Security Analysis of the MLS Key Distribution

Chris Brzuska, Eric Cornelissen, Konrad Kohbrok
Aalto University, Finland

***Abstract*—Cryptographic communication protocols provide confidentiality, integrity and authentication properties for end-to-end communication under strong corruption attacks, including, notably, post-compromise security (PCS). Most protocols are designed for one-to-one communication. Protocols for group communication are less common, less efficient, and tend to provide weaker security guarantees. This is because group communication poses unique challenges, such as coordinated key updates, changes to group membership and complex post-compromise recovery procedures.**

We need to tackle this complex challenge as a community. Thus, the Internet Engineering Task Force (IETF) has created a working group with the goal of developing a sound standard for a continuous asynchronous key-exchange protocol for dynamic groups that is secure and remains efficient for large group sizes. The current version of the *Messaging Layer Security* (MLS) security protocol is in a feature freeze, i.e., no changes are made in order to provide a stable basis for cryptographic analysis. The key schedule and TreeKEM design are of particular concern since they are crucial to distribute and combine several keys to achieve PCS.

In this work, we study the MLS *continuous group key distribution* (CGKD) which comprises the MLS key schedule, TreeKEM and their composition, as specified in Draft 11 of the MLS RFC, while abstracting away signatures, message flow and authentication guarantees. We establish the uniqueness and key indistinguishability properties of the MLS CGKD as computational security properties.

I. Introduction

We expect modern-day messaging applications to provide end-to-end security, guaranteeing confidentiality and authenticity of the transmitted messages. This expectation has been made a reality over the course of 25 years through the design of cryptographic protocols, which nowadays guarantee secure communication, potentially even after a participant's key is compromised.

Our everyday communication protocols inherit their security from the properties of the keys they use, which, in turn, are established via key-exchange protocols.

Key exchange protocols therefore have been designed to achieve strong security guarantees: *Entity Authentication* [7], [27] ensures that the key is shared only with the intended recipient. *Key Indistinguishability* [20], [27], in turn, guarantees that no one but the participants involved has information about the key. A stronger variant of key indistinguishability called *Forward Secrecy* (FS) ensures that short-term communication keys exchanged in past sessions are secure if a party's current state or long-term key gets compromised [27].

With the introduction of *continuous* key-exchange, for example by protocols such as OTR [11] or ZRTP [31], parties are able to recover from compromise if the adversary remains passive for a brief period of time. This counterpart to FS was first described as *Post-Compromise Security* (PCS) by Cohn-Gordon et al. [15], [22].

Modern messaging protocols require that messages can be sent even if a recipient is offline, requiring an *asynchronous* protocol. As a consequence, protocol sessions don't naturally end when one participant goes offline, leading to sessions that remain active for months and years. To cover this use-case, non-interactive key exchange protocols were introduced that perform key-exchange and entity authentication continuously even in an asynchronous setting. Most notably, the *Signal protocol* [26] (formerly *TextSecure*) achieves strong FS and PCS guarantees [17], [15] due to the use of double ratcheting (formerly *Axolotl*), which is based on the *Asynchronous Ratcheting Tree* (ART) by Cohn-Gordon et al. [16].

As users of messaging protocols tend to use more than one device and/or communicate with more than one party at a time, messaging protocols are commonly required to be group messaging protocols. This gives rise to the definition of continuous group key exchange protocols by Alwen et al. [2].

MLS. The *Message Layer Security* (MLS) Working Group of the Internet Engineering Task Force (IETF) aims to create a new continuous, asynchronous group key-exchange protocol [5] that is efficient even for large groups [29]. To achieve this, the current draft makes use of the TreeKEM protocol (based on [9]).

In short, TreeKEM, which is inspired by ART, is a continuous group key distribution based on a tree structure where each leaf node represents a member's Key Encapsulation Method (KEM) key pair and all other nodes represent a secret value and KEM key pair shared by the members in the node's subtree. As a result, secrets can be shared efficiently by encrypting it to individual subtrees. The secret value at the root node of the tree is a secret shared by the entire group.

Each member can update their keying material by updating the key pairs and values of their leaf-to-root path, and transfer the new key material to all other members efficiently by encrypting it to the set of subtrees below the path nodes. Each member can also remove a member or add a new member, and update the key material so that previous group members cannot access key material any longer. In Section III, we explain TreeKEM in more detail. The updating of keys allows members to achieve PCS and FS similar to ART.

Note that TreeKEM does not have an internal mechanism for members to reconcile differing views of individual groups and thus relies on members agreeing on a total order of group operations. One straightforward way of achieving this is a central distribution service enforcing that order.

Once a new secret value is distributed via TreeKEM, MLS combines the new secret with existing key material used for

communication. From these two keys, a new communication key is derived for the next *epoch*. In addition, key derivation includes information about the group and, optionally, allows to inject one or more external keys. The combination of keys allows MLS to achieve PCS and FS guarantees, and the inclusion of group parameters such as membership ensures agreement on said parameters. The processing and combining of the key material and context information is described in the *MLS key schedule*.

Related Work. The MLS design process has been accompanied by security analyses and proposals for changes to the protocol which we summarize below.

Bhargavan, Beurdouche and Naldurg (BBN, [10]) use F* to create a novel symbolic verification tool and use it to perform a symbolic analysis of MLS Draft 7. They uncover two attacks and several other weaknesses in the protocol that are subsequently fixed.

In [2], Alwen, Coretti, Dodis and Tselekounis (ACDT) introduce the Continuous Group Key Agreement (CGKA) security notion and introduce the notion of Post-Compromise Forward Security (PCFS), which encodes the security of a given session despite state compromise both before and after the session. They analyze the security of TreeKEM in MLS Draft 7, finding that while it provides the desired Post-Compromise Security (PCS) guarantees, its Forward Secrecy (FS) guarantees are very weak. They propose to modify TreeKEM in order to improve FS guarantees based on an algebraic KEM construction that requires keys to be updatable.

Alwen, Capretto, Cueto, Kamath, Klein, Markov, Pascual-Perez, Pietrzak, Walter and Yeo (ACCK+ [1]) build on ACDT by proving the CGKA security of TreeKEM in MLS Draft 9 in a stronger adversarial model. They also propose the notion of Tainted TreeKEM as an alternative to TreeKEM with a different performance profile, but with comparable security.

Following ACCK+, Alwen, Coretti, Jost and Mularczyk (ACJM, [3]) prove the CGKA security of TreeKEM in MLS Draft 9 against an even stronger adversary, allowing active interference of the adversary in the protocol flow.

Continuing the work of ACJM, Alwen, Jost and Mularczyk (AJM, [4]) analyze the CGKA security of TreeKEM in MLS Draft 10 with regard to insider security, formalizing and proving the as-of-yet strongest security notion for TreeKEM, with their work resulting in a number of changes to the MLS protocol. Even though in this work we only analyze a subset of MLS (compared to the whole protocol as analyzed by AJM) and with a different adversarial model, we reach a similar conclusion in terms of the security of MLS. See Section VI for a more detailed comparison of model and results.

Independent of other works, Weidner [30] proposes an alternative version of TreeKEM, named *Causal* TreeKEM, that does not require a strict total order of group operations. This would allow a relaxation of MLS' requirement of a strict order of handshake messages, but similar to the proposal by ACDT would require a specialized KEM construction.

In the only work considering the multi-group setting, Cremers, Hale and Kohbrok (CHK, [19]) compare MLS with a straightforward group messaging protocol made of 1:1 Signal sessions, showing that MLS provides significantly worse authentication PCS guarantees.

Our security game for the MLS Key Schedule and its analysis are inspired by the analysis of TLS 1.3 by Brzuska et al. [12] in a similar way as the MLS key schedule is inspired by the TLS 1.3 key schedule. Note, however, that the MLS key schedule enjoys better domain separation which avoids complications such as evolved invariant proofs ([12],Appendix C). Our analysis of TreeKEM and its composition with the key schedule do not build on [12].

Contributions. In this work, we study the security claims made in Draft 11 of the MLS RFC via a cryptographic analysis of the MLS key distribution, which includes the key derivation structure of both TreeKEM and the key schedule. Excluded from our analysis is authentication, as well as Secret Tree component of the key schedule. Key schedule analysis was implicitly suggested in Krawczyk's study of Sigma protocols [23] and used for the analysis of TLS 1.3 [12]. We develop security models for TreeKEM, the key schedule and their composition. Interestingly, our models themselves are composable, and the composed model is derived from the composition of the TreeKEM model and the key schedule model. This composability feature is enabled via the use of the State Separating Proofs methodology (SSP) [13] which specifies security models via modular pseudocode and enables code reuse (see Section IV and Section VIII for details).

