Key Assignment Scheme with Authenticated Encryption

Suyash Kandele and Souradyuti Paul

Indian Institute of Technology Bhilai {suyashk,souradyuti}@iitbhilai.ac.in

Abstract. The *Key Assignment Scheme (KAS)* is a well-studied cryptographic primitive used for *hierarchical access control (HAC)* in a multilevel organisation where the classes of people with higher privileges can access files of those with lower ones. Our first contribution is the formalization of a new cryptographic primitive, namely, *KAS-AE* that supports the aforementioned *HAC* solution with an additional *authenticated encryption* property. Next, we present three efficient *KAS-AE* schemes that solve the *HAC* and the associated *authenticated encryption* problem more efficiently – both with respect to time and memory – than the existing solutions that achieve it by executing *KAS* and *AE* separately. Our first *KAS-AE* construction is built by using the cryptographic primitive *MLE* (EUROCRYPT 2013) as a black box; the other two constructions (which are the most efficient ones) have been derived by cleverly tweaking the hash function *FP* (Indocrypt 2012) and the authenticated encryption scheme *APE* (FSE 2014). This high efficiency of our constructions is critically achieved by using two techniques: design of a mechanism for *reverse decryption* used for reduction of time complexity, and a novel key management scheme for optimizing storage requirements when organizational hierarchy forms an arbitrary access graph (instead of a linear graph). We observe that constructing a highly efficient *KAS-AE* scheme using primitives other than *MLE*, *FP* and *APE* is a non-trivial task. We leave it as an open problem. Finally, we provide a detailed comparison of all the *KAS-AE* schemes.

Keywords: Key assignment schemes (KAS) · Message-locked encryption (MLE) · Authenticated encryption (AE) · Hierarchical access control · Partially ordered set · Totally ordered set

1 Introduction

Hierarchical Access Control (HAC) and the Key Assignment Scheme (KAS). *Hierarchical access control* is a mechanism that allows the classes of people in an organisation with varying levels of privileges to access data based on their positions. Nowadays, since most of the organisations have hierarchical structures, and since their data is stored in public servers or on the *cloud*, secure and efficient *HAC* solutions have gained importance.

From a high level, so far, the *HAC* problem has been solved using the following two-step methodology: distribute the secret keys to various classes of people in the organization such that the people in the higher class can derive the secret keys owned by the classes below it; after the distribution of keys, all the data are encrypted using *symmetric encryption* (data authentication may also be incorporated in some way). Loosely speaking, the secure generation of these secret keys, as done in the first step of the above *HAC* methodology, is known as *Key Assignment Scheme*. The idea of *KAS* and its practical construction was introduced by Akl and Taylor in 1983 [AT83]. Since then, for over three decades, a large number of *KAS* constructions have been proposed in the literature with extensive study of their security properties [ABFF09, AFB05, CC02, CCM15, CDM10, CMW06, CSM16a, CSM$^+$16b, DSFM10, FP11, FPP13, HL90, SC02, SFM07a, WC01, YCN03].

A NEW CRYPTOGRAPHIC PRIMITIVE KAS-AE: A MOTIVATION. In all the above cases, *hierarchical access control* with *authenticated encryption* is achieved by following the same design paradigm: execute *KAS* first, and then execute *AE*. So far research in solving *HAC* mainly revolves around designing various types of *KAS* constructions. To the best of our knowledge, no attempt has been made so far to explore the possibility of building efficient *HAC* solutions by combining *KAS* and *AE* in some non-trivial ways. Our main motivation in this paper is to combine *KAS* and *AE* into a single primitive, and solve the *HAC* problem. It is very important to note at this point that a new cryptographic primitive combining *KAS* and *AE*, such as *KAS-AE* of this paper, makes little sense if it does not permit constructions that are significantly more efficient than the trivial combination of *KAS* and *AE*. Therefore, we summarize our main challenge below:

> *Can we construct a secure KAS-AE scheme that solves HAC problem more efficiently than the simple combination of KAS and AE executed in that order?*

In the remainder of the paper, we search for answers to the above question, and analyze them.

OUR CONTRIBUTION. Our first contribution is defining and formalizing a new cryptographic notion, namely, *key assignment scheme with authenticated encryption*, (or *KAS-AE* for short). To develop, motivate, analyze and easily understand this new idea, we propose a total of nine *KAS-AE* constructions – except one all are proven secure – with varying degrees of efficiencies and construction subtleties: (1) in the first construction, we show that the most natural combination of *KAS* and *AE* to generate *KAS-AE* is prone to attack; (2) in the second construction, we obviate this attack, and show a secure way of combining *KAS* and *AE* to build a *KAS-AE* scheme; (3-6) Our next four *KAS-AE* constructions are based on first building *KAS-AE* schemes for linear graphs (or totally ordered sets) and then combining them to support arbitrary access graphs (i.e. partially ordered sets); (7-9) these last three constructions are the most efficient *KAS-AE* constructions, they are based on a novel use of *Message-Locked Encryption (MLE)* [BKR13], of a hash function mode *FP* [PHG12] and of an authenticated encryption mode *APE* [ABB$^+$14], respectively.

Our best three constructions (see Table 3) outperform all other conventional *HAC* solutions (based on *KAS* and *AE* individually, as opposed to on the single primitive *KAS-AE*) with respect to running time by a factor of at least 2 (or 3) for any reasonable parameter choices; also, the *private storage* of our best performing constructions is linear, whereas they are quadratic (or cubic) in the simple combination of *KAS* and *AE*. A detailed comparison will be given in Table 1, Table 2 and Table 3.

In order to obtain this improvement in performance, our constructions exploit, among others, a very unique feature – what we call *reverse decryption* – supported by the hash function *FP* and the authenticated encryption *APE*. It turns out that the *reverse decryption* property can also be obtained by a clever use of *MLE* schemes. Besides this, our constructions also benefit from a novel key management technique to optimize the storage requirements in the very challenging scenarios where organizational access structure is non-linear (i.e., a poset, rather than a totally ordered set).

Note that the very unique *reverse decryption* property – which, to the best of our knowledge, only exists inherently in the *FP* hash mode and the *APE* authenticated encryption scheme – has also been used in [KP18] to construct efficient *file-updatable message-locked encryption (FMLE)* schemes. However, our focus in this paper is an efficient solution for the very different and fairly old *Hierarchical Access Control* problem that, unlike the *FMLE* solution, involves a significant amount of intricate graph theoretic algorithms and tools (e.g. root-finding algorithm, shortest path algorithm, etc.) to overcome crucial key management challenges. Nevertheless, our work certainly constitutes a novel and important application of the *reverse decryption* property of *FP* and *APE*.

RELATED WORK. *KAS* has been studied for over three decades. In 1983, the first *KAS* scheme was proposed by Akl and Taylor, in which each user stores one secret key, and derives the other keys using some public values [AT83]. MacKinnon *et al.* have attempted to optimize the solution proposed by Akl and Taylor [MTMA85]. Since then, a large number of *KAS* constructions have been proposed in the literature [ABFF09, AFB05, CC02, CCM15, CDM10, CMW06, CSM16a, CSM$^+$16b, DSFM10, FP11, FPP13, HL90, SC02, SFM07a, WC01, YCN03]. Crampton *et al.* have extensively studied the existing *KAS* constructions, and classified them into five generic schemes [CMW06]. They have also highlighted the advantages and disadvantages of the generic schemes.

Crampton, Daud and Martin have discussed procedure for designing *KAS* constructions by using *KAS-chains* and an innovative *chain partition* algorithm [CDM10]; this scheme was also used to construct *KAS* with useful performance-security trade-offs [FPP13]. A special type of *KAS* with expiry date and/or time for key, called *Time-bound Key Assignment Schemes*, has also been studied, and various schemes of this type have been proposed [ABF07, ASFM06, ASFM12, ASFM13, BSJ08, Chi04, HC04, PWCW15a, PWCW15b, SFM07b, SFM08, Tze02, Tze06, WL06, Yeh05, Yi05, YY03].

ORGANISATION OF THE PAPER. In Section 2, we discuss the preliminaries including the notation, basic definitions and existing constructions. In Section 3, we give the formal definition of *KAS-AE*. Section 4 describes a secure yet inefficient *KAS-AE* construction built by combining existing *KAS* and *AE*. Section 5 describes the four efficient *KAS-AE* constructions built using *modified chain partition* and *KAS-AE-chain* constructions. Section 6 describes an efficient *KAS-AE* constructions built using *MLE*. Section 7 describes two highly-efficient *KAS-AE* constructions built by tweaking existing constructions. In Section 8, we compare various *KAS-AE* schemes and conclude our paper in Section 9.

2 Preliminaries

2.1 Notation

$M := x$ denotes that the value of x is assigned to M, and $M := \mathcal{D}(x)$ denotes that the value returned by function $\mathcal{D}()$ for input x, is assigned to M. $M = x$ denotes the equality comparison of the two variables M and x, and $M = \mathcal{D}(x)$ denotes the equality comparison of the variable M with the output of $\mathcal{D}()$ on input x. The XOR or $\oplus$ denotes the bit-by-bit *exclusive-or* operation on two binary strings of same length. The concatenation operation of $p \geq 2$ strings $s_1, s_2, \cdots, s_p$ is denoted as $s_1||s_2||\cdots||s_p$. The length of string M is denoted by $|M|$. The set of all binary strings of length ℓ is denoted by $\{0,1\}^\ell$. The set of all binary strings of any length is denoted by $\{0,1\}^*$. The set of all natural numbers is denoted by $\mathbb{N}$. We denote that M is assigned a string of length k chosen randomly and uniformly by $M \xleftarrow{\$} \{0,1\}^k$. To mark any invalid string (may be input string or output string), the symbol $\perp$ is used. In a vector of strings f, the string corresponding to user i is denoted by f_i. The number of strings in f is denoted by $\|f\|$. $(f_u)_{u \in V}$ denotes the sequence of strings f_u, where $u \in V$. The symbols f_u, S_u and k_u denote the file, private information and decryption key held by user u, c_u denotes the ciphertext corresponding to f_u. $f = (f_u)_{u \in V}$, $S = (S_u)_{u \in V}$ and $k = (k_u)_{u \in V}$ denote the sequence of files, private information and keys for all the nodes in the graph $G = (V, E)$. The operation $f_1 \circ f_2 \circ \cdots \circ f_p$, for some value of p, denotes the sequence of strings $f_1, f_2, \cdots, f_p$. $(\boldsymbol{M}_0, \boldsymbol{M}_1, Z) \xleftarrow{\$} \mathcal{S}(1^\lambda)$ denotes the assignment of outputs given randomly and uniformly by $\mathcal{S}$ to $\boldsymbol{M}_0$ and $\boldsymbol{M}_1$ and supplying some auxiliary information Z. Here, $\boldsymbol{M}_0$ and $\boldsymbol{M}_1$ is a vector of strings and i-th string in

$\boldsymbol{M}$ is denoted as $\boldsymbol{M}^{(i)}$. The encryption function $\mathcal{E}$ of authenticated encryption, as defined in Subsubsection 2.2.6, that performs encryption as well as authentication, is denoted as *aencrypt*. In a graph $G = (V, E)$: if there is an edge from u to v, we say v is a child of u, or u is a parent of v; for any node u, we denote the number of children of u by $\deg(u)$; the children of u from left to right are denoted $u_1, u_2, \cdots, u_{deg(u)}$; the $level[u]$ of node u is the length of path from *root* node to u; and the maximum-depth of the tree is the maximum value of $level[\,]$ among all the nodes of the tree. The node u^i_j means in the *chain* C_i the j-th node from root. We denote an empty set by $\emptyset$ and $[s] = \{1, 2, \cdots, s\}$.

2.2 Definitions

2.2.1 Posets, Chains and Access Graphs

Suppose the users in an organisation are grouped into a set of pairwise disjoint classes $V = \{u_1, u_2, \cdots, u_n\}$; in our case, the u_i's are various *security classes.* Suppose $u, v \in V$; let $v \leq u$ imply that u can access all the data which can be accessed by v (this forms the *hierarchical access rule* for the *security classes*). Therefore, $(V, \leq)$ is a *partially ordered set (poset)*, since '$\leq$' can be easily shown to be reflexive, anti-symmetric and transitive. We say: (1) $v < u$, if u and v are two distinct classes and $v \leq u$; (2) $v \lessdot u$, if $v < u$ and $\nexists c \in V$ such that $v < c < u$; (3) $(V, \leq)$ is a *totally ordered set* or a *chain* if $\forall u, v \in V$, either $v \leq u$ or $u \leq v$; and (4) $A \subseteq V$ is an *anti-chain* in V if for all $u, v \in A$ such that $u \neq v$, we have $v \nleq u$ and $u \nleq v$. The cardinality of the largest *anti-chain* in V is called the *width* of V, denoted w.

An *access graph* is a representation of a *poset* $(V, \leq)$ by a *directed acyclic graph* $G = (V, E)$, where the vertices represent the *security classes*, and, if $v \lessdot u$, then there is an edge from u to v. So, for all $u, v \in V$, where $v < u$, there is either a directed edge or a directed path from u to v. A *partition* of set V is a collection of sets $\{V_1, V_2, \cdots V_s\}$ such that: • $V_i \subseteq V, \forall i \in [s]$; • $V_1 \cup V_2 \cup \cdots \cup V_s = V$; and • $i \neq j \Rightarrow V_i \cap V_j = \emptyset, \forall i, j \in [s]$.

According to Dilworth's Theorem, every poset $(V, \leq)$ can be partitioned into w *chains*, where w is the *width* of V [Dil50]. The partition may not be unique. Let the set of *chains* $\{C_1, C_2, \cdots, C_w\}$ denote a partition of V, $l_i = |C_i|$ (for $i \in [w]$), and $l_{max} = \max_{i \in [w]} l_i$. The *maximum* node of C_i is denoted u^i_1 (i.e. $\forall v \in C_i, v \leq u^i_1$); and the *minimum* node of C_i is denoted $u^i_{l_i}$ (i.e. $\forall v \in C_i, u^i_{l_i} \leq v$). If $C_i = \{u^i_{l_i}, u^i_{l_i - 1}, \cdots, u^i_1\}$ and $u^i_{l_i} \lessdot u^i_{l_i - 1} \lessdot \cdots \lessdot u^i_1$, then $u^i_{l_i} \lessdot u^i_{l_i - 1} \lessdot \cdots \lessdot u^i_j$ is said to be a suffix of C_i, where $j \in [l_i]$. We say that v is a successor of u, if $v \leq u$, and v is an ancestor of u, if $u \leq v$. For all $u \in V$, the set of all ancestors (and successors) of u is denoted $\uparrow u := \{v \in V : u \leq v\}$ (and $\downarrow u := \{v \in V : v \leq u\}$). Note that $\downarrow u$ has a non-empty intersection with one or more *chains* $C_1, C_2, \cdots, C_w$, and, therefore, $\downarrow u \cap C_i$ is either a suffix of C_i or an empty set $\emptyset$. Since, $\{C_1, C_2, \cdots, C_w\}$ is a disjoint partition of V, $\{\downarrow u \cap C_1, \downarrow u \cap C_2, \cdots, \downarrow u \cap C_w\}$ is also a collection of pairwise disjoint sets. The *maximum* node of $\downarrow u \cap C_i$ is denoted $\hat{u}_i$. If $\downarrow u \cap C_i = \emptyset$, then $\hat{u}_i = \perp$.

2.2.2 Ideal Permutation

Let π/π^{-1}: $\{0,1\}^n \mapsto \{0,1\}^n$ be a pair of oracles. The pair π/π^{-1} is called an *ideal permutation* if the following three properties are satisfied.
1. $\pi^{-1}(\pi(x)) = x$ and $\pi(\pi^{-1}(x)) = x$, for all $x \in \{0,1\}^n$.
2. Suppose, x_k is the k-th query ($k \geq 1$), submitted to the oracle π, and $y \in \{0,1\}^n$. Then, for the current query x_i:

$$\Pr\left[\pi(x_i) = y \middle| \pi(x_1) = y_1, \pi(x_2) = y_2, \cdots, \pi(x_{i-1}) = y_{i-1}\right]$$

$$= \begin{cases} 1, & \text{if } x_i = x_j, y = y_j, j < i. \\ 0, & \text{if } x_i = x_j, y \neq y_j, j < i, \\ 0, & \text{if } x_i \neq x_j, y = y_j, j < i, \\ \frac{1}{2^n - i + 1}, & \text{if } x_i \neq x_j, y \neq y_j, j < i. \end{cases}$$

3. Suppose, y_k is the k-th query ($k \geq 1$), submitted to the oracle π^{-1}, and $x \in \{0,1\}^n$. Then, for the current query y_i:

$$\Pr\left[\pi^{-1}(y_i) = x \middle| \pi^{-1}(y_1) = x_1, \pi^{-1}(y_2) = x_2, \cdots, \pi^{-1}(y_{i-1}) = x_{i-1}\right]$$

$$= \begin{cases} 1, & \text{if } y_i = y_j, x = x_j, j < i. \\ 0, & \text{if } y_i = y_j, x \neq x_j, j < i, \\ 0, & \text{if } y_i \neq y_j, x = x_j, j < i, \\ \frac{1}{2^n - i + 1}, & \text{if } y_i \neq y_j, x \neq x_j, j < i. \end{cases}$$

2.2.3 Random Function

Let $\mathsf{rf}\colon \{0,1\}^n \mapsto \{0,1\}^n$. Then rf is called a *random function* if the following property is satisfied. Suppose, x_k is the k-th query ($k \geq 1$), submitted to the rf, and $y \in \{0,1\}^n$. Then, for the current query x_i:

$$\Pr\left[\mathsf{rf}(x_i) = y \middle| \mathsf{rf}(x_1) = y_1, \mathsf{rf}(x_2) = y_2, \cdots, \mathsf{rf}(x_{i-1}) = y_{i-1}\right]$$

$$= \begin{cases} 1, & \text{if } x_i = x_j, y = y_j, j < i. \\ 0, & \text{if } x_i = x_j, y \neq y_j, j < i, \\ \frac{1}{2^n}, & \text{if } x_i \neq x_j, j < i. \end{cases}$$

2.2.4 Source of message $\mathcal{S}$

We are modelling the security based on an unpredictable message source which is a PT algorithm, denoted $\mathcal{S}(\cdot)$, that returns $(\boldsymbol{M}, Z)$ or $(\boldsymbol{M}_0, \boldsymbol{M}_1, Z)$ on input 1^λ, where each *vector of messages* $\boldsymbol{M} \in \{0,1\}^{**}$ (or $\boldsymbol{M}_0, \boldsymbol{M}_1 \in \{0,1\}^{**}$) and *auxiliary information* $Z \in \{0,1\}^*$. We consider that $\mathcal{S}(\cdot)$ is a public source, that is, it is known to all the parties including the adversary. Here, each *vector of messages* $\boldsymbol{M}$ has $m(1^\lambda)$ number of strings, i.e., $\|\boldsymbol{M}\| = m(1^\lambda)$ and the length of each string $\boldsymbol{M}^{(i)}$ is $l(1^\lambda, i)$, i.e., $|\boldsymbol{M}^{(i)}| = l(1^\lambda, i)$ for $i \in [m(1^\lambda)]$. Here, m and l are two functions. We require that the two strings $\boldsymbol{M}^{(i_1)} \neq \boldsymbol{M}^{(i_2)}$, for $i_1 \neq i_2$ and $i_1, i_2 \in [m(1^\lambda)]$. Associated with the *source* $\mathcal{S}(\cdot)$ is a real number $GP_{\mathcal{S}}$, namely, the *Guessing Probability of source*, which is the maximum of all the probabilities of guessing a single string in $\boldsymbol{M}$, given the auxiliary information. The formal definition is $GP_{\mathcal{S}}(1^\lambda) \overset{def}{=} \max_{i \in [m(1^\lambda)]} GP(\boldsymbol{M}^{(i)} | Z)$. The source $\mathcal{S}(\cdot)$ is said to be unpredictable if the value of $GP_{\mathcal{S}}$ is negligible. We now define the *min-entropy* $\mu_{\mathcal{S}}(\cdot)$ of the source $\mathcal{S}(\cdot)$ as $\mu_{\mathcal{S}}(1^\lambda) = -\log(GP_{\mathcal{S}}(1^\lambda))$. The source $\mathcal{S}(\cdot)$ is said to be a valid source for an *MLE* scheme Π if $\boldsymbol{M}^{(i)} \in \mathcal{M}, \forall i \in [m(1^\lambda)]$.

2.2.5 Message-locked Encryption (MLE)

The definition of *message-locked encryption (MLE)* has already been described in [BKR13]. We briefly re-discuss it below, with a few suitable changes in the notation to suit the present context.

Syntax. Suppose $\lambda \in \mathbb{N}$ is the security parameter. An *MLE* scheme $\Pi = (\Pi.\mathcal{E}, \Pi.\mathcal{D})$ is a pair of algorithms over a setup algorithm $\Pi.\mathsf{Setup}$. Π satisfies the following conditions.

1. The PPT setup algorithm $\Pi.\mathsf{Setup}(1^\lambda)$ outputs the parameter $params^{(\Pi)}$ and the sets $\mathcal{K}^{(\Pi)}$, $\mathcal{M}^{(\Pi)}$, $\mathcal{C}^{(\Pi)}$ and $\mathcal{T}^{(\Pi)}$, denoting the *key*, *message*, *ciphertext* and *tag spaces* respectively.
2. The PPT encryption algorithm $\Pi.\mathcal{E}$ takes as inputs the parameter $params^{(\Pi)}$ and $M \in \mathcal{M}^{(\Pi)}$, and returns a three-tuple $(K, C, T) := \Pi.\mathcal{E}(params^{(\Pi)}, M)$, where $K \in \mathcal{K}^{(\Pi)}$, $C \in \mathcal{C}^{(\Pi)}$ and $T \in \mathcal{T}^{(\Pi)}$.
3. The decryption algorithm $\Pi.\mathcal{D}$ is a deterministic algorithm that takes as inputs the parameter $params^{(\Pi)}$, $K \in \mathcal{K}^{(\Pi)}$, $C \in \mathcal{C}^{(\Pi)}$ and $T \in \mathcal{T}^{(\Pi)}$, and returns $\Pi.\mathcal{D}(params^{(\Pi)}, K, C, T) \in \mathcal{M}^{(\Pi)} \cup \{\perp\}$. The decryption algorithm $\Pi.\mathcal{D}$ returns $\perp$ if the key K, ciphertext C and tag T are not generated from a valid message.
4. We restrict $|C|$ to be a linear function of $|M|$.

KEY CORRECTNESS. Let $M, M' \in \mathcal{M}^{(\Pi)}$. Suppose:
- $(K, C, T) := \Pi.\mathcal{E}(params^{(\Pi)}, M)$, and
- $(K', C', T') := \Pi.\mathcal{E}(params^{(\Pi)}, M')$.

Then *key correctness* of Π requires that if $M = M'$, then $K = K'$, for all $\lambda \in \mathbb{N}$ and all $M, M' \in \mathcal{M}^{(\Pi)}$.

DECRYPTION CORRECTNESS. Let $M \in \mathcal{M}^{(\Pi)}$. Suppose:
- $(K, C, T) := \Pi.\mathcal{E}(params^{(\Pi)}, M)$.

Then *decryption correctness* of Π requires that $\Pi.\mathcal{D}(params^{(\Pi)}, K, C, T) = M$, for all $\lambda \in \mathbb{N}$ and all $M \in \mathcal{M}^{(\Pi)}$.

TAG CORRECTNESS. Let $M, M' \in \mathcal{M}^{(\Pi)}$. Suppose:
- $(K, C, T) := \Pi.\mathcal{E}(params^{(\Pi)}, M)$, and
- $(K', C', T') := \Pi.\mathcal{E}(params^{(\Pi)}, M')$.

Then *tag correctness* of Π requires that if $M = M'$, then $T = T'$, for all $\lambda \in \mathbb{N}$ and all $M, M' \in \mathcal{M}^{(\Pi)}$.

For an *MLE* scheme, here, we define four security games PRV-CDA, STC, TC and KR-CDA. The game PRV-CDA is designed for the *privacy* security, STC and TC for the *tag consistency* security, and KR-CDA for the *key recovery* security in Figure 1. The first three games have already been described in [BKR13]; we define a new security notion of *key recovery* useful for our purpose. It is easy to show that an *MLE* scheme secure against PRV-CDA attack is also secure against KR-CDA attack. Below, we discuss the PRV-CDA, STC, TC and KR-CDA security games in detail.

```
Game $\mathsf{PRV\text{-}CDA}^{\mathcal{A}}_{\Pi,\mathcal{S}}(1^\lambda, b)$
$(\boldsymbol{M}_0, \boldsymbol{M}_1, Z) \xleftarrow{\$} \mathcal{S}(1^\lambda)$;
for $(i := 1, 2, \cdots m(1^\lambda))$
    $(\boldsymbol{K}^{(i)}, \boldsymbol{C}^{(i)}, \boldsymbol{T}^{(i)}) :=$
        $\Pi.\mathcal{E}(params^{(\Pi)}, \boldsymbol{M}_b^{(i)})$;
$b' := \mathcal{A}(1^\lambda, \boldsymbol{C}, \boldsymbol{T}, Z)$;
return $b'$;
```

```
Game $\mathsf{STC}^{\mathcal{A}}_{\Pi}(1^\lambda)$   $\boxed{\mathsf{TC}^{\mathcal{A}}_{\Pi}(1^\lambda)}$
$(M, C', T') := \mathcal{A}(1^\lambda)$;
If $(M = \perp) \vee (C' = \perp)$
    return 0;
$(K, C, T) := \Pi.\mathcal{E}(params^{(\Pi)}, M)$;
$M' := \Pi.\mathcal{D}(params^{(\Pi)}, K, C', T')$;
If $(T = T') \wedge (M \neq M') \boxed{\wedge (M' \neq \perp)}$
    return 1;
Else return 0;
```

```
Game $\mathsf{KR\text{-}CDA}^{\mathcal{A}}_{\Pi,\mathcal{S}}(1^\lambda)$
$(M, Z) \xleftarrow{\$} \mathcal{S}(1^\lambda)$;
$(K, C, T) :=$
    $\Pi.\mathcal{E}(params^{(\Pi)}, M)$;
$K' := \mathcal{A}(1^\lambda, C, T, Z)$;
return $K = K'$;
```

Figure 1: Games defining PRV-CDA, STC, TC and KR-CDA security of *MLE* scheme $\Pi = (\Pi.\mathcal{E}, \Pi.\mathcal{D})$.

PRIVACY. Let $\Pi = (\Pi.\mathcal{E}, \Pi.\mathcal{D})$ be an *MLE* scheme. Since, no *MLE* scheme can provide *privacy* security for predictable messages (even if the scheme is randomized), we use an unpredictable message source $\mathcal{S}$, as defined in Subsubsection 2.2.4, to design our security notion. For an *MLE* scheme, we design the *privacy against chosen distribution attack* PRV-CDA security game in Figure 1. Here, the challenger generates two vector of messages $\boldsymbol{M}_0 = (\boldsymbol{M}_0^{(1)} \circ \boldsymbol{M}_0^{(2)} \circ \cdots \circ \boldsymbol{M}_0^{(m(1^\lambda))})$ and $\boldsymbol{M}_1 = (\boldsymbol{M}_1^{(1)} \circ \boldsymbol{M}_1^{(2)} \circ \cdots \circ \boldsymbol{M}_1^{(m(1^\lambda))})$, and some auxiliary information Z using the source $\mathcal{S}(1^\lambda)$, encrypts the string $\boldsymbol{M}_b^{(i)}$, where $i \in [m(1^\lambda)]$ and the value of b depends upon the input, using $\Pi.\mathcal{E}$ to obtain $(\boldsymbol{K}^{(i)}, \boldsymbol{C}^{(i)}, \boldsymbol{T}^{(i)})$, and sends $(\boldsymbol{C}, \boldsymbol{T}, Z)$ to the adversary. The adversary has to return a bit b' indicating whether the ciphertext $\boldsymbol{C}$ and tag $\boldsymbol{T}$ corresponds to message $\boldsymbol{M}_0$ or message $\boldsymbol{M}_1$. If the values of b and b' coincide, then the adversary wins the game.

