Linear Polynomial-Based Hierarchical Access Control

Dr. Ashok Kumar Das

IEEE Senior Member
Associate Professor
Center for Security, Theory and Algorithmic Research
(Department of Computer Science and Engineering)
International Institute of Information Technology, Hyderabad
(Formerly **Indian Institute of Information Technology, Hyderabad**)

E-mail: *ashok.das@iiit.ac.in*
Homepage: http://www.iiit.ac.in/people/faculty/ashokkdas
Personal Homepage: https://sites.google.com/view/iitkgpakdas/

September 25, 2022

Linear Polynomial-Based Hierarchical Access Control

Linear polynomial over a finite field $GF(p)$

- A linear polynomial $f(x) = a_1 x + a_0 \pmod{p}$ is a polynomial of degree one, where $a_1, a_0 \in GF(p)$ are the coefficients and $GF(p)$ is the Galois field over the prime p. We call $f(x)$ is monic linear polynomial, if $a_1 = 1$. We can store this polynomial for storing only one coefficient a_0 since the coefficient of x is always 1 for such polynomial.
- We consider an m-degree polynomial in $GF(p)$ of the form $g(x) = \sum_{i=0}^{m} a_i x^i \pmod{p}$, where $a_i \in GF(p)$. This polynomial $g(x)$ becomes the monic polynomial if $a_m = 1$. It is also clear to note that we can store this polynomial by storing m coefficients a_i, for $i = 0, 1, 2, \ldots, m-1$, and omitting the coefficient of x^m since it is always 1 in such case.

Linear polynomial over a finite field $GF(p)$

- We consider the special form of an m-degree polynomial as follows

$$h(x) = \prod_{i=1}^{m}(x - a_i) + b \pmod{p} \tag{1}$$

where p is prime, $a_i \in GF(p)$ and $b \in GF(p)$. Note that $h(x)$ is clearly a monic polynomial over $GF(p)$, which requires the storage space for storing m coefficients only since the coefficient of x^m is always 1. If we now consider the following m linear polynomials of the forms

$$\begin{aligned}
h_1(x) &= (x - a_1) + b \pmod{p} \\
&= x + (-a_1 + b) \pmod{p} \\
h_2(x) &= (x - a_2) + b \pmod{p} \\
&= x + (-a_2 + b) \pmod{p} \\
\vdots \quad & \quad \vdots \\
h_m(x) &= (x - a_m) + b \pmod{p} \\
&= x + (-a_m + b) \pmod{p} \qquad (2)
\end{aligned}$$

Linear polynomial over a finite field $GF(p)$

- Each $h_i(x)$ also requires to store only one coefficient $(-a_i + b)$ which is enough as they are monic polynomials. From Equations (1) and (2), it is clear to observe that if we split an m-degree polynomial into m individual linear polynomials, the required storage space for m linear polynomials is same as that for the m-degree polynomial.
- The cost of constructing an interpolating polynomial of degree m over a finite field $GF(p)$ requires exactly m additions, $2m^2 + 2$ subtractions, $2m^2 + m - 1$ multiplications and $m + 1$ divisions. Also, the computational complexity for constructing an m-degree interpolating polynomial by applying Fast Fourier Transformation is $O(m(\log\ m)^2)$. Constructing a linear polynomial requires only one modular addition. For evaluating the m-degree polynomial $h(x)$ at a point $x = c$ requires m modular additions and m modular multiplications, whereas for evaluating a linear polynomial $h_i(x)$ at a point $x = c$ requires only one modular addition.
- **SUMMARY:** If we use m linear polynomials instead of using an m-degree polynomial, the storage overhead remains same for both cases and computational overheads for constructing polynomials and evaluating them are significantly reduced in former situation.

Linear Polynomial-Based Hierarchical Access Control

Research Paper: Vanga Odelu, Ashok Kumar Das, and Adrijit Goswami. “A novel linear polynomial-based dynamic key management scheme for hierarchical access control,” in ***International Journal of Trust Management in Computing and Communications (Inderscience)***, Vol. 1, No. 2, pp. 156-174, 2013, DOI: 10.1504/IJTMCC.2013.053278.