Assumptions. We make standard key indistinguishability and collision-resistance assumptions on the key derivation functions (KDF) and assume indistinguishability under chosen-ciphertext attacks (IND-CCA) secure public-key encryption, as well as that the Extract function in Krawczyk's HKDF design [24] is a dual pseudorandom function and thus, we assume that HKDF is a dual KDF, which has also been assumed in the analysis of Noise [21] and TLS 1.3 [12]. Following the approach of [14], [12], we model the security of our symmetric primitives as multi-instance primitives with static corruptions. Our analysis relies on inclusion of the group context into the derivation of the joiner secret, as we suggested in a pull request awaiting to be merged [18].

Methodology. We rely on modular proofs and pseudocode-reuse as specified by the state-separating proofs (SSP) methodology [13]. SSPs allow us to iterate the proof quickly as the MLS draft is developed. Given an initial model of the protocol, we make and verify incremental changes to the model with relative ease. This is what allows us to release this analysis of the protocol shortly after the current feature freeze and it will also allow us to quickly analyse the security of future features.

Overview. Section II introduces notation. Section III explains the TreeKEM, the MLS key schedule and their composition. Section IV introduces the state-separating proofs methodology and our assumptions. Section V explains our security model, Section VII states our three main theorems, Section VIII explains our proof methodology, and Section VIII-A and the appendix contain the proofs of the theorems.

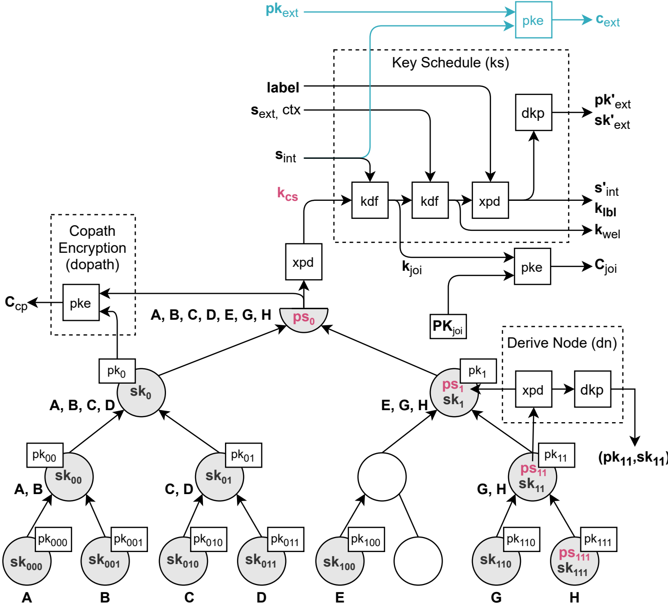


Figure 1: The tree represents a group with members A, B, C, D, E, G and H. Inside a node, we display TreeKEM secrets known to members in the corresponding sub-tree. The pink values are used in the derivation and not stored long-term. The blue part is only used for external add operations.

II. Preliminaries and Notation

Binary Trees. A *binary tree* is a finite, connected, and directed graph with a single source — called *root* — such that each node has at most two outgoing edges. For two nodes n_0 and n_1, if there is an edge $n_0 \rightarrow n_1$, then we call n_0 the *parent* of n_1 and n_1 the *child* of n_0. All nodes which can be reached from n_0 are its *descendants*, and all nodes from which n_1 can be reached are its *ancestors*. Every node can be reached starting from the root, and we call the set of ancestors of a leaf node its *direct path*. We allow for nodes to be marked as *blank* (the marking will later take on the meaning of no associated data). We call the *co-path* a list of the closest non-blank descendent nodes of the nodes along the direct path.

We leave the encoding of the tree abstract in this paper and only assume that given the size of a tree and the index i of a node, we can find the indices of its direct path.

Notation. We use pseudocode to describe algorithms. We denote sets and tables by capital letters, e.g. S or T. We denote algorithms in lowercase and sans serif. $x \leftarrow a$ assigns value a to variable x, and $x \leftarrow \mathsf{algo}(a)$ runs algo on value a and assigns the result to variable x. When algo is a randomized algorithm, $x \leftarrow\!\!\$\, \mathsf{algo}(a)$ runs algo on value a with fresh randomness and assigns the result to variable x. When S is a set, $x \leftarrow\!\!\$\, S$ samples a uniformly random value from S and assigns that value to variable x. A common set is $\{0,1\}^\lambda$, the set of bitstrings of length λ. We use $\overline{a}$ for a vector of variables, and $\overline{x} \overset{\text{vec}}{\leftarrow} \mathsf{algo}(\overline{a})$ means, we run algo separately on each value in the vector $\overline{a}$ and then assign the results to the corresponding indices in vector $\overline{x}$. Analogously, $\overline{x} \overset{\text{vec}\,\$}{\leftarrow} S$ means that we sample each value in the vector $\overline{x}$ independently and uniformly

Input	Description	U	P	J
PK_{mem}	current public view of the group	●	●	●
PK_{rem}	public key(s) to remove	●	●	
PK_{joi}	public key(s) to add	●	●	
PK_{upd}	new public key(s) of updater		●	
SK_{own}	own current secret key(s)	●	●	●
i_{own}	own index	●	●	●
i_{upd}	updater's index		●	●
k_{upd}	seed for updating	○		
CP	indices in PK_{mem} to encrypt to	●		
pk_{ext}	external public key	○		
s_{ext}	external secret randomness(es)	●	●	●
s_{int}	internal secret randomness(es)	●	●	
ctx	public context	●	●	●
c	ciphertext from a member		○	●/○
c_{ext}	ciphertext from an outsider		○	
Output	**Description**	**U**	**P**	**J**
C_{CP}	ciphertext(s) for members	●		
C_{joi}	ciphertext(s) for joiners	●		
c_{ext}	ciphertext from outsider	○		
PK'_{upd}	new public key(s) of updater	●		
PK'_{mem}	new view of the group	●	●	●
SK'_{own}	new own secret key(s)	●	●	●
k	new shared secret key(s)	●	●	●
s'_{int}	new s_{int} for the next update	●	●	●
pk'_{ext}	new external public key	●	●	●

Figure 2: ● and ○ indicate the value can be an input/output of the algorithms of cgkd and wcgkd according to our syntax, with grey rows only available in cgkd. Values marked by ○ are optional; the algorithm might behave differently if provided. Input c is required for join but optional for wjoin.

at random. For two tables T and U, the operation $T \overset{\text{merge}}{\leftarrow} U$ overwrites values $T[i]$ by values $U[i]$ whenever $U[i]$ is defined, and the operation $T \overset{\text{rem}}{\leftarrow} U$ removes all values $T[i]$ from T where $U[i]$ is defined. **assert** *cond* abbreviates that a special error message is sent unless *cond* holds.

III. MLS Continuous Group Key Distribution

A *continuous group key distribution* protocol (CGKD) allows *clients* to form a *group*. A client that is in a group is called a *member*. A group allows members to maintain a continuous session which

- distributes a secret value among group members in each epoch;
- allows for efficient updates to the key material of a member and the group;
- allows for efficiently adding clients as new members to the group;
- allows for efficiently removing members from the group.

Assuming pseudorandomness and uniqueness of input keys, a CGKD achieves pseudorandomness and uniqueness of output keys. It does not model signatures and attacks against authentication, but instead allows the adversary to combine keys (almost) arbitrarily and provide guarantees whenever at least one of the base keys is honest—see Section VI

for how the security guarantees of the CGKD relate to the security properties of a protocol which uses the CGKD which seeks to achieve forward-secrecy, post-compromise security and security against insider attacks. In Appendix A, we define the syntax of a CGKD formally, see Figure 2 for the inputs and outputs of the algorithms process, process and join. In the body, we only describe the MLS CGKD informally, since we provide security models which are specialized to the MLS CGKD. We split the MLS CGKD into a weak variant of continuous group key distribution (wCGKD) and the syntax of a key schedule (KS). The main difference between a CGKD and a wCGKD (often referred to as TreeKEM) is that the group keys of a wCGKD are corrupt whenever one group member's current private keys are corrupted. This property is, indeed, rather weak, since in large groups, individual members might use weak randomness, and for insecurity, it suffices if a single member of the group relies on weak randomness for generating their current own keys. In turn, a (strong) CGKD relies on key material which is chained across *epochs*. If the key material from a previous epoch was secure, then security in the current epoch is maintained—unless the key material is leaked by encrypting it to a client with corrupt keys that joins the group. As we will see for MLS, a wCGKD can be combined with a key schedule to obtain a CGKD.

Updates of the key material are initiated by one group member and we refer to this procedure as (w)update in (w)CGKD. In order for other group members to keep their key material in sync with an updating party, they need to process the messages generated by the update algorithm. Since new group members might perform processing in different way than existing group members, the join algorithm models the computation performed by new joiners. Last, but not least, pke.kgen allows to generate new public keys. We now describe the MLS CGKD, MLS wCGKD and MLS KS.