Now, we define the advantage of a PRV-CDA adversary $\mathcal{A}$ against Π as:

$$Adv_{\Pi,\mathcal{S},\mathcal{A}}^{\mathsf{PRV\text{-}CDA}}(1^\lambda) \overset{def}{=} \Big| \Pr[\mathsf{PRV\text{-}CDA}_{\Pi,\mathcal{S}}^{\mathcal{A}}(1^\lambda, b=1) = 1] - \Pr[\mathsf{PRV\text{-}CDA}_{\Pi,\mathcal{S}}^{\mathcal{A}}(1^\lambda, b=0) = 1] \Big|.$$

An *MLE* scheme Π is said to be PRV-CDA secure over a set of valid PT sources for *MLE* scheme Π, $\overline{\mathcal{S}} = \{\mathcal{S}_1, \mathcal{S}_2, \cdots\}$, for all PT adversaries $\mathcal{A}$ and for all $\mathcal{S}_i \in \overline{\mathcal{S}}$, if $Adv_{\Pi,\mathcal{S}_i,\mathcal{A}}^{\mathsf{PRV\text{-}CDA}}(\cdot)$ is negligible. An *MLE* scheme Π is said to be PRV-CDA secure, for all PT adversaries $\mathcal{A}$, if $Adv_{\Pi,\mathcal{S},\mathcal{A}}^{\mathsf{PRV\text{-}CDA}}(\cdot)$ is negligible, for all valid PT source $\mathcal{S}$ for Π.

Tag Consistency. Let $\Pi = (\Pi.\mathcal{E}, \Pi.\mathcal{D})$ be an *MLE* scheme. For an *MLE* scheme, we design the STC and TC security games in Figure 1, which aims to provide security against duplicate faking attacks. In a duplicate faking attack, two unidentical messages – one fake message produced by an adversary and a legitimate one produced by an honest client – produce the same tag, thereby causing loss of message and hampers the integrity. In an erasure attack, the adversary replaces the ciphertext with a fake message that decrypts successfully.

The adversary returns a message M, a ciphertext C' and a tag T'. If the message or ciphertext is invalid, the adversary loses the game. Otherwise, the challenger computes encryption key K, ciphertext C and tag T corresponding to message M using $\Pi.\mathcal{E}$, and computes the message M' corresponding to key K, ciphertext C' and tag T' using $\Pi.\mathcal{D}$. If the two tags are equal, i.e. $T = T'$, the message M' is valid, i.e. $M' \neq \perp$, and the two messages are unequal, i.e. $M \neq M'$, then the adversary wins the TC game.

Now, we define the advantage of a TC adversary $\mathcal{A}$ against Π as:

$$Adv_{\Pi,\mathcal{A}}^{\mathsf{TC}}(1^\lambda) \overset{def}{=} \Pr[\mathsf{TC}_{\Pi}^{\mathcal{A}}(1^\lambda) = 1].$$

Now, we define the advantage of an STC adversary $\mathcal{A}$ against Π as:

$$Adv_{\Pi,\mathcal{A}}^{\mathsf{STC}}(1^\lambda) \overset{def}{=} \Pr[\mathsf{STC}_{\Pi}^{\mathcal{A}}(1^\lambda) = 1].$$

An *MLE* scheme Π is said to be TC (or STC) secure, for all PT adversaries $\mathcal{A}$, if $Adv_{\Pi,\mathcal{A}}^{\mathsf{TC}}(\cdot)$ (or $Adv_{\Pi,\mathcal{A}}^{\mathsf{STC}}(\cdot)$) is negligible.

Key Recovery. Let $\Pi = (\Pi.\mathcal{E}, \Pi.\mathcal{D})$ be an *MLE* scheme. Since, no *MLE* scheme can provide *key recovery* security (even if it is randomized) for predictable messages, we use an unpredictable message source $\mathcal{S}$, as defined in Subsubsection 2.2.4, to design our *key recovery against chosen distribution attack* KR-CDA security game in Figure 1. Here, the challenger generates a message M and some auxiliary information Z using the source $\mathcal{S}(1^\lambda)$, encrypts M using $\Pi.\mathcal{E}(params^{(\Pi)}, \cdot)$ and sends (C, T, Z) to the adversary. The adversary has to return a key K' corresponding to ciphertext C and tag T. If the keys K and K' match, then the adversary wins the game.

Now, we define the advantage of a KR-CDA adversary $\mathcal{A}$ against Π as:

$$Adv^{\mathsf{KR\text{-}CDA}}_{\Pi,\mathcal{S},\mathcal{A}}(1^\lambda) \overset{def}{=} \Pr[\mathsf{KR\text{-}CDA}^{\mathcal{A}}_{\Pi,\mathcal{S}}(1^\lambda) = 1].$$

An *MLE* scheme Π is said to be KR-CDA secure, if $Adv^{\mathsf{KR\text{-}CDA}}_{\Pi,\mathcal{S},\mathcal{A}}(\cdot)$ is negligible, for all valid PT source $\mathcal{S}$ and all PT adversaries $\mathcal{A}$.

2.2.6 Authenticated Encryption (AE)

Syntax. Suppose $\lambda \in \mathbb{N}$ is the security parameter. An *authenticated encryption (AE)* scheme $\Pi = (\Pi.\mathcal{K}_{\text{GEN}}, \Pi.\mathcal{E}, \Pi.\mathcal{D})$ is a three-tuple of algorithms over a setup algorithm $\Pi.\mathsf{Setup}$. Π satisfies the following conditions.

1. The PPT setup algorithm $\Pi.\mathsf{Setup}(1^\lambda)$ outputs the parameter $params^{(\Pi)}$ and the sets $\mathcal{K}^{(\Pi)}$, $\mathcal{M}^{(\Pi)}$, $\mathcal{C}^{(\Pi)}$ and $\mathcal{T}^{(\Pi)}$, denoting the *key*, *message*, *ciphertext* and *tag spaces* respectively.

2. The PPT key-generation algorithm $\Pi.\mathcal{K}_{\text{GEN}}\colon \mathbb{N} \to \mathcal{K}^{(\Pi)}$ takes as input the parameter $params^{(\Pi)}$, and outputs $K := \Pi.\mathcal{K}_{\text{GEN}}(params^{(\Pi)})$, where $K \in \mathcal{K}^{(\Pi)}$.

3. The PPT encryption algorithm $\Pi.\mathcal{E}\colon \mathcal{K}^{(\Pi)} \times \mathcal{M}^{(\Pi)} \to \mathcal{C}^{(\Pi)} \times \mathcal{T}^{(\Pi)}$ takes as inputs the parameter $params^{(\Pi)}$, $K \in \mathcal{K}^{(\Pi)}$ and $M \in \mathcal{M}^{(\Pi)}$, and outputs a pair $(C, T) := \Pi.\mathcal{E}(params^{(\Pi)}, K, M)$, where $C \in \mathcal{C}^{(\Pi)}$ and $T \in \mathcal{T}^{(\Pi)}$. It is possible that the tag is incorporated in the ciphertext itself, in this case, T is an empty string.

4. The decryption algorithm $\Pi.\mathcal{D}\colon \mathcal{K}^{(\Pi)} \times \mathcal{C}^{(\Pi)} \times \mathcal{T}^{(\Pi)} \to \mathcal{M}^{(\Pi)} \cup \{\bot\}$ is a deterministic algorithm that takes as inputs the parameter $params^{(\Pi)}$, $K \in \mathcal{K}^{(\Pi)}$, $C \in \mathcal{C}^{(\Pi)}$ and $T \in \mathcal{T}^{(\Pi)}$, and returns $\Pi.\mathcal{D}(params^{(\Pi)}, K, C, T) \in \mathcal{M}^{(\Pi)} \cup \{\bot\}$. The decryption algorithm $\Pi.\mathcal{D}$ returns $\bot$ if the ciphertext C and tag T are not generated using the key K.

Here, we make a note that, when the tag is incorporated in the ciphertext itself, we observe an obvious and intuitive expansion of the ciphertext, therefore, we restrict $|C|$ to be a linear function of $|M|$.

Decryption Correctness. Let $M \in \mathcal{M}^{(\Pi)}$. Suppose:
- $K := \Pi.\mathcal{K}_{\text{GEN}}(params^{(\Pi)})$, and
- $(C, T) := \Pi.\mathcal{E}(params^{(\Pi)}, K, M)$.

Then *decryption correctness* of Π requires that $\Pi.\mathcal{D}(params^{(\Pi)}, K, C, T) = M$, for all $\lambda \in \mathbb{N}$, all $M \in \mathcal{M}^{(\Pi)}$ and all $K \in \mathcal{K}^{(\Pi)}$.

For an *AE* scheme, here, we define two security games, namely, IND-PRV and INT for the *privacy* and *tag consistency* security in Figure 2. Below, we discuss the IND-PRV and INT security games in detail.

Privacy. Let $\Pi = (\Pi.\mathcal{K}_{\text{GEN}}, \Pi.\mathcal{E}, \Pi.\mathcal{D})$ be an *AE* scheme. For an *AE* scheme, we design the *indistinguishability privacy* IND-PRV security game in Figure 2. Here, the challenger generates the encryption key using $\Pi.\mathcal{K}_{\text{GEN}}(params^{(\Pi)})$ and receives two messages M_0 and M_1 from the adversary, such that $|M_0| = |M_1|$. The challenger encrypts M_0 or M_1 according to the value of b, the input parameter, to obtain (C, T) and sends (C, T) to the adversary. The adversary has to return a bit b' indicating whether the ciphertext C corresponds to message 0 or message 1. If the values of b and b' coincide, then the adversary wins the game.

```
Game $\mathsf{IND\text{-}PRV}_{\Pi}^{\mathcal{A}}(1^\lambda, b)$
$K := \Pi.\mathcal{K}_{\text{GEN}}(params^{(\Pi)})$;
$(M_0, M_1) := \mathcal{A}_1(1^\lambda)$;
**If** $(|M_0| \neq |M_1|)$, **then** return **Error**;
$(C, T) := \Pi.\mathcal{E}(params^{(\Pi)}, K, M_b)$;
$b' := \mathcal{A}_2(1^\lambda, C, T, M_0, M_1)$;
return $b'$;
```

```
Game $\mathsf{INT}_{\Pi}^{\mathcal{A}}(1^\lambda, \sigma)$
$K \xleftarrow{\$} \Pi.\mathcal{K}_{\text{GEN}}(params^{(\Pi)})$;
$(C_0, C_1, T) := \mathcal{A}^{\Pi.\mathcal{E}(params^{(\Pi)}, K, \cdot)}(1^\lambda, \sigma)$;
**If** $C_0 = C_1$, **then** return 0;
$M_0 := \Pi.\mathcal{D}(params^{(\Pi)}, K, C_0, T)$;
$M_1 := \Pi.\mathcal{D}(params^{(\Pi)}, K, C_1, T)$;
**If** $(M_0 \neq \perp) \wedge (M_1 \neq \perp)$, **then** return 1;
**Else** return 0;
```

Figure 2: Games defining IND-PRV and INT security of AE scheme $\Pi = (\Pi.\mathcal{K}_{\text{GEN}}, \Pi.\mathcal{E}, \Pi.\mathcal{D})$. Here, in IND-PRV game, the adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$.

Now, we define the advantage of an IND-PRV adversary $\mathcal{A}$ against Π as:

$$Adv_{\Pi,\mathcal{A}}^{\mathsf{IND\text{-}PRV}}(1^\lambda) \overset{def}{=} \left| \Pr[\mathsf{IND\text{-}PRV}_{\Pi}^{\mathcal{A}}(1^\lambda, b = 1) = 1] - \Pr[\mathsf{IND\text{-}PRV}_{\Pi}^{\mathcal{A}}(1^\lambda, b = 0) = 1] \right|.$$

An AE scheme Π is said to be IND-PRV secure, for all PT adversaries $\mathcal{A}$, if $Adv_{\Pi,\mathcal{A}}^{\mathsf{IND\text{-}PRV}}(\cdot)$ is negligible.

Tag Consistency. Let $\Pi = (\Pi.\mathcal{K}_{\text{GEN}}, \Pi.\mathcal{E}, \Pi.\mathcal{D})$ be an AE scheme. For an AE scheme, we design the *integrity* INT security game in Figure 2. Here, the challenger generates the encryption key using $\Pi.\mathcal{K}_{\text{GEN}}(params^{(\Pi)})$ and receives two ciphertexts C_0 and C_1, and one tag T from the adversary. The challenger declares the defeat of adversary if the two ciphertexts are identical, otherwise, the challenger decrypts (C_0, T) and (C_1, T) using $\Pi.\mathcal{D}(params^{(\Pi)}, K, \cdot, \cdot)$. The adversary wins if both the messages are *valid*, i.e., $M_0 \neq \perp$ and $M_1 \neq \perp$.

Now, we define the advantage of an INT adversary $\mathcal{A}$ against Π as:

$$Adv_{\Pi,\mathcal{A}}^{\mathsf{INT}}(1^\lambda) \overset{def}{=} \Pr[\mathsf{INT}_{\Pi}^{\mathcal{A}}(1^\lambda, \sigma) = 1].$$

An AE scheme Π is said to be INT secure, for all PT adversaries $\mathcal{A}$, if $Adv_{\Pi,\mathcal{A}}^{\mathsf{INT}}(\cdot)$ is negligible.

2.2.7 Key Assignment Scheme (KAS)

The definition of *key assignment scheme (KAS)* has already been described in [FPP13]. We briefly re-discuss it below, with a few suitable changes in the notation to suit the present context.

Syntax. Suppose $\lambda \in \mathbb{N}$ is the security parameter. A KAS scheme $\Pi = (\Pi.\mathcal{GEN}, \Pi.\mathcal{DER})$ is a pair of algorithms over a setup algorithm $\Pi.\mathsf{Setup}$. Π satisfies the following conditions.

1. The PPT setup algorithm $\Pi.\mathsf{Setup}(1^\lambda)$ outputs the parameter $params^{(\Pi)}$, a set of access graphs $\Gamma^{(\Pi)}$ and the set $\mathcal{K}^{(\Pi)}$ denoting the *key space*.

2. The PPT key generation algorithm $\Pi.\mathcal{GEN}$ takes as inputs the parameter $params^{(\Pi)}$ and graph G, and returns a three-tuple $(S, k, pub) := \Pi.\mathcal{GEN}(params^{(\Pi)}, G)$, where $S = (S_u)_{u \in V}$ and $k = (k_u)_{u \in V}$. The variables S, k and pub are called private information, key and public information vectors, respectively.

   Note that $S_u \in \{0,1\}^*$, $k_u \in \mathcal{K}^{(\Pi)}$ and $pub \in \{0,1\}^*$, for all $u \in V$ and all $G \in \Gamma^{(\Pi)}$.

3. The key derivation algorithm $\Pi.\mathcal{DER}$ is a deterministic PT algorithm such that $k_v := \Pi.\mathcal{DER}(params^{(\Pi)}, G, u, v, S_u, pub)$, where $v \leq u$ are two nodes of the access

graph G, S_u is u's private information, pub is the public information, and k_v is v's decryption key.

Note that $S_u \in \{0,1\}^*$, $pub \in \{0,1\}^*$ and $k_v \in \mathcal{K}^{(\Pi)} \cup \{\perp\}$.

Correctness. The *correctness* of Π requires that for all $\lambda \in \mathbb{N}$, all $G \in \Gamma^{(\Pi)}$, all (S, k, pub) output by $\Pi.\,\mathcal{GEN}(params^{(\Pi)}, G)$, and all nodes $v \le u$, we have:

$$\Pi.\,\mathcal{DER}(params^{(\Pi)}, G, u, v, S_u, pub) = k_v.$$

For a *KAS* scheme, here, we define three security games KI-ST, S-KI-ST and KR-ST. The games KI-ST and S-KI-ST are designed for the *key indistinguishability* security, and KR-ST for the *key recovery* security in Figure 3. These notions have already been described in [ABFF09, ASFM06, SFM07a, FPP13, DSFM10].

```
Game $\mathsf{KI\text{-}ST}^{\mathcal{A}}_{\Pi}(1^\lambda, G, b)$
$u := \mathcal{A}_1(1^\lambda, G)$;
$(S, k, pub) :=$
    $\Pi.\,\mathcal{GEN}(params^{(\Pi)}, G)$;
$P_u := \{S_v \in S | v < u\}$;
If $b = 1$, then $T := k_u$;
Else $T \xleftarrow{\$} \{0,1\}^{|k_u|}$;
$b' := \mathcal{A}_2(1^\lambda, G, pub, P_u, T)$;
return $b'$;
```

```
Game $\mathsf{S\text{-}KI\text{-}ST}^{\mathcal{A}}_{\Pi}(1^\lambda, G, b)$
$u := \mathcal{A}_1(1^\lambda, G)$;
$(S, k, pub) :=$
    $\Pi.\,\mathcal{GEN}(params^{(\Pi)}, G)$;
$P_u := \{S_v \in S | v < u\}$;
$K_u := \{k_v \in k | u < v\}$;
If $b = 1$ then $T := k_u$;
Else $T \xleftarrow{\$} \{0,1\}^{|k_u|}$;
$b' := \mathcal{A}_2(1^\lambda, G, pub, P_u, K_u, T)$;
return $b'$;
```

```
Game $\mathsf{KR\text{-}ST}^{\mathcal{A}}_{\Pi}(1^\lambda, G)$
$u := \mathcal{A}_1(1^\lambda, G)$;
$(S, k, pub) :=$
    $\Pi.\,\mathcal{GEN}(params^{(\Pi)}, G)$;
$P_u := \{S_v \in S | v < u\}$;
$k'_u := \mathcal{A}_2(1^\lambda, G, pub, P_u)$;
return $k_u = k'_u$;
```

Figure 3: Games defining KI-ST, S-KI-ST and KR-ST security of *KAS* scheme $\Pi = (\Pi.\,\mathcal{GEN}, \Pi.\,\mathcal{DER})$. Here, the adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$.

Key Indistinguishability. Let $\Gamma^{(\Pi)}$ be a set of access graphs and $\Pi = (\Pi.\,\mathcal{GEN}, \Pi.\,\mathcal{DER})$ be the *KAS* for $\Gamma^{(\Pi)}$. For a *KAS* we have designed a *key indistinguishability with respect to static adversary* KI-ST (and *strong key indistinguishability with respect to static adversary* S-KI-ST) security game in Figure 3. The static adversary $\mathcal{A}$, when given access to the graph $G = (V, E)$, returns a security class $u \in V$ to the challenger, that $\mathcal{A}$ chooses to attack. The challenger then performs the following operations: calculates (S, k, pub) using the $\Pi.\,\mathcal{GEN}(params^{(\Pi)}, G)$; computes P_u as the set of private information S_v for the classes $v \in V$ such that $v < u$; (computes K_u as the set of keys k_v for the classes $v \in V$ such that $u < v$;) if the value of b is 1, then the value T is the value of k_u, otherwise, the value of T is chosen to be a random string of same length as k_u; and sends (pub, P_u, T) (and K_u) to the adversary. The adversary has to return a bit b' indicating whether T corresponds to key or is it a random string. If the values of b and b' coincide, then the adversary wins the game.

Now, we define the advantage of a KI-ST adversary $\mathcal{A}$ against Π on a graph $G \in \Gamma^{(\Pi)}$ as:

$$Adv^{\mathsf{KI\text{-}ST}}_{\Pi,\mathcal{A},G}(1^\lambda) \overset{def}{=} \left| \Pr[\mathsf{KI\text{-}ST}^{\mathcal{A}}_{\Pi}(1^\lambda, G, b = 1) = 1] - \Pr[\mathsf{KI\text{-}ST}^{\mathcal{A}}_{\Pi}(1^\lambda, G, b = 0) = 1] \right|.$$

Now, we define the advantage of an S-KI-ST adversary $\mathcal{A}$ against Π on a graph $G \in \Gamma^{(\Pi)}$ as:

$$Adv^{\mathsf{S\text{-}KI\text{-}ST}}_{\Pi,\mathcal{A},G}(1^\lambda) \overset{def}{=} \left| \Pr[\mathsf{S\text{-}KI\text{-}ST}^{\mathcal{A}}_{\Pi}(1^\lambda, G, b = 1) = 1] - \Pr[\mathsf{S\text{-}KI\text{-}ST}^{\mathcal{A}}_{\Pi}(1^\lambda, G, b = 0) = 1] \right|.$$

A *KAS* scheme Π is said to be $\mathsf{KI\text{-}ST}$ (or $\mathsf{S\text{-}KI\text{-}ST}$) secure, for all PT static adversaries $\mathcal{A}$, if $Adv^{\mathsf{KI\text{-}ST}}_{\Pi,\mathcal{A},G}(\cdot)$ (or $Adv^{\mathsf{S\text{-}KI\text{-}ST}}_{\Pi,\mathcal{A},G}(\cdot)$) is negligible.

KEY RECOVERY. Let $\Gamma^{(\Pi)}$ be a set of access graphs and $\Pi = (\Pi.\mathcal{GEN}, \Pi.\mathcal{DER})$ be the *KAS* for $\Gamma^{(\Pi)}$. For a *KAS* scheme we have designed a *key recovery with respect to static adversary* KR-ST security game in Figure 3. The static adversary $\mathcal{A}$, when given access to the graph $G = (V, E)$, returns a security class $u \in V$ to the challenger, that $\mathcal{A}$ chooses to attack. The challenger then performs the following operations: calculates (S, k, pub) using the $\Pi.\mathcal{GEN}(params^{(\Pi)}, G)$; computes P_u as the set of private information S_v for the classes $v \in V$ such that $v < u$; and sends (pub, P_u) to the adversary. The adversary has to return a key k'_u. If the values of k_u and k'_u coincide, then the adversary wins the game.

Now, we define the advantage of a KR-ST adversary $\mathcal{A}$ against Π on a graph $G \in \Gamma^{(\Pi)}$ as:

$$Adv^{\mathsf{KR\text{-}ST}}_{\Pi,\mathcal{A},G}(1^\lambda) \overset{def}{=} \Pr[\mathsf{KR\text{-}ST}^{\mathcal{A}}_{\Pi}(1^\lambda, G) = 1].$$

A *KAS* scheme Π is said to be KR-ST secure, for all PT static adversaries $\mathcal{A}$, if $Adv^{\mathsf{KR\text{-}ST}}_{\Pi,\mathcal{A},G}(\cdot)$ is negligible.

Remark. Note that a *KAS-chain* is a special type of *KAS* where the access graph is a *totally ordered set*.

2.2.8 Graph Algorithms used in the Paper

In this paper, we frequently use some graph-based algorithms that we describe below. Their algorithmic description is given in Figure 4. In the access graph $G = (V, E) \in \Gamma^{(\Pi)}$ for the poset $(V, \leq)$, we represent the *security classes* by nodes $u_1, u_2, \cdots, u_n \in V$, where $n = |V|$.

all_succ(u, G)**:** Given the node u and graph G as input, this outputs the set of all successor nodes $\downarrow u = \{v \in V | v \leq u\}$. This can be implemented by using *Breadth First Search (BFS)* (or *Depth First Search (DFS)*) traversal on the graph G with u as the *source/root* node. The running time of all_succ(u, G) is $\mathcal{O}(|V| + |E|)$.

ch_seq(u, G)**:** Given the node u and graph G as input, this function outputs a sequence of nodes $\tilde{u} = (u_{j_1}, u_{j_2}, \cdots, u_{j_d})$ – that are children of node u in G – in the ascending order of their indices. Therefore, u_{j_1} has the lowest index, u_{j_2} the second lowest, and so on. We say that $\tilde{u}$ is NULL, if u is a leaf node. The algorithm works in the following way: the children of node u are extracted from the set of edges E; a sorting algorithm is run on this set; and, finally, the sorted sequence is returned. The running time of ch_seq(u, G) is $\mathcal{O}(|V| + \mathsf{deg}(u) \log(\mathsf{deg}(u)))$.

ext_cipher(pub, u)**:** Given the public information pub and node u as input, this outputs the extracted ciphertext c_u corresponding to u from pub.

ext_secret(S_u, v)**:** Given the private information S_u of node u and a node $v \leq u$ as input, this function outputs the extracted secret value corresponding to v from S_u.

ext_tag(S_u, v)**:** Given the private information S_u of node u and a node $v \leq u$ as input, this function outputs the extracted tag t_v corresponding to v from S_u.

height(G)**:** Given a (directed acyclic) graph G as input, this function first assigns to $level[u]$ the maximum level of node u for all $u \in V$, and then returns $level[\,]$ and $h = \max_{u \in V} level[u]$. We, first, find the root node u of the graph and assign the $level[u] = 1$. Note that there is exactly one root in a connected DAG. Now, we execute *BFS* traversal on the graph G with u as the *root* node, with a slight modification that whenever we encounter a previously discovered node, we update its $level[\,]$ value with the current value. Since, the graph is acyclic, the value of $level[v]$, for all $v \in V$, can be at most n. We calculate the height of the graph $h = \max_{v \in V} level[v]$. The running time of $\mathsf{height}(G)$ is $\mathcal{O}(|V|^2 + |V| + |E|)$.

max_isect(u, C)**:** Given a node u and a *chain* C as input, this function outputs the *maximum element* of $\downarrow u \cap C$. This can be implemented by first calculating the set $\downarrow u$ using $\mathsf{all\_succ}(u, G)$ function (as defined above), and then performing the set intersection between $\downarrow u$ and C, and finally finding the *maximum* element in the resulting set. The running time of $\mathsf{max\_isect}(u, C)$ is $\mathcal{O}(|V| + |E|)$.

max_isect_chs(u, G)**:** Given a node u and the graph G as input, this outputs a sequence of nodes $(\hat{u}_1, \hat{u}_2, \cdots, \hat{u}_w)$ who are the *maximum elements* of $\downarrow u \cap C_1$, $\downarrow u \cap C_2$, $\cdots$, $\downarrow u \cap C_w$. This can be implemented in the same way as $\mathsf{max\_isect}(u, C)$ with different *chains* in different iterations and some trivial running time optimization. The running time of $\mathsf{max\_isect\_chs}(u, G)$ is $\mathcal{O}(|V| + |E|)$.

nodes_at_level$(V, level[\,], x)$**:** This function takes a graph G, the array $level[\,]$ storing the levels of nodes, and a level x as input, and outputs the set of nodes in G that are at level x. We have already assigned the values of levels of the nodes to the array $level[\,]$, during the execution of $\mathsf{height}(G)$ function. Now, we need to compare the levels of all the nodes, and build the set of those elements whose levels are x. Finally, we return this set. The running time of $\mathsf{nodes\_at\_level}(V, level[\,], x)$ is $\mathcal{O}(|V|)$.

partition(G)**:** This function takes as input a graph G, and outputs the number of partitions w and the set of *chains* $C_1, C_2, \cdots, C_w$ (as used by Freire *et al.* [FPP13]). The running time of $\mathsf{partition}(G)$ is $poly(n)$.

path(G, u, v)**:** This function takes as input a graph G, the source and the destination nodes u and v, and outputs a sequence of nodes $(u, u_{i_1}, u_{i_2}, \cdots, u_{i_\ell}, v)$ such that $u_{i_1} \lessdot u, u_{i_2} \lessdot u_{i_1}, \cdots, v \lessdot u_{i_\ell}$. In order to do this, we invoke the *Dijkstra's Algorithm* on graph G with u as source node, and get the array $dist[\,]$, defining the distance of any node from u, and array $parent[\,]$, defining the parent of any node in the graph [Dij59]. Then, we start to find the parent of v as u_{i_ℓ}, then the parent of u_{i_ℓ} as $u_{i_{\ell-1}}$, and so on, until we find the parent of u_{i_1} as u. So, the path from u to v is $(u, u_{i_1}, u_{i_2}, \cdots, u_{i_\ell}, u_{i_{\ell+1}} = v)$. The running time of $\mathsf{path}(G, u, v)$ is $\mathcal{O}(|E| + |V| \cdot \log |V| + |V|)$.

vertex_in_order(G)**:** This function takes as input the access graph $G = (V, E)$ corresponding to a *totally ordered set*, and outputs a sequence of nodes $(u_1, u_2, \cdots, u_n)$ such that $u_n \lessdot u_{n-1} \lessdot \cdots \lessdot u_1$, where $n = |V|$. We, first, find the root node u_1 of the graph. Since, in a totally ordered set there is only one child of each node, we find the edges (u_1, u_2), then (u_2, u_3), and so on up to (u_{n-1}, u_n), and compute the sequence of nodes $(u_1, u_2, \cdots, u_n)$. The running time of $\mathsf{vertex\_in\_order}(G)$ is $\mathcal{O}(|V|^2 + |E|)$.

```
all_succ($(u, G)$)
$U := \{u\}$, $Q := \emptyset$;
ENQUEUE($Q, u$);
while $Q \neq \emptyset$
  $node$ := DEQUEUE($Q$);
  for all $(node, v) \in E$
    $U := U \cup \{v\}$;
    ENQUEUE($Q, v$);
return $U$;
```

```
nodes_at_level($(V, level[\,], x)$)
$U := \emptyset$;
For all $u \in V$
  If $level[u] = x$
    $U := U \cup \{u\}$;
return $U$;
```

```
path($(G, u, v)$)
$(dist[\,], parent[\,])$
        := DIJKSTRA($G, u$);
$path := (v)$, $w := v$;
while $w \neq u$
  $path := (parent[w] \circ path)$;
  $w := parent[w]$;
return $path$;
```

```
max_isect($(u, C)$)
$\downarrow u$ := all_succ($u, G$);
$U \overset{def}{=} (u_{i_1}, u_{i_2}, \cdots, u_{i_\ell})$
      $:= \downarrow u \cap C$;
$max := 1$;
for $(j := 2, 3, \cdots, \ell)$
  If $u_{i_{max}} \leq u_{i_j}$
    $max := j$;
return $u_{i_{max}}$;
```

```
ch_seq($(u, G)$)
$U := \emptyset$;
for all $(u, v) \in E$
  $U := U \cup \{v\}$;
$U$ := SORT($U$);
return $U$;
```

```
vertex_in_order($(G)$)
$u_1$ := FIND_ROOT($G$);
for $(i := 1, 2, \cdots, m-1)$
  Find $u_{i+1} \in V$ such that
        $(u_i, u_{i+1}) \in E$;
return $(u_1, u_2, \cdots, u_m)$;
```

```
max_isect_chs($(u, G)$)
$\downarrow u$ := all_succ($u, G$);
$W := \emptyset$;
for $(k := 1, 2, \cdots, w)$
  $U \overset{def}{=} (u_{i_1}, u_{i_2}, \cdots, u_{i_\ell})$
        $:= \downarrow u \cap C_k$;
  $max := 1$;
  for $(j := 2, 3, \cdots, \ell)$
    If $u_{i_{max}} \leq u_{i_j}$
      $max := j$;
  $W := (W \circ u_{i_{max}})$;
return $W$;
```

```
height($(G)$)
$u$ := FIND_ROOT($G$);
$level[u] := 1$, $h := 1$, $Q := \emptyset$;
ENQUEUE($Q, u$);
while $Q \neq \emptyset$
  $u$ := DEQUEUE($Q$);
  for all $(u, v) \in E$
    $level[v] := level[u] + 1$;
    If $h < level[v]$
      $h := level[v]$;
    ENQUEUE($Q, v$);
return $(level[\,], h)$;
```

Figure 4: Graph algorithms used in the paper. Here: ENQUEUE(Q, u) operation appends the element u in the *queue* data structure Q; DEQUEUE(Q) operation removes the first element from the *queue* Q, and returns the element; FIND_ROOT(G) function takes the graph G, and finds its root node (this node has no incoming edges); SORT(U) operation takes a list of elements, and returns a sorted list of elements based on their index values; and DIJKSTRA(G, u) is the *Single-Source Shortest Path* algorithm that takes the Graph G and source u as input, and gives the lengths of shortest paths (as $dist[\,]$) from u to all the nodes, and the parents of all nodes (as $parent[\,]$) [Dij59].

