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,
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, PWCCW15a, PWCCW15b, 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

Let $M := x$ denote that the value of $x$ is assigned to $M$, and $M := D(x)$ denotes that the value returned by function $D()$ for input $x$, is assigned to $M$. $M := x$ denotes the equality comparison of the two variables $M$ and $x$, and $M := D(x)$ denotes the equality comparison of the variable $M$ with the output of $D()$ on input $x$. The XOR or $\oplus$ denotes the bit-by-bit exclusive-or 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 $\ell$ 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 $M \triangleleft \{0, 1\}^k$. 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 $f_u$. $S = (S_u)_{u \in V}$ 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 $f_1, f_2, \cdots, f_p$. $(M_0, M_1, Z)$ denotes the assignment of outputs given randomly and uniformly by $S$ to $M_0$ and $M_1$ and supplying some auxiliary information $Z$. Here, $M_0$ and $M_1$ is a vector of strings and $i$-th string in
\( M \) is denoted as \( M^{(q)} \). 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 \( \text{aencrypt} \). 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 \( \text{level}[u] \) of \( u \) is the length of path from \( \text{root} \) node to \( u \); and the maximum-depth of the tree is the maximum value of \( \text{level}[\cdot] \) among all the nodes of the tree. The node \( u_j \) means in the \text{chain} \( C_i \) the \( j \)-th node from \( \text{root} \). 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. \( v \in C \) \( \iff \) \( v \) is an ancestor of \( u \) or \( v \) is a successor of \( u \), if \( v \leq u \) and \( \exists \in V \) 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 \( v \preceq u \) and \( u \preceq v \). 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 \( v < u \), 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 \( V \) is a collection of sets \( \{V_1, V_2, \cdots, V_s\} \) such that:

- \( \forall \in V, \forall i \in [s] \), \( \{\forall \} \subseteq V_i \);
- \( \forall i \in [s], V_i \cup V_{i+1} \cdots \cup V_s = V \); 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 \( V \), \( t_i = |C_i| \) (for \( i \in [w] \)), and \( t_{\max} = \max_{i \in [w]} t_i \). The maximum node of \( C_i \) is denoted \( u_{i}^1 \) (i.e. \( \forall v \in C_i, v \leq u^1 \)); and the minimum node of \( C_i \), denoted \( u_{i}^\downarrow \) (i.e. \( \forall v \in C_i, u_{i}^\downarrow \leq v \)). If \( C_i = \{u_{i}^1, u_{i}^\downarrow, \cdots, u_{i}^i\} \) and \( u_{i}^1 < u_{i}^2 < \cdots < u_{i}^i \) then \( u_{i}^j < u_{i}^\downarrow \leq \cdots < u_{i}^\downarrow \). A partition \( \{C_1, C_2, \cdots, C_w\} \) is said to be a suffix of \( C_i \), where \( j \in [t_i] \). 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 \( \uparrow u := \{v \in E : v \leq u\} \) (and \( \downarrow v := \{v \in E : v \geq v\} \)). 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 \( u_i^\downarrow \). If \( \downarrow u \cap C_i = \emptyset \), then \( u_i^\downarrow = 1 \).

2.2.2 Ideal Permutation

Let \( \pi^{-1} : \{0,1\}^n \mapsto \{0,1\}^n \) be a pair of oracles. The pair \( \pi/\pi^{-1} \) 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 \( \pi \), and \( y \in \{0,1\}^n \). Then, for the current query \( x_i \):

\[
\Pr \left[ \pi(x_i) = y \mid \pi(x_1) = y_1, \pi(x_2) = y_2, \cdots, \pi(x_{i-1}) = y_{i-1} \right]
\]
We briefly re-discuss it below, with a few suitable changes in the notation to suit the present context.

We are modelling the security based on an unpredictable message source which is a random function. Here, each string $M$ has has $m(1^k)$ number of strings, i.e., $\|M\| = m(1^k)$ and the length of each string $M^{(i)}$ is $l(1^k, i)$, i.e., $|M^{(i)}| = l(1^k, i)$ for $i \in [m(1^k)]$. Here, $m$ and $l$ are two functions. We require that the two strings $M^{(i_1)} \neq M^{(i_2)}$, for $i_1 \neq i_2$ and $i_1, i_2 \in [m(1^k)]$. Associated with the source $S(\cdot)$ is a real number $GP_S$, namely, the Guessing Probability of source, which is the maximum of all the probabilities of guessing a single string in $M$, given the auxiliary information. The formal definition is $GP_S(1^k) \triangleq \max_{i \in [m(1^k)]} GP(M^{(i)}|Z)$. The source $S(\cdot)$ is said to be unpredictable if the value of $GP_S$ is negligible. We now define the min-entropy $\mu_S(\cdot)$ of the source $S(\cdot)$ as $\mu_S(1^k) = -\log(GP_S(1^k))$. The source $S(\cdot)$ is said to be a valid source for an $MLE$ scheme $\Pi$ if $M^{(i)} \in M$, $\forall i \in [m(1^k)]$.

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 $\Pi. \text{Setup}$. $\Pi$ satisfies the following conditions.

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

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

$$\Pr \left[ \pi^{-1}(y_i) = x \mid \pi^{-1}(y_1) = x_1, \pi^{-1}(y_2) = x_2, \ldots, \pi^{-1}(y_{i-1}) = x_{i-1} \right] = \begin{cases} 1, & \text{if } y_i = y_j, x = x_j, j < i, \\ 0, & \text{if } y_i = y_j, x \neq x_j, j < i, \\ 0, & \text{if } y_i \neq y_j, x = x_j, j < i, \\ \frac{1}{2n-\pi+1}, & \text{if } y_i \neq y_j, x \neq x_j, j < i. \end{cases}$$

### 2.2.3 Random Function

Let $rf: \{0,1\}^n \mapsto \{0,1\}^n$. 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 $x_i$:

$$\Pr \left[ rf(x_i) = y \mid rf(x_1) = y_1, rf(x_2) = y_2, \ldots, rf(x_{i-1}) = y_{i-1} \right] = \begin{cases} 1, & \text{if } x_i = x_j, y = y_j, j < i, \\ 0, & \text{if } x_i = x_j, y \neq y_j, j < i, \\ 0, & \text{if } x_i \neq x_j, y = y_j, j < i, \\ \frac{1}{2n}, & \text{if } x_i \neq x_j, y \neq y_j, j < i. \end{cases}$$

### 2.2.4 Source of message $S$

We are modelling the security based on an unpredictable message source which is a PT algorithm, denoted $S(\cdot)$, that returns $(M, Z)$ or $(M_0, M_1, Z)$ on input $1^k$, where each vector of messages $M \in \{0,1\}^n$ (or $M_0, M_1 \in \{0,1\}^n$) and auxiliary information $Z \in \{0,1\}^*$. We consider that $S(\cdot)$ is a public source, that is, it is known to all the parties including the adversary. Here, each vector of messages $M$ has $m(1^k)$ number of strings, i.e., $\|M\| = m(1^k)$ and the length of each string $M^{(i)}$ is $l(1^k, i)$, i.e., $|M^{(i)}| = l(1^k, i)$ for $i \in [m(1^k)]$. Here, $m$ and $l$ are two functions. We require that the two strings $M^{(i_1)} \neq M^{(i_2)}$, for $i_1 \neq i_2$ and $i_1, i_2 \in [m(1^k)]$. Associated with the source $S(\cdot)$ is a real number $GP_S$, namely, the Guessing Probability of source, which is the maximum of all the probabilities of guessing a single string in $M$, given the auxiliary information. The formal definition is $GP_S(1^k) = \max_{i \in [m(1^k)]} GP(M^{(i)}|Z)$. The source $S(\cdot)$ is said to be unpredictable if the value of $GP_S$ is negligible. We now define the min-entropy $\mu_S(\cdot)$ of the source $S(\cdot)$ as $\mu_S(1^k) = -\log(GP_S(1^k))$. The source $S(\cdot)$ is said to be a valid source for an $MLE$ scheme $\Pi$ if $M^{(i)} \in M$, $\forall i \in [m(1^k)]$.

### 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 $\Pi. \text{Setup}$. $\Pi$ satisfies the following conditions.
1. The PPT setup algorithm II.\textit{Setup}(1^\lambda) outputs the parameter \(\text{params}^{(\Pi)}\) 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 II.\textit{E} takes as inputs the parameter \(\text{params}^{(\Pi)}\) and \(M \in \mathcal{M}^{(\Pi)}\), and returns a three-tuple \((K, C, T) := \Pi.\text{E}(\text{params}^{(\Pi)}, M)\), where \(K \in \mathcal{K}^{(\Pi)}\), \(C \in \mathcal{C}^{(\Pi)}\) and \(T \in \mathcal{T}^{(\Pi)}\).

3. The decryption algorithm II.\textit{D} is a deterministic algorithm that takes as inputs the parameter \(\text{params}^{(\Pi)}\), \(K \in \mathcal{K}^{(\Pi)}\), \(C \in \mathcal{C}^{(\Pi)}\) and \(T \in \mathcal{T}^{(\Pi)}\), and returns \(\Pi.\text{D}(\text{params}^{(\Pi)}, K, C, T) \in \mathcal{M}^{(\Pi)} \cup \{\bot\}\). The decryption algorithm II.\textit{D} returns \(\bot\) 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 \(|M|\).

\textbf{Key Correctness.} Let \(M, M' \in \mathcal{M}^{(\Pi)}\). Suppose:
- \((K, C, T) := \Pi.\text{E}(\text{params}^{(\Pi)}, M)\), and
- \((K', C', T') := \Pi.\text{E}(\text{params}^{(\Pi)}, M')\).

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

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

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

\textbf{Tag Correctness.} Let \(M, M' \in \mathcal{M}^{(\Pi)}\). Suppose:
- \((K, C, T) := \Pi.\text{E}(\text{params}^{(\Pi)}, M)\), and
- \((K', C', T') := \Pi.\text{E}(\text{params}^{(\Pi)}, M')\).