The Proposed Scheme

Table: Notations

Symbol	Description
ID_{CA}	Identity of CA
SC_i	i-th security class in hierarchy
SC	User hierarchy, $SC = \{SC_1, SC_2, \ldots, SC_N\}$
N	Number of security classes in hierarchy
$H(\cdot)$	Secure collision-resistant one-way hash function
Ω	Symmetric key cryptosystem
$E_k(\cdot)/D_k(\cdot)$	Symmetric-key encryption/decryption using key k
$\|\|$	Bit concatenation operator

Relationship building phase

- In this phase, CA first needs to build the hierarchical structure for controlling access according to the given relationships among the security classes in the hierarchy.
- Let $SC_i \in SC$ and $SC_j \in SC$ be two security classes such that the relationship $SC_j \leq SC_i$ hold, that is, SC_i have a higher security clearance than that for SC_j.
- A legitimate relationship $(SC_i, SC_j) \in R_{i,j}$ between SC_i and SC_j will exist in the hierarchy if SC_i can access SC_j.

Key generation phase

Once the relationship building phase is completed by the CA, the CA can execute the following steps for distributing the secret and public keys to security classes in the given hierarchy:

- Step 1. CA first selects a secure collision-resistant one-way hash function $H(\cdot)$ and then a finite field $GF(m)$, where m is either odd prime or prime power. CA also chooses a symmetric key cryptosystem Ω (for example, AES).
- Step 2. CA randomly selects its own secret key k_{CA}. After that CA needs to select randomly the secret key sk_i and sub-secret key d_i for each security class SC_i $(1 \leq i \leq N)$ in the hierarchy.
- Step 3. Once Step 2 is completed, for each security class SC_i, CA computes the signature $Sign_i$ on sk_i as $Sign_i = H(ID_{CA}||sk_i)$. The purpose of signature is used later by the CA and security classes for verification of the secret key sk_i of SC_i. CA declares them as public.

Key generation phase

- Step 4. For each SC_i such that $(SC_i, SC_j) \in R_{i,j}$, CA then constructs the linear polynomials $f_{i,j}(x) = (x - H(ID_{CA}||Sign_j||d_i)) + sk_j \pmod{m}$, and declares them publicly.
- Step 5. Finally, CA sends d_i to SC_i via a secure channel.

CA now encrypts the sub-secret key d_i of each security class SC_i as $S_i = E_{k_{CA}}(d_i)$ using the secret key k_{CA} of the CA, computes its signature Sd_i as $Sd_i = H(ID_{CA}||d_i)$ for the purpose of the signature verification of d_i and then stores the pair (S_i, Sd_i) in the public domain. For security reasons, CA then deletes all the secret keys sk_i and d_i generated during this phase. We observe that whenever CA needs to update the secret keys sk_i's of SC_i's, CA first obtains d_i's from public parameters S_i's by decrypting them with its secret key k_{CA} and then verifies signatures by calculating the hash values as $Sd_i' = H(ID_{CA}||d_i)$, and verifies the condition $Sd_i' = Sd_i$. If it is valid, CA confirms that derived secret key d_i is legitimate.

Key generation phase

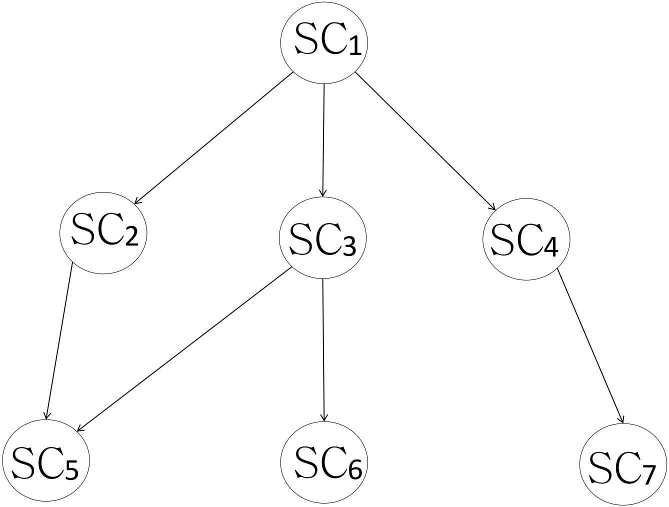

Figure: Figure 1. An example of a poset in a user hierarchy

Key generation phase

Example 1. The public parameters corresponding to each security class in the proposed scheme to be stored in public domain are given in following table.