2.3 Existing Constructions of *AE*, *MLE* and *KAS*

2.3.1 Existing *AE* schemes

We refer the reader to [AAB15, ABB+14, ACS15, BDPA09, BDP+14, BRW04, Rog02] to know about the various existing *AE* constructions in detail.

2.3.2 Existing *MLE* schemes

We refer the reader to [ABM+13, BKR13, CMYG15, DAB+02] to know about the various existing *MLE* constructions in detail.

2.3.3 Existing *KAS* schemes

Since, our work mainly focuses on *KAS-AE*, we briefly revisit various *KAS* schemes below. *KAS* is usually built in following two ways:

1. Constructing *KAS* from scratch: We refer the reader to [ABFF09, AFB05, AT83, CC02, CH05, CHW92, Gud80, HL90, SC02, SFM07a, TC95, YL04, ZRM01] to know about the various existing *KAS* constructions in detail. Crampton, Martin and Wild have classified the *KAS* constructions into five generic schemes [CMW06]. These schemes differ in: (1) the way encryption key k_u (for file f_u) corresponding to node $u \in V$ is selected; (2) the method for generation and distribution of the secret and public information $S = (S_u)_{u \in V}$ and *pub* respectively; and (3) the working of key derivation algorithm where the node u recomputes the key corresponding to the node $v \leq u$. These schemes are as follows:

Scheme 1: TKAS. A *trivial key assignment scheme (TKAS)* has the following properties: • All k_u's are chosen independently; • $S_u := (k_v)_{v \leq u}$; • $pub := \emptyset$; and • $k_v \in S_u \; \forall v \leq u$, so deriving the key k_v is trivial.

Scheme 2: TKEKAS. A *trivial key-encrypting-key assignment scheme (TKEKAS)* has the following properties: • For all $u \in V$, k_u's and K_u's are chosen independently, where K_u is a key used to encrypt k_u; • $S_u := (K_v)_{v \leq u}$; • $pub := (\mathcal{E}_{K_u}(k_u))_{u \in V}$; and • k_v is obtained by decrypting $\mathcal{E}_{K_v}(k_v) \in pub$ using $K_v \in S_u$.

Scheme 3: DKEKAS. A *direct key-encrypting-key assignment scheme (DKEKAS)* has the following properties: • All k_u's are chosen independently; • $S_u := k_u$; • $pub := (\mathcal{E}_{k_u}(k_v))_{v < u, u \in V}$; and • k_v is obtained by decrypting $\mathcal{E}_{k_u}(k_v) \in pub$ using $k_u \in S_u$.

Scheme 4: IKEKAS. An *iterative key-encrypting-key assignment scheme (TKEKAS)* has the following properties: • All k_u's are chosen independently; • $S_u := k_u$; • $pub := (\mathcal{E}_{k_u}(k_v))_{v \lessdot u, u \in V}$; and • there exists a path $(u, z_0), (z_0, z_1) \cdots (z_m, v)$ and we calculate $k_{z_0} := \mathcal{D}_{k_u}(\mathcal{E}_{k_u}(k_{z_0})), k_{z_1} := \mathcal{D}_{k_{z_0}}(\mathcal{E}_{k_{z_0}}(k_{z_1})), \cdots, k_v := \mathcal{D}_{k_{z_m}}(\mathcal{E}_{k_{z_m}}(k_v))$, to obtain k_v.

Scheme 5: NBKAS. A *node-based key assignment scheme (NBKAS)* has the following properties: • $k_u := f(e_u)$ are keys such that $g(f(e_u), e_u, e_v) = k_v$; • $S_u := k_u$; • $pub := (e_u)_{u \in V}$; and • $k_v := g(k_u, e_u, e_v)$ can be calculated using $e_u, e_v \in pub$ and $k_u \in S_u$.

2. Constructing *KAS* from *KAS-chain*: This paradigm has two phases: (1) building *KAS-chain* from scratch, and (2) combining *KAS-chains* to build *KAS* using *chain partition* algorithm. We refer the reader to [CDM10, FP11, FPP13] to know about the various existing *KAS* constructions build from *KAS-chain* in detail.

(1) BUILDING *KAS-chain* FROM SCRATCH: Crampton *et al.* described two *KAS-chain*, one based on *iterated hashing* and the other based on *RSA* [CDM10]. Freire and Paterson also gave a *KAS-chain* based on Factoring problem in [FP11]. Freire *et al.* described two *KAS-chain* schemes, one based on *pseudorandom functions* and the other based on *forward-secure pseudorandom generators* [FPP13].

(2) CHAIN PARTITION: This paradigm builds a *KAS* from *KAS-chains* for an arbitrary access graph. Crampton, Daud and Martin have discussed procedures for designing efficient *KAS* schemes, from *KAS-chains*, using an innovative *chain partition* algorithm in [CDM10]. The main idea behind their *chain partition* algorithm is the following: partition the access graph into *disjoint chains*, and design *KAS-chains* corresponding to these *chains*; finally, securely join these *KAS-chains* to form the *KAS* for the full access graph. The detailed description of *chain partition* algorithm is given below:

Let $(V, \leq)$ be a poset represented by the access graph $G = (V, E)$; suppose the set of *chains* $\{C_1, C_2, \cdots, C_w\}$ is a partition of G; let $\lambda \in \mathbb{N}$ be the security parameter, and $\pi = (\pi.\mathcal{GEN}, \pi.\mathcal{DER})$ be the *KAS-chain* for a totally ordered set of length at most l_{max}.

The *chain partition* algorithm $\Pi = (\Pi.\mathcal{GEN}, \Pi.\mathcal{DER})$ is a pair of algorithms over a setup algorithm $\Pi.\mathsf{Setup}$. Π satisfies the following conditions.

1. The PPT setup algorithm $\Pi.\mathsf{Setup}(1^\lambda)$ outputs the parameter $params^{(\Pi)}$, a set of

access graphs $\Gamma^{(\Pi)}$ and the set $\mathcal{K}^{(\Pi)}$ denoting the *key space*.

Here, $\mathcal{K}^{(\Pi)} = \{0,1\}^{p(\lambda)}$, where $p(\cdot)$ is some polynomial.

2. The PPT key generation algorithm $\Pi.\,\mathcal{GEN}$ takes as inputs the parameter $params^{(\Pi)}$, the access graph $G = (V, E) \in \Gamma^{(\Pi)}$, and the *KAS-chain* π, and returns a three-tuple (S, k, pub), where $S = (S_u)_{u \in V}$, $k = (k_u)_{u \in V}$ and pub are the sequence of private information, keys and public values respectively.

   Note that $k_u \in \mathcal{K}^{(\Pi)}$, $S_u \in \{0,1\}^*$ and $pub \in \{0,1\}^*$, for all $u \in V$.

3. The key derivation algorithm $\Pi.\,\mathcal{DER}$ is a deterministic PT algorithm such that $k_{u^g_h} := \Pi.\,\mathcal{DER}(params^{(\Pi)}, G, u^i_j, u^g_h, S_{u^i_j}, pub_g, \pi)$. Here: $u^g_h \leq u^i_j$ are two nodes of the access graph G; $S_{u^i_j}$ is u^i_j's private information; pub_g is the public information; π is the *KAS-chain*; and $k_{u^g_h}$ is u^g_h's decryption key.

   Note that $S_{u^i_j} \in \{0,1\}^*$, $pub_g \in \{0,1\}^*$ and $k_{u^g_h} \in \mathcal{K}^{(\Pi)} \cup \perp$.

The pseudo-code for the *chain partition* algorithm Π is described in Figure 5. The subroutines used by the algorithm are described in Subsubsection 2.2.8. These subroutines are identical to the subroutines used in [FPP13], but we reproduce them for the sake of completeness.

```
$\Pi.\,\mathcal{GEN}(params^{(\Pi)}, G, \pi)$
$(w, C[\,]) :=$ partition$(G)$;
for $(i := 1, 2, \cdots, w)$
  $(T^i, k^i, pub^i) := \pi.\,\mathcal{GEN}(params^{(\pi)}, C_i)$;
for $u \in V$
  $(\hat{u}_1, \hat{u}_2, \cdots, \hat{u}_w) :=$ max_isect_chs$(u, G)$;
  $S_u := T_{\hat{u}_1} \cup T_{\hat{u}_2} \cup \cdots \cup T_{\hat{u}_w}$;
$S := (S_u)_{u \in V}$, $k := (k_u)_{u \in V}$, $pub := (pub^i)_{i \in [w]}$;
return $(S, k, pub)$;
```

```
$\Pi.\,\mathcal{DER}(params^{(\Pi)}, G, u^i_j, u^g_h, S_{u^i_j}, pub_g, \pi)$
$\hat{u}_g :=$ max_isect$(u^i_j, C_g)$;
$k_{u^g_h} := \pi.\,\mathcal{DER}(params^{(\pi)}, C_g, \hat{u}_g, u^g_h,$
        $T_{\hat{u}_g, pub_g})$;
return $k_{u^g_h}$;
```

Figure 5: *Chain partition* algorithm for building *KAS*. The functions partition, max_isect_chs and max_isect are described in Subsubsection 2.2.8.

3 A New Cryptographic Primitive: *KAS-AE*

We have already discussed the *key assignment scheme (KAS)* in Subsubsection 2.2.7. This new primitive *KAS-AE* can, loosely, be viewed as a *KAS* plugged with an additional functionality, namely, *authenticated encryption*. We observe that *KAS* consists of two algorithms, namely, key generation and key derivation. The keys generated by *KAS* are later used to encrypt messages in various use-cases. The motivation for *KAS-AE* is to combine the *KAS* and *(authenticated) encryption* together, and view them as a single cryptographic primitive. Therefore, in *KAS-AE*, we target three goals: a combined key generation and authenticated encryption algorithm; a key derivation algorithm, which is identical to the one in *KAS*; and a decryption algorithm, which is necessitated by the authenticated encryption already included in the scheme. This new cryptographic primitive allows us to construct schemes that are more efficient than trivial execution of *KAS* followed by *AE*. In Section 1, we have discussed it in great detail. The full technical description of *KAS-AE* is as follows.

SYNTAX. Suppose $\lambda \in \mathbb{N}$ is the security parameter. A *KAS-AE* scheme $\Pi = (\Pi.\,\mathcal{E}, \Pi.\,\mathcal{DER}, \Pi.\,\mathcal{D})$ is a three-tuple of algorithms over a setup algorithm $\Pi.\,\mathsf{Setup}$. Π satisfies the following conditions.

1. The PPT setup algorithm $\Pi.\mathsf{Setup}(1^\lambda)$ outputs the parameter $params^{(\Pi)}$, a set of *access graphs* $\Gamma^{(\Pi)}$ and the sets $\mathcal{K}^{(\Pi)}$ and $\mathcal{M}^{(\Pi)}$, denoting the *key* and *message spaces* respectively.

2. The PPT encryption algorithm $\Pi.\mathcal{E}$ takes as inputs the parameter $params^{(\Pi)}$, a graph $G \in \Gamma^{(\Pi)}$ and a vector of files $f = (f_u)_{u \in V}$, and returns a three-tuple $(S, k, pub) := \Pi.\mathcal{E}(params^{(\Pi)}, G, f)$, where $S = (S_u)_{u \in V}$ and $k = (k_u)_{u \in V}$. The variables S, k and pub are called private information, key and public information vectors respectively.

   Note that $f_u \in \mathcal{M}^{(\Pi)}$, $k_u \in \mathcal{K}^{(\Pi)}$, $S_u \in \{0,1\}^*$ and $pub \in \{0,1\}^*$, for all $u \in V$.

3. The key derivation algorithm $\Pi.\mathcal{DER}$ is a deterministic PT algorithm such that $k_v := \Pi.\mathcal{DER}(params^{(\Pi)}, G, u, v, S_u, pub)$. Here: $v \le u$ are two nodes of the access graph G; S_u is u's private information; pub is the public information; and k_v is v's decryption key.

   Note that $S_u \in \{0,1\}^*$, $pub \in \{0,1\}^*$ and $k_v \in \mathcal{K}^{(\Pi)} \cup \{\perp\}$.[1]

4. The decryption algorithm $\Pi.\mathcal{D}$ is a deterministic PT algorithm such that $f_v := \Pi.\mathcal{D}(params^{(\Pi)}, G, u, v, S_u, pub)$. Here: node u decrypts the ciphertext corresponding to node v such that $v \le u$ in the access graph G; S_u is u's private information; pub is the public information; and f_v is v's decrypted file.

   Note that $S_u \in \{0,1\}^*$, $pub \in \{0,1\}^*$ and $f_v \in \mathcal{M}^{(\Pi)} \cup \{\perp\}$. A special case $f_v = \perp$ occurs, if ciphertext of v is not valid, that is, ciphertext is not generated by encrypting a valid message.

Remark. In principle, KAS-AE should also have an update function, allowing the users to encrypt modified plaintext efficiently. Note that such a function is absent in the definition. In fact, an update function, rather a trivial one, is implicitly present, and works in the following way: any update to original file is considered a new file requiring a fresh encryption.

Design and analysis of non-trivial update functions is a deeper issue in its own right, and, would shift the focus of the work of this paper. Therefore, this requires a separate discussion.

CORRECTNESS. The correctness of Π requires that for all $\lambda \in \mathbb{N}$, all $G = (V, E) \in \Gamma^{(\Pi)}$, all $f \in {\mathcal{M}^{(\Pi)}}^{|V|}$, all (S, k, pub) output by $\Pi.\mathcal{E}(params^{(\Pi)}, G, f)$, and all nodes $v \le u$, we have:

- $\Pi.\mathcal{DER}(params^{(\Pi)}, G, u, v, S_u, pub) = k_v$, and
- $\Pi.\mathcal{D}(params^{(\Pi)}, G, u, v, S_u, pub) = f_v$.

SECURITY. The security notions of *KAS-AE* are influenced by those of *KAS*[FPP13] and *AE*[Rog02, BRW03, BN08]. So, we should have four security notions, namely, *key indistinguishability*, *key recovery*, *privacy* and *tag consistency* using the KI-ST & S-KI-ST, KR-ST, IND-PRV and INT games. However, the notion of of *key indistinguishability*, as described in [FPP13], is not relevant for *KAS-AE* since the key used for decryption is the private information itself, and the *pub* value contains the ciphertext. Taking into consideration the scenarios, we target the three security goals: *key recovery*, *privacy* and *integrity*. All the games are written in a challenger-adversary framework.

---

[1] The fact that key derivation is used within decryption gives an impression that it does not have a separate existence. This assumption is not true. For example, when a new member joins a class (without changing hierarchical access structure), only key derivation is needed to compute his/her key. Thus, both key derivation and decryption are required in the definition.

```
Game $\mathsf{KR\text{-}ST}_{\Pi}^{\mathcal{A}}(1^\lambda, G)$
$(u, f) := \mathcal{A}_1(1^\lambda, G)$;
$(S, k, pub) :=$
    $\Pi.\mathcal{E}(params^{(\Pi)}, G, f)$;
$P_u := \{S_v \in S | v < u\}$;
$k'_u := \mathcal{A}_2(1^\lambda, G, pub, P_u)$;
return $k_u = k'_u$;
```

```
Game $\mathsf{IND\text{-}PRV}_{\Pi}^{\mathcal{A}}(1^\lambda, G, b)$
$(f^0, f^1) := \mathcal{A}_1(1^\lambda, G)$;
$(S, k, pub) :=$
    $\Pi.\mathcal{E}(params^{(\Pi)}, G, f^b)$;
$b' := \mathcal{A}_2(1^\lambda, G, pub,$
    $f^0, f^1)$;
return $b'$;
```

```
Game $\mathsf{INT}_{\Pi}^{\mathcal{A}}(1^\lambda, G)$
$(u, pub^0, pub^1, S, k) := \mathcal{A}(1^\lambda, G)$;
$f_u^0 := \Pi.\mathcal{D}(params^{(\Pi)}, G, u, u, S_u, pub^0)$;
$f_u^1 := \Pi.\mathcal{D}(params^{(\Pi)}, G, u, u, S_u, pub^1)$;
If $(f_u^0 \neq \perp) \wedge (f_u^1 \neq \perp) \wedge (f_u^0 \neq f_u^1)$
    return 1;
Else return 0;
```

Figure 6: Games defining KR-ST, IND-PRV and INT security for *KAS-AE* $\Pi = (\Pi.\mathcal{E}, \Pi.\mathcal{DER}, \Pi.\mathcal{D})$. Here, in KR-ST and IND-PRV games, the adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$.

KEY RECOVERY. Let $\Gamma^{(\Pi)}$ be a set of access graphs and $\Pi = (\Pi.\mathcal{E}, \Pi.\mathcal{DER}, \Pi.\mathcal{D})$ be the *KAS-AE* for $\Gamma^{(\Pi)}$. For a *KAS-AE* scheme we have designed a *key recovery with respect to static adversary*[2] KR-ST security game in Figure 6. The static adversary $\mathcal{A}$, when given access to the graph $G = (V, E)$, returns a security class $u \in V$, that $\mathcal{A}$ chooses to attack, and a sequence of files f to the challenger. The challenger then performs the following operations: computes (S, k, pub) using the $\Pi.\mathcal{E}(params^{(\Pi)}, G, f)$; computes P_u as the set of private information S_v for the classes $v \in V$ such that $v < u$; and sends (pub, P_u) to the adversary. The adversary has to return a key k'_u corresponding to the ciphertext for node u. If the keys k_u and k'_u match, then the adversary wins the game.

Now, we define the advantage of a KR-ST adversary $\mathcal{A}$ against Π on a graph $G \in \Gamma^{(\Pi)}$ as:

$$Adv_{\Pi,\mathcal{A},G}^{\mathsf{KR\text{-}ST}}(1^\lambda) \overset{def}{=} \Pr[\mathsf{KR\text{-}ST}_{\Pi}^{\mathcal{A}}(1^\lambda, G) = 1].$$

A *KAS-AE* scheme Π is said to be KR-ST secure, if $Adv_{\Pi,\mathcal{A},G}^{\mathsf{KR\text{-}ST}}(\cdot)$ is negligible, for all PT static adversaries $\mathcal{A}$.

PRIVACY. Let $\Gamma^{(\Pi)}$ be a set of access graphs and $\Pi = (\Pi.\mathcal{E}, \Pi.\mathcal{DER}, \Pi.\mathcal{D})$ be the *KAS-AE* for $\Gamma^{(\Pi)}$. For a *KAS-AE* scheme we have designed an *indistinguishability privacy* IND-PRV security game in Figure 6. The adversary $\mathcal{A}$, when given access to the graph $G = (V, E)$, returns two sequences of files f^0 and f^1, such that $\forall u \in V$, $|f_u^0| = |f_u^1|$. The challenger encrypts f^0 or f^1 according to the value of the input parameter b to obtain (S, k, pub) and sends (pub) to the adversary. The adversary has to return a bit b' indicating whether the ciphertext corresponds to file sequence f^0 or f^1. If the values of b and b' match, then the adversary wins the game.

Now, we define the advantage of an IND-PRV adversary $\mathcal{A}$ against Π on a graph $G \in \Gamma^{(\Pi)}$ as:

$$Adv_{\Pi,\mathcal{A},G}^{\mathsf{IND\text{-}PRV}}(1^\lambda) \overset{def}{=} \left| \Pr[\mathsf{IND\text{-}PRV}_{\Pi}^{\mathcal{A}}(1^\lambda, G, b = 1) = 1] - \Pr[\mathsf{IND\text{-}PRV}_{\Pi}^{\mathcal{A}}(1^\lambda, G, b = 0) = 1] \right|.$$

A *KAS-AE* scheme Π is said to be IND-PRV secure, if $Adv_{\Pi,\mathcal{A},G}^{\mathsf{IND\text{-}PRV}}(\cdot)$ is negligible, for all adversaries $\mathcal{A}$.

TAG CONSISTENCY. Let $\Gamma^{(\Pi)}$ be a set of access graphs and $\Pi = (\Pi.\mathcal{E}, \Pi.\mathcal{DER}, \Pi.\mathcal{D})$ be the *KAS-AE* for $\Gamma^{(\Pi)}$. For a *KAS-AE* scheme we have designed the *tag consistency* INT security game in Figure 6. Here, the challenger receives the target security class u, two

---

[2] A static adversary is polynomially equivalent to a dynamic adversary. The dynamic adversary is different from a static adversary in the way that, unlike the latter, the former can make adaptive queries to gather information from the nodes[FPP13].

public information vectors pub^0 and pub^1, secret information vector S and key vector k from the adversary. The challenger computes files $f_u^0 := \Pi.\mathcal{D}(params^{(\Pi)}, G, u, u, S_u, pub^0)$ and $f_u^1 := \Pi.\mathcal{D}(params^{(\Pi)}, G, u, u, S_u, pub^1)$. The adversary wins if both the files are *valid*, i.e., $f_u^0 \neq \perp$ and $f_u^1 \neq \perp$, and the two files are unidentical, i.e. $f_u^0 \neq f_u^1$.

Now, we define the advantage of an INT adversary $\mathcal{A}$ against Π on a graph $G \in \Gamma^{(\Pi)}$ as:

$$Adv^{\mathsf{INT}}_{\Pi,\mathcal{A},G}(1^\lambda) \overset{def}{=} \Pr[\mathsf{INT}^{\mathcal{A}}_{\Pi}(1^\lambda, G) = 1].$$

A *KAS-AE* scheme Π is said to be INT secure, if $Adv^{\mathsf{INT}}_{\Pi,\mathcal{A},G}(\cdot)$ is negligible, for all adversaries $\mathcal{A}$.

Remark. Note that a *KAS-AE-chain* is a special type of *KAS-AE* where the access graph is a *totally ordered set*.

4 *KAS-AE* from *KAS* and *AE*

In this section, we design *KAS-AE* schemes from *KAS* and *AE* schemes.

4.1 A natural construction, and an attack

We now attempt to construct *KAS-AE* constructions from *KAS* in the most intuitive way. Later we show that how this natural *KAS-AE* construction is vulnerable to an attack.

A *KAS-AE* scheme guarantees authentication of encrypted messages, in addition to the security properties of a *KAS* (note that *KAS* security properties alone do not guarantee authenticated encryption). A natural way to include this property in *KAS* could have been to use an *authenticated encryption (AE)* scheme to *aencrypt* the messages of the nodes using the keys distributed to them by the *KAS*. Such a natural *KAS-AE* scheme $\Pi = (\Pi.\mathcal{E}, \Pi.\mathcal{DER}, \Pi.\mathcal{D})$ is constructed below using the *KAS* $\Psi = (\Psi.\mathcal{GEN}, \Psi.\mathcal{DER})$ and the *AE* scheme $\Omega = (\Omega.\mathcal{K}_{\text{GEN}}, \Omega.\mathcal{E}, \Omega.\mathcal{D})$.

```
$\Pi.\mathcal{E}(params^{(\Pi)}, G, f)$
$(S, k, pub) := \Psi.\mathcal{GEN}(params^{(\Psi)}, G)$;
for all $u \in V$
    $(c_u, t_u) := \Omega.\mathcal{E}(params^{(\Omega)}, k_u, f_u)$;
$c := (c_u||t_u)_{u \in V}$;
$pub := c||pub$;
return $(S, k, pub)$;

$\Pi.\mathcal{DER}(params^{(\Pi)}, G, u, v, S_u, pub)$
$k_v := \Psi.\mathcal{DER}(params^{(\Psi)}, G, u, v, S_u, pub)$;
return $k_v$;

$\Pi.\mathcal{D}(params^{(\Pi)}, G, u, v, S_u, pub)$
$k_v := \Psi.\mathcal{DER}(params^{(\Psi)}, G, u, v, S_u, pub)$;
$c_v||t_v :=$ ext_cipher$(pub, v)$;
$f_v := \Omega.\mathcal{D}(params^{(\Omega)}, k_v, c_v, t_v)$;
return $f_v$;
```

Figure 7: Algorithmic description of the *KAS-AE* scheme of Subsection 4.1: simple combination of *KAS* and *AE*.

A SIMPLE ATTACK ON THE TAG CONSISTENCY SECURITY OF Π. The attack works as follows: a node v replaces the original ciphertext $c_v||t_v$ with a different ciphertext $c'_v||t'_v$ computed under the original key k_v and file $f'_v \neq f_v$; now, a senior node u (*i.e.*, $v \leq u$) decrypts c'_v without any error message.

