boldsymbol{V}_{r_1,r_2,\ldots,r_\kappa}) = W(\boldsymbol{V}_{r_1,r_2,\ldots,r_\kappa}) - W(\boldsymbol{I})\,, \tag{2}$$

where $\boldsymbol{I}$ *denotes the root of* T. *Moreover, the* depth of T *is defined as being the depth of the element of greatest depth.*

Definition 4. *A* κ-OWCT T *is said* complete *if all of its leaves have the same weight.*

Proposition 1 is the fundamental property of a one-way cross-tree. It indicates that given one element $\boldsymbol{V}_{r_1,r_2,\ldots,r_\kappa}$, only the elements of the *κ-ary one-way subcross-tree* generated by $\boldsymbol{V}_{r_1,r_2,\ldots,r_\kappa}$ can be evaluated. In other words, when given *one* element, no elements of lower or equal weight can be computed.

Proposition 2. *A complete* κ-OWCT T *of depth* δ *has exactly*

$$N(\kappa,\delta)=\binom{\delta+\kappa-1}{\delta} \tag{3}$$

(distinct) elements of depth δ*.*

Proof. We use induction on κ. The case $\kappa=1$ corresponds to one-way chains and we obviously have $N(1,\delta)=1=\binom{\delta}{\delta}$. Suppose now that $\kappa\geq 2$ and that Eq. (3) holds for $\kappa-1$. Let $\boldsymbol{R}=(R_1,R_2,\ldots,R_\kappa)$ be the root of T. Then, the number of elements of depth δ is given by

$$\begin{aligned}
N(\kappa,\delta) &= \#\{(h_1^{s_1}(R_1),h_2^{s_2}(R_2),\ldots,h_\kappa^{s_\kappa}(R_\kappa)) \mid \textstyle\sum_{i=1}^{\kappa} s_i=\delta\}\\
&= \#\{(h_1^{s_1}(R_1),h_2^{s_2}(R_2),\ldots,h_\kappa^{s_\kappa}(R_\kappa)) \mid 0\leq s_1\leq\delta, \textstyle\sum_{i=2}^{\kappa} s_i=\delta-s_1\}\\
&= \textstyle\sum_{s_1=0}^{\delta} N(\kappa-1,\delta-s_1)\\
&= \textstyle\sum_{s_1=0}^{\delta}\binom{\delta-s_1+\kappa-2}{\delta-s_1} \quad \text{by the induction assumption}\\
&= \binom{\delta+\kappa-1}{\delta}\ .
\end{aligned}$$

□

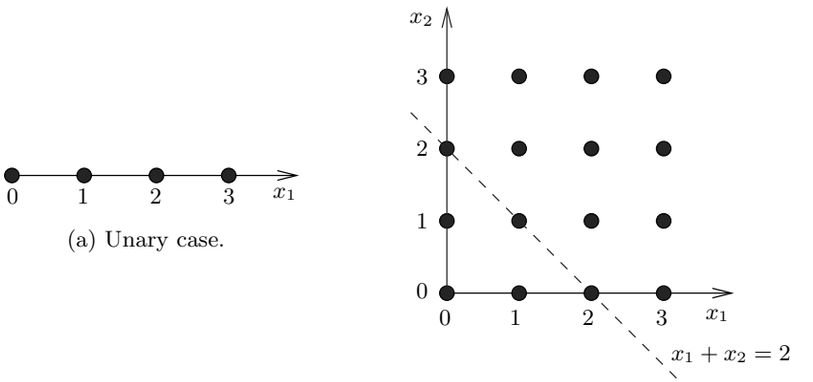
Corollary 1. *A complete* κ-OWCT T *of depth* δ *has* $\binom{\delta+\kappa}{\delta}$ *elements.*

Proof. Obvious, since $\sum_{d=0}^{\delta} N(\kappa,d)=\sum_{d=0}^{\delta}\binom{d+\kappa-1}{d}=\binom{\delta+\kappa}{\delta}$. □