Table: Key generation for the hierarchy in Figure 1.

Security class	Corresponding public parameters to be stored in public domain
SC_1	(S_1, Sd_1), $Sign_1$, $f_{1,1}(x)$, $f_{1,2}(x)$, $f_{1,3}(x)$, $f_{1,4}(x)$, $f_{1,5}(x)$, $f_{1,6}(x)$, $f_{1,7}(x)$
SC_2	(S_2, Sd_2), $Sign_2$, $f_{2,2}(x)$, $f_{2,5}(x)$
SC_3	(S_3, Sd_3), $Sign_3$, $f_{3,3}(x)$, $f_{3,5}(x)$, $f_{3,6}(x)$
SC_4	(S_4, Sd_4), $Sign_4$, $f_{4,4}(x)$, $f_{4,7}(x)$
SC_5	(S_5, Sd_5), $Sign_5$, $f_{5,5}(x)$
SC_6	(S_6, Sd_6), $Sign_6$, $f_{6,6}(x)$
SC_7	(S_7, Sd_7), $Sign_7$, $f_{7,7}(x)$

Key derivation phase

In this phase, a security class SC_i can derive the secret key sk_j of its successor SC_j with $(SC_i, SC_j) \in R_{i,j}$. SC_i needs to proceed the following steps for this:

- Step 1. SC_i first computes the hash value $H(ID_{CA}||Sign_j||d_i)$ using its own sub-secret key d_i, signature $Sign_j$ available in the public domain and the identity ID_{CA} of the CA.
- Step 2. SC_i then obtains the secret key sk_j of SC_j's (including SC_i) as $sk_j = f_{i,j}(H(ID_{CA}\,||Sign_j||d_i))$ by evaluating the corresponding public linear polynomial $f_{i,j}(x)$ for SC_j at the point $x = H(ID_{CA}\,||Sign_j||d_i)$.
- Step 3. SC_i verifies the signature of sk_j as follows. SC_i computes the signature $Sign_j^{'} = H(ID_{CA}||sk_j)$ using the derived secret key sk_j and then checks if the condition $Sign_j^{'} = Sign_j$ holds. If it holds, SC_i ensures that the derived secret key sk_j is correct and valid.

Adding new security classes phase

Let a new security class SC_l with the relationships $SC_j \leq SC_l \leq SC_i$ to be added into the existing hierarchy. CA performs the following steps to manage the accessibility of SC_l:

- Step 1. CA first randomly selects the secret key sk_l and the sub-secret key d_l for the new class SC_l.
- Step 2. For SC_l, CA needs to compute the signature $Sign_l$ on the secret key sk_l as $Sign_l = H(ID_{CA} || sk_l)$ for the purpose of signature verification of sk_l. CA then publicly declares it.
- Step 3. For each SC_i for which the relationship $(SC_i, SC_l) \in R_{i,l}$ holds in the hierarchy, CA constructs the linear polynomials $f_{i,l}(x)$ $= (x - H(ID_{CA}||Sign_l||d_i)) + sk_l \pmod{m}$, and declares them publicly.
- Step 4. For each SC_j such that the relationship $(SC_l, SC_j) \in R_{l,j}$ holds, CA constructs the linear polynomials $f_{l,j}(x)$ $= (x - H(ID_{CA}||Sign_j||d_l)) + sk_j \pmod{m}$, and declares them publicly.

Adding new security classes phase

- Step 5. CA finally sends the sub-secret key d_l to SC_l via a secure channel.

At the end of the above steps, CA further encrypts the sub-secret key d_l of SC_l as $S_l = E_{k_{CA}}(d_l)$, computes signature Sd_l on d_l as $Sd_l = H(ID_{CA}||d_l)$ for the purpose of signature verification of d_l later and stores the pair (S_l, Sd_l) in the public domain. CA also deletes secret keys sk_l and d_l for security reasons.