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

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

\begin{figure}[h]
\centering
\begin{align*}
\text{Game PRV-CDA}^{(\Pi)}_{\mathcal{A}, \mathcal{B}}(1^\lambda, b) &:
\begin{cases}
(M_0, M_1, Z) \xleftarrow{\$} \mathcal{S}(1^\lambda); \\
\text{for } (i := 1, 2, \ldots, m(1^\lambda)) &:
(K^{(i)}, C^{(i)}, T^{(i)}) := \Pi.\text{E}(\text{params}^{(\Pi)}, M^{(i)}_b); \\
&: b' = \mathcal{A}(1^\lambda, C, T, Z); \\
&: \text{return } b'; \\
\end{cases}
\end{align*}
\end{figure}

\begin{figure}[h]
\centering
\begin{align*}
\text{Game STC}^{(\Pi)}_{\mathcal{A}}(1^\lambda) &:
\begin{cases}
(M, C', T') := \mathcal{A}(1^\lambda); \\
\text{If } (M = \bot) \vee (C' = \bot) &: \text{return } 0; \\
\text{Else } (K, C, T) := \Pi.\text{E}(\text{params}^{(\Pi)}, M); \\
M' := \Pi.\text{D}(\text{params}^{(\Pi)}, K, C, T); \\
&: (T = T') \wedge (M' = M'') \wedge (M' \neq \bot) \text{return } 1; \\
&: \text{return } 0; \\
\end{cases}
\end{align*}
\end{figure}

\begin{figure}[h]
\centering
\begin{align*}
\text{Game KR-CDA}^{(\Pi)}_{\mathcal{A}, \mathcal{B}}(1^\lambda) &:
\begin{cases}
(M, Z) \xleftarrow{\$} \mathcal{S}(1^\lambda); \\
\text{(K, C, T)} := \Pi.\text{E}(\text{params}^{(\Pi)}, M); \\
K' := \mathcal{A}(1^\lambda, C, T, Z); \\
&: \text{return } K' \wedge \text{return } 0; \\
\end{cases}
\end{align*}
\end{figure}

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

\textbf{Privacy.} Let \(\Pi = (\Pi.\text{E}, \Pi.\text{D})\) be an \textit{MLE} scheme. Since, no \textit{MLE} scheme can provide \textit{privacy} security for predictable messages (even if the scheme is randomized), we use an
unpredictable message source $S$, 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 $M_0 = (M_0^{(1)} \circ M_0^{(2)} \circ \cdots \circ M_0^{(m(1^\lambda))})$ and $M_1 = (M_1^{(1)} \circ M_1^{(2)} \circ \cdots \circ M_1^{(m(1^\lambda))})$, and some auxiliary information $Z$ using the source $S(1^\lambda)$, encrypts the string $M_0^i$, where $i \in [m(1^\lambda)]$ and the value of $b$ depends upon the input, using $\Pi.E$ to obtain $(K^{(i)}, C^{(i)}, T^{(i)})$, and sends $(C, T, Z)$ to the adversary. The adversary has to return a bit $b'$ indicating whether the ciphertext $C$ and tag $T$ corresponds to message $M_0$ or message $M_1$. If the values of $b$ and $b'$ coincide, then the adversary wins the game.

Now, we define the advantage of a PRV-CDA adversary $A$ against $\Pi$ as:

$$Adv_{\Pi.S,A}^{\text{PRV-CDA}}(1^\lambda) \overset{df}{=} \left| \Pr_{[\Pi]}[\text{PRV-CDA}_{\Pi.S}(1^\lambda, b = 1)] - \Pr_{[\Pi]}[\text{PRV-CDA}_{\Pi.S}(1^\lambda, b = 0) = 1] \right|.$$  

An MLE scheme $\Pi$ is said to be PRV-CDA secure over a set of valid PT sources for MLE scheme $\Pi$, $\mathcal{S} = \{S_1, S_2, \ldots \}$, for all PT adversaries $A$ and for all $S_i \in \mathcal{S}$, if $Adv_{\Pi.S_i,A}^{\text{PRV-CDA}}(\cdot)$ is negligible. An MLE scheme $\Pi$ is said to be PRV-CDA secure, for all PT adversaries $A$, if $Adv_{\Pi,S,A}^{\text{PRV-CDA}}(\cdot)$ is negligible, for all valid PT source $S$ for $\Pi$.

**TAG CONSISTENCY.** Let $\Pi = (\Pi.E, \Pi.D)$ 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 hampering the integrity. In an erasure attack, the adversary replaces the ciphertext with a fake message that decrpyts 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 $\Pi.E$, and computes the message $M'$ corresponding to key $K$, ciphertext $C'$ and tag $T'$ using $\Pi.D$. If the two tags are equal, i.e. $T = T'$, the message $M'$ is valid, i.e. $M' \neq \bot$, 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 $A$ against $\Pi$ as:

$$Adv_{\Pi,A}^{\text{TC}}(1^\lambda) \overset{df}{=} \Pr_{[\Pi]}[\text{TC}_{\Pi}(1^\lambda) = 1].$$

Now, we define the advantage of an STC adversary $A$ against $\Pi$ as:

$$Adv_{\Pi,A}^{\text{STC}}(1^\lambda) \overset{df}{=} \Pr_{[\Pi]}[\text{STC}_{\Pi}(1^\lambda) = 1].$$

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

**KEY RECOVERY.** Let $\Pi = (\Pi.E, \Pi.D)$ 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 $S$, 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 $S(1^\lambda)$, encrypts $M$ using $\Pi.E(\text{params}^{(\Pi)}, \cdot)$ 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 $\Pi$ as:

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

An MLE scheme $\Pi$ is said to be KR-CDA secure, if $Adv^{KR\text{-CDA}}_{\Pi,S,\mathcal{A}}(\cdot)$ is negligible, for all valid PT source $S$ 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 $\Pi$. $\Pi$ satisfies the following conditions:

1. The PPT setup algorithm $\Pi.\text{Setup}(1^\lambda)$ outputs the parameter $\text{params}^{(\Pi)}$ 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 $\Pi.\mathcal{K}_\text{GEN}: \mathbb{N} \rightarrow \mathcal{K}^{(\Pi)}$ takes as input the parameter $\text{params}^{(\Pi)}$, and outputs $K := \Pi.\mathcal{K}_\text{GEN}(\text{params}^{(\Pi)})$, where $K \in \mathcal{K}^{(\Pi)}$.

3. The PPT encryption algorithm $\Pi.\mathcal{E}: \mathcal{K}^{(\Pi)} \times \mathcal{M}^{(\Pi)} \rightarrow \mathcal{C}^{(\Pi)} \times \mathcal{T}^{(\Pi)}$ takes as inputs the parameter $\text{params}^{(\Pi)}$, $K \in \mathcal{K}^{(\Pi)}$ and $M \in \mathcal{M}^{(\Pi)}$, and outputs a pair $(C, T) := \Pi.\mathcal{E}(\text{params}^{(\Pi)}, K, M)$, 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 $\Pi.\mathcal{D}: \mathcal{K}^{(\Pi)} \times \mathcal{C}^{(\Pi)} \times \mathcal{T}^{(\Pi)} \rightarrow \mathcal{M}^{(\Pi)} \cup \{\bot\}$ is a deterministic algorithm that takes as inputs the parameter $\text{params}^{(\Pi)}$, $K \in \mathcal{K}^{(\Pi)}$, $C \in \mathcal{C}^{(\Pi)}$ and $T \in \mathcal{T}^{(\Pi)}$, and returns $\Pi.\mathcal{D}(\text{params}^{(\Pi)}, K, C, T) \in \mathcal{M}^{(\Pi)} \cup \{\bot\}$. 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:

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

Then decryption correctness of $\Pi$ requires that $\Pi.\mathcal{D}(\text{params}^{(\Pi)}, K, C, T) = M$, 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 $\Pi.\mathcal{K}_\text{GEN}(\text{params}^{(\Pi)})$ 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.
Now, we define the advantage of an IND-PRV adversary $A$ against $\Pi$ as:

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

An AE scheme $\Pi$ is said to be IND-PRV secure, for all PT adversaries $A$, if $Adv_{\Pi,A}^{\text{IND-PRV}}(\cdot)$ 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 $\Pi.\mathcal{K}_\text{GEN}(\text{params}(\Pi))$ 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 $\Pi.\mathcal{D}(\text{params}(\Pi), K, C_i, T)$.

Now, we define the advantage of an INT adversary $A$ against $\Pi$ as:

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

An AE scheme $\Pi$ is said to be INT secure, for all PT adversaries $A$, if $Adv_{\Pi,A}^{\text{INT}}(\cdot)$ 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 $\Pi = (\Pi.\mathcal{G\mathcal{E}\mathcal{N}}, \Pi.\mathcal{D\mathcal{E}\mathcal{R}})$ is a pair of algorithms over a setup algorithm $\Pi.\text{Setup}$. $\Pi$ satisfies the following conditions.

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

2. The PPT key generation algorithm $\Pi.\mathcal{G\mathcal{E}\mathcal{N}}$ takes as inputs the parameter $\text{params}(\Pi)$ and graph $G$, and returns a three-tuple $(S, k, pub) := \Pi.\mathcal{G\mathcal{E}\mathcal{N}}(\text{params}(\Pi), G)$, 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.

3. 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 $\Pi.\mathcal{D\mathcal{E}\mathcal{R}}$ is a deterministic PT algorithm such that $k_u := \Pi.\mathcal{D\mathcal{E}\mathcal{R}}(\text{params}(\Pi), G, u, v, S_u, pub)$, 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 $\Pi$ requires that for all $\lambda \in \mathbb{N}$, all $G \in \Gamma(\Pi)$, all $(S,k,pub)$ output by $\Pi.\mathcal{GE}(\text{params}(\Pi), G)$, and all nodes $v \leq u$, we have:

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

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

**Figure 3:** Games defining $\text{KI-ST, S-KI-ST}$ and $\text{KR-ST}$ security of $\text{KAS}$ scheme $\Pi = (\Pi.\mathcal{GE}, \Pi.\mathcal{DER})$. Here, the adversary $A = (A_1, A_2)$.