Proposition 3. *Let* T *be a* κ-OWCT*. Given one or several* κ*-tuples of* T*, all the* κ*-tuples in the smallest one-way subcross-tree containing the given* κ*-tuples can be evaluated.*

Proof. Let $\mathcal{S}=\left\{\boldsymbol{V}_{r_1^{(j)},r_2^{(j)},\ldots,r_\kappa^{(j)}}=\left(h_1^{r_1^{(j)}}(I_1),h_2^{r_2^{(j)}}(I_2),\ldots,h_\kappa^{r_\kappa^{(j)}}(I_\kappa)\right)\right\}_{1\leq j\leq\ell}$ be the subset of the given κ-tuples. If $\tilde{r}_i$ denotes the smallest $r_i^{(j)}$ $(1\leq j\leq\ell)$ such that $h_i^{r_i^{(j)}}(I_i)$ is a component of a κ-tuple in $\mathcal{S}$, then the one-way subcross-tree generated by $\boldsymbol{V}_{\tilde{r}_1,\tilde{r}_2,\ldots,\tilde{r}_\kappa}$ will contain all the κ-tuples of $\mathcal{S}$ and is the smallest one. From the root $\boldsymbol{V}_{\tilde{r}_1,\tilde{r}_2,\ldots,\tilde{r}_\kappa}=(h_1^{\tilde{r}_1}(I_1),h_2^{\tilde{r}_2}(I_2),\ldots,h_\kappa^{\tilde{r}_\kappa}(I_\kappa))$ of this subcross-tree, all its κ-tuples can be evaluated. □

Alternatively, one-way cross-trees may be described in terms of integer lattices (see Fig. 3). Let T be a κ-OWCT and let $(I_1,I_2,\ldots,I_\kappa)$ be its root. Each element $(h_1^{r_1}(I_1),h_2^{r_2}(I_2),\ldots,h_\kappa^{r_\kappa}(I_\kappa))\in T$ may uniquely be represented by the



(a) Unary case.

(b) Binary case.

Fig. 3. Lattice interpretation.

integer vector $(r_1, r_2, \ldots, r_\kappa)$. We can therefore define a lattice L containing those integer vectors: $L = \{\sum_{i=1}^{\kappa} x_i \boldsymbol{e_i} \mid x_i \in \mathbb{N}\}$ where $\{\boldsymbol{e}_i\}$ is the standard basis—that is, $\boldsymbol{e}_i$ is a vector with a '1' in the i^{th} position and '0' elsewhere. Figure 3 depicts the lattice analogues of a one-way chain and of a binary one-way cross-tree. The elements of depth δ correspond to the lattice vectors lying in the hyper-plan $\Pi_\delta \equiv x_1 + x_2 + \cdots + x_\kappa = \delta$. Increasing the depth of one unit means "jumping forward" away from the hyper-plan Π_δ towards the parallel hyper-plan $\Pi_{\delta+1}$. Moreover, each lattice vector $(s_1, s_2, \ldots, s_\kappa) \in L$ defines a sublattice $L_{\boldsymbol{s}} = \{\sum_{i=1}^{\kappa} x_i \boldsymbol{e}_i \mid x_i \in \mathbb{N}, x_i \geq s_i\}$. The one-wayness of cross-tree T implies that from the only knowledge of the κ-tuple $(h_1^{s_1}(I_1), h_2^{s_2}(I_2), \ldots, h_\kappa^{s_\kappa}(I_\kappa))$, only the κ-tuples $(h_1^{x_1}(I_1), h_2^{x_2}(I_2), \ldots, h_\kappa^{x_\kappa}(I_\kappa))$ with $(x_1, x_2, \ldots, x_\kappa) \in L_{\boldsymbol{s}}$ may be computed. Note that, although less practical in higher dimension, the lattice formulation can be useful for theoretical purposes. For example, $N(\kappa, \delta)$ (see Proposition 2) may be considered as $|L \cap \Pi_\delta|$.