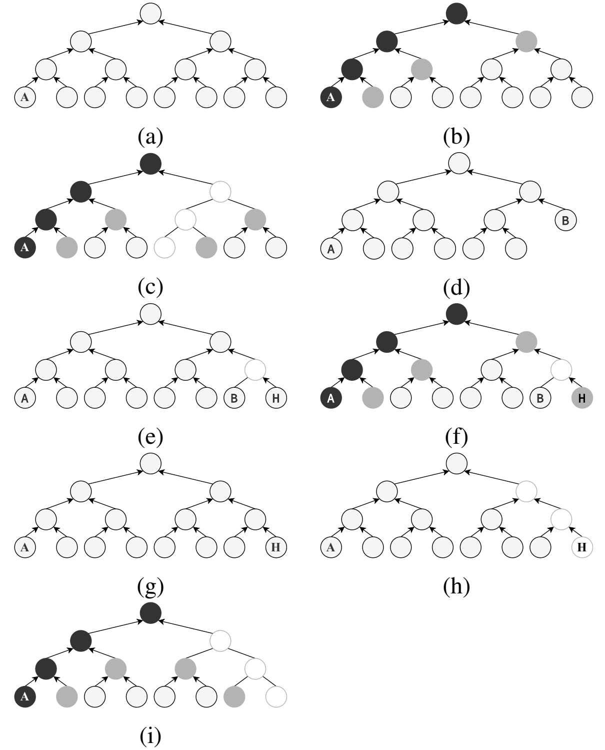


Figure 3: **(a-c)**: Member A updates their their key material. **(a)** original tree for the group. **(b)** tree during the update, highlighting the *direct path* in black and the *co-path* in dark grey. **(c)** tree during the update if there are *blank nodes* in the tree.
(d-f) Adding a new member to a group. **(d)** tree for the original group. **(e)** tree with H added to the group in an *add-only* operation, highlighting blanked nodes in white. **(f)** tree for the update performed by A, highlighting the *direct path* in black and the *co-path* in dark grey. Note that H's unmerged leaf is part of the co-path.
(g-i) Removing a member from a group. **(g)** tree for the original group. **(h)** tree with H removed. **(i)** tree for the update after the removal, highlighting the *direct path* in black and the *co-path* in dark grey. Blanked nodes are highlighted in white in both **(h)** and **(i)**.

MLS Key Schedule. The purpose of the key schedule is to combine existing key material into new key material and, in particular, to chain keys across epochs via the *internal key* s_{int}, which is referred to as the *init secret* in MLS terminology. In addition, the MLS key schedule allows to combine s_{int} with an *external key* s_{ext}, PSK in the MLS RFC, and a *commit secret* k_{cs}, which is derived via the MLS wCGKD. We abstract away the details of how MLS pre-processes the external key material in case that several external keys are combined and work directly with a single s_{ext}.

The MLS key schedule ks uses a key derivation function (KDF) which is constructed based on an extract (XTR) and an expand (XPD) function, following Krawczyk's HKDF standard [24]. ks chains two full KDFs and then a single XPD function, see top of Figure 1. In OPTLS [25], it was suggested to use the salt position in HKDF to combine two keys, and therefore, the chained call of two KDFs allows to combine the aforementioned three secrets s_{int}, s_{ext}, and k_{cs}. See Figure 1 for an illustration of the MLS Key Schedule algorithm ks. Note that the join algorithm of the MLS CGKD only knows k_{joi}, but not k_{cs} and thus starts the computation of the key schedule only at k_{joi}. The key schedule derives several secrets and, in particular, uses one of the secrets as input to a *key pair derivation function* (DKP), which is a *deterministic* algorithm that corresponds to running pke.kgen using its input as explicit randomness. We now turn to the MLS wCGKD.

MLS wCGKD. The MLS wCGKD is called TreeKEM, and we use the terms MLS wCGKD and TreeKEM interchangeably in this work. The main purpose of TreeKEM is to provide new k_{cs} values to ks. I.e., k_{cs} is fresh key material which is shared between all group members. As the name suggests, key material in TreeKEM is structured and stored in a tree data structure. The leaf nodes of the tree represent group members, the other nodes represent shared secrets known by members in the subtree, i.e., members which correspond to the leaves in the subtree below a node. E.g., k_{cs} is shared by all members of the group and indeed derived from the value ps which is stored at the root of the tree. Figure 1 illustrates the data associated

with each node as well as the members to which the secret node data is known. Every node in the tree has associated with it a KEM key pair known to the members represented by the leaves in the sub-tree. To share a secret value with some members, a member can encrypt the value either to the public key of a node whose secret key all intended members know or encrypt to several public keys for a more fine-grained information-sharing.

The main sub-procedure of the TreeKEM is the Derive Node procedure dn, see Figure 1. dn uses an expand function (XPD) and a key encapsulation method (KEM) as building blocks. dn takes as input the secret of a node, referred to as the *path secret* ps_i of node i, and uses XPD to derive two random values: The path secret $ps_{p(i)}$ (where $p(i)$ refers to the parent of node i) and randomness for (DKP) to computes a KEM key pair (pk_i, sk_i) associated with node i.

Roughly, in the tree of Figure 1, each arrow can be interpreted as a possible application of dn with the restriction that if a node i has two non-blank children, then its path secret was only derived from one of them. The keys associated with each node evolve over time via *updates* which start from a leaf node and proceed along the direct path to the root. The path secret ps_i of a node i is derived from the child which was last contained in the direct path of an update.

We next explain key updates. In addition, one can also combine an update with the *adding* and/or the *removing* of one or more members. While adding and removing is implemented simultaneously, we describe the conceptual idea for each separately below.

Updating Key Material. Consider the group represented by the tree in Figure 3a. The group member Alice (A) can update the group secret by the following process. First, A samples a random value k_{upd} for the key update and uses it as the new path secret associated with A's leaf node, i.e., $ps \leftarrow k_{\text{upd}}$. Then, Alice runs $(ps', pk, sk) \leftarrow \mathsf{dn}(ps)$ and stores pk in $PK'_{mem}[i_{cur}]$ and $PK_{upd}[i_{cur}]$, where i_{cur} is the index of Alice's leaf node, PK'_{mem} is a table which represents Alice's view of the public keys of the group members, and PK'_{upd} is a table which contains the public-keys that Alice will tell other group members to change. Additionally, Alice also stores the secret key in $SK'_{own}[i_{cur}]$. SK is a table of secret keys which Alice knows. Then, Alice computes the parent index of i_{cur} and proceeds in the same way for the parent index: First, she computes $(ps'', pk, sk) \leftarrow \mathsf{dn}(ps')$, then stores pk and sk in the adequate tables, then moves to the parent node and applies dn to ps'' etc. until reaching the root node. This loop is captured by dopath which is hinted at in Figure 1. Note that dopath needs to encrypt to the *entire* co-path and not only to a single node as hinted at in Figure 1. Jumping ahead to the pseuodcode provided in Figure 17, encryption to the co-path is only executed if k_{upd} is non-empty. Note that the assert in line 2 of the wupdate code only forces k_{upd} to be defined if a member is removed or if an external add is performed (since in both of these cases, the keys needs to be updated). In the end, Alice has replaced all the path secrets, public-keys and secret-keys of the direct path, marked in black in Figure 3b.

As only Alice knows the path secrets used along this path, she now also shares the path secret with the other group members. Thus, for each node i_{cur} on her direct path, she considers the child i_{child} of i_{cur} which is not on her direct path (but, instead, is on the *co-path*) and encrypts the path secret of i_{cur} under $PK_{mem}[i_{child}]$, the public-key associated with this node. In this way, the entire sub-tree below $PK_{mem}[i_{child}]$ can decrypt the ciphertext. In Figure 3b, the co-path is marked in grey, i.e., for each grey node, Alice encrypts to the public-key associated with it.

Figure 3a and Figure 3b now illustrate a case where there are no *blank nodes* in the tree—blank nodes are nodes which do not have data associated with it. Figure 3c depicts the case that the tree contains some blank nodes. In this case, Alice encrypts to the nodes marked in grey in Figure 3c. Note that we nevertheless refer to these nodes as co-path even though our co-path definition does not coincide with the graph-theoretic notion of co-path. In order to simplify the computation of the co-path, we give a description of the co-path CP as input to update. CP is a table of sets, i.e., for each tree index i, $CP[i]$ contains a set of the indices to which Alice needs to encrypt. It is a set because Alice may need to encrypt to multiple public keys due to blank and unmerged nodes, see Figure 17 for details.

Adding a New Member. There are two ways to add a new member. One can either add a member in an *add-only* operation, or one can add a member and perform an update at the same time. (In this case, one can also remove members simultaneously, but let us ignore this operation for now.) When performing an *add* operation, we either find the left-most empty leaf node (there might be empty leaf nodes due to previous removals) or, if the tree is full, we replace a leaf node by a 3-node sub-tree. E.g., consider the original group in Figure 3d. Here, to add Henry (H) to this group, the group member Alice (A) blanks Bob's (B) node, creates below it a node for H and a new node for B and associates H's node with a KEM public-key which Alice knows belong to H, and associates B's new node with the same public-key which B had before. Moreover, Alice runs the key schedule with an empty k_{upd} and encrypts k_{joi} to H.