**Key Indistinguishability.** Let $\Gamma(\Pi)$ be a set of access graphs and $\Pi = (\Pi.\mathcal{GE}, \Pi.\mathcal{DER})$ be the $\text{KAS}$ for $\Gamma(\Pi)$. For a $\text{KAS}$ we have designed a key indistinguishability with respect to static adversary $\text{KI-ST}$ (and strong key indistinguishability with respect to static adversary $\text{S-KI-ST}$) security game in Figure 3. The static adversary $A$, when given access to the graph $G = (V,E)$, returns a security class $u \in V$ to the challenger, that $A$ chooses to attack. The challenger then performs the following operations: calculates $(S,k,pub)$ using the $\Pi.\mathcal{GE}(\text{params}(\Pi), G)$; 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 $\text{KI-ST}$ adversary $A$ against $\Pi$ on a graph $G \in \Gamma(\Pi)$ as:

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

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

$$\text{Adv}^{\text{S-KI-ST}}_{\Pi, A, G}(1^\lambda) \overset{\text{def}}{=} \Pr[\text{S-KI-ST}_{\Pi}^{\mathcal{A}}(1^\lambda, G, b = 1) = 1] - \Pr[\text{S-KI-ST}_{\Pi}^{\mathcal{A}}(1^\lambda, G, b = 0) = 1].$$
A KAS scheme $\Pi$ is said to be KI-ST (or S-KI-ST) secure, for all PT static adversaries $A$, if $\text{Adv}_{\Pi,A,G}^{\text{KI-ST}}(\cdot)$ (or $\text{Adv}_{\Pi,A,G}^{\text{S-KI-ST}}(\cdot)$) is negligible.

**Key Recovery.** Let $\Gamma[\Pi]$ be a set of access graphs and $\Pi = (\Pi, \mathcal{EN}, \Pi, \mathcal{DER})$ 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 $A$, when given access to the graph $G = (V,E)$, returns a security class $u \in V$ to the challenger, that $A$ chooses to attack. The challenger then performs the following operations: calculates $(S, k, pub)$ using the $\Pi.\mathcal{EN}(\text{params}[\Pi], G)$; 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 $A$ against $\Pi$ on a graph $G \in \Gamma[\Pi]$ as:

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

A KAS scheme $\Pi$ is said to be KR-ST secure, for all PT static adversaries $A$, if $\text{Adv}_{\Pi,A,G}^{\text{KR-ST}}(\cdot)$ 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 $\downarrow u = \{v \in V \mid v \leq u\}$. 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 $O(|V| + |E|)$.

- **ch_seq($u, G$):** Given the node $u$ and graph $G$ as input, this function outputs a sequence of nodes $\tilde{u} = (u_1, u_2, \cdots, u_m)$ — that are children of node $u$ in $G$ — in the ascending order of their indices. Therefore, $u_1$ has the lowest index, $u_2$ 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 $O(|V| + \deg(u) \log(\deg(u)))$.

- **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 $\text{level}[u]$ the maximum level of node $u$ for all $u \in V$, and then returns $\text{level}[]$ and $h = \max_{u \in V} \text{level}[u]$. We, first, find the root node $u$ of the graph and assign the $\text{level}[u] = 1$. 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 $\text{level}[]$ value with the current value. Since, the graph is acyclic, the value of $\text{level}[v]$, for all $v \in V$, can be at most $n$. We calculate the height of the graph $h = \max_{u \in V} \text{level}[v]$. The running time of $\text{height}(G)$ is $O(|V|^2 + |V| + |E|)$.

$max_{\text{isect}}(u, C)$: 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 $\text{all\_succ}(u, G)$ 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 $\text{max\_isect}(u, C)$ is $O(|V| + |E|)$.

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

$\text{nodes\_at\_level}(V, \text{level}[], x)$: This function takes a graph $G$, the array $\text{level}[]$ 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 $\text{level}[]$, during the execution of $\text{height}(G)$ 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 $\text{nodes\_at\_level}(V, \text{level}[], x)$ is $O(|V|)$.

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

$\text{path}(G, u, v)$: This function takes as input a graph $G$, the source and the destination nodes $u$ and $v$, and outputs a sequence of nodes $(u, u_1, u_2, \ldots, u_{i_1}, v)$ such that $u_1 < u, u_2 < u_1, \ldots, v < u_{i_1}$. In order to do this, we invoke the Dijkstra’s Algorithm on graph $G$ with $u$ as source node, and get the array $\text{dist}[]$, defining the distance of any node from $u$, and array $\text{parent}[]$, defining the parent of any node in the graph [Dij59]. Then, we start to find the parent of $v$ as $u_{i_1}$, then the parent of $u_{i_1}$ as $u_{i_1-1}$, and so on, until we find the parent of $u_1$ as $u$. So, the path from $u$ to $v$ is $(u, u_1, u_2, \ldots, u_{i_1}, u_{i_1+1} = v)$. The running time of $\text{path}(G, u, v)$ is $O(|E| + |V| \cdot \log |V| + |V|)$.

$\text{vertex\_in\_order}(G)$: This function takes as input the access graph $G = (V, E)$ corresponding to a totally ordered set, and outputs a sequence of nodes $(u_1, u_2, \ldots, u_n)$ such that $u_n < u_{n-1} < \cdots < u_1$, 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 $(u_1, u_2, \ldots, u_n)$. The running time of $\text{vertex\_in\_order}(G)$ is $O(|V|^2 + |E|)$. 

1. Constructing KAS

We refer the reader to [2.3.1 Existing AE] 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 se-