3 Generation/Release of Secrets

Suppose that a sequence of n secrets $(S_1, S_2, \ldots, S_n)$ has to be generated. There are basically two ways to do it:

(M1) A first method consists in randomly choosing $S_1, \ldots, S_n$.
(M2) Another method is to randomly choose S_1 and then evaluate $S_i = h(S_{i-1})$ for $i = 2, \ldots, n$, where h is a one-way function.

In this section, we will see that κ-OWCTs unify these two approaches and offer a fine control on both efficiency and security.

3.1 Efficiency

Method (M1) is computationally more efficient while Method (M2) is more efficient in terms of storage—only S_1 has to be stored and S_i is computed as $S_i = h^{i-1}(S_1)$. Note that, in (M1), $S_1, \ldots, S_n$ can be considered as the root of a n-OWCT and, in (M2), as the elements of a 1-OWCT (one-way chain). Therefore, if the secrets are constructed as elements of a κ-OWCT where κ varies between 1 and n, we obtain a full range of possibilities, enabling to choose the best trade-off between computational speed and storage requirements.

However, one-way functions such as SHA [1] or MD5 [15] are very fast; the storage limitation is thus more restrictive. Consequently, Method (M2) seems to be optimal since it only requires the storage of one secret, but the following paragraph brings the opposite consideration.

3.2 Security Considerations

From a security point of view, Method (M1) is superior because the secrets are totally independent; in Method (M2), from a secret S_r, anyone is able to compute $S_{r+1}, S_{r+2}, \ldots, S_n$. Note that this property is sometimes desired in certain applications such as Lamport's one-time password scheme (see Section 1).

3.3 Generation/Release of Secrets in a κ-OWCT

In this paragraph, we discuss in more details how to generate and release a sequence of n secrets in a κ-OWCT T with $2 \leq \kappa \leq n-1$. A first idea is to use the elements of T as secrets. However, special precautions must be taken: the elements in a κ-OWCT are not completely independent (see Proposition 3). Another idea is to only use the leaves of a complete κ-OWCT. Even in that case, independence between secrets is not guaranteed. Consider for example a complete 3-OWCT of depth 4 with root (I_1, I_2, I_3). The leaf $\boldsymbol{V}_{2,1,1} = (h_1^2(I_1), h_2(I_2), h_3(I_3))$ may for example be obtained from leaves $\boldsymbol{V}_{2,0,2} = (h_1^2(I_1), I_2, h_3^2(I_3))$ and $\boldsymbol{V}_{1,2,1} = (h_1(I_1), h_2^2(I_2), h_3(I_3))$ (which are also of depth 4).

Consequently, the elements of a κ-OWCT T may not be used like this as secrets, they have first to be passed through a one-way hash function H, i.e. the secrets will be

$$S_i = H(\boldsymbol{V}_{r_1,r_2,\ldots,r_\kappa}), \tag{4}$$

where $\boldsymbol{V}_{r_1,r_2,\ldots,r_\kappa} \in T$. The use of the hash function H also results in better performances since it reduces the size of the secrets.

To release a subsequence of secrets $\{S_i, \ldots, S_j\}$, it suffices to reveal their common predecessor of highest weight, say $\boldsymbol{P} \in T$. Note however that, by Proposition 3, this allows to construct all the secrets in the subcross-tree generated by $\boldsymbol{P}$. So, several elements of T must sometimes be released in order to reconstruct *only* the secrets in $\{S_i, \ldots, S_j\}$. Consequently, the secrets have to be carefully arranged into the OWCT in order to minimize the number of elements to release.

3.4 Binary-OWCT vs. OWCT of Higher Dimension

We already learned in §3.1 that a small parameter κ enhances the storage efficiency in the construction of a κ-OWCT. It is worth noting that it may also enhance the overall security, simply because the database containing the root element $\boldsymbol{I} = (I_1, I_2, \ldots, I_\kappa)$ is smaller (κ secret components have to be stored, the other ones are computed), making its maintenance easier.

Another advantage of binary-OWCTs is that each elements has at most two successors. Therefore the release of one element of depth δ enables to derive at most 2 elements of depth $(\delta + 1)$ (and hence at most 2 secrets instead of κ for a κ-OWCT).