In the case that A wants to simultaneously perform an update of the key material, she now updates her path as described in Section III and encrypts the relevant path-secrets to the co-path, marked in grey. Again, the co-path is not the graph-theoretic co-path, since A needs to encrypt a message to H separately. In addition to sharing the relevant path secrets with the co-path, A also encrypts the joiner secret k_{joi} derived via the MLS Key Schedule (see Figure 1)to H. If Alice performs an add-only (without an update), she performs the computations in the same way except that k_{cs} is now a fixed zero value. It is important to execute the ks function also in an add-only operation to ensure that the new joiners cannot read messages from the time before they joined.

A special case is the situation where the add operation is not performed from *inside* the group, but instead, an outside member adds itself to the group. In this case, the external group public-key pk_{ext} is used (marked in blue in Figure 1

and Figure 17) to transmit s_{int}. Note that the pink code in Figure 17 is only run if pk_{ext} is non-empty, which requires that k_{upd} is non-empty, too.

Removing a Member. Consider the group represented by the tree in Figure 3g. To remove a group member Henry (*H*) from this group, the group member Alice (*A*) blanks the nodes on the path from node *H* to the root, shown in Figure 3h. As a result, a co-path that included those nodes before, now, instead includes their non-blank child nodes.

Notice that the removal by itself does not update the group secret and therefore, *A* also needs to perform an update to ensure that *H* cannot decrypt messages sent by group members anymore. The requirement to perform an update is captured by

$$\textbf{if } |PK_{rem}| > 0 : \textbf{assert } k_{upd} \neq \perp$$

in line 2 of the update procedure in Figure 17. As the tree now contains blanked nodes, the path secrets along the new direct path must be encrypted under the keys of a larger co-path, as shown in Figure 3i.

MLS CGKD. Combining TreeKEM and ks as depicted in Figure 1 yields the MLS CGKD. As mentioned in the beginning of the section, we can now see that the MLS CGKD indeed achieves significantly stronger security properties than the wCGKD. Namely, key secrecy is violated if *any* of the group keys are compromised (and not yet updated). In turn, the MLS CGKD also yields security if the s_{ext} or the s_{int} from a previous epoch are fresh. See Section VI for a discussion of *key freshness*.

IV. Assumptions

Game-Based Security Notions. We formulate security properties via indistinguishability between games, a *real* game which models the real behaviour of a cryptographic primitive, and an *ideal* game which models an ideally secure variant of the primitive. For example, security of a pseudorandom function is captured by demanding that the input-output behaviour of the (keyed) pseudorandom function is indistinguishable from the input-output behavior of a truly random function which draws a uniformly random output for each input.

Especially since Bellare and Rogaway promoted code-based game-hopping [8], it has been popular to formulate games in pseudocode, and we follow this approach here. I.e., a game $\mathtt{G}$ provides a set of *oracles* to the adversary $\mathcal{A}$, these oracles are specified by pseudocode and operate on a common state which is secret to the adversary. This hidden state might contain, e.g., a secret key. In this work, we denote by $\mathcal{A} \circ \mathtt{G}$ that the adversary $\mathcal{A}$ interacts with the oracles of game $\mathtt{G}$. I.e., the adversary is the *main procedure* that makes queries to the oracles of $\mathtt{G}$, which returns an answer to $\mathcal{A}$, and then, in the end, the adversary $\mathcal{A}$ returns a bit indicating their guess whether the game is indeed real or ideal. The *advantage* $\mathsf{Adv}_{\mathtt{G}^0,\mathtt{G}^1}(\mathcal{A})$ measures an adversary $\mathcal{A}$'s ability to distinguish between $\mathtt{G}^0$ (real) and $\mathtt{G}^1$ (ideal).

Definition IV.1 (Advantage)**.** *For an adversary $\mathcal{A}$ and two games $\mathtt{G}^0, \mathtt{G}^1$, the advantage $\mathsf{Adv}_{\mathtt{G}^0,\mathtt{G}^1}(\mathcal{A})$ denotes the term*

$$\left|\Pr\left[1 = \mathcal{A} \circ \mathtt{G}^0\right] - \Pr\left[1 = \mathcal{A} \circ \mathtt{G}^1\right]\right|.$$

```
KEY^{b,λ}
Package Parameters
b:  idealization bit
λ:  key length
i:  package index

Package State
K[h ↦ k]:  key table
H[h ↦ b]:  honesty table

SET(h, hon, s)
assert |s| = λ
h ← (h, i)
if K[h] ≠⊥:
  ret h
if b ∧ hon:
  s ←$ {0,1}^λ
h' ← UNQ(h, hon, s)
if h' ≠ h : ret h'
K[h] ← s
H[h] ← hon
ret h

REG(hon, s)
assert |s| = λ
h ← reg⟨|K|, i⟩
if K[h] ≠⊥:
  ret h
if hon:
  s ←$ {0,1}^λ
h' ← UNQ(h, hon, s)
if h' ≠ h : ret h'
K[h] ← s
H[h] ← hon
ret h

GET(h)
assert K[h] ≠⊥
ret K[h]

CGET(h)
assert H[h] = 0
ret K[h]

HON(h)
assert H[h] ≠⊥
ret H[h]
```

```
LOG^b
Package Parameters
b:  idealization bit

Package State
L:  log table

UNQ(h, hon, s)
if L[h] ≠ ⊥ :
  (h', hon', s') ← L[h]
  assert s' = s
  if hon' = hon:
    ret h'
for (h', hon', s') ∈ L
        with s = s':
  r ← (level(h) = set)
  r' ← (level(h') = set)
  if r ≠ r' ∧ M[s] = ⊥
     ∧ hon = hon' = 0:
    M[s] ← 1
    L[h] ← (h', hon, s)
    ret h'
  if hon = hon' = 0
     ∧ r = r': abort
  if b ∧ r = r' = 1:
    abort
Log[h] ← (h, hon, s)
ret h
```

Figure 4: Definition of the KEY and LOG package, where **reg**$\langle\cdot\rangle$ is an injective handle constructor, creating a tuple from the inputs. Oracle REG is useful since it can be exposed to the adversary without leading to malformed handles, unlike SET which allows the caller to specify arbitrary handles. The `level` function returns whether the handle is from a SET or REG query.

We formulate security statements by relating advantages. E.g., we upper bound an adversary $\mathcal{A}$'s advantage against a game pair for MLS key distribution security by an adversary $\mathcal{B}$'s advantage against a game pair for the base primitive (e.g., a pseudorandom function) security. Importantly, the runtime of $\mathcal{A}$ and $\mathcal{B}$ will be similar. Namely, we often write $\mathcal{B}$ as $\mathcal{A} \circ \mathcal{R}$ where $\mathcal{R}$ is referred to as a *reduction* which translates $\mathcal{A}$'s oracle queries to its own game (in this case, the game pair for MLS key distribution security) into oracle queries to the game for the base primitive.

State-Separating Proofs. The previous description lacks a natural way to compose games. However, if games are not described as a single block of pseudocode but rather sliced into individual packages of code, then the same package of code can be reused. This notion of code-reuse, together with a separate state for each package results in a natural notion of package composition to form games.

A frequent example of a natural, re-usable code package is the KEY package described in Figure 4. Cryptographic

$\underline{\texttt{PKEY}^{b,\mathsf{pke}}}$

Package Parameters

b: idealization bit
pke: pub. key enc. sch.
i: package index

Package State

$K[h \mapsto k]$: key table
$H[h \mapsto b]$: honesty table
$Q[h \mapsto h]$: handle table

```
SET(h, hon, pk, sk)
assert pke.valid(pk, sk)
if K[h] ≠ ⊥:
  ret Q[h]
if b ∧ hon:
  (pk, sk) <-$ pke.kgen()
h' <- (h, i, pk)
K[h'] <- sk
H[h'] <- hon
Q[h] <- h'
ret h'
```

```
REG(hon, pk, sk)
assert pke.valid(pk, sk)
if hon:
  (pk, sk) <-$ pke.kgen()
h' <- reg⟨(|K|, i, pk)⟩
K[h'] <- sk
H[h'] <- hon
Q[h] <- h'
ret h'
```

```
GET(h)
assert K[h] ≠ ⊥
ret K[h]
```

```
CGET(h)
assert H[h] = 0
ret K[h]
```

```
HON(h)
assert H[h] ≠ ⊥
ret H[h]
```

Figure 5: Definition of the PKEY package. **reg**$\langle\cdot\rangle$ is an injective handle constructor, creating a tuple from the inputs.

primitives often share key material (e.g., in the MLS key schedule, KDF produces outputs which become keys of KDF which produces outputs which become keys of XPD etc., see Figure 1). Now, if KDF computes keys and stores them in a KEY package using the SET oracle, then XPD can retrieve them via the GET oracle.