---

```plaintext
all_succ(u, G)
U := \{u\}; Q := \emptyset;
ENQUEUE(Q, u);
while Q ≠ \emptyset
  node := DEQUEUE(Q);
  for all (node, v) ∈ E
    U := U ∪ {v};
    ENQUEUE(Q, v);
  return U;

nodes_at_level(V, level[\_], x)
U := \emptyset;
for all u ∈ V
  if level[u] = x
    U := U ∪ \{u\};
  return U;

path(G, u, v)
  dist[\_], parent[\_]
  := DIJKSTRA(G, u);
  path := (v, w := v);
while w ≠ u
  path := (parent[w] \_ o path);
  w := parent[w];
return path;

max_isect(u, C)
\[ \downarrow u := all_succ(u, G); \]
\[ U := (u_1, u_2, \ldots, u_i); \]
\[ \max := 1; \]
for \( j := 2, 3, \ldots, \ell \)
  if \( u_{i_{\max}} ≤ u_j \)
    \[ \max := j; \]
    return \( u_{i_{\max}} \);

max_isect_chs(u, G)
\[ \downarrow u := all_succ(u, G); \]
\[ W := \emptyset; \]
for \( k := 1, 2, \ldots, \ell \)
  \[ U := (u_1, u_2, \ldots, u_i); \]
  \[ \max := 1; \]
  for \( j := 2, 3, \ldots, \ell \)
    if \( u_{i_{\max}} ≤ u_j \)
      \[ \max := j; \]
      \[ W := (W ∪ u_{i_{\max}}); \]
return W;

height(G)
u := FIND_ROOT(G);
level[u] := 1, h := 1, Q := \emptyset;
ENQUEUE(Q, u);
while Q ≠ \emptyset
  u := DEQUEUE(Q);
  for all (u, v) ∈ 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, BDP09, BDPM+14, BRW04, R02] 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 se-
lected; (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; 
- $S_u := (k_u)_{v \leq u}$; 
- $pub := \emptyset$; 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 $k_u$; 
- $S_u := (K_v)_{v \leq u}$; 
- $pub := (K_u(k_u))_{u \in V}$; and 
- $k_v$ is obtained by decrypting $K_u(k_u) \in pub$ using $K_u \in S_u$.

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

**Scheme 4: IKEKAS.** An *iterative key-encrypting-key assignment scheme* (IKEKAS) has the following properties:
- All $k_u$’s are chosen independently; 
- $S_u := k_v$; 
- $pub := (K_u(k_v))_{v < u, u \in V}$; and 
- there exists a path $(u, z_0, z_1 \cdots z_m, v)$ and we calculate $k_u := D_{k_{u}}(K_{u}(z_{m})), k_{z_1} := D_{k_{z_1}}(K_{k_{z_1}}(z_{2})), \cdots, k_{v} := D_{k_{v}}(K_{u}(z_{m})), to obtain k_v.$

**Scheme 5: NBKAS.** A *node-based key assignment scheme* (NBKAS) has the following properties:
- $k_u := f(e_u)$ are keys such that $g(f(e_u), k_u, e_v) = k_v$; 
- $S_u := k_v$; 
- $pub := (e_u)_{u \in V}$; 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 $l_{\text{max}}$. The chain partition algorithm $\Pi = (\Pi.\mathcal{GEN}, \Pi.\mathcal{DER})$ is a pair of algorithms over a setup algorithm $\Pi.\text{Setup}$. $\Pi$ satisfies the following conditions:

1. The PPT setup algorithm $\Pi.\text{Setup}(1^\lambda)$ outputs the parameter $\text{params}^{(\Pi)}$, 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 II, $\mathcal{G}\mathcal{E}\mathcal{N}$ takes as inputs the parameter $\text{params}(\Pi)$, the access graph $G = (V, E) \in \Gamma(\Pi)$, and the KAS-chain $\pi$, 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 II, $\mathcal{D}\mathcal{E}\mathcal{R}$ is a deterministic PT algorithm such that

$k^*_h := \Pi.\mathcal{D}\mathcal{E}\mathcal{R}(\text{params}(\Pi), G, u^*_j, u^*_h, S_{u^*_j}, pub^*_g, \pi)$. Here: $u^*_h \leq u^*_j$ are two nodes of the access graph $G$; $S_{u^*_j}$ is $u^*_j$’s private information; $pub^*_g$ is the public information; $\pi$ is the KAS-chain; and $k^*_h$ is $u^*_h$’s decryption key.

Note that $S_{u^*_j} \in \{0,1\}^*$, $pub^*_g \in \{0,1\}^*$ and $u^*_h \in \mathcal{K}(\Pi) \cup \bot$.

The pseudo-code for the chain partition algorithm II 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.

![Figure 5: Chain partition algorithm for building KAS. The functions partition, max_isect_chs and max_isect are described in Subsubsection 2.2.8.](image)

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 II = ($\Pi.\mathcal{E}$, $\Pi.\mathcal{D}\mathcal{E}\mathcal{R}$, $\Pi.\mathcal{D}$) is a three-tuple of algorithms over a setup algorithm $\Pi.\text{Setup}$. $\Pi$ satisfies the following conditions.
1. The PPT setup algorithm II.Setup(1^λ) outputs the parameter \textit{params}^{(II)}, a set of access graphs \( \Gamma^{(II)} \) and the sets \( \mathcal{K}^{(II)} \) and \( \mathcal{M}^{(II)} \), denoting the key and message spaces respectively.

2. The PPT encryption algorithm II.\( \mathcal{E} \) takes as inputs the parameter \textit{params}^{(II)}, a graph \( G \in \Gamma^{(II)} \) and a vector of files \( f = (f_u)_{u \in V} \), and returns a three-tuple \( (S, k, pub) := II.\mathcal{E}(\textit{params}^{(II)}, G, f) \), 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.

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

4. The key derivation algorithm II.\( \mathcal{D}\mathcal{E}R \) is a deterministic PT algorithm such that \( k_v := II.\mathcal{D}\mathcal{E}R(\textit{params}^{(II)}, G, u, v, S_u, pub) \). Here: node \( u \) decrypts the ciphertext corresponding to node \( v \) such that \( v \leq u \) in the access graph \( G \); \( S_u \) is \( u \)'s private information; \( pub \) is the public information; and \( k_v \) is \( v \)'s decryption key.

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.

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

- \( II.\mathcal{D}\mathcal{E}R(\textit{params}^{(II)}, G, u, v, S_u, pub) = k_v \), and
- \( II.\mathcal{D}(\textit{params}^{(II)}, G, u, v, S_u, pub) = f_v \).

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 \( K\text{-ST} \& S-K\text{-ST}, K\text{-ST}, \text{IND-PRV} \) and INT games. However, the notion of \textit{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.

Footnote: 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.
Figure 6: Games defining KR-ST, IND-PRV and INT security for KAS-AE \( \Pi = (\Pi, E, I, D_E, I, D) \). Here, in KR-ST and IND-PRV games, the adversary \( A = (A_1, A_2) \).

**Key Recovery.** Let \( \Gamma^{(\Pi)} \) be a set of access graphs and \( \Pi = (\Pi, E, I, D_E, I, D) \) 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 \( A \), when given access to the graph \( G = (V, E) \), returns a security class \( u \in V \), that \( A \) 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 \( \Pi. E(params^{(\Pi)}), G, f) \); 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 \( A \) against \( \Pi \) on a graph \( G \in \Gamma^{(\Pi)} \) as:

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

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

**Privacy.** Let \( \Gamma^{(\Pi)} \) be a set of access graphs and \( \Pi = (\Pi, E, I, D_E, I, D) \) 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 \( A \), when given access to the graph \( G = (V, E) \), returns two sequences of files \( f^0 \) and \( f^1 \), such that \( \forall u \in V, |f^0_u| = |f^1_u| \). 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 \( A \) against \( \Pi \) on a graph \( G \in \Gamma^{(\Pi)} \) as:

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

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

**Tag Consistency.** Let \( \Gamma^{(\Pi)} \) be a set of access graphs and \( \Pi = (\Pi, E, I, D_E, I, D) \) 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\cite{FPP13}.
public information vectors $pub^0$ and $pub^1$, secret information vector $S$ and key vector $k$ from the adversary. The challenger computes files $f_u^0 := \Pi. D(params^{(I)}(G, u, u, S_u, pub^0))$ and $f_u^1 := \Pi. D(params^{(I)}(G, u, u, S_u, pub^1))$. The adversary wins if both the files are valid, i.e., $f_u^0 \neq \bot$ and $f_u^1 \neq \bot$, and the two files are unidentical, i.e. $f_u^0 \neq f_u^1$.

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

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

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

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 $\Pi = (\Pi. E, \Pi. D\mathcal{E}R, \Pi. D)$ is constructed below using the KAS $\Psi = (\Psi. GEN, \Psi. D\mathcal{E}R)$ and the AE scheme $\Omega = (\Omega. KGen, \Omega. E, \Omega. D)$.

![Figure 7: Algorithmic description of the KAS-AE scheme of Subsection 4.1: simple combination of KAS and AE.](image)

A simple attack on the TAG CONSISTENCY security of $\Pi$. 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 $\Pi = (\Pi.\mathcal{E}, \Pi.\mathcal{D}, \Pi.\mathcal{R}, \Pi.\mathcal{D})$ be the KAS-AE scheme, $\Psi = (\Psi.\mathcal{G}\mathcal{E}\mathcal{N}, \Psi.\mathcal{D}, \Psi.\mathcal{R})$ be the KAS and $\Omega = (\Omega.\mathcal{G}\mathcal{E}\mathcal{N}, \Omega.\mathcal{E}, \Omega.\mathcal{D})$ 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 $\Pi$ is shown in Figure 8. Here, it is important to note that $\Gamma^{(\Pi)} = \Gamma^{(\Psi)}$ and $\text{params}^{(\Pi)} = (\text{params}^{(\Psi)}, \text{params}^{(\Omega)})$.

\begin{figure}[h]
\centering
\begin{tabular}{|l|}
\hline
\Pi.\mathcal{E}(\text{params}^{(\Pi)}, G, f) \\
\hline
$(S, k, pub) := \Psi.\mathcal{G}\mathcal{E}\mathcal{N}(\text{params}^{(\Psi)}, G)$; \\
for $u \in V$ \\
$(c_u, t_u) := \Omega.\mathcal{E}(\text{params}^{(\Omega)}, k_u, f_u)$; \\
for $v \in \downarrow u$ \\
$S_v := S_u \circ t_u$; \\
e := $(c_v)_{u \in V}$; \\
$S := (S_u)_{u \in V}$; \\
pub := $f(pub)$; \\
return $(S, k, pub)$; \\
\hline
\Pi.\mathcal{D}(\text{params}^{(\Pi)}, G, u, v, S_u, pub) \hline
$k_v := \Psi.\mathcal{D}, \mathcal{R}(\text{params}^{(\Psi)}, G, u, v, S_u, pub)$; \\
return $k_v$; \\
\hline
\Pi.\mathcal{D}(\text{params}^{(\Pi)}, G, u, v, S_u, pub) \hline
$k_v := \Psi.\mathcal{D}, \mathcal{R}(\text{params}^{(\Psi)}, G, u, v, S_u, pub)$; \\
t_v := \text{ext\_tag}(S_v, v); \\
c_v := \text{ext\_cipher}(pub, v); \\