4.2 A secure (yet inefficient) scheme

We have shown an attack on the most intuitive construction of *KAS-AE* built from *KAS* and *AE* construction. In this section, we design a generic *KAS-AE* scheme by combining a generic *KAS* scheme and an *AE* scheme in a different way than done in Subsection 4.1, so that the attack of Subsection 4.1 is avoided. Although, it generates a secure *KAS-AE* scheme, the high memory requirements make it unsuitable for any practical applications. Let $\Pi = (\Pi.\mathcal{E}, \Pi.\mathcal{DER}, \Pi.\mathcal{D})$ be the *KAS-AE* scheme, $\Psi = (\Psi.\mathcal{GEN}, \Psi.\mathcal{DER})$ be the *KAS* and $\Omega = (\Omega.\mathcal{K}_{\text{GEN}}, \Omega.\mathcal{E}, \Omega.\mathcal{D})$ denote the *AE* scheme. As opposed to considering the authentication tag being a part of the ciphertext, here, we assume that tag and ciphertext are distinct. The core idea behind this construction is that tag is a secret value, and that every node stores the tags of itself and its successors. The full construction of Π is shown in Figure 8. Here, it is important to note that $\Gamma^{(\Pi)} = \Gamma^{(\Psi)}$ and $params^{(\Pi)} = (params^{(\Psi)}, params^{(\Omega)})$.

```
$\Pi.\mathcal{E}(params^{(\Pi)}, G, f)$
$(S, k, pub) := \Psi.\mathcal{GEN}(params^{(\Psi)}, G)$;
for $u \in V$
    $(c_u, t_u) := \Omega.\mathcal{E}(params^{(\Omega)}, k_u, f_u)$;
for $u \in V$
    $\downarrow u :=$ all_succ$(u, G)$;
    for $v \in \downarrow u$
        $S_u := S_u \circ t_v$;
$c := (c_u)_{u \in V}$, $S := (S_u)_{u \in V}$;
$pub := c || pub$;
return $(S, k, pub)$;

$\Pi.\mathcal{DER}(params^{(\Pi)}, G, u, v, S_u, pub)$
$k_v := \Psi.\mathcal{DER}(params^{(\Psi)}, G, u, v, S_u, pub)$;
return $k_v$;

$\Pi.\mathcal{D}(params^{(\Pi)}, G, u, v, S_u, pub)$
$k_v := \Psi.\mathcal{DER}(params^{(\Psi)}, G, u, v, S_u, pub)$;
$t_v :=$ ext_tag$(S_u, v)$;
$c_v :=$ ext_cipher$(pub, v)$;
$f_v := \Omega.\mathcal{D}(params^{(\Omega)}, k_v, c_v, t_v)$;
return $f_v$;
```

Figure 8: A Framework for building a *KAS-AE* scheme from *KAS* and *AE* schemes, used separately. By plugging an existing *KAS* scheme $\mathsf{X} \in \{\mathsf{TKAS}, \mathsf{TKEKAS}, \mathsf{DKEKAS}, \mathsf{IKEKAS}, \mathsf{NBKAS}\}$ into the framework, we get a concrete *KAS-AE* construction, namely, X-AE scheme. In Subsubsection 2.3.3, various existing *KAS* constructions have been described.

Theorem 1. *If the underlying KAS (or AE) is* KR-ST *(or* IND-PRV *or* INT*) secure, then the KAS-AE construction is also* KR-ST *(or* IND-PRV *or* INT*) secure.*

Proof sketch. We can prove the KR-ST (or IND-PRV or INT) security of this construction by using reduction. So, we can show that if the adversary $\mathcal{A}$ can break the KR-ST (or IND-PRV or INT) security of *KAS-AE*, then an adversary $\mathcal{B}$, using $\mathcal{A}$, can break the KR-ST (or IND-PRV or INT) security of *KAS* (or *AE*) scheme. By using the contrapositive argument, this would show that if the underlying *KAS* (or *AE*) scheme is secure, so is the *KAS-AE* scheme.

5 Building *KAS-AE* using *Modified Chain Partition*

In this section, we design *KAS-AE* schemes by using *KAS-AE-chain* schemes in the *modified chain partition* algorithm. *KAS-AE-chain* has already been described in Section 3. The *modified chain partition* algorithm will be described in detail shortly.

5.1 *KAS-AE-chain* constructions

The first ingredient to construct *KAS-AE* scheme is a *KAS-AE-chain* scheme. We describe four different types of *KAS-AE-chain*, namely, $\mathsf{A}_{\mathsf{Chain}}$, $\mathsf{B}_{\mathsf{Chain}}$, $\mathsf{C}_{\mathsf{Chain}}$ and $\mathsf{D}_{\mathsf{Chain}}$, based on *KAS-chain* [FPP13] & *AE* [Rog02, BRW03, BN08], *MLE* [BKR13], *APE* [ABB+14] and *FP* [PHG12] respectively.

5.1.1 $\mathsf{A}_{\mathsf{Chain}}$: ***KAS-AE-chain*** based on ***KAS-chain*** and ***AE***

An $\mathsf{A}_{\mathsf{Chain}}$ $\Pi = (\Pi.\mathcal{E}, \Pi.\mathcal{DER}, \Pi.\mathcal{D})$ is a *KAS-AE-chain* built from a *KAS-chain* $\Psi = (\Psi.\mathcal{GEN}, \Psi.\mathcal{DER})$ and an *AE* scheme $\Omega = (\Omega.\mathcal{K}_{\mathrm{GEN}}, \Omega.\mathcal{E}, \Omega.\mathcal{D})$ following the framework described in Figure 9.

```
$\Pi.\mathcal{E}(params^{(\Pi)}, G, f)$

$(u_1, u_2, \cdots, u_m) :=$ vertex_in_order$(G)$;
$(f_1 \circ f_2 \circ \cdots \circ f_m) := f$;
$(S, k, pub) := \Psi.\mathcal{GEN}(params^{(\Psi)}, G)$;
for $(i := m, m-1, \cdots, 1)$
  $(c_i, t_i) := \Omega.\mathcal{E}(params^{(\Omega)}, k_i, f_i)$;
  $S_i = S_i \circ t_i \circ t_{i+1} \circ \cdots \circ t_m$;
$pub := (c_1 \circ c_2 \circ \cdots \circ c_m)||pub$;
$S := (S_1 \circ S_2 \circ \cdots \circ S_m)$;
$k := (k_1 \circ k_2 \circ \cdots \circ k_m)$;
return $(S, k, pub)$;
```

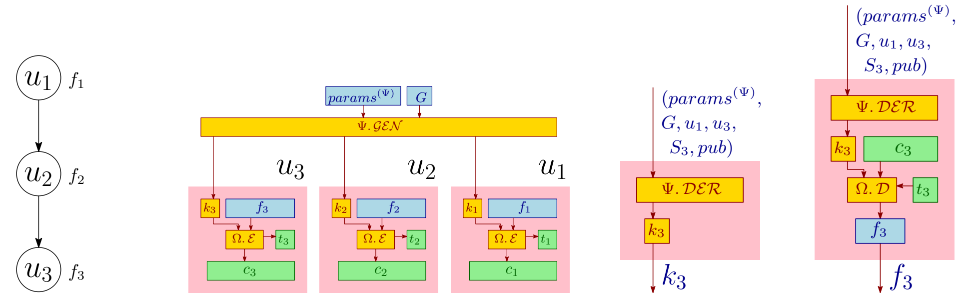
```
$\Pi.\mathcal{DER}(params^{(\Pi)}, G, u, v, S_u, pub)$

$u_i := u,\ u_j := v$;
$k_j := \Psi.\mathcal{DER}(params^{(\Psi)}, G, u, v, S_u, pub)$;
return $k_j$;
```

```
$\Pi.\mathcal{D}(params^{(\Pi)}, G, u, v, S_u, pub)$

$u_i := u,\ u_j := v$;
$k_j := \Psi.\mathcal{DER}(params^{(\Psi)}, G, u, v, S_u, pub)$;
$c_j :=$ ext_cipher$(pub, u_j)$;
$t_j :=$ ext_tag$(S_u, u_j)$;
$f_j := \Omega.\mathcal{D}(params^{(\Omega)}, k_j, c_j, t_j)$;
return $f_j$;
```

(a) Algorithmic description of building $\mathsf{A}_{\mathsf{Chain}}$ Π using the *KAS-chain* Ψ and *AE* scheme Ω. For pictorial description with an example, see 9(b)–9(e)



(b) The access graph G with 3 nodes u_1, u_2, u_3 and their corresponding files f_1, f_2, f_3.

(c) Pictorial description of $\Pi.\mathcal{E}(params^{(\Pi)}, G, f)$, where $f = (f_1 \circ f_2 \circ f_3)$ and G is shown in 9(b).

(d) Pictorial description of $\Pi.\mathcal{DER}(params^{(\Pi)}, G, u_1, u_3, S_1, pub)$.

(e) Pictorial description of $\Pi.\mathcal{D}(params^{(\Pi)}, G, u_1, u_3, S_1, pub)$.

Figure 9: Building $\mathsf{A}_{\mathsf{Chain}}$.

Security of $\mathrm{A}_{\mathrm{Chain}}$

Theorem 2. *If the underlying KAS-chain scheme is* KR-ST *secure, then the Construction* A_{Chain} *is also* KR-ST *secure.*

Proof. The proof is by using reduction. So, we can show that if an adversary $\mathcal{A}$ can break the KR-ST security of Construction $\mathrm{A}_{\mathrm{Chain}}$, then an adversary $\mathcal{B}$, using $\mathcal{A}$, can break the KR-ST security of the underlying *KAS-chain* Ψ. By using the contrapositive argument, this would show that if the underlying *KAS* is secure, so is the Construction $\mathrm{A}_{\mathrm{Chain}}$. ☐

Theorem 3. *If the underlying AE scheme is* IND-PRV *secure, then the Construction* A_{Chain} *is also* IND-PRV *secure.*

Proof. The proof is by using reduction. So, we can show that if an adversary $\mathcal{A}$ can break the IND-PRV security of Construction $\mathrm{A}_{\mathrm{Chain}}$, then an adversary $\mathcal{B}$, using $\mathcal{A}$, can break the IND-PRV security of the underlying *AE* Ω. By using the contrapositive argument, this would show that if the underlying *AE* is secure, so is the Construction $\mathrm{A}_{\mathrm{Chain}}$. ☐

Theorem 4. *If the underlying AE scheme is INT secure, then the Construction A_{Chain} is also INT secure.*

Proof. The proof is by using reduction. So, we can show that if an adversary $\mathcal{A}$ can break the INT security of Construction $\mathrm{A}_{\mathsf{Chain}}$, then an adversary $\mathcal{B}$, using $\mathcal{A}$, can break the INT security of the underlying *AE* Ω. By using the contrapositive argument, this would show that if the underlying *AE* is secure, so is the Construction $\mathrm{A}_{\mathsf{Chain}}$. ☐

5.1.2 $\mathsf{B}_{\mathsf{Chain}}$: *KAS-AE-chain* based on *MLE*

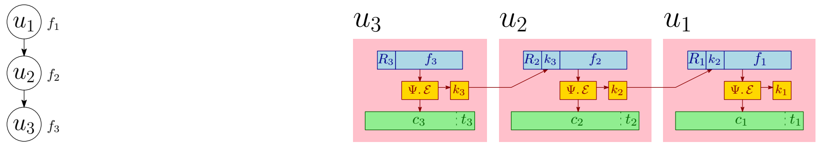
A $\mathsf{B}_{\mathsf{Chain}}$ $\Pi = (\Pi.\mathcal{E}, \Pi.\mathcal{DER}, \Pi.\mathcal{D})$ is a *KAS-AE-chain* built from an *MLE* scheme $\Psi = (\Psi.\mathcal{E}, \Psi.\mathcal{D})$ following the framework described in Figure 10.

```
$\Pi.\mathcal{E}(params^{(\Pi)}, G, f)$
$(u_1, u_2, \cdots, u_m) :=$ vertex_in_order$(G)$;
$(f_1 \circ f_2 \circ \cdots \circ f_m) := f$;
$k_{m+1} := \epsilon$;
for $(i := m, m-1, \cdots, 1)$
    $R_i \xleftarrow{\$} \{0,1\}^{\lambda}$;
    $f'_i := R_i || k_{i+1} || f_i$;
    $(k_i, c_i, t_i) := \Psi.\mathcal{E}(params^{(\Psi)}, f'_i)$;
$pub := (c_1||t_1 \circ c_2||t_2 \circ \cdots \circ c_m||t_m)$;
$S := k := (k_1 \circ k_2 \circ \cdots \circ k_m)$;
return $(S, k, pub)$;
```

```
$\Pi.\mathcal{DER}(params^{(\Pi)}, G, u, v, S_u, pub)$
$u_i := u$, $u_j := v$, $k_i := S_u$;
if $(u < v)$, then return $\perp$;
if $u = v$, then return $k_j := k_i$;
$(c_1||t_1 \circ c_2||t_2 \circ \cdots \circ c_m||t_m) := pub$;
for $(\ell := i, i+1, \cdots, j-1)$
    $f'_\ell := \Psi.\mathcal{D}(params^{(\Psi)}, k_\ell, c_\ell, t_\ell)$;
    If $f'_\ell = \perp$, then return $\perp$;
    $R_\ell || k_{\ell+1} || f_\ell := f'_\ell$;
return $k_j$;
```

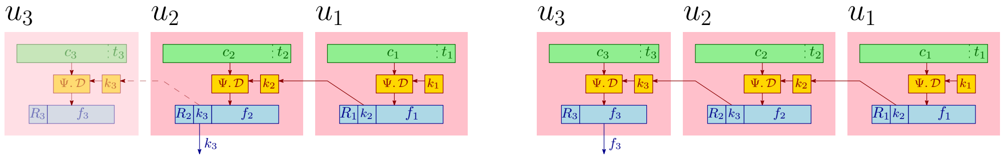
```
$\Pi.\mathcal{D}(params^{(\Pi)}, G, u, v, S_u, pub)$
$u_i := u$, $u_j := v$, $m := |V|$;
$k_j := \Pi.\mathcal{DER}(params^{(\Pi)}, G, u, v, S_u, pub)$;
If $k_j = \perp$, then return $\perp$;
$(c_1||t_1 \circ c_2||t_2 \circ \cdots \circ c_m||t_m) := pub$;
$f'_j := \Psi.\mathcal{D}(params^{(\Psi)}, k_j, c_j, t_j)$;
If $f'_j = \perp$, then return $\perp$;
If $j = m$, then $R_j || f_j := f'_j$;
Else $R_j || k_{j+1} || f_j := f'_j$;
return $f_j$;
```

(a) Algorithmic description of building $\mathsf{B}_{\mathsf{Chain}}$ Π using the *MLE* scheme Ψ. For the pictorial description with an example, see 10(b)–10(e).



(b) The access graph G with 3 nodes u_1, u_2, u_3 and their corresponding files f_1, f_2, f_3.

(c) Pictorial description of $\Pi.\mathcal{E}(params^{(\Pi)}, G, f)$, where $f = (f_1 \circ f_2 \circ f_3)$ and G is shown in 10(b).



(d) Pictorial description of $\Pi.\mathcal{DER}(params^{(\Pi)}, G, u_1, u_3, S_1, pub)$.

(e) Pictorial description of $\Pi.\mathcal{D}(params^{(\Pi)}, G, u_1, u_3, S_1, pub)$.

Figure 10: Building $\mathsf{B}_{\mathsf{Chain}}$.

Security of Construction $\mathrm{B}_{\mathsf{Chain}}$

Theorem 5. *If the underlying MLE scheme is KR-CDA secure, then the Construction B_{Chain} is also KR-ST secure.*

Proof. The proof is by using reduction as shown in Figure 11. So, we show that if an adversary $\mathcal{A}$ can break the KR-ST security of Construction $\mathrm{B}_{\mathsf{Chain}}$, then an adversary $\mathcal{B}$, using $\mathcal{A}$, can break the KR-CDA security of the underlying *MLE* scheme Ψ. By using the contrapositive argument, this would show that if the underlying *MLE* scheme is secure, so is the Construction $\mathrm{B}_{\mathrm{Chain}}$.

Our message source $\mathcal{S}_{u,m}$ works in the following way: for $i = m, m-1, \cdots, 1$, generate a message f_i and a λ-bit random number R_i, and computes $(k_i, c_i, t_i) := \Psi.\mathcal{E}(params^{(\Psi)}, R_i||k_{i+1}||f_i)$, where $k_{m+1} := \epsilon$; creates $Z := c_1||t_1 \circ c_2||t_2 \circ \cdots \circ c_{u-1}||t_{u-1} \circ c_{u+1}||t_{u+1} \circ \cdots \circ c_m||t_m \circ k_{u+1} \circ k_{u+2} \circ \cdots \circ k_m$ and message $M = R_u||k_{u+1}||f_u$; and returns (M, Z). Here, u is the security class that $\mathcal{A}$ chooses to attack and m is the number of nodes (or security classes) in the graph G. ☐

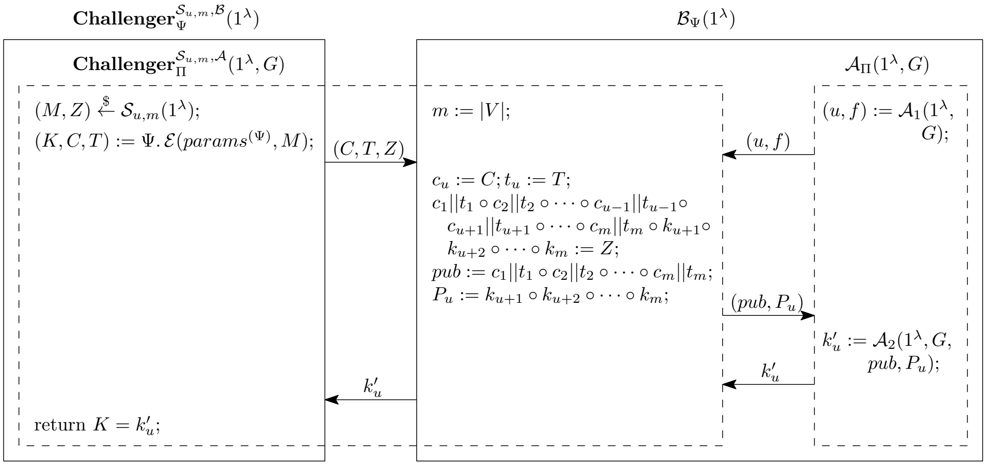


Figure 11: The reduction used in Theorem 5: *MLE* adversary is constructed using *KAS-AE-chain* adversary.

Theorem 6. *If the underlying MLE scheme is* PRV-CDA *secure, then the Construction B_{Chain} is also* IND-PRV *secure.*

Proof. The proof is by using reduction as shown in Figure 12. So, we show that if an adversary $\mathcal{A}$ can break the IND-PRV security of Construction $\mathrm{B}_{\mathrm{Chain}}$, then an adversary $\mathcal{B}$, using $\mathcal{A}$, can break the PRV-CDA security of the underlying *MLE* scheme Ψ. By using the contrapositive argument, this would show that if the underlying *MLE* scheme is secure, so is the Construction $\mathrm{B}_{\mathrm{Chain}}$.

Our message source $\mathcal{S}_{f^0,f^1}$ mimics the functioning of the *KAS-AE-chain* scheme but instead of giving (S, k, pub) as output, it performs the following operations: for $i = m, m-1, \cdots, 1$ and $b \in \{0,1\}$, generate a λ-bit random number $R_b^{(i)}$, and computes $M_b^{(i)} := R_b^{(i)}||k_b^{(i+1)}||f_b^{(i)}$ and $(k_b^{(i)}, c_b^{(i)}, t_b^{(i)}) := \Psi.\mathcal{E}(params^{(\Psi)}, M_b^{(i)})$; and returns the sequence of strings $M_0 := (M_0^{(1)} \circ M_0^{(2)} \circ \cdots \circ M_0^{(m)})$ and $M_1 := (M_1^{(1)} \circ M_1^{(2)} \circ \cdots \circ M_1^{(m)})$ along with auxiliary information Z. Here, m is the number of nodes (or security classes) in the graph G and the adversary $\mathcal{A}$ generates two sequence of files f^0 and f^1 such that for $i = m, m-1, \cdots, 1$, $|f_i^0| = |f_i^1|$, which results into $|M_0^{(i)}| = |M_1^{(i)}|$ when $\mathcal{S}_{f^0,f^1}(1^\lambda)$ generates M_0, M_1 and Z. ☐

Theorem 7. *If the underlying MLE scheme is* TC *secure, then the Construction B_{Chain} is also* INT *secure.*

Proof. The proof is by using reduction as shown in Figure 13. So, we show that if an adversary $\mathcal{A}$ can break the INT security of Construction $\mathrm{B}_{\mathrm{Chain}}$, then an adversary $\mathcal{B}$, using $\mathcal{A}$, can break the TC security of the underlying *MLE* scheme Ψ. By using the

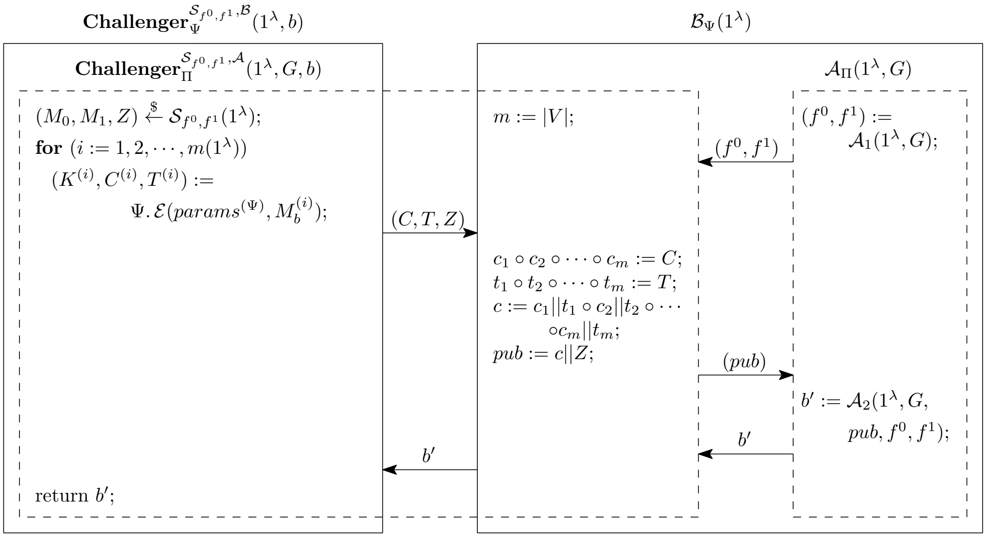


Figure 12: The reduction used in Theorem 6: *MLE* adversary is constructed using *KAS-AE-chain* adversary.

contrapositive argument, this would show that if the underlying *MLE* scheme is secure, so is the Construction $\mathrm{B}_{\mathsf{Chain}}$. □

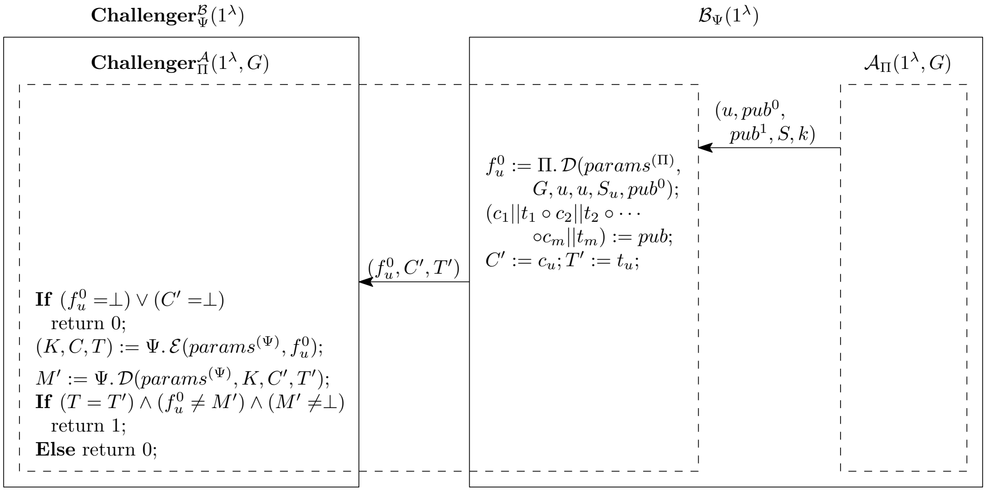


Figure 13: The reduction used in Theorem 7: *MLE* adversary is constructed using *KAS-AE-chain* adversary.

5.1.3 $\mathrm{C}_{\mathsf{Chain}}$: *KAS-AE-chain* based on *APE*

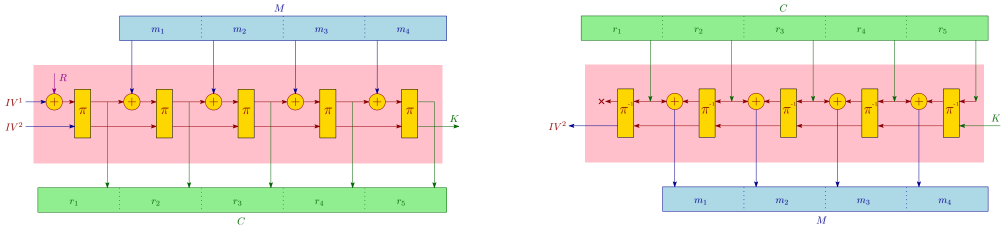
Functionalities based on *APE*

In this section, we are designing two functionalities – $\mathcal{F}_1^\pi$ and $\mathcal{F}_2^\pi$ – that are motivated by the encryption and decryption of authenticated encryption algorithm *APE* [ABB+14]. Let us first be very clear that the *APE* variant used by us is marginally different from the original *APE* construction by Andreeva *et al.* The main difference is: in the original *APE*, the encryption and decryption keys are identical, because of the XOR operation on the lower-half bits with the encryption key K, in the last round; whereas, in our variant, we remove this XOR. In the entire paper, by *APE* we refer to the variant used by us. The algorithmic and diagrammatic descriptions of $\mathcal{F}_1^\pi$ and $\mathcal{F}_2^\pi$ are shown in Figure 14.

```
$\mathcal{F}_1^\pi(1^\lambda, M, IV^1, IV^2)$
$p := |M|/\lambda$, $r_0 := IV^1$, $s_0 := IV^2$;
$m_1||m_2||\cdots||m_p := M$;
$R \xleftarrow{\$} \{0,1\}^\lambda$, $m_0 := R$;
for $(j := 0, 1, \cdots, p)$
  $r'_j := m_j \oplus r_j$;
  $(r_{j+1}||s_{j+1}) := \pi(r'_j||s_j)$;
$C := r_1||r_2||\cdots||r_{p+1}$, $K := s_{p+1}$;
return $(K, C)$;
```

```
$\mathcal{F}_2^\pi(1^\lambda, K, C)$
$p := |C|/\lambda - 1$, $s_{p+1} := K$;
$r_1||r_2||\cdots||r_{p+1} := C$;
for $(j := p, p-1, \cdots, 0)$
  $(r'_j||s_j) := \pi^{-1}(r_{j+1}||s_{j+1})$;
  If $j \neq 0$, then $m_j := r'_j \oplus r_j$;
$M := m_1||m_2||\cdots||m_p$, $IV^2 := s_0$;
return $(M, IV^2)$;
```

(a) Algorithmic description of functionalities $\mathcal{F}_1^\pi$ and $\mathcal{F}_2^\pi$.



(b) Diagrammatic description of functionality $(K, C) := \mathcal{F}_1^\pi(1^\lambda, M, IV^1, IV^2)$.

(c) Diagrammatic description of functionality $(M, IV^2) := \mathcal{F}_2^\pi(1^\lambda, K, C)$.

Figure 14: Algorithmic and diagrammatic descriptions of the functionalities $\mathcal{F}_1^\pi$ and $\mathcal{F}_2^\pi$ are shown in (a), (b) and (c); here, π is a 2λ-bit easy-to-invert permutation. Each wire in (b) and (c) represents λ bits. The function $\mathcal{F}_1^\pi$ takes as inputs parameter 1^λ, message M and two other values IV^1 and IV^2, and returns the decryption key K and the ciphertext C. Similarly, $\mathcal{F}_2^\pi$ takes as inputs parameter 1^λ, the decryption key K and the ciphertext C, and outputs the message M and value IV^2. For the sake of simplicity, we assume that $|M|$ is a multiple of security parameter λ.