We now explain how to use KEY packages to model security. For convenience we use orange boxes to indicate $b = 0$ and blue boxes to indicate $b = 1$. Figure 6 and Figure 7 describe the security games for KDF, XPD and HPKE, decomposed into several packages. For KDF and XPD, the upper KEY models the input key, the lower KEY package models the output keys. KDF and XPD are stateless packages, (see Figure 6d) which retrieve an input key from the upper KEY package, perform a computation, and store the result in the lower KEY package. The lower KEY package models security, namely pseudorandomness of the output keys.

Recall that our security notion aims to capture that output keys are indistinguishable from uniformly random values of the same length. Thus, if $b = 0$ in $\texttt{KEY}^{b,\lambda}$, the concrete key values are stored, and if $b = 1$ in $\texttt{KEY}^{b,\lambda}$, then uniformly random keys of the same length are drawn. Finally, the adversary can retrieve the output keys via the GET oracle.

Note, that we want to model multi-instance security, where some of the keys might be known to the adversary. We track the adversary's knowledge of a key by marking the key as *dishonest* (sometimes also called corrupt). In that case we will say that the honesty value hon is 0. In turn, for honest keys, hon is 1. In the case that $hon = 0$, the concrete keys (i.e. the real result of the key computations) are stored regardless of whether $b = 1$ or $b = 0$. It is important that the adversary can access also honest keys via GET and not only dishonest keys, since the game models pseudorandomness of *honest* keys, and for *dishonest* keys, the ideal and real game behave identically.

We model multi-instance security with static corruptions on the input keys and corrupt-upon-derivation (CUD) on the output keys. If all input keys are corrupt, then the output value is marked as corrupt as well. Moreover, if the adversary uses $hon = 0$ as a parameter in its oracle query to EVAL, then the key will be marked as corrupt, too. Else, keys are marked as honest. Note, that in the upper KEY package, honest keys are sampled uniformly at random and secret from the adversary, as the adversary cannot access them via GET, because the GET query cannot be asked by the adversary—the arrows specify which package may call which oracles of which package. Importantly, this means that the graphs are part of the formal game definitions.

Handles. If an adversary wants to interact with a specific key, e.g., to retrieve it via a GET or have it be the subject of another oracle query such as EVAL, they use the *handle* of that key. We construct handles in such a way that each handle injectively maps to a key. When a key is initially randomly generated via the REG oracle, its handle **reg**$\langle\cdot\rangle$ is composed of the index of the package KEY and a counter, implemented as $|K|$ where K is the table of keys already stored in the KEY package, making it globally unique. We denote by **reg** the corresponding and injective handle constructor, see Figure 4.

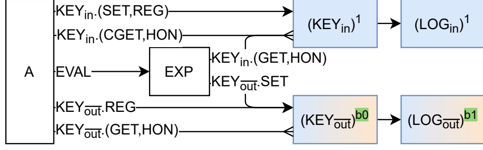
If a key is derived from one or more other keys, the handle of the output key is constructed from the handles of the input key(s), as well as any other inputs to the key derivation. See, e.g., the description of the EVAL oracles in Figure 6d. This guarantees that the handle-to-key mapping is *computationally injective*, i.e., will not have collisions unless we found a collision in the key derivation function used to derive the key.

Finally, let us elaborate on the role of the KEY package in HPKE. Here, KEY models a message that shall remain secret. If the HPKE is idealized, the message is instead replaced with an all-zero string before encryption (Figure 7). Note that pairs of secret keys and public keys are stored in the PKEY package, see Figure 5 and Figure 7.

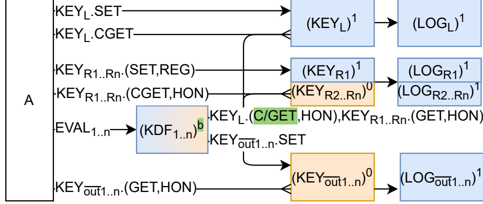
Logging. We now turn to how we model collision resistance and key uniqueness. We call two handles h and h' such that they correspond to the same key in the K table in KEY a *key collision*. For honest, uniformly random keys, such collisions are unlikely, but for keys registered by the adversary or known to the adversary, such collisions can trivially occur. We thus 1) disallow the adversary from registering the same dishonest key value k twice under two different handles (since identical input keys would lead to identical output keys), 2) remove collisions between registered and set dishonest keys (once) and 3) prove that then derived keys do not have collisions. We model both properties using a LOG package which keeps track of the keys and handles used so far and sends a special abort message when a collision occurs.

This modular style of code-writing has recently been developed by Brzuska, Delignat-Lavaud, Fournet, Kohbrok and Kohlweiss (BDFKK [13]) and been coined *state-separating proofs* (SSP) since the state separation of the games enables proofs which rely on code-separation and state-separation. We now state all our assumptions formally using the composed games style and return to proofs in Section VIII-A.

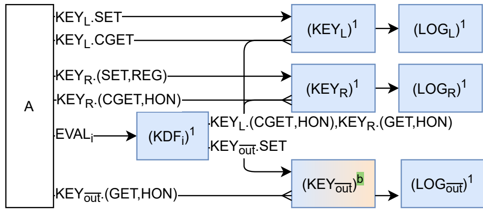
Expand. We consider Krawczyk's XPD function as a function $\mathsf{xpd} : (\{0,1\}^\lambda \times \{0,1\}^*) \mapsto \{0,1\}^{\lambda'}$, where $\lambda' = \lambda'(\lambda)$ is hardcoded (rather than provided as an explicit input). The second input is a pair (lbl, ctx), where in some cases, ctx might be empty. We parametrize our security game for XPD



(a) Game $\text{GEXP}^{b_0,b_1}_{\text{xpd}}$.



(b) Game $\text{GKDFL}^{b}_{\text{kdf}}$ where KDF_i for $i \in \{1, \dots, n\}$ interacts only with KEY_{Ri} and $\text{KEY}_{\overline{\text{out}}i}$.



(c) Game $\text{GKDFR}^{b}_{\text{kdf}}$.

<u>EXP</u>

Package Parameters

xpd : a PRF
$\overline{lbl}$: labels
in : input index
$\overline{out}$: output indices

Package State

no state

$\text{EVAL}(h, \overline{ctx}, \overline{hon})$
$k \leftarrow \text{KEY}_{in}.\text{GET}(h)$
$\overline{hon'} \overset{\text{vec}}{\leftarrow} \text{KEY}_{in}.\text{HON}(h) \wedge \overline{hon}$
$\overline{k'} \overset{\text{vec}}{\leftarrow} \text{xpd}(s, (\overline{lbl}, \overline{ctx}))$
$\overline{h'} \overset{\text{vec}}{\leftarrow} (h, (\overline{lbl}, \overline{ctx}))$
$\overline{h'} \overset{\text{vec}}{\leftarrow} \text{KEY}_{\overline{out}}.\text{SET}(\overline{h'}, \overline{hon'}, \overline{k'})$
ret $\overline{h'}$

<u>KDF^{b}_{i}</u>

Package Parameters

kdf : a KDF
$\overline{lbl}$: labels
b : idealization bit
i : package index
L : left index
R : right index
$\overline{out}$: output indices
λ : output key length

Package State

no state

$\text{EVAL}(h_L, h_R, \overline{ctx}, \overline{hon})$
$hon_L \leftarrow \text{KEY}_L.\text{HON}(h_L)$
$hon_R \leftarrow \text{KEY}_R.\text{HON}(h_R)$
if $b \wedge hon_L$:
  $\overline{k'} \overset{\text{vec}\,\$}{\leftarrow} \{0,1\}^{\lambda}$
else :
  if b: $k_L \leftarrow \text{KEY}_L.\text{CGET}(h_L)$
  else : $k_L \leftarrow \text{KEY}_L.\text{GET}(h_L)$
  $k_R \leftarrow \text{KEY}_R.\text{GET}(h_R)$
  $\overline{ctx'} \overset{\text{vec}}{\leftarrow} (\overline{ctx}, i)$
  $\overline{k'} \overset{\text{vec}}{\leftarrow} \text{kdf}(k_L, k_R, (\overline{lbl}, \overline{ctx'}))$
$\overline{hon'} \overset{\text{vec}}{\leftarrow} (hon_L \vee hon_R) \wedge \overline{hon}$
$\overline{h'} \overset{\text{vec}}{\leftarrow} (h_L, h_R, (\overline{lbl}, \overline{ctx'}))$
$\overline{h'} \overset{\text{vec}}{\leftarrow} \text{KEY}_{\overline{out}}.\text{SET}(\overline{h'}, \overline{hon'}, \overline{k'})$
ret $\overline{h'}$

(d) Code of packages EXP and KDF_i.