Adding new security classes phase

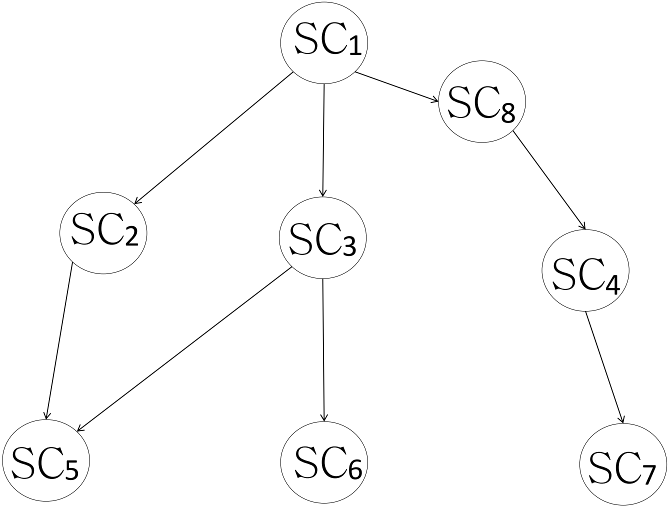

Figure: Hierarchy after adding a new security class SC_8 in Figure 1.

Adding new security classes phase

Example 2. Suppose a new security class SC_8 be added into the hierarchy shown in Figure 1. The resultant hierarchy after addition of SC_8 is given in Figure 2 with the added relationships $SC_4 \leq SC_8 \leq SC_1$. CA selects the sub-secret key d_8 and secret key sk_8 for the security class SC_8 and then calculates the signature $Sign_8 = H(ID_{CA}||sk_8)$ on sk_8, the public linear polynomials $f_{8,8}(x)$ $= (x - H(ID_{CA}||Sign_8||d_8) + sk_8 (\bmod m)$, $f_{8,4}(x)$ $= (x - H(ID_{CA}||Sign_4||d_8) + sk_4 (\bmod m)$, $f_{8,7}(x)$ $= (x - H(ID_{CA}||Sign_7||d_8) + sk_7 (\bmod m)$ and updates the predecessor's public linear polynomial $f_{1,8}(x)$ $= (x - H(ID_{CA}||Sign_8||d_1) + sk_8 (\bmod m)$. Finally, CA computes the public parameter $S_8 = E_{K_{CA}}(d_8)$ and signature $Sd_8 = H(ID_{CA}||d_8)$ corresponding to sub-secret key d_8 for future use in signature verification.

Deleting existing security classes

Deleting an existing security class SC_l with the relationship $SC_j \leq SC_l \leq SC_i$ from the hierarchy works in the proposed scheme as follows.

CA executes the following steps in order to remove SC_l so that the forward security is preserved:

- Step 1. CA removes all the parameters corresponding to SC_l.
- Step 2. CA renews the secret keys sk_j's of successors SC_j's of SC_l as sk_j^*, and signatures $Sign_j$'s as $Sign_j^* = H(ID_{CA}||sk_j^*)$ and replaces $Sign_j$ with $Sign_j^*$ in the public domain.
- Step 3. For each security class SC_i such that $SC_j \leq SC_i$ $(\neq SC_l)$ in the hierarchy, CA constructs the linear polynomials $f_{i,j}^*(x)$ $= (x - H(ID_{CA}||Sign_j^*||d_i)) + sk_j^* \pmod{m}$ and declares them publicly.

Deleting existing security classes

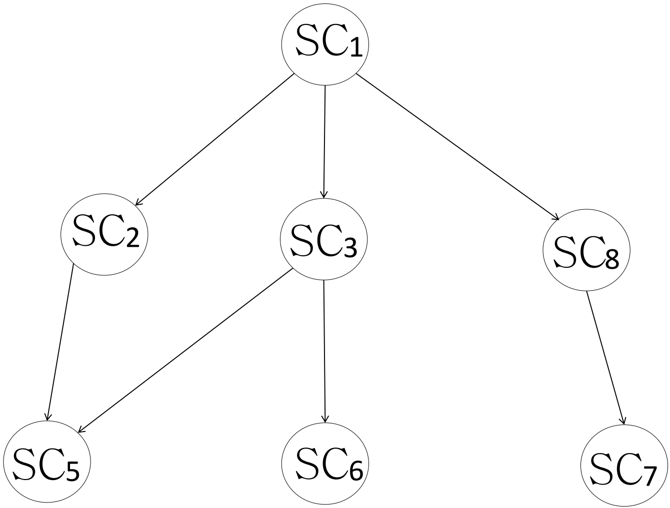

Figure: Hierarchy after deleting the security class SC_4 from the hierarchy in Figure 2.