KAS-AE-chain scheme based on functionalities $\mathcal{F}_1^\pi$ and $\mathcal{F}_2^\pi$

A $\mathsf{C}_{\mathsf{Chain}}$ $\Pi = (\Pi.\mathcal{E}, \Pi.\mathcal{DER}, \Pi.\mathcal{D})$ is a *KAS-AE-chain* built from the functionalities $\mathcal{F}_1^\pi$ and $\mathcal{F}_2^\pi$ following the framework described in Figure 15.

Security of Construction $\mathbf{C}_{\mathbf{Chain}}$

Theorem 8. *If π is the ideal permutation in Construction $\mathsf{C}_{\mathsf{Chain}}$, then*

$$Adv^{\textsf{\textit{KR-ST}}}_{\mathsf{C}_{\mathsf{Chain}},\mathcal{A},G}(1^\lambda) \leq \frac{\sigma(\sigma-1)}{2^{2\lambda-1}} + \frac{\sigma(\sigma-1)}{2^\lambda}$$

Proof. We prove security by constructing successive games (or hybrids) and finding adversarial advantages between them.

Game 0: This game is identical to KR-ST game where Construction $\mathsf{C}_{\mathsf{Chain}}$ is used. (see Figure 15).

Game 1: This **Game 1** is identical to **Game 0** except that we replace the 2λ-bit permutation π with 2λ-bit random function rf.

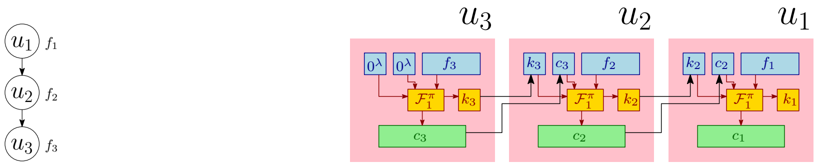
Using PRP/PRF Switching Lemma [BR06], for an adversary limited by σ queries to the permutation (or random function), the following equation can be obtained.

```
$\Pi.\mathcal{E}(params^{(\Pi)}, G, f)$
$(u_1, u_2, \cdots, u_m) :=$ vertex_in_order$(G)$;
$(f_1 \circ f_2 \circ \cdots \circ f_m) := f$;
$IV^1 := IV^2 := 0^\lambda$;
for $(i := m, m-1, \cdots, 1)$
    $(k_i, c_i) := \mathcal{F}_1^\pi(1^\lambda, f_i, IV^1, IV^2)$;
    $IV^1 := c_i[last\_block]$;
    $IV^2 := k_i$;
$pub := (c_1 \circ c_2 \circ \cdots \circ c_m)$;
$S := k := (k_1 \circ k_2 \circ \cdots \circ k_m)$;
return $(S, k, pub)$;
```

```
$\Pi.\mathcal{DER}(params^{(\Pi)}, G, u, v, S_u, pub)$
$u_i := u$, $u_j := v$, $k_i := S_u$;
if $(u < v)$, then return $\perp$;
if $u = v$, then return $k_j := k_i$;
$(c_1 \circ c_2 \circ \cdots \circ c_m) := pub$;
for $(\ell := i, i+1, \cdots, j-1)$
    $(f_\ell, k_{\ell+1}) := \mathcal{F}_2^\pi(1^\lambda, k_\ell, c_\ell)$;
return $k_j$;
```

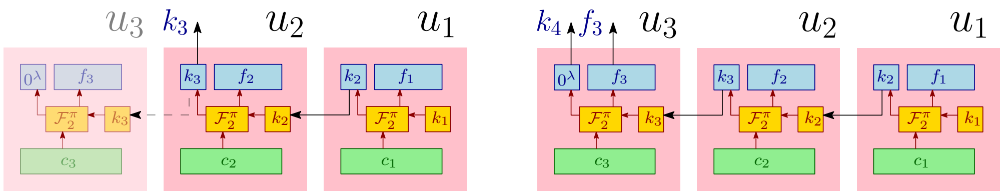
```
$\Pi.\mathcal{D}(params^{(\Pi)}, G, u, v, S_u, pub)$
$u_j := v$;
$(c_1 \circ c_2 \circ \cdots \circ c_m) := pub$;
$k_j := \Pi.\mathcal{DER}(params^{(\Pi)}, G, u, v, S_u, pub)$;
If $k_j = \perp$, then return $\perp$;
for $(\ell := j, j+1, \cdots, m)$
    $(f_\ell, k_{\ell+1}) := \mathcal{F}_2^\pi(1^\lambda, k_\ell, c_\ell)$;
If $k_{m+1} = 0^\lambda$, then return $f_j$;
Else return $\perp$;
```

(a) Algorithmic description of building $\mathsf{C}_{\mathsf{Chain}}$ Π using the functionalities $\mathcal{F}_1^\pi$ and $\mathcal{F}_2^\pi$. For the pictorial description with an example, see 15(b)–15(e).



(b) The access graph G with 3 nodes u_1, u_2, u_3 and their corresponding files f_1, f_2, f_3.

(c) Pictorial description of $\Pi.\mathcal{E}(params^{(\Pi)}, G, f)$, where $f = (f_1, f_2, f_3)$ and G is shown in 15(b).



(d) Pictorial description of $\Pi.\mathcal{DER}(params^{(\Pi)}, G, u_1, u_3, S_1, pub)$.

(e) Pictorial description of $\Pi.\mathcal{D}(params^{(\Pi)}, G, u_1, u_3, S_1, pub)$.

Figure 15: Building $\mathsf{C}_{\mathsf{Chain}}$.

$$\begin{aligned}\left|Adv^{\mathsf{Game\ 0}}_{\Pi^{\pi,\pi^{-1}},\mathcal{A},G}(1^\lambda) - Adv^{\mathsf{Game\ 1}}_{\Pi^{\mathsf{rf},\pi^{-1}},\mathcal{A},G}(1^\lambda)\right| &= \left|Adv^{\mathsf{KR\text{-}ST}}_{\Pi^{\pi,\pi^{-1}},\mathcal{A},G}(1^\lambda) - Adv^{\mathsf{KR\text{-}ST}}_{\Pi^{\mathsf{rf},\pi^{-1}},\mathcal{A},G}(1^\lambda)\right| \\ &\leq Adv^{\mathsf{IND\text{-}PRV}}_{\pi,\mathsf{rf},\mathcal{A}}(1^\lambda,\sigma) \leq \frac{\sigma(\sigma-1)}{2^{2\lambda}}\end{aligned} \tag{1}$$

Game 2: This **Game 2** is identical to **Game 1** except that here we change 2λ-bit permutation π^{-1} by a 2λ-bit random function rf'.

Using PRP/PRF Switching Lemma [BR06], for an adversary limited by σ queries to the permutation (or random function), the following equation can be obtained.

$$\begin{aligned}\left|Adv^{\mathsf{Game\ 1}}_{\Pi^{\mathsf{rf},\pi^{-1}},\mathcal{A},G}(1^\lambda) - Adv^{\mathsf{Game\ 2}}_{\Pi^{\mathsf{rf},\mathsf{rf}'},\mathcal{A},G}(1^\lambda)\right| &= \left|Adv^{\mathsf{KR\text{-}ST}}_{\Pi^{\mathsf{rf},\pi^{-1}},\mathcal{A},G}(1^\lambda) - Adv^{\mathsf{KR\text{-}ST}}_{\Pi^{\mathsf{rf},\mathsf{rf}'},\mathcal{A},G}(1^\lambda)\right| \\ &\leq Adv^{\mathsf{IND\text{-}PRV}}_{\pi^{-1},\mathsf{rf}',\mathcal{A}}(1^\lambda,\sigma) \leq \frac{\sigma(\sigma-1)}{2^{2\lambda}}\end{aligned} \tag{2}$$

Game 3: This **Game 3** is identical to **Game 2** except that the game aborts whenever there is a collision in the lower λ bits of rf or of rf'. The event of collision in the lower λ bits of rf or rf' is called a *bad* event.

Using Code-Based Game Playing Technique [BR06], for an adversary limited by σ queries to the random functions, the following equation can be obtained.

$$\begin{aligned}\left|Adv^{\mathsf{Game\ 2}}_{\Pi^{\mathsf{rf},\mathsf{rf}'},\mathcal{A},G}(1^\lambda) - Adv^{\mathsf{Game\ 3}}_{\hat{\Pi}^{\mathsf{rf},\mathsf{rf}'},\mathcal{A},G}(1^\lambda)\right| &= \left|Adv^{\mathsf{KR\text{-}ST}}_{\Pi^{\mathsf{rf},\mathsf{rf}'},\mathcal{A},G}(1^\lambda) - Adv^{\mathsf{KR\text{-}ST}}_{\hat{\Pi}^{\mathsf{rf},\mathsf{rf}'},\mathcal{A},G}(1^\lambda)\right| \\ &\leq Pr[\mathcal{A} \text{ sets } Bad] \leq \frac{\sigma(\sigma-1)}{2^{\lambda}}\end{aligned} \tag{3}$$

Using Triangle Inequality [BR06] and the Equation 1, Equation 2 and Equation 3, the following equation can be obtained.

$$\begin{aligned}\left|Adv^{\mathsf{Game\ 0}}_{\Pi^{\pi,\pi^{-1}},\mathcal{A},G}(1^\lambda) - Adv^{\mathsf{Game\ 3}}_{\hat{\Pi}^{\mathsf{rf},\mathsf{rf}'},\mathcal{A},G}(1^\lambda)\right| &= \Big|Adv^{\mathsf{Game\ 0}}_{\Pi^{\pi,\pi^{-1}},\mathcal{A},G}(1^\lambda) - Adv^{\mathsf{Game\ 1}}_{\Pi^{\mathsf{rf},\pi^{-1}},\mathcal{A},G}(1^\lambda) \\ &\quad + Adv^{\mathsf{Game\ 1}}_{\Pi^{\mathsf{rf},\pi^{-1}},\mathcal{A},G}(1^\lambda) - Adv^{\mathsf{Game\ 2}}_{\Pi^{\mathsf{rf},\mathsf{rf}'},\mathcal{A},G}(1^\lambda) \\ &\quad + Adv^{\mathsf{Game\ 2}}_{\Pi^{\mathsf{rf},\mathsf{rf}'},\mathcal{A},G}(1^\lambda) - Adv^{\mathsf{Game\ 3}}_{\hat{\Pi}^{\mathsf{rf},\mathsf{rf}'},\mathcal{A},G}(1^\lambda)\Big| \\ &\leq \left|Adv^{\mathsf{Game\ 0}}_{\Pi^{\pi,\pi^{-1}},\mathcal{A},G}(1^\lambda) - Adv^{\mathsf{Game\ 1}}_{\Pi^{\mathsf{rf},\pi^{-1}},\mathcal{A},G}(1^\lambda)\right| \\ &\quad + \left|Adv^{\mathsf{Game\ 1}}_{\Pi^{\mathsf{rf},\pi^{-1}},\mathcal{A},G}(1^\lambda) - Adv^{\mathsf{Game\ 2}}_{\Pi^{\mathsf{rf},\mathsf{rf}'},\mathcal{A},G}(1^\lambda)\right| \\ &\quad + \left|Adv^{\mathsf{Game\ 2}}_{\Pi^{\mathsf{rf},\mathsf{rf}'},\mathcal{A},G}(1^\lambda) - Adv^{\mathsf{Game\ 3}}_{\hat{\Pi}^{\mathsf{rf},\mathsf{rf}'},\mathcal{A},G}(1^\lambda)\right| \\ &\leq \frac{\sigma(\sigma-1)}{2^{2\lambda}} + \frac{\sigma(\sigma-1)}{2^{2\lambda}} + \frac{\sigma(\sigma-1)}{2^{\lambda}}\end{aligned}$$

Because the output of Game 3 is releasing no non-trivial information to the adversary.

$$Adv^{\mathsf{Game\ 3}}_{\hat{\Pi}^{\mathsf{rf},\mathsf{rf}'}\mathcal{A},G}(1^\lambda) = 0$$

$$Adv^{\mathsf{KR\text{-}ST}}_{\mathsf{C_{Chain}},\mathcal{A},G}(1^\lambda) \leq Adv^{\mathsf{Game\ 0}}_{\Pi^{\pi,\pi^{-1}},\mathcal{A},G}(1^\lambda) \leq \frac{\sigma(\sigma-1)}{2^{2\lambda-1}} + \frac{\sigma(\sigma-1)}{2^{\lambda}}$$

□

Theorem 9. *If π is the ideal permutation in Construction $\mathsf{C}_{\mathit{Chain}}$, then*

$$Adv^{\textit{IND-PRV}}_{\mathsf{C}_{\mathit{Chain}},\mathcal{A},G}(1^\lambda) \leq \frac{\sigma(\sigma-1)}{2^{2\lambda-1}} + \frac{\sigma(\sigma-1)}{2^\lambda}$$

Proof. We prove security by constructing successive games (or hybrids) and finding adversarial advantages between them.

Game 0: This game is identical to $\mathsf{IND\text{-}PRV}$ game where Construction $\mathsf{C}_{\mathsf{Chain}}$ is used. (see Figure 15).

Game 1: This **Game 1** is identical to **Game 0** except that we replace the 2λ-bit permutation π with 2λ-bit random function rf.

Using PRP/PRF Switching Lemma [BR06], for an adversary limited by σ queries to the permutation (or random function), the following equation can be obtained.

$$\begin{aligned}\left|Adv^{\mathsf{Game\ 0}}_{\Pi^{\pi,\pi^{-1}},\mathcal{A},G}(1^\lambda) - Adv^{\mathsf{Game\ 1}}_{\Pi^{\mathsf{rf},\pi^{-1}},\mathcal{A},G}(1^\lambda)\right| &= \left|Adv^{\mathsf{IND\text{-}PRV}}_{\Pi^{\pi,\pi^{-1}},\mathcal{A},G}(1^\lambda) - Adv^{\mathsf{IND\text{-}PRV}}_{\Pi^{\mathsf{rf},\pi^{-1}},\mathcal{A},G}(1^\lambda)\right| \\ &\leq Adv^{\mathsf{IND\text{-}PRV}}_{\pi,\mathsf{rf},\mathcal{A}}(1^\lambda,\sigma) \leq \frac{\sigma(\sigma-1)}{2^{2\lambda}}\end{aligned} \tag{4}$$

Game 2: This **Game 2** is identical to **Game 1** except that here we change 2λ-bit permutation π^{-1} by a 2λ-bit random function rf'.

Using PRP/PRF Switching Lemma [BR06], for an adversary limited by σ queries to the permutation (or random function), the following equation can be obtained.

$$\begin{aligned}\left|Adv^{\mathsf{Game\ 1}}_{\Pi^{\mathsf{rf},\pi^{-1}},\mathcal{A},G}(1^\lambda) - Adv^{\mathsf{Game\ 2}}_{\Pi^{\mathsf{rf},\mathsf{rf}'},\mathcal{A},G}(1^\lambda)\right| &= \left|Adv^{\mathsf{IND\text{-}PRV}}_{\Pi^{\mathsf{rf},\pi^{-1}},\mathcal{A},G}(1^\lambda) - Adv^{\mathsf{IND\text{-}PRV}}_{\Pi^{\mathsf{rf},\mathsf{rf}'},\mathcal{A},G}(1^\lambda)\right| \\ &\leq Adv^{\mathsf{IND\text{-}PRV}}_{\pi^{-1},\mathsf{rf}',\mathcal{A}}(1^\lambda,\sigma) \leq \frac{\sigma(\sigma-1)}{2^{2\lambda}}\end{aligned} \tag{5}$$

Game 3: This **Game 3** is identical to **Game 2** except that the game aborts whenever there is a collision in the lower λ bits of rf or of rf'. The event of collision in the lower λ bits of rf or rf' is called a *bad* event.

Using Code-Based Game Playing Technique [BR06], for an adversary limited by σ queries to the random functions, the following equation can be obtained.

$$\begin{aligned}\left|Adv^{\mathsf{Game\ 2}}_{\Pi^{\mathsf{rf},\mathsf{rf}'},\mathcal{A},G}(1^\lambda) - Adv^{\mathsf{Game\ 3}}_{\hat{\Pi}^{\mathsf{rf},\mathsf{rf}'},\mathcal{A},G}(1^\lambda)\right| &= \left|Adv^{\mathsf{IND\text{-}PRV}}_{\Pi^{\mathsf{rf},\mathsf{rf}'},\mathcal{A},G}(1^\lambda) - Adv^{\mathsf{IND\text{-}PRV}}_{\hat{\Pi}^{\mathsf{rf},\mathsf{rf}'},\mathcal{A},G}(1^\lambda)\right| \\ &\leq Pr[\mathcal{A}\text{ sets } Bad] \leq \frac{\sigma(\sigma-1)}{2^{\lambda}}\end{aligned} \tag{6}$$

Using Triangle Inequality [BR06] and the Equation 4, Equation 5 and Equation 6, the following equation can be obtained.

$$\begin{aligned}
\left| Adv^{\mathsf{Game\ 0}}_{\Pi^{\pi,\pi^{-1}}\mathcal{A},G}(1^{\lambda}) - Adv^{\mathsf{Game\ 3}}_{\hat{\Pi}^{\mathsf{rf},\mathsf{rf}'}\mathcal{A},G}(1^{\lambda}) \right| &= \left| Adv^{\mathsf{Game\ 0}}_{\Pi^{\pi,\pi^{-1}}\mathcal{A},G}(1^{\lambda}) - Adv^{\mathsf{Game\ 1}}_{\Pi^{\mathsf{rf},\pi^{-1}}\mathcal{A},G}(1^{\lambda}) \right. \\
&\quad + Adv^{\mathsf{Game\ 1}}_{\Pi^{\mathsf{rf},\pi^{-1}}\mathcal{A},G}(1^{\lambda}) - Adv^{\mathsf{Game\ 2}}_{\Pi^{\mathsf{rf},\mathsf{rf}'}\mathcal{A},G}(1^{\lambda}) \\
&\quad \left. + Adv^{\mathsf{Game\ 2}}_{\Pi^{\mathsf{rf},\mathsf{rf}'}\mathcal{A},G}(1^{\lambda}) - Adv^{\mathsf{Game\ 3}}_{\hat{\Pi}^{\mathsf{rf},\mathsf{rf}'}\mathcal{A},G}(1^{\lambda}) \right| \\
&\leq \left| Adv^{\mathsf{Game\ 0}}_{\Pi^{\pi,\pi^{-1}}\mathcal{A},G}(1^{\lambda}) - Adv^{\mathsf{Game\ 1}}_{\Pi^{\mathsf{rf},\pi^{-1}}\mathcal{A},G}(1^{\lambda}) \right| \\
&\quad + \left| Adv^{\mathsf{Game\ 1}}_{\Pi^{\mathsf{rf},\pi^{-1}}\mathcal{A},G}(1^{\lambda}) - Adv^{\mathsf{Game\ 2}}_{\Pi^{\mathsf{rf},\mathsf{rf}'}\mathcal{A},G}(1^{\lambda}) \right| \\
&\quad + \left| Adv^{\mathsf{Game\ 2}}_{\Pi^{\mathsf{rf},\mathsf{rf}'}\mathcal{A},G}(1^{\lambda}) - Adv^{\mathsf{Game\ 3}}_{\hat{\Pi}^{\mathsf{rf},\mathsf{rf}'}\mathcal{A},G}(1^{\lambda}) \right| \\
&\leq \frac{\sigma(\sigma-1)}{2^{2\lambda}} + \frac{\sigma(\sigma-1)}{2^{2\lambda}} + \frac{\sigma(\sigma-1)}{2^{\lambda}}
\end{aligned}$$

Because the output of $\mathsf{Game\ 3}$ is releasing no non-trivial information to the adversary.

$$Adv^{\mathsf{Game\ 3}}_{\hat{\Pi}^{\mathsf{rf},\mathsf{rf}'},\mathcal{A},G}(1^{\lambda}) = 0$$

$$Adv^{\mathsf{IND\text{-}PRV}}_{\mathsf{C_{Chain}},\mathcal{A},G}(1^{\lambda}) \leq Adv^{\mathsf{Game\ 0}}_{\Pi^{\pi,\pi^{-1}},\mathcal{A},G}(1^{\lambda}) \leq \frac{\sigma(\sigma-1)}{2^{2\lambda-1}} + \frac{\sigma(\sigma-1)}{2^{\lambda}}$$

□

Theorem 10. *If π is the ideal permutation in Construction $\mathsf{C_{Chain}}$, then*

$$Adv^{\mathsf{INT}}_{\mathsf{C_{Chain}},\mathcal{A},G}(1^{\lambda}) \leq \frac{\sigma^2}{2^{\lambda-1}} + \frac{\sigma^2}{2^{2\lambda-1}}$$

Proof. We replace the random permutation π used in the Construction $\mathsf{C_{Chain}}$ by the random function rf, to obtain the Construction $\mathsf{C'_{Chain}}$ shown in Figure 16.

The variables $L_0, L_1, \cdots, L_\sigma$ represent the lower λ-bit input in the permutation π or random function rf and are generated during the generation of C (see Figure 16). Here, σ is the maximum block-length of the ciphertext C.

The variables $L'_0, L'_1, \cdots, L'_{\sigma'}$ represent the lower λ-bit input in the permutation π or random function rf and are generated during the generation of C' (see Figure 16). Here, σ' is the maximum block-length of the ciphertext C'.

Suppose that we are using the construction $\mathsf{C'_{Chain}}$, we define the following events:
A is the event that at least one collision occurs in the values of $L_0, L_1, \cdots, L_\sigma$.
A_i is the event that $L_0, L_1, \cdots, L_i$ are all distinct, for $i \in [\sigma - 1]$.

So, we calculate the Probability of event A as follows:

$$\begin{aligned}
\Pr[A] &\leq \Pr[L_1 = L_0] + \Pr[L_2 = L_1 \vee L_2 = L_0 | A_1] \\
&\quad + \Pr[L_3 = L_2 \vee L_3 = L_1 \vee L_3 = L_0 | A_2] \\
&\quad + \cdots + \Pr[L_\sigma = L_{\sigma-1} \vee L_\sigma = L_{\sigma-2} \vee \cdots \vee L_\sigma = L_0 | A_{\sigma-1}] \\
&\leq \frac{1}{2^\lambda} + \frac{2}{2^\lambda} + \frac{3}{2^\lambda} + \cdots + \frac{\sigma}{2^\lambda} \\
&\leq \frac{\sigma^2}{2^\lambda}
\end{aligned}$$

Suppose that we are using the construction $\mathsf{C_{Chain}}$, we define the following events:

B is the event that $L'_j = L_i$ for some $i \in \{0, 1, \cdots, \sigma\}$ and $j \in \{0, 1, \cdots, \sigma'\}$.
C is the event that $L_0, L_1, \cdots, L_\sigma$ are all distinct.

$$
\begin{aligned}
Adv^{\mathsf{INT}}_{\mathsf{C_{Chain}},\mathcal{A},G}(1^\lambda) &\overset{def}{=} \left|\Pr[\mathsf{INT}^{\mathcal{A}}_{\Pi}(1^\lambda, G) = 1]\right| \\
&\leq \Pr[B] \\
&\leq \Pr[B|C] \cdot \Pr[C] + \Pr[B|\bar{C}] \cdot \Pr[\bar{C}] \\
&\leq \Pr[B|C] + \Pr[\bar{C}] \\
&\leq \left(\frac{\sigma^2}{2^\lambda} + Adv^{\mathsf{IND\text{-}PRV}}_{\pi,\mathsf{rf},\mathcal{A}}(1^\lambda, \sigma)\right) + \left(\Pr[A] + Adv^{\mathsf{IND\text{-}PRV}}_{\pi,\mathsf{rf},\mathcal{A}}(1^\lambda, \sigma)\right) \\
&\text{Using PRP/PRF Switching Lemma [BR06]} \\
&\leq \left(\frac{\sigma^2}{2^\lambda} + \frac{\sigma(\sigma-1)}{2^{2\lambda}}\right) + \left(\frac{\sigma^2}{2^\lambda} + \frac{\sigma(\sigma-1)}{2^{2\lambda}}\right) \\
&\leq \frac{\sigma^2}{2^{\lambda-1}} + \frac{\sigma^2}{2^{2\lambda-1}}
\end{aligned}
$$

□

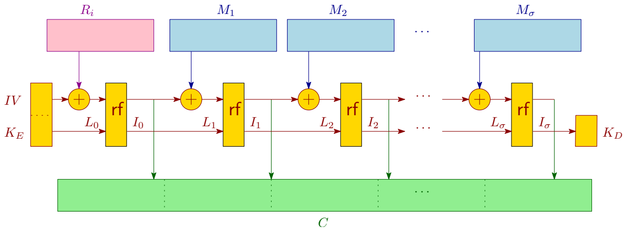


Figure 16: Construction $\mathsf{C'_{Chain}}$ obtained by replacing the random permutation π used in the Construction $\mathsf{C_{Chain}}$ by the random function rf.

5.1.4 $\mathsf{D_{Chain}}$: *KAS-AE-chain* based on *FP*

Functionalities based on *FP*

In this section, we are designing two functionalities – namely $\mathcal{G}_1^\pi$ and $\mathcal{G}_2^\pi$ – that are motivated by the mode of operation of hash function *FP* [PHG12] (Note that they are not identical). The algorithmic and diagrammatic descriptions of $\mathcal{G}_1^\pi$ and $\mathcal{G}_2^\pi$ are shown in Figure 17.

***KAS-AE-chain* scheme based on functionalities $\mathcal{G}_1^\pi$ and $\mathcal{G}_2^\pi$**

A $\mathsf{D_{Chain}}$ $\Pi = (\Pi.\mathcal{E}, \Pi.\mathcal{DER}, \Pi.\mathcal{D})$ is a *KAS-AE-chain* built from the functionalities $\mathcal{G}_1^\pi$ and $\mathcal{G}_2^\pi$ following the framework described in Figure 18.

Security of Construction $\mathsf{D_{Chain}}$

Theorem 11. *If π is the ideal permutation in Construction $\mathsf{D_{Chain}}$, then*

$$Adv^{\mathit{KR\text{-}ST}}_{\mathsf{D_{Chain}},\mathcal{A},G}(1^\lambda) \leq \frac{\sigma(\sigma-1)}{2^{2\lambda-1}} + \frac{\sigma(\sigma-1)}{2^\lambda}$$

Proof. This proof is similar to the proof of Construction $\mathsf{C_{Chain}}$ (see Theorem 8). □

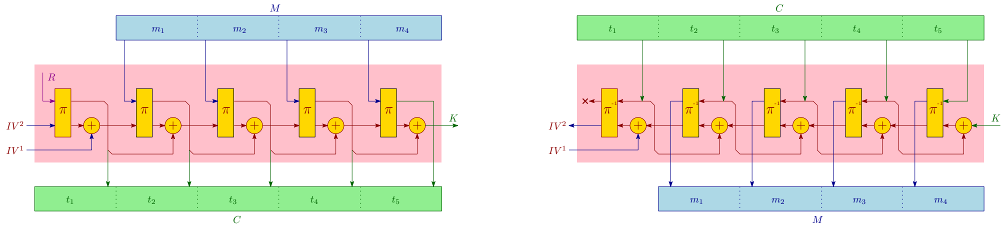
Theorem 12. *If π is the ideal permutation in Construction $\mathsf{D_{Chain}}$, then*

$$Adv^{\mathit{IND\text{-}PRV}}_{\mathsf{D_{Chain}},\mathcal{A},G}(1^\lambda) \leq \frac{\sigma(\sigma-1)}{2^{2\lambda-1}} + \frac{\sigma(\sigma-1)}{2^\lambda}$$

```
$\mathcal{G}_1^\pi(1^\lambda, M, IV^1, IV^2)$
$p := |M|/\lambda$, $s_0 := IV^2$, $t_0 := IV^1$;
$m_1||m_2||\cdots||m_p := M$;
$R \xleftarrow{\$} \{0,1\}^\lambda$, $m_0 := R$;
for $(j := 0, 1, \cdots, p)$
    $r_j := m_j$, $(r'_j, s'_j) := \pi(r_j, s_j)$;
    $t_{j+1} := r'_j$, $s_{j+1} := s'_j \oplus t_j$;
$C := t_1||t_2||\cdots t_{p+1}$, $K := s_{p+1}$;
return $(K, C)$;
```

```
$\mathcal{G}_2^\pi(1^\lambda, K, C, IV^1)$
$p := |C|/\lambda - 1$, $s_{p+1} := K$, $t_0 := IV^1$;
$t_1||t_2||\cdots t_{p+1} := C$;
for $(j := p, p-1, \cdots, 0)$
    $r'_j := t_{j+1}$, $s'_j := s_{j+1} \oplus t_j$;
    $(r_j, s_j) := \pi^{-1}(r'_j, s'_j)$;
    $m_j := r_j$;
$M := m_1||m_2||\cdots||m_p$, $IV^2 := s_0$;
return $(M, IV^2)$;
```

(a) Algorithmic description of functionalities $\mathcal{G}_1^\pi$ and $\mathcal{G}_2^\pi$.