Finally, we can remark that, contrary to OWCT of higher dimension, all the components of elements of same depth in a 2-OWCT tree are different. So, for efficiency purposes, if only the leaves of a complete 2-OWCT are used then *one-way* hash function H (see Eq. (4)) may advantageously be replaced by the XOR (exclusive OR) operator in the construction of the secrets. More explicitly, if T is a complete 2-OWCT of depth $n-1$ with root (I_1, I_2), then secrets $S_1, \ldots, S_n$ are given by $S_i = h_1^{i-1}(I_1) \oplus h_2^{n-i}(I_2)$.

4 Applications

4.1 Key Escrow

It is well-known that *key escrow* systems are proposed to reach a balance between the user's privacy and the society security if those systems are employed by an organization or a country. Briefly, a key escrow system goes as follows. A trusted escrow agent is assumed to hold the secret key of each person in the group and it will be asked to reveal the secret key under authorized law enforcement when required. Unfortunately, the main problem in key escrow systems, especially hardware-oriented systems, is that once the personal secret key has been disclosed, all the past and future communications are no longer secure. Consequently, a scheme with time-constrained release of personal secret keys would be very useful. The proposed construction (namely, generation and release of secrets using κ-OWCTs) can successfully be applied to construct such a scheme.

Imagine a company where user's secret keys are periodically updated for security reasons, say each day. Consider a complete 2-OWCT T of depth 364. By Proposition 2, T has 365 leaves $\boldsymbol{V}_{i,364-i} = (h_1^i(I_1), h_2^{364-i}(I_2))$ with $0 \le i \le 364$. As remarked in §3.4, the XOR operator may be used to construct the secrets; therefore we define secret S_j by

$$S_j = h^{j-1}(I_1) \oplus h^{365-j}(I_2) \qquad (1 \le j \le 365) \ . \tag{5}$$

Note that the same one-way function $h_1 = h_2 := h$ has been chosen.

The scheme can therefore be described as follows. At the beginning of each year, each user A receives a tamper-proof hardware from the trusted center or

key escrow agent. Note that this is also the assumption made in the original key escrow standard. In the hardware, there is a real-time clock or a counter that can be used as input to compute the sequence of numbers $\{1, 2, 3, \ldots, 365\}$. The tamper-proof device also stores a secret value I and user's identity ID_A. Define $I_1 = H(I\|1)$ and $I_2 = H(I\|2)$ where H is a one-way hash function. When user A wishes to encrypt a message m to receiver B using a shared session key K_{AB}, both the message and the session key K_{AB} are sent into the hardware and the hardware outputs

$$C = \big(j, \{K_{AB}\}_{SK_j}, ID_A, \{m\}_{K_{AB}}\big) \tag{6}$$

where $1 \le j \le 365$ is the number of the day and $\{p\}_k$ denotes the encryption (any encryption algorithm adopted by the hardware) of message p under secret key k. From integer j (computed from the embedded real-time clock or counter) and the identity ID_A, the hardware computes A's secret key SK_j as

$$SK_j = h^{j-1}(I_1\|ID_A) \oplus h^{365-j}(I_2\|ID_A) \ . \tag{7}$$

On the receiver's (i.e., B) side, his hardware contains the same information except that the identity is now ID_B. From the transmitted C (Eq. (6)), A's secret key SK_j can be recovered (by Eq. (7)) and thus the session key K_{AB} can be correctly extracted. If the computed session key and the entered session key are the same, then receiver's hardware starts to decrypt the ciphertext $\{m\}_{K_{AB}}$, otherwise rejects the decryption request.