f_v := \Omega.\mathcal{D}(\text{params}^{(\Omega)}, k_v, c_v, t_v); \\
return $f_v$; \\
\hline
\end{tabular}
\caption{A Framework for building a KAS-AE scheme from KAS and AE schemes, used separately. By plugging an existing KAS scheme $X \in \{\text{TKAS, TKEKAS, DKEKAS, IKEKAS, NBKAS}\}$ 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.}
\end{figure}

**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 $A$ can break the KR-ST (or IND-PRV or INT) security of KAS-AE, then an adversary $B$, using $A$, 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, $A_{\text{Chain}}, B_{\text{Chain}}, C_{\text{Chain}}$ and $D_{\text{Chain}}$, based on KAS-chain [FPP13] & AE [Rog02, BRW03, BN08], MLE [BKR13], APE [ABB14] and FP [PHG12] respectively.
5.1.1 AChain: KAS-AE-chain based on KAS-chain and AE

An AChain $\Pi = (\Pi. E, \Pi. D\mathcal{E}R, \Pi. D)$ is a KAS-AE-chain built from a KAS-chain $\Psi = (\Psi. \mathcal{G}\mathcal{E}\mathcal{N}, \Psi. D\mathcal{E}R)$ and an AE scheme $\Omega = (\Omega. \mathcal{G}E\mathcal{N}, \Omega. E, \Omega. D)$ following the framework described in Figure 9.

![Diagram of AChain](image)

(a) Algorithmic description of building AChain $\Pi$ using the KAS-chain $\Psi$ and AE scheme $\Omega$. 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 $\Pi. E(params^{(i)}, G, f)$, where $f = (f_1 \circ f_2 \circ f_3)$ and $G$ is shown in 9(b).

(d) Pictorial description of $\Pi. D\mathcal{E}R(params^{(i)}, G, u, v, S_u, pub)$.

(e) Pictorial description of $\Pi. D(params^{(i)}, G, u, v, S_u, pub)$.

Figure 9: Building AChain.

Security of AChain

Theorem 2. If the underlying KAS-chain scheme is KR-ST secure, then the Construction AChain is also KR-ST secure.

Proof. The proof is by using reduction. So, we can show that if an adversary $A$ can break the KR-ST security of Construction AChain, then an adversary $B$, using $A$, can break the KR-ST security of the underlying KAS-chain $\Psi$. By using the contrapositive argument, this would show that if the underlying KAS is secure, so is the Construction AChain.

Theorem 3. If the underlying AE scheme is IND-PRV secure, then the Construction AChain is also IND-PRV secure.

Proof. The proof is by using reduction. So, we can show that if an adversary $A$ can break the IND-PRV security of Construction AChain, then an adversary $B$, using $A$, can break the IND-PRV security of the underlying AE $\Omega$. By using the contrapositive argument, this would show that if the underlying AE is secure, so is the Construction AChain.
Theorem 4. If the underlying AE scheme is INT secure, then the Construction $A_{\text{Chain}}$ is also INT secure.

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

5.1.2 $B_{\text{Chain}}$: KAS-AE-chain based on MLE

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

![Algorithmic description of building $B_{\text{Chain}}$ $\Pi$ using the MLE scheme $\Psi$. For the pictorial description with an example, see 10(b)–10(c).](image1)

![The access graph $G$ with 3 nodes $u_1, u_2, u_3$ and their corresponding files $f_1, f_2, f_3$.](image2)

![Pictorial description of $\Pi.\mathcal{E}(\text{params}^{(\Pi)}, G, f)$.](image3)

![Pictorial description of $\Pi.\mathcal{D}(\text{params}^{(\Pi)}, G, u, v, S_{u}, \text{pub})$.](image4)

![Pictorial description of $\Pi.\mathcal{D}(\text{params}^{(\Pi)}, G, u, v, S_{u}, \text{pub})$.](image5)

Figure 10: Building $B_{\text{Chain}}$.

Security of Construction $B_{\text{Chain}}$

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

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

Our message source S\textsubscript{u,m} works in the following way: for i = m, m - 1, \cdots, 1, generate a message f\textsubscript{i} and a \lambda-bit random number R\textsubscript{i}, and computes (k\textsubscript{i}, c\textsubscript{i}, t\textsubscript{i}) := \Psi \cdot E(\mathcal{P}arams(\Psi), R\textsubscript{i}||k\textsubscript{i+1}||f\textsubscript{i}), where R\textsubscript{m+1} := \epsilon; creates Z := c\textsubscript{1}||f\textsubscript{1}||k\textsubscript{2}||c\textsubscript{2}||f\textsubscript{2}||\cdots ||c\textsubscript{u-1}||f\textsubscript{u-1}||c\textsubscript{u}||f\textsubscript{u}||k\textsubscript{u+1}||k\textsubscript{u+2}||\cdots ||k\textsubscript{m} and message M = R\textsubscript{u}||k\textsubscript{u+1}||f\textsubscript{u}; and returns (M, Z). Here, u is the security class that A chooses to attack and m is the number of nodes (or security classes) in the graph G.

\[\text{Figure 11: The reduction used in Theorem 5: MLE adversary is constructed using KAS-AE-chain adversary.}\]

**Theorem 6.** If the underlying MLE scheme is PRV-CDA secure, then the Construction B\textsubscript{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 A can break the IND-PRV security of Construction B\textsubscript{Chain}, then an adversary B, using A, can break the PRV-CDA security of the underlying MLE scheme \Psi. By using the contrapositive argument, this would show that if the underlying MLE scheme is secure, so is the Construction B\textsubscript{Chain}.

Our message source S\textsubscript{f,i} 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 \lambda-bit random number R\textsubscript{b} and computes M\textsubscript{b}(i) := R\textsubscript{b}||k\textsubscript{b}(i+1)||f\textsubscript{b}(i) and (k\textsubscript{b}(i), c\textsubscript{b}(i), t\textsubscript{b}(i)) := \Psi \cdot E(\mathcal{P}arams(\Psi), M\textsubscript{b}(i)); and returns the sequence of strings M\textsubscript{0} := (M\textsubscript{0}(1) \circ M\textsubscript{0}(2) \circ \cdots \circ M\textsubscript{0}(m)) and M\textsubscript{1} := (M\textsubscript{1}(1) \circ M\textsubscript{1}(2) \circ \cdots \circ M\textsubscript{1}(m)) along with auxiliary information Z. Here, m is the number of nodes (or security classes) in the graph G and the adversary A generates two sequence of files f\textsubscript{0} and f\textsubscript{1} such that for i = m, m - 1, \cdots, 1, |f\textsubscript{0}(i)| = |f\textsubscript{1}(i)|, which results into \(|M\textsubscript{0}(i)| = |M\textsubscript{1}(i)|\) when S\textsubscript{f,i}(1\textsuperscript{\lambda}) generates M\textsubscript{0}, M\textsubscript{1} and Z.

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

*Proof.* The proof is by using reduction as shown in Figure 13. So, we show that if an adversary A can break the INT security of Construction B\textsubscript{Chain}, then an adversary B, using A, can break the TC security of the underlying MLE scheme \Psi. By using the
contrapositive argument, this would show that if the underlying MLE scheme is secure, so is the Construction $B_{\text{Chain}}$.

\[ \text{Challenger}_{\Pi}^{b'}(1^\lambda) \]

\[ \text{B}_{\Pi}(1^\lambda) \]

\[ \text{Challenger}_{\Pi}^{b'}(1^\lambda, G) \]

\[ \text{B}_{\Pi}(1^\lambda, G) \]

\[ \text{Challenger}_{\Pi}^{b'}(1^\lambda, G, b) \]

\[ \text{B}_{\Pi}(1^\lambda, G, b) \]

\[ \text{Figure 12:} \text{ The reduction used in Theorem 6: MLE adversary is constructed using KAS-AE-chain adversary.} \]

\[ \text{Figure 13:} \text{ The reduction used in Theorem 7: MLE adversary is constructed using KAS-AE-chain adversary.} \]

5.1.3 $C_{\text{Chain}}$: KAS-AE-chain based on APE

Functionalities based on APE

In this section, we are designing two functionalities – $F_1^\ast$ and $F_2^\ast$ – 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
KAS-AE-chain scheme based on functionalities $F_1^\pi$ and $F_2^\pi$

$F_1^\pi(1^\lambda, M, IV^1, IV^2)$
\begin{align*}
p := |M|/\lambda, & \quad s_0 := IV^1; \\
m_1[m_2]|\cdots|m_p := M; \\
R \leftarrow \{0, 1\}^\lambda, & \quad m_0 := R; \\
\text{for } (j := 0, 1, \cdots, p) \\
\quad r_j := m_j \oplus r_j; \\
\quad (r_j||s_{j+1}) := \pi(r_j||s_j); \\
C := r_1||r_2||\cdots||r_p, & \quad K := s_{p+1}; \\
\text{return } (K, C); \end{align*}

$F_2^\pi(1^\lambda, K, C)$
\begin{align*}
p := |C|/\lambda - 1, & \quad s_{p+1} := K; \\
r_1||r_2||\cdots||r_{p+1} := C; \\
\text{for } (j := p, p - 1, \cdots, 0) \\
\quad (r_j'||s_j) := \pi^{-1}(r_{j+1}||s_{j+1}); \\
\text{If } j \neq 0, \text{ then } m_j := r_j || r_j'; \\
M := m_1[m_2]|\cdots|m_p, IV^2 := s_0; \\
\text{return } (M, IV^2); \end{align*}

Figure 14: Algorithmic and diagrammatic descriptions of the functionalities $F_1^\pi$ and $F_2^\pi$ are shown in (a), (b) and (c); here, $\pi$ is a 2\lambda-bit easy-to-invert permutation. Each wire in (b) and (c) represents \lambda bits. The function $F_1^\pi$ takes as inputs parameter $1^\lambda$, message $M$ and two other values $IV^1$ and $IV^2$, and returns the decryption key $K$ and the ciphertext $C$. Similarly, $F_2^\pi$ takes as inputs parameter $1^\lambda$, 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 $\lambda$.

Security of Construction $C_{\text{Chain}}$

Theorem 8. If $\pi$ is the ideal permutation in Construction $C_{\text{Chain}}$, then

\[
\text{Adv}^{\text{KR-ST}}_{C_{\text{Chain}}}(1^\lambda) \leq \frac{\sigma(\sigma - 1)}{2^{2\lambda-1}} + \frac{\sigma(\sigma - 1)}{2^\lambda}
\]

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 $C_{\text{Chain}}$ is used. (see Figure 15).

Game 1: This Game 1 is identical to Game 0 except that we replace the 2\lambda-bit permutation $\pi$ with 2\lambda-bit random function $\kappa$.

Using PRP/PRF Switching Lemma [BR06], for an adversary limited by $\sigma$ queries to the permutation (or random function), the following equation can be obtained.
(d) Pictorial description of II. $\mathcal{DER}(\text{params}(\Pi), G, u_1, u_3, S_1, pub)$.

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

Figure 15: Building $C_{\text{Chain}}$. 