(b) Diagrammatic description of functionality $\mathcal{G}_1^\pi$.

(c) Diagrammatic description of functionality $\mathcal{G}_2^\pi$.

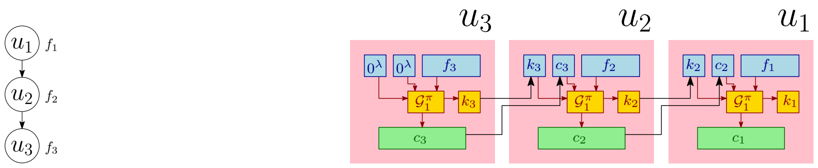
Figure 17: Algorithmic and diagrammatic descriptions of the functionalities $\mathcal{G}_1^\pi$ and $\mathcal{G}_2^\pi$ are shown in (a), (b) and (c); here, π is a 2λ-bit easy-to-invert permutation. Each wire in (b) and (c) represents λ bits. The function $\mathcal{G}_1^\pi$ takes as inputs parameter 1^λ, message M and two other values IV^1 and IV^2, and returns the decryption key K and the ciphertext C. Similarly, $\mathcal{G}_2^\pi$ takes as inputs parameter 1^λ, decryption key K, the ciphertext C and value IV^1, and outputs the message M and value IV^2. For the sake of simplicity, we assume that $|M|$ is a multiple of security parameter λ.

```
$\Pi.\mathcal{E}(params^{(\Pi)}, G, f)$
$(u_1, u_2, \cdots, u_m) :=$
        vertex_in_order$(G)$;
$(f_1 \circ f_2 \circ \cdots \circ f_m) := f$;
$IV^1 := IV^2 := 0^\lambda$;
for $(i := m, m-1, \cdots, 1)$
  $(k_i, c_i) := \mathcal{G}_1^\pi(1^\lambda, f_i, IV^1, IV^2)$;
  $IV^1 := c_i[last\_block]$;
  $IV^2 := k_i$;
$pub := (c_1 \circ c_2 \circ \cdots \circ c_m)$;
$S := k := (k_1 \circ k_2 \circ \cdots \circ k_m)$;
return $(S, k, pub)$;
```

```
$\Pi.\mathcal{DER}(params^{(\Pi)}, G, u, v, S_u, pub)$
$u_i := u$, $u_j := v$, $k_i := S_u$;
if $(u < v)$, then return $\perp$;
if $u = v$, then return $k_j := k_i$;
$(c_1 \circ c_2 \circ \cdots \circ c_m) := pub$;
for $(\ell := i, i+1, \cdots, j-1)$
  $IV^1 := c_{\ell+1}[last\_block]$;
  $(f_\ell, k_{\ell+1}) := \mathcal{G}_2^\pi(1^\lambda, k_\ell, c_\ell, IV^1)$;
return $k_j$;
```

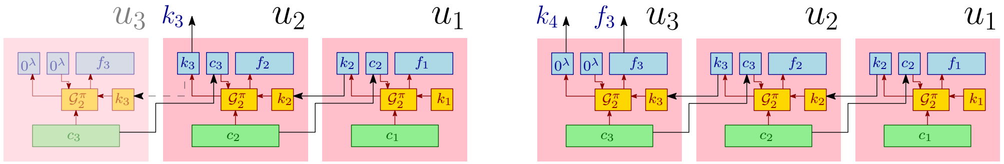
```
$\Pi.\mathcal{D}(params^{(\Pi)}, G, u, v, S_u, pub)$
$u_j := v$;
$(c_1 \circ c_2 \circ \cdots \circ c_m) := pub$;
$k_j := \Pi.\mathcal{DER}(params^{(\Pi)}, G, u, v, S_u, pub)$;
If $k_j = \perp$, then return $\perp$;
for $(\ell := j, j+1, \cdots, m)$
  If $\ell = m$ then $IV^1 := 0^\lambda$;
  Else
    $IV^1 := c_{\ell+1}[last\_block]$;
  $(f_\ell, k_{\ell+1}) := \mathcal{G}_2^\pi(1^\lambda, k_\ell, c_\ell, IV^1)$;
If $k_{m+1} = 0^\lambda$, then return $f_v$;
Else return $\perp$;
```

(a) Algorithmic description of building $\mathsf{D}_{\mathsf{Chain}}$ Π using the functionalities $\mathcal{G}_1^\pi$ and $\mathcal{G}_2^\pi$. For the pictorial description with an example, see 18(b)–18(e).



(b) The access graph G with 3 nodes u_1, u_2, u_3 and their corresponding files f_1, f_2, f_3.

(c) Pictorial description of $\Pi.\mathcal{E}(params^{(\Pi)}, G, f)$, where $f = (f_1, f_2, f_3)$ and G is shown in 18(b).



(d) Pictorial description of $\Pi.\mathcal{DER}(params^{(\Pi)}, G, u_1, u_3, S_1, pub)$.

(e) Pictorial description of $\Pi.\mathcal{D}(params^{(\Pi)}, G, u_1, u_3, S_1, pub)$.

Figure 18: Building $\mathsf{D}_{\mathsf{Chain}}$.

Proof. This proof is similar to the proof of Construction $\mathsf{C}_{\mathsf{Chain}}$ (see Theorem 9). ☐

Theorem 13. *If π is the ideal permutation in Construction $\mathsf{D}_{\mathsf{Chain}}$, then*

$$Adv^{\mathsf{INT}}_{\mathsf{D}_{\mathsf{Chain}},\mathcal{A},G}(1^{\lambda}) \leq \frac{\sigma^2}{2^{\lambda-1}} + \frac{\sigma^2}{2^{2\lambda-1}}$$

Proof. This proof is similar to the proof of Construction $\mathsf{C}_{\mathsf{Chain}}$ (see Theorem 10). ☐

5.2 *Modified Chain Partition* using *KAS-AE-chains*

Modified chain partition algorithm can be viewed as an adaptation of the *chain partition* algorithm which is used for constructing *KAS* schemes as described in Subsubsection 2.3.3.

Let $(V, \leq)$ and $G = (V, E)$ be, respectively, a poset and the access graph corresponding to it. Let λ be the security parameter. A *chain partition* of V into w *chains* $C_1, C_2, \cdots, C_w$ is selected in such a way that C_i contains nodes (or classes) $u^i_1, u^i_2, \cdots, u^i_{l_i}$, where $l_i = |C_i|$, $u^i_{j+1} < u^i_j$ for $1 \leq j < l_i$. We set $l_{max} = \max_{i\in[w]} l_i$. Let $\pi = (\pi.\mathcal{E}, \pi.\mathcal{DER}, \pi.\mathcal{D})$ be a *KAS-AE-chain* scheme of length at most l_{max}.

Suppose $\lambda \in \mathbb{N}$ is the security parameter. A *modified chain partition* algorithm $\Pi = (\Pi.\mathcal{E}, \Pi.\mathcal{DER}, \Pi.\mathcal{D})$ is a three tuple of algorithms over a setup algorithm $\Pi.\mathsf{Setup}$. Π satisfies the following conditions.

1. The PPT setup algorithm $\Pi.\mathsf{Setup}(1^{\lambda})$ outputs the parameter $params^{(\Pi)}$, a set of access graphs $\Gamma^{(\Pi)}$ and the sets $\mathcal{K}^{(\Pi)}$ and $\mathcal{M}^{(\Pi)}$, denoting the *key* and *message spaces* respectively.

   Here, $\mathcal{K}^{(\Pi)} = \{0,1\}^{p(\lambda)}$ and $\mathcal{M}^{(\Pi)} = \{0,1\}^*$, where $p(\cdot)$ is some polynomial.

2. The PPT encryption algorithm $\Pi.\mathcal{E}$ takes as inputs the parameter $params^{(\Pi)}$, the access graph $G = (V, E) \in \Gamma^{(\Pi)}$, the sequence of files $f = (f_u)_{u\in V}$ and the *KAS-AE-chain* scheme π, and return a three-tuple $(S, k, pub) := \Pi.\mathcal{E}(params^{(\Pi)}, G, f, \pi)$, where $S = (S_u)_{u\in V}$, $k = (k_u)_{u\in V}$ and pub are the sequence of private information, keys and public values respectively.

   Note that $f_u \in \mathcal{M}^{(\Pi)}$, $k_u \in \mathcal{K}^{(\Pi)}$, $S_u \in \{0,1\}^*$ and $pub \in \{0,1\}^*$, for all $u \in V$.

3. The key-derive algorithm $\Pi.\mathcal{DER}$ is a deterministic PT algorithm such that $k_{u^g_h} := \Pi.\mathcal{DER}(params^{(\Pi)}, G, u^i_j, u^g_h, S_{u^i_j}, pub_g, \pi)$. Here: $u^g_h \leq u^i_j$ are two nodes of the access graph G; $S_{u^i_j}$ is u^i_j's private information; pub_g is the public information; π is the *KAS-AE-chain* scheme; and $k_{u^g_h}$ is u^g_h's decryption key.

   Note that $S_{u^i_j} \in \{0,1\}^*$, $pub_g \in \{0,1\}^*$ and $k_{u^g_h} \in \mathcal{K}^{(\Pi)} \cup \perp$.

4. The decryption algorithm $\Pi.\mathcal{D}$ is a deterministic PT algorithm such that $f_{u^g_h} := \Pi.\mathcal{D}(params^{(\Pi)}, G, u^i_j, u^g_h, S_{u^i_j}, pub_g, \pi)$. Here: $u^g_h \leq u^i_j$ are two nodes of the access graph G; $S_{u^i_j}$ is u^i_j's private information; pub_g is the public information; π is the *KAS-AE-chain* scheme; and $f_{u^g_h}$ is u^g_h's decrypted file.

   Note that $S_{u^i_j} \in \{0,1\}^*$, $pub_g \in \{0,1\}^*$ and $f_{u^g_h} \in \mathcal{M}^{(\Pi)} \cup \perp$.

Detailed internal workings of the *modified chain partition* algorithm are given in Figure 19. The subroutines used by the algorithm are described in Subsubsection 2.2.8. These subroutines are identical to the subroutines used in [FPP13], but we reproduce them for the sake of completeness.

By instantiating π with the *KAS-AE-chain* schemes $\mathsf{A}_{\mathsf{Chain}}$, $\mathsf{B}_{\mathsf{Chain}}$, $\mathsf{C}_{\mathsf{Chain}}$ and $\mathsf{D}_{\mathsf{Chain}}$, in the *modified chain partition* algorithm, we construct the *KAS-AE* schemes Construction A, B, C and D respectively (see Figure 19).

```
$\Pi.\mathcal{E}(params^{(\Pi)}, G, f, \pi)$
$(w, C[\,]) :=$ partition$(G)$;
for $(i := 1, 2, \cdots, w)$
    $f^i := (f_u)_{u \in C_i}$;
    $(T^i, k^i, pub^i) := \pi.\mathcal{E}(params^{(\pi)}, C_i, f^i)$;
for $u \in V$
    $(\hat{u}_1, \hat{u}_2, \cdots, \hat{u}_w) :=$ max_isect_chs$(u, G)$;
    $S_u := T_{\hat{u}_1} \cup T_{\hat{u}_2} \cup \cdots \cup T_{\hat{u}_w}$;
$S := (S_u)_{u \in V}$, $k := (k_u)_{u \in V}$;
$pub := (pub_i)_{i \in [w]}$;
return $(S, k, pub)$;

$\Pi.\mathcal{DER}(params^{(\Pi)}, G, u^i_j, u^g_h, S_{u^i_j}, pub_g, \pi)$
$\hat{u}_g :=$ max_isect$(u^i_j, C_g)$;
$T_{\hat{u}_g} :=$ ext_secret$(S_{u^i_j}, \hat{u}_g)$;
$k_{u^g_h} := \pi.\mathcal{DER}(params^{(\pi)}, C_g, \hat{u}_g, u^g_h, T_{\hat{u}_g}, pub_g)$;
return $k_{u^g_h}$;

$\Pi.\mathcal{D}(params^{(\Pi)}, G, u^i_j, u^g_h, S_{u^i_j}, pub_g, \pi)$
$k_{u^g_h} := \Pi.\mathcal{DER}(params^{(\Pi)}, G, u^i_j, u^g_h, S_{u^i_j}, pub_g, \pi)$;
$f_{u^g_h} := \pi.\mathcal{D}(params^{(\pi)}, C_g, u^g_h, u^g_h, k_{u^g_h}.pub_g)$;
return $f_{u^g_h}$;
```

Figure 19: Algorithmic description of *modified chain partition* algorithm to build a *KAS-AE* scheme $\Pi = (\Pi.\mathcal{E}, \Pi.\mathcal{DER}, \Pi.\mathcal{D})$ using the *KAS-AE-chain* scheme $\pi = (\pi.\mathcal{E}, \pi.\mathcal{DER}, \pi.\mathcal{D})$.

5.3 Security of *KAS-AE* built using *KAS-AE-chain* and *modified chain partition* algorithm

Proof sketch. We can prove the KR-ST, IND-PRV and INT security of this construction by using reduction as used by Freire *et al.* [FPP13]. So, we can show that if the adversary $\mathcal{A}$ can break the KR-ST (or IND-PRV or INT) security of *KAS-AE* secure built using *KAS-AE-chain* and *modified chain partition*, then an adversary $\mathcal{B}$, using $\mathcal{A}$, can break the KR-ST (or IND-PRV or INT) security of *KAS-AE-chain* scheme. By using the contrapositive argument, this would show that if the underlying *KAS-AE-chain* scheme is secure, so is the *KAS-AE* scheme.

6 Building *KAS-AE* from *MLE*

In this section, we describe a *KAS-AE* scheme built using *MLE* scheme referred to as Construction 1. This scheme is more efficient than the *KAS-AE* constructions described in Section 4 and Section 5. This scheme exploits the self-sufficiency of *MLE* schemes to provide the integrity along with the confidentiality. This results in the huge reduction in memory of the *private information* that has to be stored securely by the members of each security class, especially in the cases when the *width* of the access graph (as described in Subsubsection 2.2.1) is huge.

6.1 Construction 1: A *KAS-AE* scheme based on *MLE*

The pseudo-code for building a *KAS-AE* scheme $\Pi = (\Pi.\mathcal{E}, \Pi.\mathcal{DER}, \Pi.\mathcal{D})$ from the functionalities $\Psi.\mathcal{E}$ and $\Psi.\mathcal{D}$ of an *MLE* scheme $\Psi = (\Psi.\mathcal{E}, \Psi.\mathcal{D})$ (described in Subsubsection 2.2.5) is given in Figure 20, which also contains the diagrammatic representation of the pseudocode. Below we give the full description of the *KAS-AE* scheme Π.

- $\Pi.\mathcal{E}(params^{(\Pi)}, G, f)$ is a randomised algorithm. This encryption function is designed in such a way that any node u is able to decrypt the files of its successors. In order to do that, for each node u, we encrypt the file f_u as well as the decryption keys of the children of u. Therefore, the algorithm: assigns level to each node as $level[\,]$ and calculates maximum-depth of the tree h, which are returned by the function height(G); and starts by encrypting the files at level h, followed by the encryption of the files at level $h-1$, and so on, until the root node is reached. For each node u, the following operations are executed: the function ch_seq(u, G)

returns the sequence of children $(u_{j_1}, u_{j_2}, \cdots, u_{j_d})$ of u (in ascending order); then a λ-bit random number R is generated; then f'_u is obtained by prepending R and the decryption keys $k_{u_{j_1}}, k_{u_{j_2}}, \cdots, k_{u_{j_d}}$ – which have been already generated in the previous iterations – to the file f_u; and finally, $(k_u, c_u, t_u) := \Psi.\mathcal{E}(params^{(\Psi)}, f'_u)$ is computed, where k_u, c_u and t_u are the decryption key, ciphertext and tag. The vectors S, k and pub are computed as $pub := (c_u||t_u)_{u\in V}$, and $S := k := (k_u)_{u\in V}$. Pictorial description of this algorithm on an access graph G is given in 20(c).

- $\Pi.\mathcal{DER}(params^{(\Pi)}, G, u, v, S_u, pub)$ is a deterministic algorithm in which a node u computes the decryption key of a successor node v. The node u uses its private information S_u and the public information of the system pub. First, the function $\mathsf{path}(G, u, v)$ returns a sequence of nodes $(u, u_{i_1}, u_{i_2}, \cdots, u_{i_\ell}, u_{i_{\ell+1}} = v)$ representing the path from u to v. S_u contains the decryption key k_u, and therefore can be used to start the decryption procedure. For all the successive nodes $w = u, u_{i_1}, u_{i_2}, \cdots, u_{i_\ell}, v$ the following operations are executed: the ciphertext c_w and the tag t_w is extracted; $f'_w := \Psi.\mathcal{D}(params^{(\Psi)}, k_w, c_w, t_w)$ is computed; the function $\mathsf{ch\_seq}(w, G)$ returns the sequence of children $(u_{j_1}, u_{j_2}, \cdots, u_{j_d})$ of w (in ascending order); the values of $R, k_{u_{j_1}}, k_{u_{j_2}}, \cdots, k_{u_{j_d}}$ and f_w are extracted from f'_w, where R is the random number used during the encryption; and the next node in the path is searched in the sequence $\tilde{w}$, and the key corresponding to it is extracted, before the next iteration begins. Pictorial description of this algorithm on an access graph G is given in 20(d).
- $\Pi.\mathcal{D}(params^{(\Pi)}, G, u, v, S_u, pub)$ is a deterministic algorithm that allows u to decrypt the file stored by its successor v. Like before, u uses the private information S_u and the public information of the system pub. In the first step, the decryption key $k_v := \Pi.\mathcal{DER}(params^{(\Pi)}, G, u, v, S_u, pub)$ is computed. Then, the ciphertext c_v and tag t_v are extracted from pub using the function $\mathsf{ext\_cipher}$. After that, the file $f'_v := \Psi.\mathcal{D}(params^{(\Psi)}, k_v, c_v, t_v)$ is computed, and the random number and the keys of the children of v are removed from the head of file f'_v to get the original file f_v. Pictorial description of this algorithm on an access graph G is given in 20(e).

6.2 Security of Construction 1

Theorem 14. *If the underlying MLE is* KR-CDA *(or* PRV-CDA *or* TC*) secure, then the KAS-AE scheme* Construction 1 *is also* KR-ST *(or* IND-PRV *or* INT*) secure against static adversaries.*

Proof. The proof of this is identical to the KR-ST (or IND-PRV or INT) security proof of *KAS-AE-chain* construction $\mathsf{B_{Chain}}$ in the Subsubsection 5.1.2. ☐

7 Building *KAS-AE* by Tweaking *APE* and *FP*

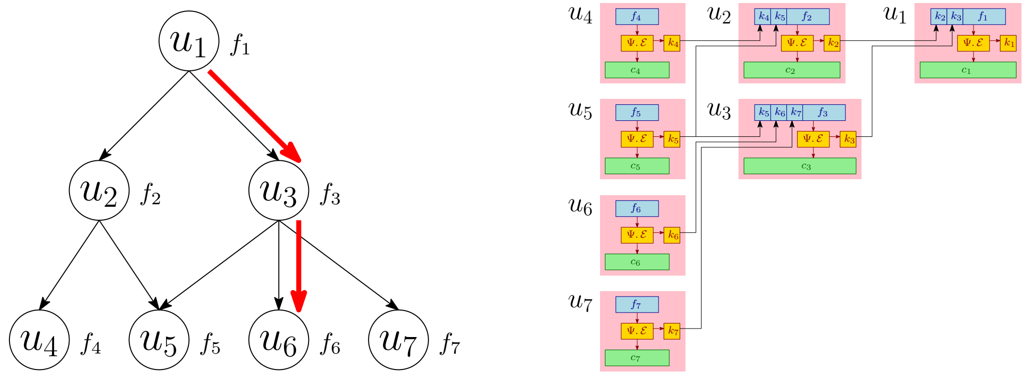
So far we have constructed the *KAS-AE* schemes using the existing schemes used as black boxes. Here we take a focused look on generating the *KAS-AE* schemes from scratch and we describe two *KAS-AE* schemes, namely, Construction 2 and Construction 3. These two schemes are much more efficient than all the *KAS-AE* constructions described in the paper. They exploit the very unique property of *reverse decryption* of *APE* authenticated encryption and *FP* hash mode of operation to integrate the key and message, and provide authenticated encryption. This trick has been used earlier by Kandele and Paul to come up with *FMLE* schemes [KP18]. This results in the huge reduction in the memory requirement for the *private information* – that has to be stored securely by the members of each security class – and the *ciphertext expansion* that is stored in the public storage, especially in the cases when the *width* of the access graph (as described in Subsubsection 2.2.1) is huge.

```
$\Pi.\mathcal{E}(params^{(\Pi)}, G, f)$
$(level[\,], h) :=$ height$(G)$;
while $h \geq 0$
  $V_h :=$ nodes_at_level$(V, level[\,], h)$;
  For all $u \in V_h$
    $\tilde{u} \overset{def}{=} (u_{j_1}, u_{j_2}, \cdots, u_{j_d}) :=$ ch_seq$(u, G)$;
    $R \overset{\$}{\leftarrow} \{0,1\}^{\lambda}$;
    If $\tilde{u} =$ NULL
      $f'_u := R||f_u$;
    Else
      $f'_u := R||k_{u_{j_1}}||k_{u_{j_2}}||\cdots \cdots||k_{u_{j_d}}||f_u$;
    $(k_u, c_u, t_u) := \Psi.\mathcal{E}(params^{(\Psi)}, f'_u)$;
  $h := h - 1$;
$pub := (c_u||t_u)_{u \in V}$;
$S := k := (k_u)_{u \in V}$;
return $(S, k, pub)$;
```

```
$\Pi.\mathcal{DER}(params^{(\Pi)}, G, u, v, S_u, pub)$
$(u, u_{i_1}, u_{i_2}, \cdots, u_{i_\ell}, u_{i_{\ell+1}} = v) :=$ path$(G, u, v)$;
$p := 1,\ w := u,\ k_w := S_u$;
while $w \neq v$
  $c_w||t_w :=$ ext_cipher$(pub, w)$;
  $f'_w := \Psi.\mathcal{D}(params^{(\Psi)}, k_w, c_w, t_w)$;
  If $f'_w = \perp$, then return $\perp$;
  $\tilde{w} \overset{def}{=} (u_{j_1}, u_{j_2}, \cdots, u_{j_d}) :=$ ch_seq$(w, G)$;
  $R||k_{u_{j_1}}||k_{u_{j_2}}||\cdots||k_{u_{j_d}}||f_w := f'_w$;
  Find $u_{j_q} \in \tilde{w}$, s.t. $j_q = i_p$;
  $k_w := k_{u_{j_q}},\ w := u_{j_q}$;
  $p := p + 1$;
return $k_w$;
```

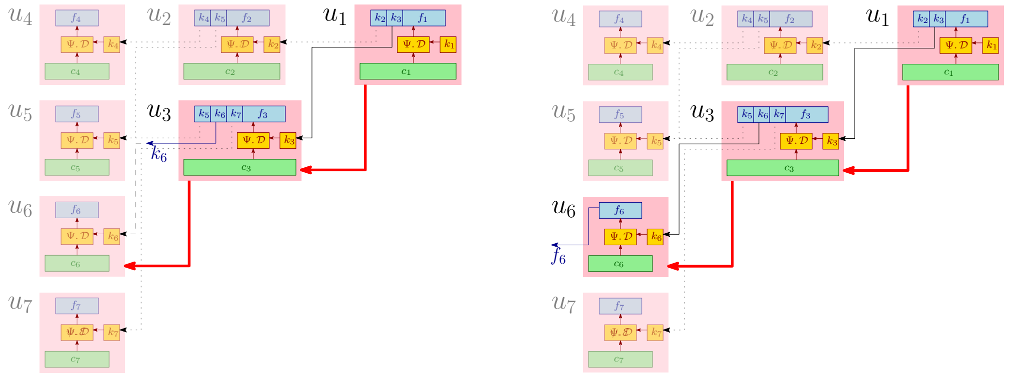
```
$\Pi.\mathcal{D}(params^{(\Pi)}, G, u, v, S_u, pub)$
$k_v := \Pi.\mathcal{DER}(params^{(\Pi)}, G, u, v, S_u, pub)$;
$c_v||t_v :=$ ext_cipher$(pub, v)$;
$f'_v := \Psi.\mathcal{D}(params^{(\Psi)}, k_v, c_v, t_v)$;
If $f'_v = \perp$, then return $\perp$;
$\tilde{v} \overset{def}{=} (u_{j_1}, u_{j_2}, \cdots, u_{j_d}) :=$ ch_seq$(v, G)$;
If $\tilde{v} =$ NULL,
  $R||f_v := f'_v$;
Else
  $R||k_{u_{j_1}}||k_{u_{j_2}}||\cdots||k_{u_{j_d}}||f_v := f'_v$;
return $f_v$;
```

(a) Algorithmic description of building *Construction 1* Π using the *MLE* scheme Ψ. For the pictorial description with an example, see 20(b)–20(e).



(b) The access graph G with 7 nodes $u_1, u_2, \cdots, u_7$ and their corresponding files $f_1, f_2, \cdots, f_7$.

(c) Pictorial description of $\Pi.\mathcal{E}(params^{(\Pi)}, G, f)$, where $f = (f_1, f_2, \cdots, f_7)$ and G is shown in 20(b).



(d) Pictorial description of $\Pi.\mathcal{DER}(params^{(\Pi)}, G, u_1, u_6, S_1, pub)$ using the path (u_1, u_3, u_6) which is shown in red line in graph G in 20(b).

(e) Pictorial description of $\Pi.\mathcal{D}(params^{(\Pi)}, G, u_1, u_6, S_1, pub)$ using the path (u_1, u_3, u_6) which is shown in red line in graph G in 20(b).

Figure 20: Building *Construction 1*.

7.1 Construction 2: *KAS-AE* from *APE*

We design two functionalities $\mathcal{F}_1^\pi$ and $\mathcal{F}_2^\pi$ from the APE authenticated encryption. The details of these functionalities are described in Subsubsection 5.1.3.

7.1.1 A *KAS-AE* scheme based on functionalities $\mathcal{F}_1^\pi$ and $\mathcal{F}_2^\pi$

The pseudo-code for building a $KAS\text{-}AE$ scheme $\Pi = (\Pi.\mathcal{E}, \Pi.\mathcal{DER}, \Pi.\mathcal{D})$ from the functionalities $\mathcal{F}_1^\pi$ and $\mathcal{F}_2^\pi$ (as described in Subsubsection 5.1.3) is given in Figure 21, which also contains the diagrammatic representation of the pseudocode. Below we give the full description of the $KAS\text{-}AE$ scheme Π.