Deleting existing security classes

Example 3. Suppose the security class SC_4 be deleted from the hierarchy in Figure 2. Then the resultant hierarchy is shown in Figure 3. CA renews the secret key sk_7 as sk_7^* and computes the signature $Sign_7^* = H(ID_{CA}||sk_7^*)$.
CA then updates the public linear polynomials $f_{7,7}^*(x)$ $= (x - H(ID_{CA}||Sign_7||d_7)) + sk_7^* \pmod m$, $f_{8,7}^*(x)$ $= (x - H(ID_{CA}||Sign_7||d_8)) + sk_7^* \pmod m$ and $f_{1,7}^*(x)$ $= (x - H(ID_{CA}||Sign_7||d_1)) + sk_7^* \pmod m$ of the security class SC_7 and its predecessors SC_8 and SC_1, respectively.

Creating new relationships

- Suppose a new relationship $SC_j \leq SC_i$ between two immediate security classes SC_j and SC_i be added in the hierarchy.
- Further assume that $SC_i \leq SC_l$ and $SC_y \leq SC_j$ (SC_y is not successor of SC_l before creating this relationship).
- For this purpose, CA needs to compute the linear polynomials $f_{l,y}(x) = (x - H(ID_{CA}||Sign_y||d_l)) + sk_y \pmod m$ and publicly declare them.

Creating new relationships

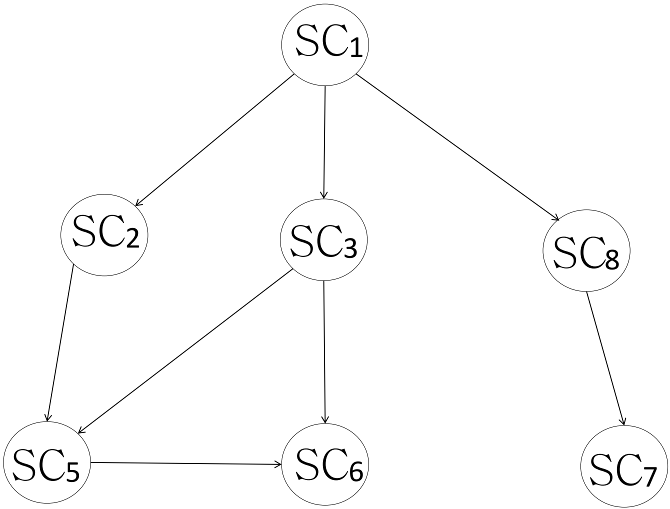

Figure: After creating relationship from SC_5 to SC_6 in Figure 1

Creating new relationships

Example 4. Consider again the hierarchy in Figure 1. The predecessors of the class SC_5 are SC_3, SC_2 and SC_1, whereas the predecessors of SC_6 are SC_3 and SC_1. Suppose the new relationship be created between two security classes SC_5 and SC_6. The resultant hierarchy is shown in Figure 4. The security class SC_6 is the successor of SC_1 and also the successor of the class SC_3 before creating new relationship from SC_5 to SC_6. After creating this new relationship from class SC_5 to SC_6, CA does not need to construct the linear polynomial $f_{1,6}(x)$ and $f_{3,6}$ again. It is enough only to compute the linear polynomials $f_{5,6}(x)$ and $f_{2,6}(x)$ for the predecessors SC_5 and SC_2 of security class SC_6.