Figure 6: Games $\text{GEXP}^{b_0 b_1}_{\text{xpd}}$, $\text{GKDFL}^{b}_{\text{kdf}}$ and $\text{GKDFR}^{b}_{\text{kdf}}$.

with a list of labels $\overline{lbl}$ and measure the security of the XPD function by the advantage which adversaries $\mathcal{A}$ have in distinguishing the real game $\text{GEXP}^{0,0}_{\text{xpd}}$ and the ideal game $\text{GEXP}^{1,1}_{\text{xpd}}$, defined in Figure 6a. For an adversary $\mathcal{A}$, we define the advantage function $\text{Adv}^{\text{xpd}}_{\text{EXP}}(\mathcal{A}) :=$

$$\left|\Pr\left[1 \leftarrow \mathcal{A} \circ \text{GEXP}^{0,0}_{\text{xpd}}\right] - \Pr\left[1 \leftarrow \mathcal{A} \circ \text{GEXP}^{1,1}_{\text{xpd}}\right]\right|.$$

In asymptotic terminology, if the advantage $\text{Adv}^{\text{xpd}}_{\text{EXP}}(\mathcal{A})$ is negligible for all PPT adversaries, then xpd is a secure pseudorandom function. $\text{Adv}^{\text{xpd}}_{\text{EXP}}(\mathcal{A})$ incorporates both, pseudorandomness *and* collision-resistance into a single assumption, since the bit in the LOG also changes from 0 to 1. While this is convenient in proofs, in some cases, we need pseudorandomness only, since collision resistance has already been taken care of. For an adversary $\mathcal{A}$, we write $\text{Adv}^{\text{xpd}}_{\text{PREXP}}(\mathcal{A}) :=$

$$\left|\Pr\left[1 \leftarrow \mathcal{A} \circ \text{GEXP}^{1,1}_{\text{xpd}}\right] - \Pr\left[1 \leftarrow \mathcal{A} \circ \text{GEXP}^{0,1}_{\text{xpd}}\right]\right|.$$

KDF. We model Krawczyk's HKDF as a function kdf : $(\{0,1\}^{\lambda} \times \{0,1\}^{\lambda} \times \{0,1\}^{*}) \mapsto \{0,1\}^{\lambda'}$ for some hard-coded $\lambda' = \lambda'(\lambda)$ such that a triple (k, lbl, ctx) is mapped to a string of length λ'. We assume that kdf is a *dual* KDF. I.e., if either of the inputs is random and secret from the adversary, then kdf produces a pseudorandom output. Idealization based on the *left* input is modeled by a bit in the code of KDF. For an adversary $\mathcal{A}$, we define the advantage $\text{Adv}^{\text{kdf}}_{\text{KDFL}}(\mathcal{A}) :=$

$$\left|\Pr\left[1 \leftarrow \mathcal{A} \circ \text{GKDFL}^{0}_{\text{kdf}}\right] - \Pr\left[1 \leftarrow \mathcal{A} \circ \text{GKDFL}^{1}_{\text{kdf}}\right]\right|,$$

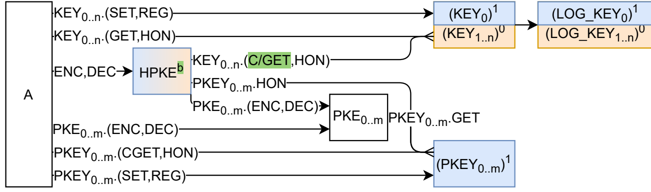
where $\text{GKDFL}^{b}_{\text{kdf}}$ is defined in Figure 6b. Note that here, right inputs might come from one of many packages. In turn, idealization based on the *right* input is modeled by a bit in the lower KEY and for an adversary $\mathcal{A}$, is captured by the advantage $\text{Adv}^{\text{kdf}}_{\text{KDFR}}(\mathcal{A}) :=$

$$\left|\Pr\left[1 \leftarrow \mathcal{A} \circ \text{GKDFR}^{0}_{\text{kdf}}\right] - \Pr\left[1 \leftarrow \mathcal{A} \circ \text{GKDFR}^{1}_{\text{kdf}}\right]\right|,$$

where $\text{GKDFR}^{b}_{\text{kdf}}$ is defined in Figure 6c.

Public-Key Encryption. Lastly, we assume the existence of a public-key encryption (PKE) scheme pke, which consists of three algorithms pke.kgen, pke.enc and pke.dec with standard correctness and confidentiality properties, namely indistinguishability under chosen ciphertext attacks (IND-CCA2). IND-CCA2 captures that encryptions of (adversarially chosen) messages m are computationally indistinguishable from encryptions of $0^{|m|}$, even when the adversary can use a decryption oracle (except on the challenge ciphertexts). We use the PKE scheme exclusively to model Hybrid Public Key Encryption (HPKE), i.e. we only encrypt symmetric keys with the PKE scheme. The length of the symmetric keys is λ. For an adversary $\mathcal{A}$, we define the advantage as $\text{Adv}^{\text{pke}}_{\text{HPKE}}(\mathcal{A}) :=$

$$\left|\Pr\left[1 \leftarrow \mathcal{A} \circ \text{GHPKE}^{0}_{\text{pke}}\right] - \Pr\left[1 \leftarrow \mathcal{A} \circ \text{GHPKE}^{1}_{\text{pke}}\right]\right|,$$



(a) Game $\text{GHPKE}^b_{\text{pke}}$.

```
PKE
Package Parameters
pke : pub. key enc. sch.
pkey‾ : PKEY indices

Package State
H[c ↦ b] : hon. ciph.

ENC(h_kp, m)
pk ← getpk(h_kp)
c ←$ pke.enc(pk, m)
H[c] ← 1
ret c

DEC(h_kp, c)
j ← parse(h_kp, pkey‾)
if H[c] = 1:
  ret ⊥
sk ← PKEY_j.GET(h_kp)
m ← pke.dec(sk, c)
ret m
```

```
HPKE
Package Parameters
b : idealisation bit      pkey‾ : PKEY indices
pke‾ : PKE indices        key‾ : KEY indices
λ : encrypted key length

Package State
Q[c ↦ h] : handle table

ENC(h_kp, h_k)
j ← parse(h_kp, pkey‾)
i ← parse(h_k, key‾)
assert PKEY_j.HON(h_kp)
   ∨ ¬KEY_i.HON(h_k)
if b ∧ PKEY_j.HON(h_pk):
  m ← 0^λ
else if b:
  m ← KEY_i.CGET(h_k)
else :
  m ← KEY_i.GET(h_k)
c ←$ PKE_j.ENC(h_kp, m)
Q[c] ← h_k
ret c

DEC(h_kp, c)
j ← parse(h_kp, pkey‾)
if Q[c] ≠⊥:
  ret Q[c]
m ← PKE_j.DEC(h_kp, c)
ret m
```

(b) Packages PKE and HPKE (Hybrid Public Key Encryption). getpk gets the public key from a handle and parse gets the package index from a handle.

Figure 7: Game $\text{GHPKE}^b_{\text{pke}}$.

where the game $\text{GHPKE}^b_{\text{pke}}$ is defined in Figure 7 and n and m refers to an upper bound on the number of secret-key public-key pairs, and symmetric-keys to be encrypted, respectively. Recall from Section III, that we rely on a *derive key pair function* (DKP), which maps random coins to the output of pke.kgen. As a DKP may fail without producing an error [6] we define the advantage of distinguishing pke.kgen from the output of a DKP as $\mathsf{Adv}^{\mathsf{pke}}_{\mathsf{DKP}}(\mathcal{A})$.

Collision-Resistance. In addition to assuming that xpd and kdf produce pseudorandom outputs, we also assume their collision-resistance. In particular, we encode collision-resistance by assuming that xpd and kdf pass on the uniqueness of their input keys to their output keys, i.e., if the bits in the logs of their input keys have bit 1, then the bits in the log packages of their output keys can also be flipped from 0 to 1. For an adversary $\mathcal{A}$, we denote the advantage of finding a collision either for xpd or kdf by $\mathsf{Adv}^{\mathsf{xpd,kdf}}_{\mathsf{CR}}$.

V. Security Models

We now introduce our security model for the MLS CGKD. We start with a high-level overview which considers the MLS CGKD, defined as a composition of TreeKEM and the key schedule and then turn to modeling TreeKEM and the key schedule each on their own.

CGKD. A crucial component in the modeling are different keys, inputs and outputs as described in Figure 1. Based on the rationale outlined in Section IV, storing keys in separate packages is useful for composition, and so, our model stores each type of key in a separate KEY package, i.e., there are two main packages wCGKD (modeling TreeKEM) and KS, each reading and writing keys from/into the different KEY and PKEY packages, see Figure 8a for the composed game GWCGKD and Figure 8c for GKS. Figure 8b and 8c respectively show the security definition of TreeKEM and the Key Schedule on their own. Note that each KEY package has a different subscript and so have their oracles to ensure that queries are unique across the entire graph. We now map the inputs and outputs described in Figure 1 to the KEY packages.