\[
\left| \text{Adv}^{\text{Game 0}}_{\Pi^r,\sigma^{-1},A,G}(1^\lambda) - \text{Adv}^{\text{Game 1}}_{\Pi^r,\sigma^{-1},A,G}(1^\lambda) \right|
= \left| \text{Adv}^{\text{KR-ST}}_{\Pi^r,\sigma^{-1},A,G}(1^\lambda) - \text{Adv}^{\text{KR-ST}}_{\Pi^r,\sigma^{-1},A,G}(1^\lambda) \right|
\leq \text{Adv}^{\text{IND-PRV}}_{\pi,rf,A}(1^\lambda, \sigma) \leq \frac{\sigma(\sigma - 1)}{2^\lambda}
\]

(1)

**Game 2:** This Game 2 is identical to Game 1 except that here we change 2\(\lambda\)-bit permutation \(\pi^{-1}\) by a 2\(\lambda\)-bit random function \(rf\).

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

\[
\left| \text{Adv}^{\text{Game 1}}_{\Pi^r,\sigma^{-1},A,G}(1^\lambda) - \text{Adv}^{\text{Game 2}}_{\Pi^r,rf^{-1},A,G}(1^\lambda) \right|
= \left| \text{Adv}^{\text{KR-ST}}_{\Pi^r,\sigma^{-1},A,G}(1^\lambda) - \text{Adv}^{\text{KR-ST}}_{\Pi^r,rf^{-1},A,G}(1^\lambda) \right|
\leq \text{Adv}^{\text{IND-PRV}}_{\pi,rf,A}(1^\lambda, \sigma) \leq \frac{\sigma(\sigma - 1)}{2^\lambda}
\]

(2)

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

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

\[
\left| \text{Adv}^{\text{Game 2}}_{\Pi^r,\sigma^{-1},A,G}(1^\lambda) - \text{Adv}^{\text{Game 3}}_{\Pi^r,rf^{-1},A,G}(1^\lambda) \right|
= \left| \text{Adv}^{\text{KR-ST}}_{\Pi^r,\sigma^{-1},A,G}(1^\lambda) - \text{Adv}^{\text{KR-ST}}_{\Pi^r,rf^{-1},A,G}(1^\lambda) \right|
\leq \text{Pr}[A \text{ sets Bad}] \leq \frac{\sigma(\sigma - 1)}{2^\lambda}
\]

(3)

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

\[
\left| \text{Adv}^{\text{Game 0}}_{\Pi^r,\sigma^{-1},A,G}(1^\lambda) - \text{Adv}^{\text{Game 3}}_{\Pi^r,\sigma^{-1},A,G}(1^\lambda) \right|
= \left| \text{Adv}^{\text{Game 0}}_{\Pi^r,\sigma^{-1},A,G}(1^\lambda) - \text{Adv}^{\text{Game 1}}_{\Pi^r,\sigma^{-1},A,G}(1^\lambda) \right|
+ \left| \text{Adv}^{\text{Game 1}}_{\Pi^r,\sigma^{-1},A,G}(1^\lambda) - \text{Adv}^{\text{Game 2}}_{\Pi^r,\sigma^{-1},A,G}(1^\lambda) \right|
+ \left| \text{Adv}^{\text{Game 2}}_{\Pi^r,\sigma^{-1},A,G}(1^\lambda) - \text{Adv}^{\text{Game 3}}_{\Pi^r,\sigma^{-1},A,G}(1^\lambda) \right|
\leq \left| \text{Adv}^{\text{Game 0}}_{\Pi^r,\sigma^{-1},A,G}(1^\lambda) - \text{Adv}^{\text{Game 1}}_{\Pi^r,\sigma^{-1},A,G}(1^\lambda) \right|
+ \left| \text{Adv}^{\text{Game 1}}_{\Pi^r,\sigma^{-1},A,G}(1^\lambda) - \text{Adv}^{\text{Game 2}}_{\Pi^r,\sigma^{-1},A,G}(1^\lambda) \right|
+ \left| \text{Adv}^{\text{Game 2}}_{\Pi^r,\sigma^{-1},A,G}(1^\lambda) - \text{Adv}^{\text{Game 3}}_{\Pi^r,\sigma^{-1},A,G}(1^\lambda) \right|
\leq \frac{\sigma(\sigma - 1)}{2^\lambda} + \frac{\sigma(\sigma - 1)}{2^\lambda} + \frac{\sigma(\sigma - 1)}{2^\lambda}
\]

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

\[
\text{Adv}^{\text{Game 3}}_{\Pi^r,\sigma^{-1},A,G}(1^\lambda) = 0
\]

\[
\text{Adv}^{\text{KR-ST}}_{\Pi^r,\sigma^{-1},A,G}(1^\lambda) \leq \text{Adv}^{\text{Game 0}}_{\Pi^r,\sigma^{-1},A,G}(1^\lambda) \leq \frac{\sigma(\sigma - 1)}{2^\lambda} + \frac{\sigma(\sigma - 1)}{2^\lambda}
\]
Theorem 9. If $\pi$ is the ideal permutation in Construction $C_{\text{Chain}}$, then

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

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

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

Game 1: This Game 1 is identical to Game 0 except that we replace the $2\lambda$-bit permutation $\pi$ with $2\lambda$-bit random function $rf$.

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

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

Game 2: This Game 2 is identical to Game 1 except that here we change $2\lambda$-bit permutation $\pi^{-1}$ by a $2\lambda$-bit random function $rf'$.

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

$$\left| \text{Adv}^{\text{Game 1}}_{\Pi, \pi^{-1}, A, G}(1^\lambda) - \text{Adv}^{\text{Game 2}}_{\Pi, \pi^{-1}, A, G}(1^\lambda) \right| = \left| \text{Adv}^{\text{IND-PRV}}_{\Pi, \pi^{-1}, A, G}(1^\lambda) - \text{Adv}^{\text{IND-PRV}}_{\Pi, \pi^{-1}, A, G}(1^\lambda) \right| \leq \text{Adv}^{\text{IND-PRV}}_{\pi, rf, \lambda}(1^\lambda, \sigma) \leq \frac{\sigma(\sigma - 1)}{2^{2\lambda}} \quad (5)$$

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

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

$$\left| \text{Adv}^{\text{Game 2}}_{\Pi, \pi^{-1}, A, G}(1^\lambda) - \text{Adv}^{\text{Game 3}}_{\Pi, \pi^{-1}, A, G}(1^\lambda) \right| = \left| \text{Adv}^{\text{IND-PRV}}_{\Pi, \pi^{-1}, A, G}(1^\lambda) - \text{Adv}^{\text{IND-PRV}}_{\Pi, \pi^{-1}, A, G}(1^\lambda) \right| \leq \Pr[A \text{ sets } \text{Bad}] \leq \frac{\sigma(\sigma - 1)}{2^\lambda} \quad (6)$$

Using Triangle Inequality [BR06] and the Equation 4, Equation 5 and Equation 6, the following equation can be obtained.
Theorem 10. If \( \sigma \) random function is the event that at least one collision occurs in the values of \( \gamma \) is the maximum block-length of the ciphertext \( L \) is the ideal permutation in Construction \( C_{\text{Chain}} \), then

\[
Adv_{C_{\text{Chain}},A,G}(1^\lambda) \leq Adv_{\text{Game 0}}(1^\lambda) \leq \frac{\sigma(\sigma - 1)}{22^\lambda} + \frac{\sigma(\sigma - 1)}{2^\lambda}
\]

Proof. We replace the random permutation \( \pi \) used in the Construction \( C_{\text{Chain}} \) by the random function \( rf \), to obtain the Construction \( C_{\text{Chain}} \) shown in Figure 16.

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

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

Suppose that we are using the construction \( C_{\text{Chain}} \), 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:

\[
Pr[A] \leq Pr[L_1 = L_0] + Pr[L_2 = L_1 \lor L_2 = L_0 | A_1] \\
+ Pr[L_3 = L_2 \lor L_3 = L_1 \lor L_3 = L_0 | A_2] \\
+ \cdots + Pr[L_\sigma = L_{\sigma - 1} \lor L_\sigma = L_{\sigma - 2} \lor \cdots \lor L_\sigma = L_0 | A_{\sigma - 1}]
\]

\[
\leq \frac{2^\lambda}{2^\lambda} + \frac{2^\lambda}{2^\lambda} + \frac{3^\lambda}{2^\lambda} + \cdots + \frac{\sigma^\lambda}{2^\lambda}
\]

\[
\leq \frac{\sigma^\lambda}{2^\lambda}
\]

Suppose that we are using the construction \( C_{\text{Chain}} \), we define the following events:
B is the event that $L_j^i = L_i$ 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.

\[
Adv_{\text{Chain}, A, G}^{\text{INT}}(1^\lambda) \overset{\Delta}{=} \left| \Pr[I\text{NT}_{\Pi}^A(1^\lambda, G) = 1] \right| 
\leq \Pr[B] 
\leq \Pr[B|C] \cdot \Pr[C] + \Pr[B|\bar{C}] \cdot \Pr[\bar{C}] 
\leq \left( \frac{\sigma^2}{2^\lambda} + Adv_{\text{IND-PRV}}^{\text{IND-PRV}}(1^\lambda, \sigma) \right) + \left( \Pr[A] + Adv_{\text{IND-PRV}}^{\text{IND-PRV}}(1^\lambda, \sigma) \right)
\]

Using PRP/PRF Switching Lemma [BR06]

\[
\leq \left( \frac{\sigma^2}{2^\lambda} + \frac{\sigma(\sigma - 1)}{2^\lambda} \right) + \left( \frac{\sigma^2}{2^\lambda} + \frac{\sigma(\sigma - 1)}{2^\lambda} \right) 
\leq \frac{\sigma^2}{2^\lambda - 1} + \frac{\sigma(\sigma - 1)}{2^\lambda - 1}
\]

\[\square\]

\[\text{Figure 16: Construction } C'_\text{Chain} \text{ obtained by replacing the random permutation } \pi \text{ used in the Construction } C_\text{Chain} \text{ by the random function } rf.\]

5.1.4 DChain: KAS-AE-chain based on FP

Functionalities based on FP

In this section, we are designing two functionalities – namely $G_1^\pi$ and $G_2^\pi$ – 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 $G_1^\pi$ and $G_2^\pi$ are shown in Figure 17.

**KAS-AE-chain scheme based on functionalities $G_1^\pi$ and $G_2^\pi**

A $D_{\text{Chain}} \Pi = (\Pi, \mathcal{E}, \Pi, D, \mathcal{F}, R, \Pi, D)$ is a KAS-AE-chain built from the functionalities $G_1^\pi$ and $G_2^\pi$ following the framework described in Figure 18.

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

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

\[
Adv_{D_{\text{Chain}}, A, G}^{\text{KR-ST}}(1^\lambda) \leq \frac{\sigma(\sigma - 1)}{2^\lambda - 1} + \frac{\sigma(\sigma - 1)}{2^\lambda}
\]

**Proof.** This proof is similar to the proof of Construction $C_\text{Chain}$ (see Theorem 8). \[\square\]

**Theorem 12.** If $\pi$ is the ideal permutation in Construction $D_{\text{Chain}}$ (see Theorem 8), then

\[
Adv_{D_{\text{Chain}}, A, G}^{\text{IND-PRV}}(1^\lambda) \leq \frac{\sigma(\sigma - 1)}{2^\lambda - 1} + \frac{\sigma(\sigma - 1)}{2^\lambda}
\]
$G_\pi^1(1^\lambda, M, IV^1, IV^2)$

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

$G_\pi^2(1^\lambda, K, C, IV^1)$

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

(a) Algorithmic description of functionalities $G_\pi^1$ and $G_\pi^2$.

(b) Diagrammatic description of functionality $G_\pi^1$.

(c) Diagrammatic description of functionality $G_\pi^2$.

Figure 17: Algorithmic and diagrammatic descriptions of the functionalities $G_\pi^1$ and $G_\pi^2$ are shown in (a), (b) and (c); here, $\pi$ is a $2\lambda$-bit easy-to-invert permutation. Each wire in (b) and (c) represents $\lambda$ bits. The function $G_\pi^1$ takes as inputs parameter $1^\lambda$, message $M$ and two other values $IV^1$ and $IV^2$, and returns the decryption key $K$ and the ciphertext $C$. Similarly, $G_\pi^2$ takes as inputs parameter $1^\lambda$, 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 $\lambda$. 
(a) Algorithmic description of building $D_{\text{Chain}}$ II using the functionalities $\psi_1^\pi$ and $\psi_2^\pi$. 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 $\Pi. E(\text{params}^{(\Pi)}, G, f)$, where $f = (f_1, f_2, f_3)$ and $G$ is shown in 18(b).

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

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