Suppose now that for some reason, the wire-tapped communication initiated from A during days 30 till 116 has to be decrypted. Then the law enforcement agent submits the set of numbers $\{30, 31, 32, \ldots, 116\}$ to trusted key escrow agent and receives

$$(h^{30-1}(I_1\|ID_A), h^{365-116}(I_2\|ID_A)) = (h^{29}(I_1\|ID_A), h^{249}(I_2\|ID_A)) \ . \tag{8}$$

From this information, all the secret keys between SK_{30} and SK_{116} can evidently be recomputed. Therefore, all the session keys protected by these secret keys can be extracted and the corresponding wire-tapped ciphertexts can be correctly decrypted. Note that, from Proposition 3, only *one* release of the form $(h^r(I_1\|ID_A), h^s(I_2\|ID_A))$ is permitted. For example, if the law enforcement agent requests the secret keys $SK_{301}, \ldots, SK_{310}$, he may not be provided with $(h^{300}(I_1\|ID_A), h^{55}(I_2\|ID_A))$ because he is then also able to compute $SK_{117}, \ldots, SK_{300}$. Secret keys $SK_{301}, \ldots, SK_{310}$ must thus be released individually.

To overcome this drawback, one can for example construct a complete 2-OWCT of depth 6 with root (I_1, I_2). By Corollary 1, this OWCT has $\binom{8}{6} = 28$ elements. These elements are then numbered in a publicly known way as $\boldsymbol{E}_1, \boldsymbol{E}_2, \ldots, \boldsymbol{E}_{28}$. We define

$$\boldsymbol{R}_i = (R_{i,1}, R_{i,2}) = (H(\boldsymbol{E}_{2i-1}), H(\boldsymbol{E}_{2i})) \quad (1 \le i \le 12)\,, \tag{9}$$

where H is a one-way hash function. From the discussion in §3.3, we know that all of the $H(\boldsymbol{E}_j)$ are independent. Therefore, $\boldsymbol{R}_i$ $(1 \le i \le 12)$ may be used as

roots to construct 12 independent 2-OWCTs. The secret keys of user A are now given by

$$SK_{i,j} = h^{j-1}(R_{i,1} \| ID_A) \oplus h^{31-j}(R_{i,2} \| ID_A)\,, \tag{10}$$

where $1 \leq i \leq 12$ denotes the number of the month and $1 \leq j \leq 31$ denotes the number of the day in month i. Suppose now that the law enforcement agent requests A's secret keys used during 20th March till 10th April, then he will receive

$$(h^{19}(R_{3,1} \| ID_A), R_{3,2} \| ID_A, R_{4,1} \| ID_A, h^{21}(R_{4,2} \| ID_A))\ . \tag{11}$$

The advantage of this second construction is that *one* release of a sequence of secret keys is permitted *each month* instead of each year.

4.2 Delegation

Consider as the previous paragraph that for security reasons secret keys are changed each day. Suppose an employee in a company needs to go on a business trip, and he has no special portable computer available for the access to the company. Before going on his trip, he plans to give the secret keys (for logging into his computer account, for decrypting emails, etc.) to his secretary. Since the secret keys are constructed from OWCTs, the employee has just to release some secret values and not all the secret keys. Then the secretary can compute the corresponding secret keys. She can thus decide in place of the employee according to his policy told beforehand; or if the employee can make a call to the office, she follows instructions given by him. Note that the secretary can only recover the secret keys corresponding to the period during those the employee is away from the office. She is not able to substitute the employee afterwards or to decrypt past communications.

4.3 Lamport-Like Schemes

If the elements of a κ-OWCT (with $\kappa \geq 2$) are passed through a one-way function (see Eq. (4)), they constitute independent secrets—that is, given one or several secrets, it is computationally infeasible to find another one. OWCTs provide thus a simple and efficient means to construct independent secrets. Moreover, to enhance the performance, one can attribute an element of lower weight to a secret which is more often used; the computation of this element (and thus of the corresponding secret) from the root of the OWCT is then speeded up.

Suppose that an user generates N independent secrets in a κ-OWCT ($\kappa \geq 2$). These secrets may represent the initial values of N one-way chains (see Fig. 1). Each of these chains can then for example be used to construct a micro-payment protocol with a given merchant [2,16]. The OWCT just serves as a memo to compute the initial value of a one-way chain. The advantages are (1) the user has just to remember (or store) the root of a OWCT; and (2) if for some reason one secret has been disclosed, the security of the transactions with the other merchants is not compromised. Of course, other applications based on one-way chains (e.g., [3,5,6,7,8,9,10,13,14]) may also be adapted advantageously to such a construction.