The values $(pk,\ sk)$ of the KEMs associated with different nodes in the tree are stored in PKEY_NKP packages, the path secrets are stored in the KEY_PS packages with other path secrets (incuding k_{upd} value which the updated samples). Finally, values for the s_{ext} variable are stored in KEY_PSK packages, values for the s_{int} variable are stored in KEY_IS packages, and values of each of the k variables in $\overline{k_{lbl}}$ are stored in KEY_GR.

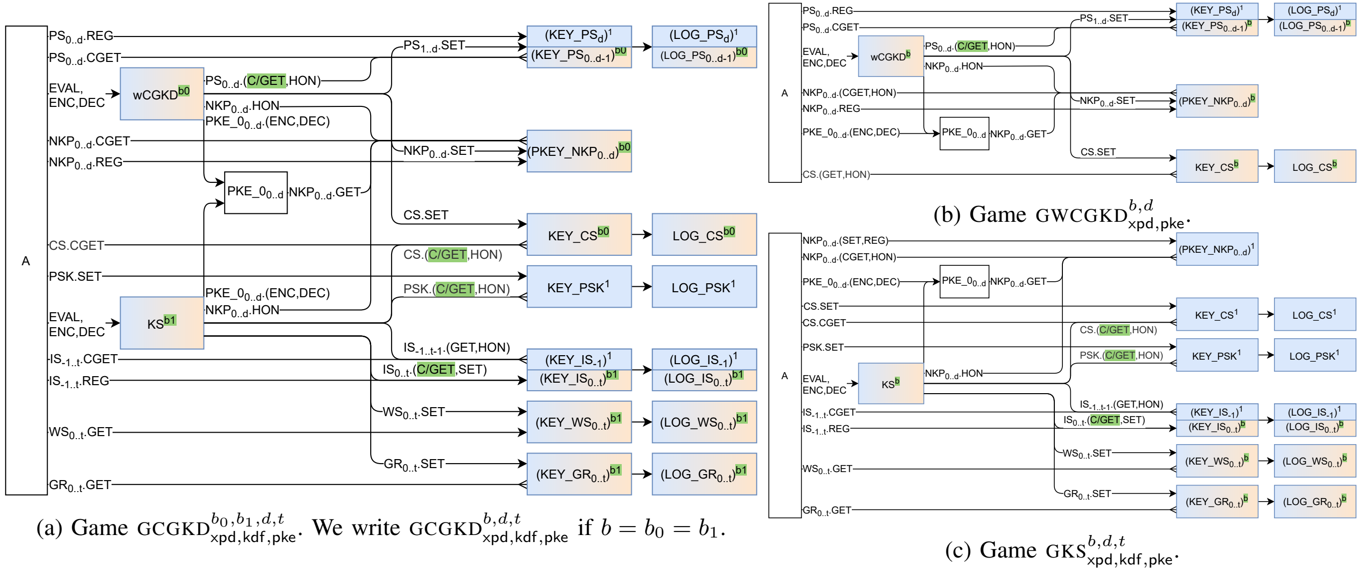
Our base keys are modeled as uniformly random, unique keys (for honest keys) or adversarially chosen (but still unique) keys, e.g., the KEY_PSK starts with an idealization bit $b = 1$, reflecting that keys are sampled uniformly at random and that the adversary cannot register the same dishonest key twice. We can then rely on the assumptions stated in Section IV to prove that a game where the idealization bits of the other KEY packages are all 0 is computationally indistinguishable from a game where the idealization bits of the other KEY packages are all 1. This establishes that the MLS CGKD provides pseudorandom and unique keys.

Definition V.1 (CGKD advantage)**.** *For an expand function* xpd*, a KDF* kdf*, a PKE* pke*, for all polynomials d and t and all adversaries $\mathcal{A}$ which generate trees of depth at most d and run groups for at most t epochs, we denote by* $\mathsf{Adv}^{\mathsf{xpd,kdf,pke}}_{CGKD}(\mathcal{A})$ *the advantage*

$$\left|\Pr\left[1 \leftarrow \mathcal{A} \circ \text{GCGKD}^{0,d,t}_{\mathsf{xpd,kdf,pke}}\right] - \Pr\left[1 \leftarrow \mathcal{A} \circ \text{GCGKD}^{1,d,t}_{\mathsf{xpd,kdf,pke}}\right]\right|.$$

wCGKD In Figure 8b, the wCGKD package represents the composition of packages shown in the center of Figure 9a ($\text{LAYER}_{0..d}$ and EXP). Note that the KEY, PKEY, and PKE packages correspond to those in Figure 8b. The LAYER packages represent an at most d-layer tree structure where each layer is encoded as a separate composition of packages. Note that for each layer, the KEY_PS package, storing path secrets, has a REG query that allows nodes on that layer to function as a leaf node by generating a random (secret) value from scratch.

We define each LAYER package as composition of packages, see Figure 9b. Again, the KEY, PKEY_NKP, and PKE packages correspond to those in Figure 8b. Figure 9b encodes both the dn function (Figure 1) through the EXP and DKP

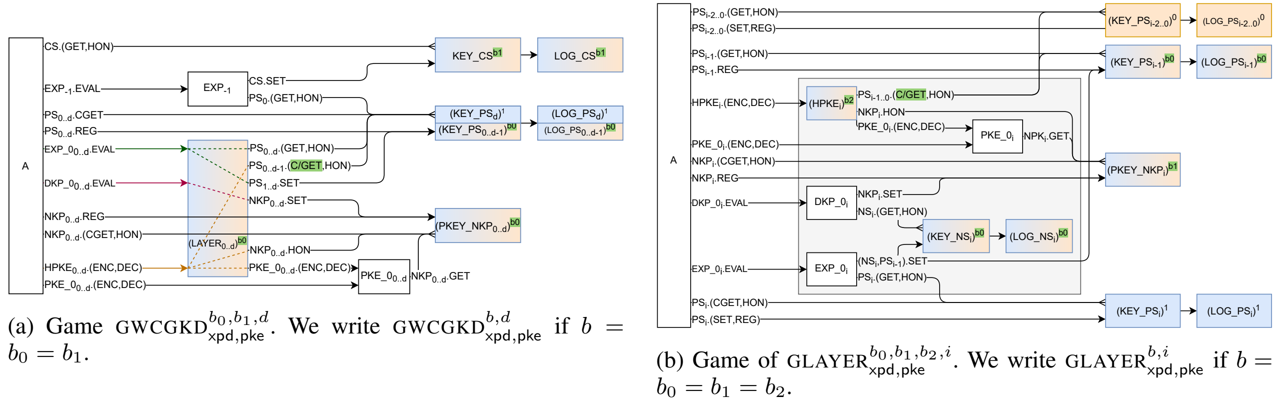


(a) Game $\text{GCGKD}^{b_0,b_1,d,t}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}$. We write $\text{GCGKD}^{b,d,t}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}$ if $b = b_0 = b_1$.

(b) Game $\text{GWCGKD}^{b,d}_{\mathsf{xpd},\mathsf{pke}}$.

(c) Game $\text{GKS}^{b,d,t}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}$.

Figure 8: Composed (left) and individual (right) TreeKEM and Key Schedule games.



(a) Game $\text{GWCGKD}^{b_0,b_1,d}_{\mathsf{xpd},\mathsf{pke}}$. We write $\text{GWCGKD}^{b,d}_{\mathsf{xpd},\mathsf{pke}}$ if $b = b_0 = b_1$.

(b) Game of $\text{GLAYER}^{b_0,b_1,b_2,i}_{\mathsf{xpd},\mathsf{pke}}$. We write $\text{GLAYER}^{b,i}_{\mathsf{xpd},\mathsf{pke}}$ if $b = b_0 = b_1 = b_2$.

Figure 9: TreeKEM subgames.

packages, and the encryption of path secrets to other members through the `HPKE`, `PKE`, `PKEY`, and `KEY_PS` packages. The GET access of `HPKE` to all layers above it encodes that path secrets at various positions in the tree might be encrypted by HPKE.

This model gives significantly more power to the adversary than the MLS TreeKEM interface would (making our security statements only stronger). The model allows the adversary to set up derivations and encryptions that encode arbitrary tree structures. The model merely enforces the concept of layers.