Figure 18: Building $D_{\text{Chain}}$. 
Proof. This proof is similar to the proof of Construction \( C_{\text{Chain}} \) (see Theorem 9).

\[ \text{Theorem 13.} \quad \text{If } \pi \text{ is the ideal permutation in Construction } D_{\text{Chain}}, \text{ then} \]

\[ \text{Adv}^{\text{INT}}_{\text{Chain-A,E}}(1^\lambda) \leq \frac{\sigma^2}{2^{k-1}} + \frac{\sigma^2}{2^{2\lambda-1}} \]

Proof. This proof is similar to the proof of Construction \( C_{\text{Chain}} \) (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 \( \lambda \) be the security parameter. A chain partition of \( V \) into \( w \) chains \( C_1, C_2, \ldots, C_w \) is selected in such a way that \( C_i \) contains nodes (or classes) \( u_1^i, u_2^i, \ldots, u_{l_i}^i \), where \( l_i = |C_i| \).

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

1. The PPT setup algorithm \( \Pi. \text{Setup}(1^\lambda) \) outputs the parameter \( \text{params}(\Pi) \), 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 \( \text{params}(\Pi) \), 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 \( \pi \), and return a three-tuple \( (S, k, \text{pub}) := \Pi. \mathcal{E}(\text{params}(\Pi), G, f, \pi) \), where \( S = (S_u)_{u \in V} \), \( k = (k_u)_{u \in V} \) and \( \text{pub} \) 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 \( \text{pub} \in \{0,1\}^* \), for all \( u \in V \).

3. The key-derive algorithm \( \Pi. \mathcal{D} \mathcal{E} \mathcal{R} \) is a deterministic PT algorithm such that \( k_{u_h}^g := \Pi. \mathcal{D} \mathcal{E} \mathcal{R}(\text{params}(\Pi), G, u_j^g, u_h^g, S_{u_j^g}, \text{pub}_{g}, \pi) \). Here: \( u_h^g \leq u_j^g \) are two nodes of the access graph \( G \); \( S_{u_j^g} \) is \( u_j^g \)'s private information; \( \text{pub}_{j} \) is the public information; \( \pi \) is the KAS-AE-chain scheme; and \( k_{u_h}^g \) is \( u_h^g \)'s decryption key.

Note that \( S_{u_j^g} \in \{0,1\}^*, \text{pub}_{g} \in \{0,1\}^* \) and \( k_{u_h}^g \in \mathcal{K}(\Pi) \cup \perp \).

4. The decryption algorithm \( \Pi. \mathcal{D} \) is a deterministic PT algorithm such that \( f_{u_h}^g := \Pi. \mathcal{D}(\text{params}(\Pi), G, u_j^g, u_h^g, S_{u_j^g}, \text{pub}_{g}, \pi) \). Here: \( u_h^g \leq u_j^g \) are two nodes of the access graph \( G \); \( S_{u_j^g} \) is \( u_j^g \)'s private information; \( \text{pub}_{j} \) is the public information; \( \pi \) is the KAS-AE-chain scheme; and \( f_{u_h}^g \) is \( u_h^g \)'s decrypted file.

Note that \( S_{u_j^g} \in \{0,1\}^*, \text{pub}_{g} \in \{0,1\}^* \) and \( f_{u_h}^g \in \mathcal{M}(\Pi) \cup \perp \).

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 \( \pi \) with the KAS-AE-chain schemes \( A_{\text{Chain}}, B_{\text{Chain}}, C_{\text{Chain}} \) and \( D_{\text{Chain}} \) in the modified chain partition algorithm, we construct the KAS-AE schemes Construction \( A, B, C \) and \( D \) respectively (see Figure 19).
Algorithm description of modified chain partition algorithm to build a KAS-AE scheme $\Pi = (\Pi.\mathcal{E}, \Pi.\mathcal{D}\mathcal{R}, \Pi.\mathcal{D})$ using the KAS-AE-chain scheme $\pi = (\pi.\mathcal{E}, \pi.\mathcal{D}\mathcal{R}, \pi.\mathcal{D})$.

### 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 $A$ 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 $B$, using $A$, 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 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 Subsection 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 $\Pi = (\Pi.\mathcal{E}, \Pi.\mathcal{D}\mathcal{R}, \Pi.\mathcal{D})$ from the functionalities $\Psi$, $\mathcal{E}$ and $\mathcal{D}$ of an MLE scheme $\Psi = (\Psi, \mathcal{E}, \Psi, \mathcal{D})$ (described in Subsection 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 $\Pi$.

- $\Pi.\mathcal{E}(\text{params}^{(T)}, G, f, \pi)$ 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 \((u_{j1}, u_{j2}, \ldots, u_{jd})\) of \(u\) (in ascending order); then a \(\lambda\)-bit random number \(R\) is generated; then \(f'_u\) is obtained by prepending \(R\) and the decryption keys \(k_{u_{j1}}, k_{u_{j2}}, \ldots, k_{u_{jd}}\) – which have been already generated in the previous iterations – to the file \(f_u\); and finally, \((k_u, c_u, t_u) := \Psi.E(params(\Psi), f'_u)\) 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).

- II. \(D\mathcal{E}R(params^{(I)}, G, u, v, S_u, pub)\) 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 \(\text{path}(G, u, v)\) returns a sequence of nodes \((u, u_{i1}, u_{i2}, \ldots, u_{il}, u_{il+1} = v)\) 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 \(w = u, u_{i1}, u_{i2}, \ldots, u_{il}, v\) the following operations are executed: the ciphertext \(c_w\) and the tag \(t_w\) is extracted; \(f'_w := \Psi.D(params(\Psi), k_w, c_w, t_w)\) is computed; the function \(\text{ch_seq}(w, G)\) returns the sequence of children \((u_{j1}, u_{j2}, \ldots, u_{jd})\) of \(w\) (in ascending order); the values of \(R, k_{u_{j1}}, k_{u_{j2}}, \ldots, k_{u_{jd}}\) 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).

- II. \(D(params^{(I)}, G, u, v, S_u, pub)\) 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 \(k_v := II. D\mathcal{E}R(params^{(I)}, G, u, v, S_u, pub)\) is computed. Then, the ciphertext \(c_v\) and tag \(t_v\) are extracted from \(pub\) using the function \(\text{ext_cipher}\). After that, the file \(f'_v := \Psi.D(params(\Psi), k_v, c_v, t_v)\) 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 \(B_{\text{Chain}}\) in the Subsubsection 5.1.2. \(\square\)

### 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 2.2.2.1) is huge.
(a) Algorithmic description of building Construction 1 II 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 u₁, u₂, ..., u₇ and their corresponding files f₁, f₂, ..., f₇.

d) Pictorial description of $\Pi.E(\text{params}^{(1)}, G, f)$ using the path (u₁, u₃, u₆) which is shown in red line in graph G in 20(b).

e) Pictorial description of $\Pi.D(\text{params}^{(1)}, G, u, v, S, pub)$ using the path (u₁, u₃, u₆) 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 $F_1^\pi$ and $F_2^\pi$ 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 $F_1^\pi$ and $F_2^\pi$

The pseudo-code for building a KAS-AE scheme II = (II, $E$, II, $\mathcal{DE\!R}$, II, $D$) from the functionalities $F_1^\pi$ and $F_2^\pi$ (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-AE scheme II.

- II. $E$(params$^{(II)}$, $G$, $f$) 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[$\cdot$] and calculating maximum-depth of tree $h$, which are returned by the function 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 $\text{ch\_seq}(u, G)$ returns the sequence of children $(u_{j_1}, u_{j_2}, \ldots, u_{j_d})$ of $u$ (in ascending order); then the key of the first child $k_{u_{j_1}}$ is assigned to $IV^2$, the last $\lambda$-bit block of ciphertext $c_{u_{j_1}}$ is assigned to $IV^1$, and the keys $k_{u_{j_2}}, k_{u_{j_3}}, \ldots, k_{u_{j_d}}$ – 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 $(k_u, c_u) := F_2^\pi(1^\lambda, f'_u, IV^1, IV^2)$. For the leaf nodes, the value of $IV^1$ and $IV^2$ are $0^\lambda$. The vectors $\mathcal{S}$, $\lambda$ and pub are computed as $pub := (c_u)_{u \in \mathcal{V}}$, and $S := k := (k_u)_{u \in \mathcal{V}}$. Pictorial description of this algorithm on an access graph $G$ is given in 21(c).

- II. $\mathcal{DE\!R}$(params$^{(II)}$, $G$, $u$, $v$, $S_u$, 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 $\text{path}(G, u, v)$ returns a sequence of nodes $(u, u_1, u_2, \ldots, u_t, v) = \mathcal{v}$ 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 $w = u, u_1, u_2, \ldots, u_t, v$ the following operations are executed: the ciphertext $c_w$ is extracted; $(f'_w, IV^2) := F_2^\pi(1^\lambda, k_w, c_w)$ is computed; the function $\text{ch\_seq}(w, G)$ returns the sequence of children $(u_{j_1}, u_{j_2}, \ldots, u_{j_d})$ of $w$ (in ascending order); the key $k_{u_{j_1}}$ is assigned the value of $IV^2$; the values of $k_{u_{j_2}}, k_{u_{j_3}}, \ldots, k_{u_{j_d}}$ and $f_w$ are extracted from $f'_w$; and the next node in the path is searched in the sequence $\mathcal{v}$, 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).

- II. $D$(params$^{(II)}$, $G$, $u$, $v$, $S_u$, 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 $k_v := II. \mathcal{DE\!R}(\text{params}^{(II)}, G, u, v, S_u, pub)$ is computed. Then, the ciphertext $c_v$ is extracted from pub using the function $\text{ext\_cipher}$. After that, the file $f'_v$ and value $IV^2$ are computed $(f'_v, IV^2) := F_2^\pi(1^\lambda, k_v, c_v)$ 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 $(f'_w, IV^2) := F_2^\pi(1^\lambda, k_w, c_w)$, 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 $0^\lambda$ for
the leaf node whose ciphertext is decrypted in the last iteration. If this condition is
satisfied, the file $f_v$ is returned, otherwise $\bot$ 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 $C_{\text{Chain}}$ in the Subsubsection 5.1.3. \qed

7.2 Construction 3: KAS-AE built from FP

We design two functionalities $G^*_1$ and $G^*_2$ 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 $G^*_1$ and $G^*_2$

The pseudo-code for building a KAS-AE scheme $Π = (Π, E, Π, D\mathcal{E}, R, Π, D)$ from the
functionalities $G^*_1$ and $G^*_2$ 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.

- $Π, E(\text{params}^{(Π)}, G, f)$ is a randomised algorithm. This encryption function is de-