5 Conclusions

This paper generalized the concept of one-way chain. The resulting construction, called one-way cross-tree, finds interesting applications in the generation and release of secrets. In a κ-OWCT, only κ secrets have to be stored; moreover, when required, some selected secrets (and only those ones) may efficiently be released.

Acknowledgments

Sung-Ming Yen was supported in part by the National Science Council of the Republic of China under contract NSC 89-2213-E-008-049.

References

1. FIPS 180-1. Secure hash standard. Federal Information Processing Standards Publication 180-1, NIST, U.S. Department of Commerce, April 1995.
2. R. Anderson, H. Manifavas, and C. Sutherland. A practical electronic cash system. Available from URL `<http://www.cl.cam.ac.uk/users/rja14/>`, 1995.
3. N. Asokan, G. Tsudik, and M. Waidner. Server-supported signatures. In E. Bertino, editor, *Fourth European Symposium on Research in Computer Security (ESORICS '96)*, volume 1146 of *Lecture Notes in Computer Science*, pages 131–143. Springer-Verlag, 1996.
4. D. Bleichenbacher and U.M. Maurer. Directed acyclic graphs, one-way functions and digital signatures. In Y.G. Desmedt, editor, *Advances in Cryptology — CRYPTO '94*, volume 839 of *Lecture Notes in Computer Science*, pages 75–82. Springer-Verlag, 1994.
5. O. Delos and J.-J. Quisquater. An identity-based signature scheme with bounded life-span. In Y.G. Desmedt, editor, *Advances in Cryptology — CRYPTO '94*, volume 839 of *Lecture Notes in Computer Science*, pages 83–94. Springer-Verlag, 1994.
6. D. de Waleffe and J.-J. Quisquater. Better login protocols for computer networks. In B. Preneel, R. Govaerts, and J. Vandewalle, editors, *Computer Security and Industrial Cryptography*, volume 741 of *Lecture Notes in Computer Science*, pages 50–70. Springer-Verlag, 1993.
7. S. Even, O. Goldreich, and S. Micali. On-line/off-line digital signatures. In G. Brassard, editor, *Advances in Cryptology — CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 263–275. Springer-Verlag, 1990.
8. N.M. Haller. The S/KEY one-time password system. In *Proc. of the ISOC Symposium on Networks and Distributed Systems Security*, 1994.
9. L. Lamport. Constructing digital signatures from a one-way function. Technical Report CSL-98, SRI International, 1979.
10. L. Lamport. Password authentication with insecure communication. *Comm. ACM*, 24(11):770–772, November 1981.
11. M. Mambo, K. Usuda, and E. Okamoto. Proxy signatures for delegating signing operations. In *Proc. of the 3rd ACM Conference on Computer and Communications Security*, pages 48–57. ACM Press, 1996.
12. R.C. Merkle. A digital signature based on a conventional encryption function. In C. Pomerance, editor, *Advances in Cryptology — CRYPTO '87*, volume 293 of *Lecture Notes in Computer Science*, pages 369–378. Springer-Verlag, 1988.

13. R.C. Merkle. A certified digital signature. In G. Brassard, editor, *Advances in Cryptology — CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 218–238. Springer-Verlag, 1990.
14. M.O. Rabin. Digitalized signatures. In D. Dobkin, A. Jones, and R. Lipton, editors, *Foundations of Secure Computation*, pages 155–168. Academic Press, 1978.
15. R. Rivest. The MD5 message digest algorithm. Internet Request for Comments RFC 1321, April 1992. Available at `<ftp://ds.internic.net/rfc/rfc1321.txt>`.
16. R.L. Rivest and A. Shamir. PayWord and MicroMint: two simple micropayment schemes. *CryptoBytes*, **2** (1), 7–11, 1996.