Revoking existing relationships

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY HYDERABAD

- Let the relationship $SC_j \leq SC_i$ between two immediate security classes SC_j and SC_i be deleted from the existing hierarchy.
- Assume that $SC_j \leq SC_l$ ($\neq SC_i$) and $SC_y \leq SC_j$. CA removes all parameters corresponding to the keys sk_y (including that for sk_j).
- CA also renews the secret keys sk_y as sk_y^* and updates the signatures $Sign_y$ as $Sign_y^* = H(ID_{CA}||sk_y^*)$ in the public domain.
- Finally, CA constructs the linear polynomials $f_{l,y}^*(x)$ $= (x - H(ID_{CA}||Sign_y^*||d_l)) + sk_y^* \pmod{m}$ and declares them publicly.

Revoking existing relationships

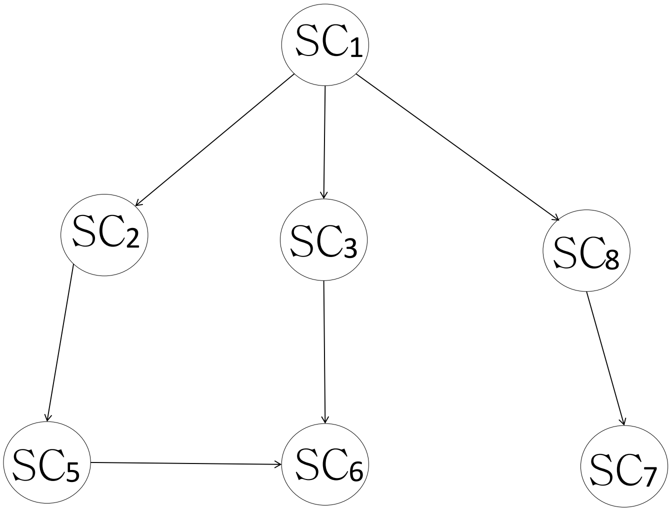

Figure: After removing relationship from SC_3 to SC_5 from the Figure 3(a)

Revoking existing relationships

Example 5. Consider the hierarchy in Figure 4 and delete the relation from SC_3 to SC_5. The resultant hierarchy is shown in Figure 5. Now, CA needs to update the secret keys of the successors SC_5 and SC_6 of the security class SC_3. However, SC_6 is a successor of SC_3 after revoking the relationship. Hence, it is enough to renew the secret key sk_5 as sk_5^* for the security class SC_5 and update its corresponding linear polynomials $f_{5,5}(x)$, $f_{2,5}(x)$ and $f_{1,5}(x)$.

Changing secret keys

If we want to change the secret key sk_j of a security class SC_j, where $SC_j \leq SC_i$, CA then needs to execute the following steps:

- Renew the secret key sk_j as sk_j^*, update the signature $Sign_j$ by $Sign_j^* = H(ID_{CA}||sk_j^*)$
- Compute the corresponding linear polynomials $f_{i,j}^*(x)$ $= (x - H(ID_{CA}||Sign_j^*||d_i)) + sk_j^* \pmod{m}$ which are declared publicly.

Security analysis

Contrary attack: In this attack, the successor class SC_j of a security class SC_i ($SC_j \leq SC_i$), being an insider attacker, tries to derive the secret key sk_i of its predecessor class SC_i from the available public parameters $f_{i,i}(x) = (x - H(ID_{CA}||Sign_i||d_i)) + sk_i \pmod{m}$ and $f_{i,j}$'s. Without knowing the sub-secret key d_i of SC_i, it is an infeasible task for SC_j to compute $H(ID_{CA}||Sign_i||d_i)$ and hence, as a result the secret key sk_i too. Note that the pairs $(Sign_j, d_i)$ are used in the construction of linear polynomials, which are distinct for two different polynomials. Further, from the public parameter $S_i = E_{k_{CA}}(d_i)$, SC_j or any other user (except CA) cannot retrieve d_i without knowing CA's private key k_{CA}. the proposed scheme is thus secure against contrary attack.

Security analysis

Exterior collecting attack: This potential attack is from an external attack, where that intruder tries to derive the secret key from lower level security classes through the accessible public parameters. However, the task of computing the secret key of a security class becomes a computationally infeasible due to collision-resistant property of the one-way hash function $H(\cdot)$. As a result, no external intruder can retrieve the secret key of any security class. Hence, the proposed scheme protects such attack.