signed 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 $\text{height}(G)$, 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 $\text{ch_seq}(u, G)$ returns the sequence of
children $(u_{j1}, u_{j2}, \cdots, u_{j\lambda})$ of $u$ (in ascending order); then the key of the first child
$k_{u_{j1}}$ is assigned to $IV^2$, the last $\lambda$-bit block of ciphertext $c_{u_{j1}}$ is assigned to $IV^1$, and
the keys $k_{u_{j1}}, k_{u_{j2}}, \cdots, k_{u_{j\lambda}}$ – 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 $(k_u, c_u) = G^*_1(1^\lambda, f_u', IV^1, IV^2)$. For the leaf
nodes, the value of $IV^1$ and $IV^2$ are $0^\lambda$. 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).

- $Π, D\mathcal{E}(\text{params}^{(Π)}, G, u, v, S_u, pub)$ 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
$\text{path}(G, u, v)$ returns a sequence of nodes $(u, u_{i1}, u_{i2}, \cdots, u_i, v)$ 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 $w = u, u_{i1}, u_{i2}, \cdots, u_i, v$
the following operations are executed: the ciphertext $c_w$ is extracted; the function
$\text{ch_seq}(w, G)$ returns the sequence of children $(u_{1j}, u_{2j}, \cdots, u_{ij})$ of $w$ (in ascending order); the value of $IV^1$ is computed as the last $\lambda$-bit block of ciphertext $c_{u_{1j}}$;
$(f_w, IV^2) = G^*_2(1^\lambda, k_w, c_w, IV^1)$ is computed; the key $k_{u_{1j}}$ is assigned the value of
(a) Algorithmic description of building Construction 2 II using the functionalities $F^\uparrow_1$ and $F^\uparrow_2$. For the pictorial description with an example, see 21(b)–21(e).

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

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

(d) Pictorial description of $\Pi.\mathcal{DERR}(\text{params}^{(I)}, G, u, v, S_u, \text{pub})$ 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 $\Pi.\mathcal{D}(\text{params}^{(II)}, G, u, v, S_u, \text{pub})$ 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 $f'_u$; and the next node in the path is searched in the sequence $u_i$, 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).

- II. $\mathcal{D}(\text{params}^{(i)}, G, u, v, S_u, pub)$ 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 $k_u = II.\mathcal{D}ER(\text{params}^{(i)}, G, u, v, S_u, pub)$ is computed. Then, the ciphertext $c_w$ is extracted from $pub$ using the function $\text{ext\_cipher}$, and the function $\text{ch\_seq}(v, G)$ 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 $\lambda$-bit block of ciphertext $c_{u_{j_1}}$.

After that, the file $f'_w$ and value $IV^2$ are computed ($f'_w, IV^2 = G^2_S(1^\lambda, k_v, c_v, IV^1)$) and the keys of children of $v$ are removed from the head of file $f'_w$ 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 $\text{ch\_seq}(w, G)$ 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 $\lambda$-bit block of ciphertext $c_{u_{j_1}}$; and (for all $j > 1$) $f'_w, IV^2 = G^2_S(1^\lambda, k_w, c_w, IV^1)$, where $IV^2$ acts as the key of the first child for the execution of next iteration. The value of $IV^2$ should be $0^\lambda$ for the leaf node whose ciphertext is decrypted in the last iteration. If this condition is satisfied, the file $f_v$ is returned, otherwise $\bot$ 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 $\mathcal{D}_{\text{Chain}}$ 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 $G$; $\text{deg}(u)$ 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 $\lambda$ is the security parameter. Also, we consider the key and tag sizes to be $\lambda$ bits each. Based on the definitions of the key derivation algorithm II. $\mathcal{D}ER$ and decryption algorithm II. $\mathcal{D}$ of the KAS-AE scheme (defined in Section 3), the chain $G_g$, and vertices $u^g_h$ and $u^g_g$ (discussed in Subsubsection 2.2.1 and Section 5), we define the sets $U_1 := \{v \in C_g | u^g_h \leq v \leq u^g_g\}$; $U_2 := \{v \in C_g | u^g_h \leq v \leq u^g_g\}$; so $U_1 \subseteq U_2$; $U_3 := \{u, u_{i_1}, u_{i_2}, \cdots, u_{i_k}, v\}$ such that $u_{i_1} < u, u_{i_2} < u_{i_1}, \cdots, v < u_{i_k}$; 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, $G_g$ is a partition of $V$ forming a chain that contains the nodes $u^g_g$ and $u^g_h$ such that $u^g_g \leq u^g_h$ (see Subsubsection 2.2.1).

For the KAS schemes II = (II. $\mathcal{G}EM$, II. $\mathcal{D}ER$): $\text{cGEM}$ is the running time of generating a $\lambda$-bit key by algorithm II. $\mathcal{G}EM$; $\text{cK}$ denote the cost of generating single $\lambda$-bit key, for the schemes X-AE, where $X \in \{TKAS, TKEKAS, DKEKAS, IKEKAS\}$; and $\text{cE}$ and $\text{cK}$ denote the cost of generating one public value $e$ and generating one $\lambda$-bit key from a given $e$ value in NBKAS-AE.
\section{Key Assignment Scheme with Authenticated Encryption}

\subsection{Algorithmic Description}

(b) Algorithmic description of building Construction 3 II using the functionalities $G_1^*$ and $G_2^*$. For the pictorial description with an example, see 22(b)–22(e).

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{construction3_ii}
\caption{Building Construction 3.}
\end{figure}
For $AE$ scheme $\Psi = (\Psi, K_{\text{GEN}}, \Psi, \mathcal{E}, \mathcal{D})$: \(c_{\mathcal{AE}^\lambda}\) and \(c_{\mathcal{DV}^\lambda}\) denote the running times of algorithms $\Psi, \mathcal{E}$, and $\Psi, \mathcal{D}$ for a $\lambda$-bit input.

For $MLE$ scheme $\Omega = (\Omega, \mathcal{E}, \Omega, \mathcal{D})$: \(c_{\mathcal{E}^\lambda}\) and \(c_{\mathcal{D}^\lambda}\) denote the running times of algorithms $\Omega, \mathcal{E}$ and $\Omega, \mathcal{D}$ for a $\lambda$-bit input.

The \(c_{\mathcal{E}}, c_{\mathcal{E}^{-1}}, c_{\mathcal{D}}\) and \(c_{\mathcal{D}^{-1}}\), denote the running times of the algorithms $\mathcal{F}_1^\lambda$, $\mathcal{F}_2^\lambda$, $G_1^\lambda$ and $G_2^\lambda$ that uses a $2\lambda$-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.

<table>
<thead>
<tr>
<th>Const. $\rightarrow$ Prop. ↓</th>
<th>TRKAS-AE (Based on [CC02])</th>
<th>TKEKAS-AE (Based on DKEKAS [Gud80])</th>
<th>DKEKAS-AE (Based on [ZRM01])</th>
<th>IKEKAS-AE (Based on [ABF05])</th>
<th>NBKAS-AE (Based on [AT83])</th>
</tr>
</thead>
<tbody>
<tr>
<td>Storage Req.:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\mathcal{P}_{\mathcal{DV}}$</td>
<td>$2n^2\lambda$</td>
<td>$2n^2\lambda$</td>
<td>$n^2\lambda + n\lambda$</td>
<td>$n^2\lambda + n\lambda$</td>
<td>$n^2\lambda + n\lambda$</td>
</tr>
<tr>
<td>$\mathcal{P}_{\mathcal{UB}}$</td>
<td>$</td>
<td>f</td>
<td>$</td>
<td>$</td>
<td>f</td>
</tr>
<tr>
<td>Running Time:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\mathcal{E}$</td>
<td>$c_{\mathcal{AE}^\lambda}(\frac{</td>
<td>f</td>
<td>}{\lambda}) + n \cdot c_{\mathcal{C}}$</td>
<td>$c_{\mathcal{AE}^\lambda}(\frac{</td>
<td>f</td>
</tr>
<tr>
<td>$\mathcal{D}_{\mathcal{CR}}$</td>
<td>$O(1)\cdot c_{\mathcal{DV}^\lambda}$</td>
<td>$O(1)\cdot c_{\mathcal{DV}^\lambda}$</td>
<td>$O(1)\cdot c_{\mathcal{DV}^\lambda}$</td>
<td>$O(1)\cdot c_{\mathcal{DV}^\lambda}$</td>
<td>$O(1)\cdot c_{\mathcal{DV}^\lambda}$</td>
</tr>
<tr>
<td>$\mathcal{D}$</td>
<td>$\frac{c_{\mathcal{DV}^\lambda}}{\lambda}</td>
<td>f_v</td>
<td>$</td>
<td>$\frac{c_{\mathcal{DV}^\lambda}}{\lambda}</td>
<td>f_v</td>
</tr>
</tbody>
</table>

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


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

<table>
<thead>
<tr>
<th>Const. → Prop. ↓</th>
<th>Construction A (Subsection 5.2 + Subsection 5.1.1)</th>
<th>Construction B (Subsection 5.2 + Subsection 5.1.2)</th>
<th>Construction C (Subsection 5.2 + Subsection 5.1.3)</th>
<th>Construction D (Subsection 5.2 + Subsection 5.1.4)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Storage Req.:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• PRIV</td>
<td>$\lambda (n^2 + nw)\lambda$</td>
<td>$nw\lambda$</td>
<td>$nw\lambda$</td>
<td>$nw\lambda$</td>
</tr>
<tr>
<td>• PUB</td>
<td>$</td>
<td>f</td>
<td>$</td>
<td>$2nw +</td>
</tr>
<tr>
<td><strong>Running Time:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• $E$</td>
<td>$\frac{</td>
<td>f</td>
<td>}{\lambda}$</td>
<td>$\frac{</td>
</tr>
<tr>
<td>• $\mathcal{D}_E$</td>
<td>$\frac{</td>
<td>f</td>
<td>}{\lambda}$</td>
<td>$\frac{</td>
</tr>
<tr>
<td>• $\mathcal{D}_R$</td>
<td>$\frac{</td>
<td>f</td>
<td>}{\lambda}$</td>
<td>$\frac{</td>
</tr>
<tr>
<td>• $\mathcal{D}$</td>
<td>$\frac{</td>
<td>f</td>
<td>}{\lambda}$</td>
<td>$\frac{</td>
</tr>
<tr>
<td><strong>Comparison:</strong></td>
<td>Secure KAS &amp; Secure AE</td>
<td>Secure MLE</td>
<td>Ideal Permutation</td>
<td>Ideal Permutation</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Const. → Prop. ↓</th>
<th>Construction 1</th>
<th>Construction 2</th>
<th>Construction 3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Storage Req.:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• PRIV</td>
<td>$\sum_{u \in V} (\deg(u) + 1) \cdot \lambda +</td>
<td>f_u</td>
<td>\lambda$</td>
</tr>
<tr>
<td>• PUB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Running Time:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• $E$</td>
<td>$\frac{</td>
<td>f</td>
<td>}{\lambda}$</td>
</tr>
<tr>
<td>• $\mathcal{D}_E$</td>
<td>$\frac{</td>
<td>f</td>
<td>}{\lambda}$</td>
</tr>
<tr>
<td>• $\mathcal{D}_R$</td>
<td>$\frac{</td>
<td>f</td>
<td>}{\lambda}$</td>
</tr>
<tr>
<td>• $\mathcal{D}$</td>
<td>$\frac{</td>
<td>f</td>
<td>}{\lambda}$</td>
</tr>
<tr>
<td><strong>Comparison:</strong></td>
<td>Secure MLE</td>
<td>Ideal Permutation</td>
<td>Ideal Permutation</td>
</tr>
</tbody>
</table>


46 Key Assignment Scheme with Authenticated Encryption