- $\Pi.\mathcal{E}(params^{(\Pi)}, G, f)$ is a randomised algorithm. This encryption function is designed in such a way that any node u is able to decrypt the files of its successors. In order to do that, for each node u, we encrypt the file f_u as well as the decryption keys of the children of u, such that, on decrypting the ciphertext corresponding to u, the decryption keys of all its children are revealed. The algorithm starts by assigning level to each node as $level[\,]$ and calculating maximum-depth of tree h, which are returned by the function $\mathsf{height}(G)$, and then encrypts the files at level h, followed by the encryption of the files at level $h-1$, and so on, until the root node is reached. For each node u, the following operations are executed: the function $\mathsf{ch\_seq}(u, G)$ returns the sequence of children $(u_{j_1}, u_{j_2}, \cdots, u_{j_d})$ of u (in ascending order); then the key of the first child $k_{u_{j_1}}$ is assigned to IV^2, the last λ-bit block of ciphertext $c_{u_{j_1}}$ is assigned to IV^1, and the keys $k_{u_{j_2}}, k_{u_{j_3}}, \cdots, k_{u_{j_d}}$ – which have been already generated in the previous iterations – are prepended to the file f_u to obtain f'_u; and then the decryption key k_u and ciphertext c_u is computed as $(k_u, c_u) := \mathcal{F}_1^\pi(1^\lambda, f'_u, IV^1, IV^2)$. For the leaf nodes, the value of IV^1 and IV^2 are 0^λ. The vectors S, k and pub are computed as $pub := (c_u)_{u\in V}$, and $S := k := (k_u)_{u\in V}$. Pictorial description of this algorithm on an access graph G is given in 21(c).

- $\Pi.\mathcal{DER}(params^{(\Pi)}, G, u, v, S_u, pub)$ is a deterministic algorithm in which a node u computes the decryption key of a successor node v. The node u uses its private information S_u and the public information of the system pub. First, the function $\mathsf{path}(G, u, v)$ returns a sequence of nodes $(u, u_{i_1}, u_{i_2}, \cdots, u_{i_\ell}, u_{i_{\ell+1}} = v)$ representing the path from u to v. S_u contains the decryption key k_u, and therefore can be used to start the decryption procedure. For all the successive nodes $w = u, u_{i_1}, u_{i_2}, \cdots, u_{i_\ell}, v$ the following operations are executed: the ciphertext c_w is extracted; $(f'_w, IV^2) := \mathcal{F}_2^\pi(1^\lambda, k_w, c_w)$ is computed; the function $\mathsf{ch\_seq}(w, G)$ returns the sequence of children $(u_{j_1}, u_{j_2}, \cdots, u_{j_d})$ of w (in ascending order); the key $k_{u_{j_1}}$ is assigned the value of IV^2; the values of $k_{u_{j_2}}, k_{u_{j_3}}, \cdots, k_{u_{j_d}}$ and f_w are extracted from f'_w; and the next node in the path is searched in the sequence $\tilde{w}$, and the key corresponding to it is extracted, before the next iteration begins. Pictorial description of this algorithm on an access graph G is given in 21(d).

- $\Pi.\mathcal{D}(params^{(\Pi)}, G, u, v, S_u, pub)$ is a deterministic algorithm that facilitates the node u to decrypt the file of its successor v. As earlier, the node u uses the private information S_u and the public information of the system pub. In the first step, the decryption key $k_v := \Pi.\mathcal{DER}(params^{(\Pi)}, G, u, v, S_u, pub)$ is computed. Then, the ciphertext c_v is extracted from pub using the function $\mathsf{ext\_cipher}$. After that, the file f'_v and value IV^2 are computed $(f'_v, IV^2) := \mathcal{F}_2^\pi(1^\lambda, k_v, c_v)$ and the keys of children of v are removed from the head of file f'_v to obtain the original file f_v. To verify the authentication of the file, the first child w of each node starting from v performs the following operations: the key $k_w := IV^2$ is computed, ciphertext c_w is extracted and decrypted to find $(f'_w, IV^2) := \mathcal{F}_2^\pi(1^\lambda, k_v, c_v)$, where IV^2 acts as the key of the first child for the execution of next iteration. The the value of IV^2 should be 0^λ for the leaf node whose ciphertext is decrypted in the last iteration. If this condition is satisfied, the file f_v is returned, otherwise $\perp$ is returned.

7.1.2 Security of Construction 2

Theorem 15. *If the underlying APE is KR-ST (or IND-PRV or INT) secure, then the KAS-AE scheme Construction 2 is also KR-ST (or IND-PRV or INT) secure against static adversaries.*

Proof. The proof of this is identical to the KR-ST (or IND-PRV or INT) security proof of *KAS-AE-chain* construction $\mathsf{C}_{\mathsf{Chain}}$ in the Subsubsection 5.1.3. ☐

7.2 Construction 3: *KAS-AE* built from FP

We design two functionalities $\mathcal{G}_1^\pi$ and $\mathcal{G}_2^\pi$ from the mode of operation of hash function *FP*. The details of these functionalities are described in Subsubsection 5.1.4.

7.2.1 A *KAS-AE* scheme based on functionalities $\mathcal{G}_1^\pi$ and $\mathcal{G}_2^\pi$

The pseudo-code for building a *KAS-AE* scheme $\Pi = (\Pi.\mathcal{E}, \Pi.\mathcal{DER}, \Pi.\mathcal{D})$ from the functionalities $\mathcal{G}_1^\pi$ and $\mathcal{G}_2^\pi$ is given in Figure 22, which also contains the diagrammatic representation of the pseudocode. Below we give the full description of the *KAS-AE* scheme Π.

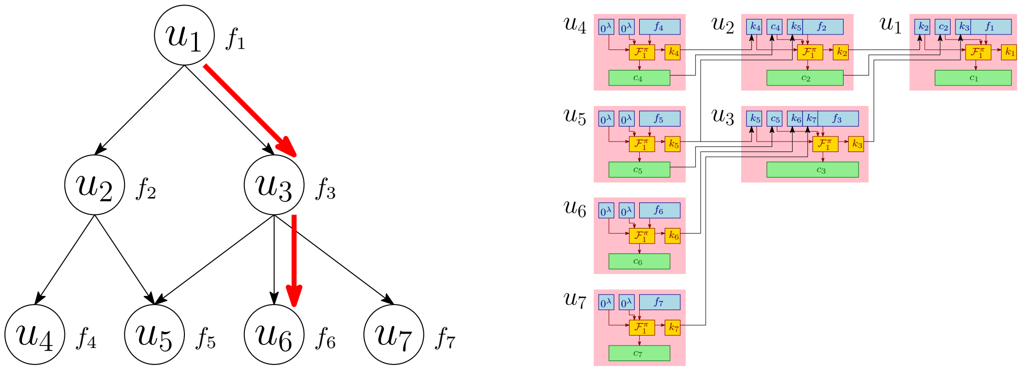
- $\Pi.\mathcal{E}(params^{(\Pi)}, G, f)$ is a randomised algorithm. This encryption function is designed in such a way that any node u is able to decrypt the files of its successors. In order to do that, for each node u, we encrypt the file f_u as well as the decryption keys of the children of u, such that, on decrypting the ciphertext corresponding to u, the decryption keys of all its children are revealed. The algorithm starts by assigning level to each node as *level* and calculating maximum-depth of tree h returned by $\mathsf{height}(G)$, and then encrypts the files at level h, after that the files at level $h-1$, and so on, until the root node is reached. For each node u, the following operations are executed: the function $\mathsf{ch\_seq}(u, G)$ returns the sequence of children $(u_{j_1}, u_{j_2}, \cdots, u_{j_d})$ of u (in ascending order); then the key of the first child $k_{u_{j_1}}$ is assigned to IV^2, the last λ-bit block of ciphertext $c_{u_{j_1}}$ is assigned to IV^1, and the keys $k_{u_{j_2}}, k_{u_{j_3}}, \cdots, k_{u_{j_d}}$ – which have been already generated in the previous iterations – are prepended to the file f_u to obtain f'_u; and then the ciphertext c_u and decryption key k_u is computed $(k_u, c_u) = \mathcal{G}_1^\pi(1^\lambda, f'_u, IV^1, IV^2)$. For the leaf nodes, the value of IV^1 and IV^2 are 0^λ. The vectors S, k and *pub* are computed as $pub := (c_u)_{u \in V}$, and $S := k := (k_u)_{u \in V}$. Pictorial description of this algorithm on an access graph G is given in 22(c).
- $\Pi.\mathcal{DER}(params^{(\Pi)}, G, u, v, S_u, pub)$ is a deterministic algorithm in which a node u computes the decryption key of a successor node v. The node u uses its private information S_u and the public information of the system *pub*. First, the function $\mathsf{path}(G, u, v)$ returns a sequence of nodes $(u, u_{i_1}, u_{i_2}, \cdots, u_{i_\ell}, v)$ representing the path from u to v. S_u contains the decryption key k_u, and therefore can be used to start the decryption procedure. For all the successive nodes $w = u, u_{i_1}, u_{i_2}, \cdots, u_{i_\ell}, v$ the following operations are executed: the ciphertext c_w is extracted; the function $\mathsf{ch\_seq}(w, G)$ returns the sequence of children $(u_{j_1}, u_{j_2}, \cdots, u_{j_d})$ of w (in ascending order); the value of IV^1 is computed as the last λ-bit block of ciphertext $c_{u_{j_1}}$; $(f'_w, IV^2) = \mathcal{G}_2^\pi(1^\lambda, k_w, c_w, IV^1)$ is computed; the key $k_{u_{j_1}}$ is assigned the value of

```
$\Pi.\mathcal{E}(params^{(\Pi)}, G, f)$
$(level[\,], h) := \mathsf{height}(G)$;
while $h \geq 0$
  $V_h := \mathsf{nodes\_at\_level}(V, level[\,], h)$;
  For all $u \in V_h$
    $\tilde{u} \overset{def}{=} (u_{j_1}, u_{j_2}, \cdots, u_{j_d}) := \mathsf{ch\_seq}(u, G)$;
    If $\tilde{u} = \mathsf{NULL}$
      $IV^1 := IV^2 := 0^\lambda$;
      $f'_u := f_u$;
    Else
      $IV^1 := c_{u_{j_1}}[last\_block]$;
      $IV^2 := k_{u_{j_1}}$;
      $f'_u := k_{u_{j_2}} || k_{u_{j_3}} || \cdots \cdots || k_{u_{j_d}} || f_u$;
    $(k_u, c_u) := \mathcal{F}_1^\pi(1^\lambda, f'_u, IV^1, IV^2)$;
  $h := h - 1$;
$pub := (c_u)_{u \in V}$;
$S := k := (k_u)_{u \in V}$;
return $(S, k, pub)$;
```

```
$\Pi.\mathcal{DER}(params^{(\Pi)}, G, u, v, S_u, pub)$
$(u, u_{i_1}, u_{i_2}, \cdots, u_{i_\ell}, u_{i_{\ell+1}} = v) := \mathsf{path}(G, u, v)$;
$p := 1,\ w := u,\ k_w := S_u$;
while $(w \neq v)$
  $c_w := \mathsf{ext\_cipher}(pub, w)$;
  $(f'_w, IV^2) := \mathcal{F}_2^\pi(1^\lambda, k_w, c_w)$;
  $\tilde{w} \overset{def}{=} (u_{j_1}, u_{j_2}, \cdots, u_{j_d}) := \mathsf{ch\_seq}(w, G)$;
  $k_{u_{j_1}} := IV^2$;
  $k_{u_{j_2}} || k_{u_{j_3}} || \cdots || k_{u_{j_d}} || f_w := f'_w$;
  Find $u_{j_q} \in \tilde{w}$, s.t. $j_q = i_p$;
  $k_w := k_{u_{j_q}},\ w := u_{j_q}$;
  $p := p + 1$;
return $k_w$;
```

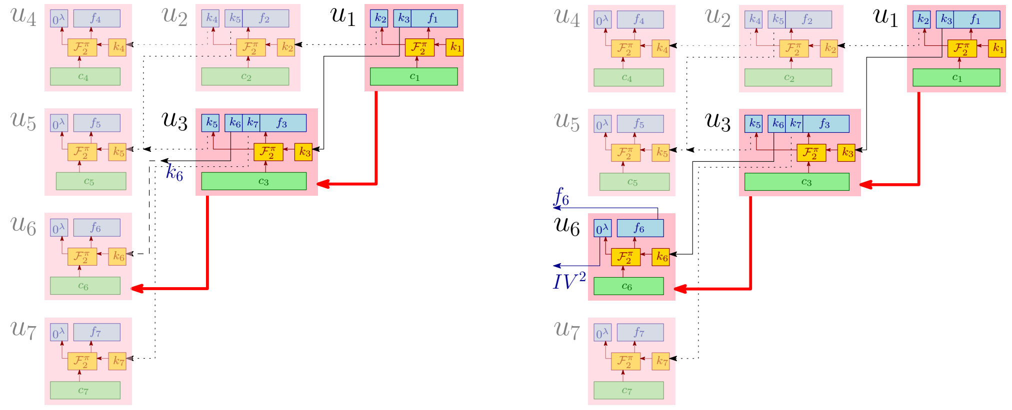
```
$\Pi.\mathcal{D}(params^{(\Pi)} G, u, v, S_u, pub)$
$k_v := \Pi.\mathcal{DER}(params^{(\Pi)}, G, u, v, S_u, pub)$;
$c_v := \mathsf{ext\_cipher}(pub, v)$;
$(f'_v, IV^2) := \mathcal{F}_2^\pi(1^\lambda, k_v, c_v)$;
$\tilde{v} \overset{def}{=} (u_{j_1}, u_{j_2}, \cdots, u_{j_d}) := \mathsf{ch\_seq}(v, G)$;
If $\tilde{v} = \mathsf{NULL}$
  If $IV^2 = 0^\lambda$,then return $f'_v$;
  Else return $\perp$;
$k_{u_{j_2}} || k_{u_{j_3}} || \cdots || k_{u_{j_d}} || f_v := f'_v$;
$\tilde{w} := \tilde{v},\ f'_w := f'_v$;
while $\tilde{w} \neq \mathsf{NULL}$
  $k_w := IV^2,\ w := u_{j_1}$;
  $c_w := \mathsf{ext\_cipher}(pub, w)$;
  $\tilde{w} \overset{def}{=} (u_{j_1}, u_{j_2}, \cdots, u_{j_d}) := \mathsf{ch\_seq}(w, G)$;
  $(f'_w, IV^2) := \mathcal{F}_2^\pi(c_w, k_w)$;
If $IV^2 = 0^\lambda$, then return $f_v$;
Else return $\perp$;
```

(a) Algorithmic description of building *Construction 2* Π using the functionalities $\mathcal{F}_1^\pi$ and $\mathcal{F}_2^\pi$. For the pictorial description with an example, see 21(b)–21(e).



(b) The access graph G with 7 nodes $u_1, u_2, \cdots, u_7$ and their corresponding files $f_1, f_2, \cdots, f_7$.

(c) Pictorial description of $\Pi.\mathcal{E}(params^{(\Pi)}, G, f)$, where $f = (f_1, f_2, \cdots, f_7)$ and G is shown in 21(b).



(d) Pictorial description of $\Pi.\mathcal{DER}(params^{(\Pi)}, G, u_1, u_6, S_1, pub)$ using the path (u_1, u_3, u_6) which is shown in red line in graph G in 21(b).

(e) Pictorial description of $\Pi.\mathcal{D}(params^{(\Pi)}, G, u_1, u_6, S_1, pub)$ using the path (u_1, u_3, u_6) which is shown in red line in graph G in 21(b).

Figure 21: Building *Construction 2*.

IV^2; the values of $k_{u_{j_2}}, k_{u_{j_3}}, \cdots, k_{u_{j_d}}$ and f_w are extracted from f'_w; and the next node in the path is searched in the sequence $\tilde{w}$, and the key corresponding to it is extracted, before the next iteration begins. Pictorial description of this algorithm on an access graph G is given in 22(d).

- $\Pi.\mathcal{D}(params^{(\Pi)}, G, u, v, S_u, pub)$ is a deterministic algorithm that facilitates the node u to decrypt the file of its successor v. As earlier, the node u uses the private information S_u and the public information of the system *pub*. In the first step, the decryption key $k_v = \Pi.\mathcal{DER}(params^{(\Pi)}, G, u, v, S_u, pub)$ is computed. Then, the ciphertext c_v is extracted from *pub* using the function ext_cipher, and the function ch_seq(v, G) returns the sequence of children $(u_{j_1}, u_{j_2}, \cdots, u_{j_d})$ of v (in ascending order); the value of IV^1 is computed as the last λ-bit block of ciphertext $c_{u_{j_1}}$. After that, the file f'_v and value IV^2 are computed $(f'_v, IV^2) = \mathcal{G}_2^\pi(1^\lambda, k_v, c_v, IV^1)$ and the keys of children of v are removed from the head of file f'_v to obtain the original file f_v. To verify the authentication of the file, the first child w of each node starting from v performs the following operations: the key $k_w = IV^2$ is computed, ciphertext c_w is extracted, the function ch_seq(w, G) returns the sequence of children $(u_{j_1}, u_{j_2}, \cdots, u_{j_d})$ of w (in ascending order); the value of IV^1 is computed as the last λ-bit block of ciphertext $c_{u_{j_1}}$; and $(f'_w, IV^2) = \mathcal{G}_2^\pi(1^\lambda, k_w, c_w, IV^1)$, where IV^2 acts as the key of the first child for the execution of next iteration. The the value of IV^2 should be 0^λ for the leaf node whose ciphertext is decrypted in the last iteration. If this condition is satisfied, the file f_v is returned, otherwise $\perp$ is returned.

7.2.2 Security of Construction 3

Theorem 16. *If the underlying FP is KR-ST (or IND-PRV or INT) secure, then the KAS-AE scheme Construction 3 is also KR-ST (or IND-PRV or INT) secure against static adversaries.*

Proof. The proof of this is identical to the KR-ST (or IND-PRV or INT) security proof of *KAS-AE-chain* construction $\mathsf{D}_{\mathsf{Chain}}$ in the Subsubsection 5.1.4. ☐

8 Comparison of various *KAS-AE* schemes

For the access graph $G = (V, E)$ and the sequence of files $f = (f_1 \circ f_2 \circ \cdots \circ f_n)$, we use the following notation: $n = |V|$; w is the *width* of the access graph G; $\mathsf{deg}(u)$ is the number of children of $u \in V$; $\uparrow u = \{v \in V | u \leq v\}$ denotes all the ancestors of $u \in V$; $|f| = \sum_{i \in [n]} |f_i|$; and λ is the security parameter. Also, we consider the key and tag sizes to be λ bits each. Based on the definitions of the key derivation algorithm $\Pi.\mathcal{DER}$ and decryption algorithm $\Pi.\mathcal{D}$ of the *KAS-AE* scheme (defined in Section 3), the *chain* C_g, and vertices u_h^g and $\hat{u}_g$ (discussed in Subsubsection 2.2.1 and Section 5), we define the sets $U_1 := \{v \in C_g | u_h^g \leq v \leq \hat{u}_g\}$; $U_2 := \{v \in C_g | v \leq \hat{u}_g\}$, so $U_1 \subseteq U_2$; $U_3 := \{u, u_{i_1}, u_{i_2}, \cdots, u_{i_\ell}, v\}$ such that $u_{i_1} \lessdot u, u_{i_2} \lessdot u_{i_1}, \cdots, v \lessdot u_{i_\ell}$; and $U_4 := U_3 \cup \{v, u_{j_1}, u_{j_2}, \cdots, u_{j_d}\}$ such that u_{j_1} is the first child of v, u_{j_2} is the first child of u_{j_1}, and so on, u_{j_d} is the first child of $u_{j_{d-1}}$.

Here, C_g is a partition of V forming a *chain* that contains the nodes $\hat{u}_g$ and u_h^g, such that $\hat{u}_g \leq u_h^g$ (see Subsubsection 2.2.1).

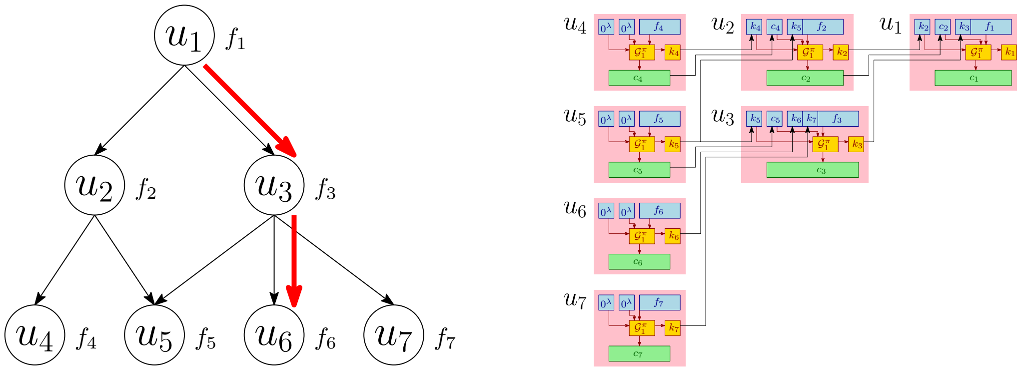
For the *KAS* schemes $\Pi = (\Pi.\mathcal{GEN}, \Pi.\mathcal{DER})$: • $\mathsf{c}_{\mathcal{GEN}}$ is the running time of generating a λ-bit key by algorithm $\Pi.\mathcal{GEN}$; • $\mathsf{c}_{\mathcal{K}}$ denote the cost of generating single λ-bit key, for the schemes X-AE, where X $\in$ {TKAS, TKEKAS, DKEKAS, IKEKAS}; and • c_e and $\mathsf{c}_{\mathcal{K}_g}$ denote the cost of generating one public value e and generating one λ-bit key from a given e value in NBKAS-AE.

```
$\Pi.\mathcal{E}(params^{(\Pi)}, G, f)$
$(level[\,], h)$ := height$(G)$;
while $h \geq 0$
  $V_h$ := nodes_at_level$(G, level[\,], h)$;
  For all $u \in V_h$
    $\tilde{u} \overset{def}{=} (u_{j_1}, u_{j_2}, \cdots, u_{j_d})$ := ch_seq$(u, G)$;
    If $\tilde{u}$ = NULL
      $IV^1 := IV^2 := 0^\lambda$;
      $f'_u := f_u$;
    Else
      $IV^1 := c_{u_{j_1}}[last\_block]$;
      $IV^2 := k_{u_{j_1}}$;
      $f'_u := k_{u_{j_2}} || k_{u_{j_3}} || \cdots \cdots || k_{u_{j_d}} || f_u$;
    $(k_u, c_u) := \mathcal{G}_1^\pi(1^\lambda, f'_u, IV^1, IV^2)$;
  $h := h - 1$;
$pub := (c_u)_{u \in V}$;
$S := k := (k_u)_{u \in V}$;
return $(S, k, pub)$;
```

```
$\Pi.\mathcal{DER}(params^{(\Pi)}, G, u, v, S_u, pub)$
$(u, u_{i_1}, u_{i_2}, \cdots, u_{i_\ell}, u_{i_{\ell+1}} = v)$ := path$(G, u, v)$;
$p := 1$, $w := u$, $k_w := S_u$;
while $(w \neq v)$
  $c_w$ := ext_cipher$(pub, w)$;
  $\tilde{w} \overset{def}{=} (u_{j_1}, u_{j_2}, \cdots, u_{j_d})$ := ch_seq$(w, G)$;
  If $\tilde{w} \neq$ NULL
    $c_{u_{j_1}}$ := ext_cipher$(pub, u_{j_1})$;
    $IV^1 := c_{u_{j_1}}[last\_block]$;
  Else $IV^1 := 0^\lambda$;
  $(f'_w, IV^2) := \mathcal{G}_2^\pi(1^\lambda, k_w, c_w, IV^1)$;
  $k_{u_{j_1}} := IV^2$;
  $k_{u_{j_2}} || k_{u_{j_3}} || \cdots || k_{u_{j_d}} || f_w := f'_w$;
  Find $u_{j_q} \in \tilde{w}$, s.t. $j_q = i_p$;
  $k_w := k_{u_{j_q}}$, $w := u_{j_q}$;
  $p := p + 1$;
return $k_w$;
```

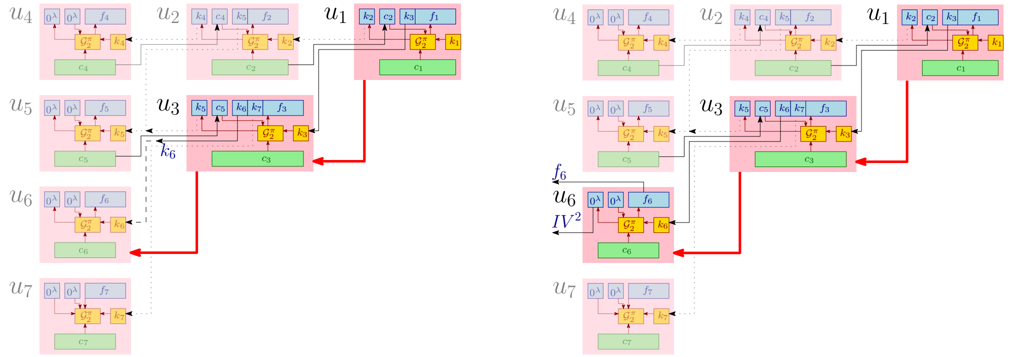
```
$\Pi.\mathcal{D}(params^{(\Pi)}, G, u, v, S_u, pub)$
$k_v := \Pi.\mathcal{DER}(params^{(\Pi)}, G, u, v, S_u, pub)$;
$c_v$ := ext_cipher$(pub, v)$;
$\tilde{v} \overset{def}{=} (u_{j_1}, u_{j_2}, \cdots, u_{j_d})$ := ch_seq$(v, G)$;
If $\tilde{v} \neq$ NULL
  $c_{u_{j_1}}$ := ext_cipher$(pub, u_{j_1})$;
  $IV^1 := c_{u_{j_1}}[last\_block]$;
Else $IV^1 := 0^\lambda$;
$(f'_v, IV^2) := \mathcal{G}_2^\pi(1^\lambda, k_v, c_v, IV^1)$;
If $\tilde{v}$ = NULL
  If $IV^2 = 0^\lambda$, then return $f'_v$;
  Else return $\perp$;
$k_{u_{j_2}} || k_{u_{j_3}} || \cdots || k_{u_{j_d}} || f_v := f'_v$;
$\tilde{w} := \tilde{v}$, $f'_w := f'_v$;
while $\tilde{w} \neq$ NULL
  $k_w := IV^2$, $w := u_{j_1}$;
  $c_w$ := ext_cipher$(pub, w)$;
  $\tilde{w} \overset{def}{=} (u_{j_1}, u_{j_2}, \cdots, u_{j_d})$ := ch_seq$(w, G)$;
  If $\tilde{w} \neq$ NULL
    $c_{u_{j_1}}$ := ext_cipher$(pub, u_{j_1})$;
    $IV^1 := c_{u_{j_1}}[last\_block]$;
  Else $IV^1 := 0^\lambda$;
  $(f'_w, IV^2) := \mathcal{G}_2^\pi(1^\lambda, k_w, c_w, IV^1)$;
If $IV^2 = 0^\lambda$, then return $f_v$;
Else return $\perp$;
```

(a) Algorithmic description of building *Construction 3* Π using the functionalities $\mathcal{G}_1^\pi$ and $\mathcal{G}_2^\pi$. For the pictorial description with an example, see 22(b)–22(e).



(b) The access graph G with 7 nodes $u_1, u_2, \cdots, u_7$ and their corresponding files $f_1, f_2, \cdots, f_7$.

(c) Pictorial description of $\Pi.\mathcal{E}(params^{(\Pi)}, G, f)$, where $f = (f_1, f_2, \cdots, f_7)$ and G is shown in 22(b).



(d) Pictorial description of $\Pi.\mathcal{DER}(params^{(\Pi)}, G, u_1, u_6, S_1, pub)$ using the path (u_1, u_3, u_6) which is shown in red line in graph G in 22(b).

(e) Pictorial description of $\Pi.\mathcal{D}(params^{(\Pi)}, G, u_1, u_6, S_1, pub)$ using the path (u_1, u_3, u_6) which is shown in red line in graph G in 22(b).

Figure 22: Building *Construction 3*.

For *AE* scheme $\Psi = (\Psi.\mathcal{K}_{\text{GEN}}, \Psi.\mathcal{E}, \Psi.\mathcal{D})$: • $\mathsf{c}_{\mathcal{AE}_\lambda}$ and $\mathsf{c}_{\mathcal{DV}_\lambda}$ denote the running times of algorithms $\Psi.\mathcal{E}$ and $\Psi.\mathcal{D}$ for a λ-bit input.