Security analysis

Collaborative attack: In this attack, several users in a hierarchy try to collaborate to launch an attack for computing their predecessor's secret key. Assume that SC_j and SC_l are two immediate successor classes of a predecessor class SC_i and they try to derive the secret key sk_i of their predecessor SC_i. In order to do this, they can exchange secret keys with each other and derive the sub-secret key d_i of SC_i in order to derive the secret key sk_i of SC_i through the public linear polynomials $f_{i,j}(x) = (x - H(ID_{CA}||Sign_j||d_i)) + sk_j \pmod{m}$ and $f_{i,l}(x) = (x - H(ID_{CA}||Sign_l||d_i)) + sk_l \pmod{m}$. However, the sub-secret key d_i of SC_i is masked with one-way hash function $H(\cdot)$ during our key generation phase. As a result, the task of deriving d_i is a computational infeasible problem due to hash function's collision-resistant properties. Thus, no successor class can obtain the secret key of a predecessor class by collaborating each other. the proposed scheme has then the ability to withstand and then such an attack.

Security analysis

Equation attack: Suppose a security class SC_j has common predecessors SC_i and SC_l, where SC_i does not have an accessibility relationship with SC_l. Let SC_i, being an insider attacker, try to access the secret key sk_l of SC_l through the public linear polynomials $f_{l,j}(x)$
$= (x - H(ID_{CA}||Sign_j||d_l)) + sk_j \pmod{m}$ and $f_{l,l}(x)$
$= (x - H(ID_{CA}||Sign_l||d_l)) + sk_l \pmod{m}$. SC_i can compute $H(ID_{CA}||Sign_j||d_l)$ from $f_{l,j}(x)$ by using the derived secret key sk_j of SC_j, but SC_i cannot compute the sk_l from $f_{l,l}(x)$, since the hash values $H(ID_{CA}||Sign_j||d_l)$ and $H(ID_{CA}||Sign_l||d_l)$ are different. Thus, the linear polynomials corresponding to one security class cannot be solvable by other security classes and as a result, the proposed scheme has also the ability to protect this attack.

Security analysis

Forward security of successors while changing $SC_j \leq SC_k \leq SC_i$ to $SC_j \leq SC_i$: Let the relationship $SC_j \leq SC_k \leq SC_i$ be modified to another relationship $SC_j \leq SC_i$ after removing the security class SC_k from an existing hierarchy. Note that CA not only deleted the accessibility relationship $SC_j \leq SC_k$, but it also updated the accessibility-link relationship between SC_i and SC_j. CA further renewed the secret keys sk_j's of SC_j's and the corresponding linear polynomials $f^*_{i,j} = (x - H(ID_{CA}||Sign^*_j||d_i)) + sk^*_j \pmod m$. Now, the hash values $H(ID_{CA}||Sign^*_j||d_i)$ can be computed only by the security class SC_i and thus, the security class SC_k cannot hack the updated key sk^*_j of SC_j later. Therefore, the authority of SC_k over SC_j is completely terminated, and the proposed scheme preserves the forward security property.

Security analysis

Man-in-the-middle attack: We refer the "man-in-the-middle" attack as the masquerade attack, where an attacker wants to be represented as an authorized central authority. Though the public domain is write-protected, if the attacker can update somehow the information in the public domain, then assume the attacker changes the public linear polynomials $f_{i,j}(x)$'s in the public domain. The derivation of the secret key sk_j of a security class SC_j becomes a computationally infeasible problem, since the sub-secret key d_j is only known to SC_j. As a result, the attacker does not have any ability to change properly the signatures $Sign_j = H(ID_{CA}||sk_j)$ and $Sd_j = H(ID_{CA}||d_j)$ in the public domain. Hence, the proposed scheme is secure against such an potential active attack.

Summary

- The proposed dynamic access control scheme in a user hierarchy utilizes the advantages of the linear polynomials over higher degree polynomials over a finite field along with symmetric-key cryptosystem in order to achieve the required goals which are required for designing an idle access control scheme.
- The designed scheme offers low computational cost and small storage space as compared with other schemes.
- Through the informal and formal security analysis it is shown that the designed scheme is secure against known attacks including the serious active attack, man-in-the-middle attack.
- The designed scheme is thus more effective than the previously proposed schemes and much appropriate for practical applications.