For *MLE* scheme $\Omega = (\Omega.\mathcal{E}, \Omega.\mathcal{D})$: • $\mathsf{c}_{\mathcal{E}_\lambda}$ and $\mathsf{c}_{\mathcal{D}_\lambda}$ denote the running times of algorithms $\Omega.\mathcal{E}$ and $\Omega.\mathcal{D}$ for a λ-bit input.

The c_π, $\mathsf{c}_{\pi^{-1}}$, $\tilde{\mathsf{c}}_\pi$ and $\tilde{\mathsf{c}}_{\pi^{-1}}$ denote the running times of the algorithms $\mathcal{F}_1^\pi$, $\mathcal{F}_2^\pi$, $\mathcal{G}_1^\pi$ and $\mathcal{G}_2^\pi$ that uses a 2λ-bit permutation.

Table 1: Comparison table for generic *KAS-AE* schemes (Subsection 4.2) built from generic *KAS* schemes (Subsubsection 2.3.3) using *AE*. Here, the assumption is that for all the *KAS-AE* schemes, the underlying *KAS* and *AE* are computationally secure.

Const.→ **Prop. ↓**	TKAS-AE (Based on TKAS [CC02] [YL04])	TKEKAS-AE (Based on TKEKAS [TC95] [SC02])	DKEKAS-AE (Based on DKEKAS [Gud80] [ZRM01])	IKEKAS-AE (Based on IKEKAS [ABFF09, AFB05] [CH05, SFM07a])	NBKAS-AE (Based on NBKAS [AT83] [HL90, CHW92])
Storage Req.:					
•Priv	$2n^2\lambda$	$2n^2\lambda$	$n^2\lambda + n\lambda$	$n^2\lambda + n\lambda$	$n^2\lambda + n\lambda$
•Pub	$\lvert f\rvert$	$\lvert f\rvert + n\lambda$	$\lvert f\rvert + n^2\lambda$	$\lvert f\rvert + n\lambda$	$\lvert f\rvert + n\lambda$
Running Time:					
•$\mathcal{E}$	$\mathsf{c}_{\mathcal{AE}_\lambda}\left(\frac{\lvert f\rvert}{\lambda}\right)$ $+n \cdot \mathsf{c}_{\mathcal{K}}$	$\mathsf{c}_{\mathcal{AE}_\lambda}\left(\frac{\lvert f\rvert}{\lambda} + n\right)$ $+2n \cdot \mathsf{c}_{\mathcal{K}}$	$\mathsf{c}_{\mathcal{AE}_\lambda}\left(\frac{\lvert f\rvert}{\lambda} + n^2\right)$ $+2n \cdot \mathsf{c}_{\mathcal{K}}$	$\mathsf{c}_{\mathcal{AE}_\lambda}\left(\frac{\lvert f\rvert}{\lambda} + n\right)$ $+n \cdot \mathsf{c}_{\mathcal{K}}$	$\mathsf{c}_{\mathcal{AE}_\lambda}\left(\frac{\lvert f\rvert}{\lambda}\right)$ $+n \cdot (\mathsf{c}_e + \mathsf{c}_{\mathcal{K}_g})$
•$\mathcal{DER}$	$\mathcal{O}(1)$	$\mathsf{c}_{\mathcal{DV}_\lambda}$	$\mathsf{c}_{\mathcal{DV}_\lambda}$	$\mathcal{O}(n) \cdot \mathsf{c}_{\mathcal{DV}_\lambda}$	$\mathsf{c}_{\mathcal{K}_g}$
•$\mathcal{D}$	$\mathcal{O}(1)$ $+\frac{\mathsf{c}_{\mathcal{DV}_\lambda}}{\lambda}\lvert f_v\rvert$	$\mathsf{c}_{\mathcal{DV}_\lambda}$ $+\frac{\mathsf{c}_{\mathcal{DV}_\lambda}}{\lambda}\lvert f_v\rvert$	$\mathsf{c}_{\mathcal{DV}_\lambda}$ $+\frac{\mathsf{c}_{\mathcal{DV}_\lambda}}{\lambda}\lvert f_v\rvert$	$\mathcal{O}(n) \cdot \mathsf{c}_{\mathcal{DV}_\lambda}$ $+\frac{\mathsf{c}_{\mathcal{DV}_\lambda}}{\lambda}\lvert f_v\rvert$	$\mathsf{c}_{\mathcal{K}_g}$ $+\frac{\mathsf{c}_{\mathcal{DV}_\lambda}}{\lambda}\lvert f_v\rvert$

9 Conclusion and Future Work

In this paper, we present a new cryptographic primitive, namely, *KAS-AE*, and design three efficient constructions of it. We showed that these constructions perform better – both with respect to time and memory – than the existing mechanisms to solve the well-known *hierarchical access control* problem relevant for any multi-layered organization. The high performance of our schemes is attributed to its very unique *reverse decryption* property. This property is difficult to find, and we leave it as an open problem to design more constructions with this property. Another future work in this line of research will be to add more functionalities to *KAS-AE*, such as *key revocation* and *file update*, and find efficient constructions.

References

[AAB15] Javad Alizadeh, Mohammad Reza Aref, and Nasour Bagheri. Artemia v1, 2015.

[ABB+14] Elena Andreeva, Begül Bilgin, Andrey Bogdanov, Atul Luykx, Bart Mennink, Nicky Mouha, and Kan Yasuda. APE: authenticated permutation-based encryption for lightweight cryptography. In Carlos Cid and Christian Rechberger, editors, *Fast Software Encryption - 21st International Workshop,*

Table 2: Comparison table for different *KAS-AE* schemes built using *KAS-AE-chain* constructions (described in Subsection 5.1) embedded into *modified chain partition* algorithm (described in Subsection 5.2).

Const. → **Prop. ↓**	**Construction A** (Subsection 5.2 + Subsubsection 5.1.1)	**Construction B** (Subsection 5.2 + Subsubsection 5.1.2)	**Construction C** (Subsection 5.2 + Subsubsection 5.1.3)	**Construction D** (Subsection 5.2 + Subsubsection 5.1.4)
Storage Req.:				
•Priv	$(n^2 + nw)\lambda$	$nw\lambda$	$nw\lambda$	$nw\lambda$
•Pub	$\lvert f\rvert$	$2n\lambda + \lvert f\rvert$	$n\lambda + \lvert f\rvert$	$n\lambda + \lvert f\rvert$
Running Time:				
•$\mathcal{E}$	$\mathsf{c}_{\mathcal{AE}_\lambda} \cdot \frac{\lvert f\rvert}{\lambda}$ $+n \cdot \mathsf{c}_{\mathcal{GEN}}$	$\mathsf{c}_{\mathcal{E}_\lambda} \cdot \frac{\lvert f\rvert}{\lambda}$ $+n \cdot \mathsf{c}_{\mathcal{E}_\lambda}$	$\mathsf{c}_\pi \cdot \left(\frac{\lvert f\rvert}{\lambda} + n\right)$	$\tilde{\mathsf{c}}_\pi \cdot \left(\frac{\lvert f\rvert}{\lambda} + n\right)$
•$\mathcal{DER}$	$\mathsf{c}_{\mathcal{GEN}} \cdot \lvert U_1\rvert$	$\frac{\mathsf{c}_{\mathcal{D}_\lambda}}{\lambda} \cdot \sum_{v \in U_1} \lvert f_v\rvert$ $+\lvert U_1\rvert \cdot \mathsf{c}_{\mathcal{D}_\lambda}$	$\frac{\mathsf{c}_{\pi-1}}{\lambda} \cdot \sum_{v \in U_1} \lvert f_v\rvert$ $+\lvert U_1\rvert \cdot \mathsf{c}_{\pi-1}$	$\frac{\tilde{\mathsf{c}}_{\pi-1}}{\lambda} \cdot \sum_{v \in U_1} \lvert f_v\rvert$ $+\lvert U_1\rvert \cdot \tilde{\mathsf{c}}_{\pi-1}$
•$\mathcal{D}$	$\frac{\mathsf{c}_{\mathcal{DV}_\lambda}}{\lambda} \cdot \lvert f_{u_h^g}\rvert$ $+\lvert U_1\rvert \cdot \mathsf{c}_{\mathcal{GEN}}$	$\frac{\mathsf{c}_{\mathcal{D}_\lambda}}{\lambda} \cdot \sum_{v \in U_1} \lvert f_v\rvert$ $+\lvert U_1\rvert \cdot \mathsf{c}_{\mathcal{D}_\lambda}$	$\frac{\mathsf{c}_{\pi-1}}{\lambda} \cdot \sum_{v \in U_2} \lvert f_v\rvert$ $+\lvert U_2\rvert \cdot \mathsf{c}_{\pi-1}$	$\frac{\tilde{\mathsf{c}}_{\pi-1}}{\lambda} \cdot \sum_{v \in U_2} \lvert f_v\rvert$ $+\lvert U_2\rvert \cdot \tilde{\mathsf{c}}_{\pi-1}$
Computation Assumption	Secure *KAS* & Secure *AE*	Secure *MLE*	Ideal Permutation	Ideal Permutation

Table 3: Comparison table for *KAS-AE* schemes built in Section 6 and Section 7.

Const. → **Prop. ↓**	**Construction 1** Section 6	**Construction 2** Subsection 7.1	**Construction 3** Subsection 7.2
Storage Req.:			
•Priv	$n\lambda$	$n\lambda$	$n\lambda$
•Pub	$\sum_{u \in V} \Big((\mathsf{deg}(u) + 1) \cdot \lambda + \lvert f_u\rvert\Big)$	$\sum_{u \in V} \Big(\mathsf{deg}(u) \cdot \lambda + \lvert f_u\rvert\Big)$	$\sum_{u \in V} \Big(\mathsf{deg}(u) \cdot \lambda + \lvert f_u\rvert\Big)$
Running Time:			
•$\mathcal{E}$	$\mathsf{c}_{\mathcal{E}_\lambda} \cdot \frac{\lvert f\rvert}{\lambda} \cdot$ $+\mathsf{c}_{\mathcal{E}_\lambda} \cdot \sum_{u \in V} (\mathsf{deg}(u) + 1)$	$\mathsf{c}_\pi \cdot \frac{\lvert f\rvert}{\lambda}$ $+\mathsf{c}_\pi \cdot \sum_{u \in V} \mathsf{deg}(u)$	$\tilde{\mathsf{c}}_\pi \cdot \frac{\lvert f\rvert}{\lambda}$ $+\tilde{\mathsf{c}}_\pi \cdot \sum_{u \in V} \mathsf{deg}(u)$
•$\mathcal{DER}$	$\frac{\mathsf{c}_{\mathcal{D}_\lambda}}{\lambda} \cdot \sum_{u_i \in U_3} \lvert f_{u_i}\rvert$ $+\mathsf{c}_{\mathcal{D}_\lambda} \cdot \sum_{u_i \in U_3} \Big(\mathsf{deg}(u_i) + 1\Big)$	$\frac{\mathsf{c}_{\pi-1}}{\lambda} \cdot \sum_{u_i \in U_3} \lvert f_{u_i}\rvert$ $+\mathsf{c}_{\pi-1} \cdot \sum_{u_i \in U_3} \mathsf{deg}(u_i)$	$\frac{\tilde{\mathsf{c}}_{\pi-1}}{\lambda} \cdot \sum_{u_i \in U_3} \lvert f_{u_i}\rvert$ $+\tilde{\mathsf{c}}_{\pi-1} \cdot \sum_{u_i \in U_3} \mathsf{deg}(u_i)$
•$\mathcal{D}$	$\frac{\mathsf{c}_{\mathcal{D}_\lambda}}{\lambda} \cdot \sum_{u_i \in U_3} \lvert f_{u_i}\rvert$ $+\mathsf{c}_{\mathcal{D}_\lambda} \cdot \sum_{u_i \in U_3} \Big(\mathsf{deg}(u_i) + 1\Big)$	$\frac{\mathsf{c}_{\pi-1}}{\lambda} \cdot \sum_{u_i \in U_4} \lvert f_{u_i}\rvert$ $+\mathsf{c}_{\pi-1} \cdot \sum_{u_i \in U_4} \mathsf{deg}(u_i)$	$\frac{\tilde{\mathsf{c}}_{\pi-1}}{\lambda} \cdot \sum_{u_i \in U_4} \lvert f_{u_i}\rvert$ $+\tilde{\mathsf{c}}_{\pi-1} \cdot \sum_{u_i \in U_4} \mathsf{deg}(u_i)$
Computation Assumption	Secure *MLE*	Ideal Permutation	Ideal Permutation

FSE 2014, London, UK, March 3-5, 2014. Revised Selected Papers, volume 8540 of *Lecture Notes in Computer Science*, pages 168–186. Springer, 2014.

[ABF07] Mikhail J. Atallah, Marina Blanton, and Keith B. Frikken. Incorporating temporal capabilities in existing key management schemes. In Joachim Biskup and Javier Lopez, editors, *Computer Security - ESORICS 2007, 12th European Symposium On Research In Computer Security, Dresden, Germany, September 24-26, 2007, Proceedings*, volume 4734 of *Lecture Notes in Computer Science*, pages 515–530. Springer, 2007.

[ABFF09] Mikhail J. Atallah, Marina Blanton, Nelly Fazio, and Keith B. Frikken. Dynamic and efficient key management for access hierarchies. *ACM Trans. Inf. Syst. Secur.*, 12(3):18:1–18:43, 2009.

[ABM$^+$13] Martín Abadi, Dan Boneh, Ilya Mironov, Ananth Raghunathan, and Gil Segev. Message-Locked Encryption for Lock-Dependent Messages. In Ran Canetti and Juan A. Garay, editors, *CRYPTO (1)*, volume 8042 of *Lecture Notes in Computer Science*, pages 374–391. Springer, 2013.

[ACS15] Megha Agrawal, Donghoon Chang, and Somitra Sanadhya. sp-aelm: Sponge based authenticated encryption scheme for memory constrained devices. In Ernest Foo and Douglas Stebila, editors, *Information Security and Privacy - 20th Australasian Conference, ACISP 2015, Brisbane, QLD, Australia, June 29 - July 1, 2015, Proceedings*, volume 9144 of *Lecture Notes in Computer Science*, pages 451–468. Springer, 2015.

[AFB05] Mikhail J. Atallah, Keith B. Frikken, and Marina Blanton. Dynamic and efficient key management for access hierarchies. In Vijay Atluri, Catherine A. Meadows, and Ari Juels, editors, *Proceedings of the 12th ACM Conference on Computer and Communications Security, CCS 2005, Alexandria, VA, USA, November 7-11, 2005*, pages 190–202. ACM, 2005.

[ASFM06] Giuseppe Ateniese, Alfredo De Santis, Anna Lisa Ferrara, and Barbara Masucci. Provably-secure time-bound hierarchical key assignment schemes. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, Ioctober 30 - November 3, 2006*, pages 288–297. ACM, 2006.

[ASFM12] Giuseppe Ateniese, Alfredo De Santis, Anna Lisa Ferrara, and Barbara Masucci. Provably-secure time-bound hierarchical key assignment schemes. *J. Cryptology*, 25(2):243–270, 2012.

[ASFM13] Giuseppe Ateniese, Alfredo De Santis, Anna Lisa Ferrara, and Barbara Masucci. A note on time-bound hierarchical key assignment schemes. *Inf. Process. Lett.*, 113(5-6):151–155, 2013.

[AT83] Selim G. Akl and Peter D. Taylor. Cryptographic solution to a problem of access control in a hierarchy. *ACM Trans. Comput. Syst.*, 1(3):239–248, 1983.

[BDP$^+$14] Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. Caesar submission: Ketje v1, 2014.

[BDPA09] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. The Keccak Hash Function. The 1st SHA-3 Candidate Conference, 2009.

[BKR13] Mihir Bellare, Sriram Keelveedhi, and Thomas Ristenpart. Message-Locked Encryption and Secure Deduplication. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT*, volume 7881 of *Lecture Notes in Computer Science*, pages 296–312. Springer, 2013.

[BN08] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. *J. Cryptology*, 21(4):469–491, 2008.

[BR06] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In Serge Vaudenay, editor, *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 409–426. Springer, 2006.

[BRW03] Mihir Bellare, Phillip Rogaway, and David A. Wagner. EAX: A conventional authenticated-encryption mode. *IACR Cryptology ePrint Archive*, 2003:69, 2003.

[BRW04] Mihir Bellare, Phillip Rogaway, and David Wagner. The EAX mode of operation. In Bimal K. Roy and Willi Meier, editors, *Fast Software Encryption, 11th International Workshop, FSE 2004, Delhi, India, February 5-7, 2004, Revised Papers*, volume 3017 of *Lecture Notes in Computer Science*, pages 389–407. Springer, 2004.

[BSJ08] Elisa Bertino, Ning Shang, and Samuel S. Wagstaff Jr. An efficient time-bound hierarchical key management scheme for secure broadcasting. *IEEE Trans. Dependable Sec. Comput.*, 5(2):65–70, 2008.

[CC02] Tzer-Shyong Chen and Yu-Fang Chung. Hierarchical access control based on chinese remainder theorem and symmetric algorithm. *Computers & Security*, 21(6):565–570, 2002.

[CCM15] Massimo Cafaro, Roberto Civino, and Barbara Masucci. On the equivalence of two security notions for hierarchical key assignment schemes in the unconditional setting. *IEEE Trans. Dependable Sec. Comput.*, 12(4):485–490, 2015.

[CDM10] Jason Crampton, Rosli Daud, and Keith M. Martin. Constructing key assignment schemes from chain partitions. In Sara Foresti and Sushil Jajodia, editors, *Data and Applications Security and Privacy XXIV, 24th Annual IFIP WG 11.3 Working Conference, Rome, Italy, June 21-23, 2010. Proceedings*, volume 6166 of *Lecture Notes in Computer Science*, pages 130–145. Springer, 2010.

[CH05] Tzer-Shyong Chen and Jen-Yan Huang. A novel key management scheme for dynamic access control in a user hierarchy. *Applied Mathematics and Computation*, 162(1):339–351, 2005.

[Chi04] Hung-Yu Chien. Efficient time-bound hierarchical key assignment scheme. *IEEE Trans. Knowl. Data Eng.*, 16(10):1301–1304, 2004.

[CHW92] Chin-Chen Chang, Ren-Junn Hwang, and Tzong-Chen Wu. Crypographic key assignment scheme for access control in a hierarchy. *Inf. Syst.*, 17(3):243–247, 1992.

[CMW06] Jason Crampton, Keith M. Martin, and Peter R. Wild. On key assignment for hierarchical access control. In *19th IEEE Computer Security Foundations Workshop, (CSFW-19 2006), 5-7 July 2006, Venice, Italy*, pages 98–111. IEEE Computer Society, 2006.

[CMYG15] Rongmao Chen, Yi Mu, Guomin Yang, and Fuchun Guo. BL-MLE: block-level message-locked encryption for secure large file deduplication. *IEEE Trans. Information Forensics and Security*, 10(12):2643–2652, 2015.

[CSM16a] Arcangelo Castiglione, Alfredo De Santis, and Barbara Masucci. Key indistinguishability versus strong key indistinguishability for hierarchical key assignment schemes. *IEEE Trans. Dependable Sec. Comput.*, 13(4):451–460, 2016.

[CSM$^+$16b] Arcangelo Castiglione, Alfredo De Santis, Barbara Masucci, Francesco Palmieri, and Aniello Castiglione. On the relations between security notions in hierarchical key assignment schemes for dynamic structures. In Joseph K. Liu and Ron Steinfeld, editors, *Information Security and Privacy - 21st Australasian Conference, ACISP 2016, Melbourne, VIC, Australia, July 4-6, 2016, Proceedings, Part II*, volume 9723 of *Lecture Notes in Computer Science*, pages 37–54. Springer, 2016.

[DAB$^+$02] John R. Douceur, Atul Adya, William J. Bolosky, Dan Simon, and Marvin Theimer. Reclaiming space from duplicate files in a serverless distributed file system. In *ICDCS*, pages 617–624, 2002.

[Dij59] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numer. Math.*, 1:269–271, 1959.

[Dil50] R. P. Dilworth. A decomposition theorem for partially ordered sets. *Ann. of Math.*, 51(2):161–166, 1950.

[DSFM10] Paolo D'Arco, Alfredo De Santis, Anna Lisa Ferrara, and Barbara Masucci. Variations on a theme by akl and taylor: Security and tradeoffs. *Theor. Comput. Sci.*, 411(1):213–227, 2010.

[FP11] Eduarda S. V. Freire and Kenneth G. Paterson. Provably secure key assignment schemes from factoring. In Udaya Parampalli and Philip Hawkes, editors, *Information Security and Privacy - 16th Australasian Conference, ACISP 2011, Melbourne, Australia, July 11-13, 2011. Proceedings*, volume 6812 of *Lecture Notes in Computer Science*, pages 292–309. Springer, 2011.

[FPP13] Eduarda S. V. Freire, Kenneth G. Paterson, and Bertram Poettering. Simple, efficient and strongly ki-secure hierarchical key assignment schemes. In Ed Dawson, editor, *Topics in Cryptology - CT-RSA 2013 - The Cryptographers' Track at the RSA Conference 2013, San Francisco,CA, USA, February 25-March 1, 2013. Proceedings*, volume 7779 of *Lecture Notes in Computer Science*, pages 101–114. Springer, 2013.

[Gud80] Ehud Gudes. The design of a cryptopgraphy based secure file system. *IEEE Trans. Software Eng.*, 6(5):411–420, 1980.

[HC04] Hui-Feng Huang and Chin-Chen Chang. A new cryptographic key assignment scheme with time-constraint access control in a hierarchy. *Computer Standards & Interfaces*, 26(3):159–166, 2004.

[HL90] Lein Harn and Hung-Yu Lin. A cryptographic key generation scheme for multilevel data security. *Computers & Security*, 9(6):539–546, 1990.

[KP18] Suyash Kandele and Souradyuti Paul. Message-locked encryption with file update. In Bart Preneel and Frederik Vercauteren, editors, *Applied Cryptography and Network Security - 16th International Conference, ACNS 2018, Leuven, Belgium, July 2-4, 2018, Proceedings*, volume 10892 of *Lecture Notes in Computer Science*, pages 678–695. Springer, 2018.

[MTMA85] Stephen J. MacKinnon, Peter D. Taylor, Henk Meijer, and Selim G. Akl. An optimal algorithm for assigning cryptographic keys to control access in a hierarchy. *IEEE Trans. Computers*, 34(9):797–802, 1985.

[PHG12] Souradyuti Paul, Ekawat Homsirikamol, and Kris Gaj. A Novel Permutation-based Hash Mode of Operation FP and the Hash Function SAMOSA. In Steven Galbraith and Mridul Nandi, editors, *Progress in Cryptology - INDOCRYPT 2012*, volume 7668 of *Lecture Notes in Computer Science*, pages 514 – 532, Kolkata,WB,INDIA, 2012. Springer-Verlag.

[PWCW15a] Jeng-Shyang Pan, Tsu-Yang Wu, Chien-Ming Chen, and Eric Ke Wang. An efficient solution for time-bound hierarchical key assignment scheme. In Thi Thi Zin, Jerry Chun-Wei Lin, Jeng-Shyang Pan, Pyke Tin, and Mitsuhiro Yokota, editors, *Genetic and Evolutionary Computing - Proceedings of the Ninth International Conference on Genetic and Evolutionary Computing, ICGEC 2015, August 26-28, 2015, Yangon, Myanmar - Volume II*, volume 388 of *Advances in Intelligent Systems and Computing*, pages 3–9. Springer, 2015.

[PWCW15b] Jeng-Shyang Pan, Tsu-Yang Wu, Chien-Ming Chen, and Eric Ke Wang. Security analysis of a time-bound hierarchical key assignment scheme. In Jeng-Shyang Pan, Ivan Lee, Hsiang-Cheh Huang, and Ching-Yu Yang, editors, *2015 International Conference on Intelligent Information Hiding and Multimedia Signal Processing, IIH-MSP 2015, Adelaide, Australia, September 23-25, 2015*, pages 203–206. IEEE, 2015.

[Rog02] Phillip Rogaway. Authenticated-encryption with associated-data. In Vijayalakshmi Atluri, editor, *Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002, Washington, DC, USA, November 18-22, 2002*, pages 98–107. ACM, 2002.

[SC02] Victor R. L. Shen and Tzer-Shyong Chen. A novel key management scheme based on discrete logarithms and polynomial interpolations. *Computers & Security*, 21(2):164–171, 2002.

[SFM07a] Alfredo De Santis, Anna Lisa Ferrara, and Barbara Masucci. Efficient provably-secure hierarchical key assignment schemes. In Ludek Kucera and Antonín Kucera, editors, *Mathematical Foundations of Computer Science 2007, 32nd International Symposium, MFCS 2007, Ceský Krumlov, Czech Republic, August 26-31, 2007, Proceedings*, volume 4708 of *Lecture Notes in Computer Science*, pages 371–382. Springer, 2007.

[SFM07b] Alfredo De Santis, Anna Lisa Ferrara, and Barbara Masucci. New constructions for provably-secure time-bound hierarchical key assignment schemes. In Volkmar Lotz and Bhavani M. Thuraisingham, editors, *12th ACM Symposium on Access Control Models and Technologies, SACMAT 2007, Sophia Antipolis, France, June 20-22, 2007, Proceedings*, pages 133–138. ACM, 2007.

[SFM08] Alfredo De Santis, Anna Lisa Ferrara, and Barbara Masucci. New constructions for provably-secure time-bound hierarchical key assignment schemes. *Theor. Comput. Sci.*, 407(1-3):213–230, 2008.

[TC95] Hui-Min Tsai and Chin-Chen Chang. A cryptographic implementation for dynamic access control in a user hierarchy. *Computers & Security*, 14(2):159–166, 1995.

[Tze02] Wen-Guey Tzeng. A time-bound cryptographic key assignment scheme for access control in a hierarchy. *IEEE Trans. Knowl. Data Eng.*, 14(1):182–188, 2002.

[Tze06] Wen-Guey Tzeng. A secure system for data access based on anonymous authentication and time-dependent hierarchical keys. In Ferng-Ching Lin, Der-Tsai Lee, Bao-Shuh Paul Lin, Shiuhpyng Shieh, and Sushil Jajodia, editors, *Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security, ASIACCS 2006, Taipei, Taiwan, March 21-24, 2006*, pages 223–230. ACM, 2006.

[WC01] Tzong-Chen Wu and Chin-Chen Chang. Cryptographic key assignment scheme for hierarchical access control. *Comput. Syst. Sci. Eng.*, 16(1):25–28, 2001.

[WL06] Shyh-Yih Wang and Chi-Sung Laih. Merging: An efficient solution for a time-bound hierarchical key assignment scheme. *IEEE Trans. Dependable Sec. Comput.*, 3(1):91–100, 2006.

[YCN03] Jyh-haw Yeh, Randy Chow, and Richard E. Newman. Key assignment for enforcing access control policy exceptions in distributed systems. *Inf. Sci.*, 152:63–88, 2003.

[Yeh05] Jyh-haw Yeh. An rsa-based time-bound hierarchical key assignment scheme for electronic article subscription. In Otthein Herzog, Hans-Jörg Schek, Norbert Fuhr, Abdur Chowdhury, and Wilfried Teiken, editors, *Proceedings of the 2005 ACM CIKM International Conference on Information and Knowledge Management, Bremen, Germany, October 31 - November 5, 2005*, pages 285–286. ACM, 2005.

[Yi05] Xun Yi. Security of chien's efficient time-bound hierarchical key assignment scheme. *IEEE Trans. Knowl. Data Eng.*, 17(9):1298–1299, 2005.

[YL04] Cungang Yang and Celia Li. Access control in a hierarchy using one-way hash functions. *Computers & Security*, 23(8):659–664, 2004.

[YY03] Xun Yi and Yiming Ye. Security of tzeng's time-bound key assignment scheme for access control in a hierarchy. *IEEE Trans. Knowl. Data Eng.*, 15(4):1054–1055, 2003.

[ZRM01] Xukai Zou, Byrav Ramamurthy, and Spyros S. Magliveras. Chinese remainder theorem based hierarchical access control for secure group communication. In Sihan Qing, Tatsuaki Okamoto, and Jianying Zhou, editors, *Information and Communications Security, Third International Conference, ICICS 2001, Xian, China, November 13-16, 2001*, volume 2229 of *Lecture Notes in Computer Science*, pages 381–385. Springer, 2001.