Definition V.2 (wCGKD advantage)**.** *For an expand function* xpd*, a PKE* pke*, for all polynomials* d *and all adversaries* $\mathcal{A}$ *which generate trees of depth at most* d*, we denote by* $\mathsf{Adv}^{\mathsf{xpd},\mathsf{pke}}_{wCGKD}(\mathcal{A})$ *the advantage*

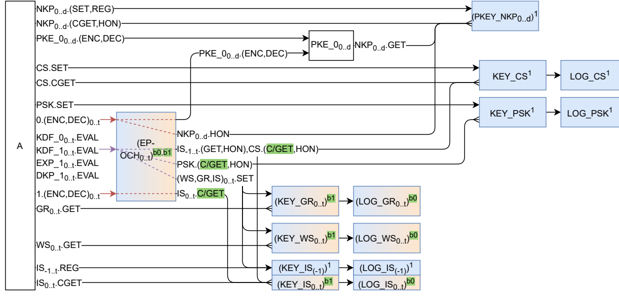
$$\left|\Pr\left[1 \leftarrow \mathcal{A} \circ \text{GWCGKD}^{0,d}_{\mathsf{xpd},\mathsf{pke}}\right] - \Pr\left[1 \leftarrow \mathcal{A} \circ \text{GWCGKD}^{1,d}_{\mathsf{xpd},\mathsf{pke}}\right]\right|.$$

Key Schedule model. We define the `KS` package in Figure 8c by the composition of packages shown in the center of Figure 10a ($\texttt{EPOCH}_{0..t}$). The `KEY`, `PKEY`, and `PKE` packages correspond to those in Figure 8c. The `EPOCH` packages represent a t-epoch group where each epoch is encoded as a separate composition of packages. Each `EPOCH` package is the composition of packages shown in the center of Figure 10b. Recall that the `KEY`, `PKEY`, and `PKE` packages correspond to those in Figure 8c.

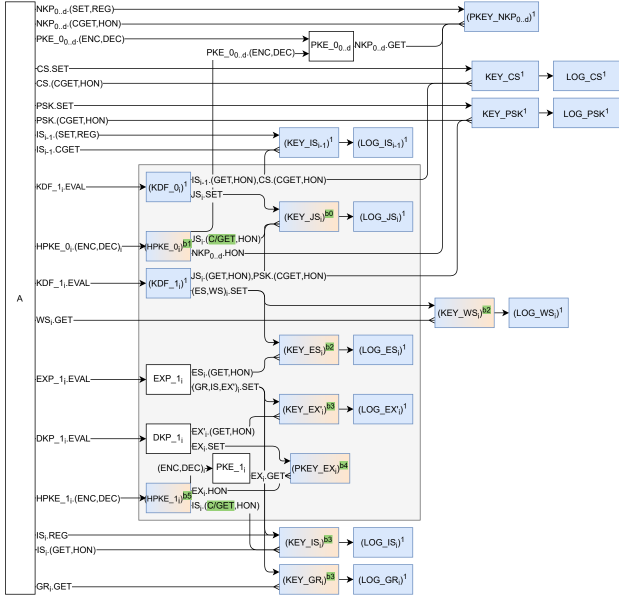
Figure 10b encodes the key schedule ks (Figure 1). The `KDF` and `EXP` packages correspond to kdf and xpd calls, and the `DKP` package corresponds to the dkp call. The `HPKE` packages encode the encryption of the joiner secret to new members (top), and the encryption of init secrets by external committers (bottom). The REG query allows the adversary to create external init secrets. The values of external key pairs, $(pk_{\text{ext}}, sk_{\text{ext}})$, are stored in the `PKEY_EX` package.

Definition V.3 (KS advantage)**.** *For an expand function* xpd*, a key derivation function* kdf*, a PKE* pke*, for all polynomials* d*,* t *and all adversaries* $\mathcal{A}$ *which generate trees of depth at most* d *and run groups for at most* t *epochs,* $\mathsf{Adv}^{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}_{KS}(\mathcal{A})$ *denotes*

$$\left|\Pr\left[1 \leftarrow \mathcal{A} \circ \text{GKS}^{0,d,t}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}\right] - \Pr\left[1 \leftarrow \mathcal{A} \circ \text{GKS}^{1,d,t}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}\right]\right|.$$



(a) Game $\text{GGKS}^{b_0,b_1,d,t}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}$. We write $\text{GGKS}^{b,d,t}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}$ if $b = b_0 = b_1$.



(b) Game $\text{GEPOCH}^{b_0,b_1,b_2,b_3,b_4,b_5,d,t,i}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}$. We write $\text{GEPOCH}^{b,d,t,i}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}$ if $b = b_0 = b_1 = b_2 = b_3 = b_4 = b_5$. Note that $\mathtt{KEY_IS}_{-1}$ should technically have a REG query, but for an ideal $\mathtt{KEY}$ package REG and SET are functionally equivalent.

Figure 10: Key Schedule subgames.

VI. Adversarial Model and Freshness

In this section, we first discuss how key distribution security properties relate to protocol-level security properties the latter of which express usually them as *freshness* conditions. We then discuss the adversarial model of CGKD, wCGKD, as well as the key schedule model and compare it to the recent model by Alwen, Jost and Mularczyk (AJM, [4]).

Honesty and Freshness. As discussed in Section IV, we use honesty tables to keep track of keys that are known to the adversary (i.e. dishonest keys) or keys which we expect to provide security (i.e. honest keys). Our notion of honesty closely relates to the *freshness* notion for keys in the context of traditional key exchange security models. We show how the structure of each of our models yields a key exchange style freshness condition for the relevant output keys of the model. We express this freshness/honesty in terms of the honesty of the base keys and the adversary's actions.

Each freshness condition follows the same pattern, namely, the honesty of the output key is a function of the honesty of the keys it is derived from. We can express the honesty of a key by a *honesty graph*, which is a directed, acyclic graph, where each edge represents the boolean honesty value of a key and each node represents an operation the values of its incoming edges. Edges without an origin or destination node represent the honesty values of input or output keys, respectively.

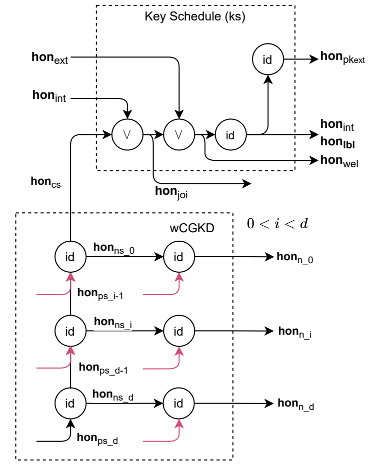


Figure 11: Honesty graph of CGKD. *id* indicates that the honesty is passed on and ∨ indicates that the result is honest if at least one of the inputs is honest. The pink arrows indicate that the input value is either a fresh input value or a derived one. Each node has an additional, implicit input, which turns the output to 0, indicating that an adversary can corrupt each value upon derivation.

If a key is derived from a single key, it either inherits that key's honesty or it is immediately corrupted by the adversary. This mechanism is visible in the definition of EVAL oracle of the $\mathtt{EXP}$ package (Figure 6), where the adversary can pass in an $\overline{hon}$ value to indicate if they want the output key to simply inherit the honesty of the input key, or if they want the output key to be dishonest.

If a key is derived from two keys, it is honest if one of the input keys is honest and the adversary doesn't decide to compromise it immediately. Similar to single-key derivation, this mechanism is visible in the definition of the EVAL oracle of the GKDF package (Figure 6).

Therefore, we can now express the honesty of any key k in the graph as a function of the honesty of the root keys from which the key is derived. Namely, the key k is honest only if there is at least one root key which is (a) honest and (b) where none of the keys on the path from the root key to k is corrupted.

We provide the honesty graph which is induced by CGKD in Figure 11. The honesty graph of each individual output key is a subgraph of the entire model's structural graph, and the honesty graphs of key schedule and wCGKD are subgraphs of the CGKD graph, see Figure 11. In addition, in each derivation, the adversary can corrupt upon derivation, and thus, the resulting honesty value of an (intermediate or output) key is always the $\wedge$ of the honesty value shown in Figure 11 and the value provided by the adversary in the derivation. For simplicity, we do not depice the CUD in the graph—it would amount to adding and $\wedge$ node right after each derivation.

Forward Secrecy (FS). Traditionally, key exchange models define forward secrecy in terms of long term keys and in a scenario where sessions terminate. The first does not apply,

as our model does not consider long term keys. The second does not apply, because MLS considers long-term group conversations that do not explicitly terminate. Thus, messaging protocols —and MLS in particular— seek to achieve fine-grained FS by constantly deriving keys, such that old keys can be deleted after use.

In our model, FS is expressed by the fact that the honesty graph is directed, i.e. the honesty of a key does not depend on keys that are derived from it. If an adversary compromises a key, keys from earlier epochs are still considered honest.

Post-Compromise Security (PCS). PCS was defined in the context of messaging protocols in [15] and expresses that a party can *recover* from compromise by introducing new key material into an existing session (and sharing that key material with group members), as long as the adversary is passive during that procedure.

PCS is expressed in the honesty graph by the fact that a key derived from two input keys (one corresponds to the old session key and one to the new key material) is honest if at least one of the input keys is honest (and the adversary doesn't decide to compromise again).

For example, the PCS property of MLS is expressed throught the fact that even if the init secret s_{int} is dishonest, the joiner secret k_{joi} of the subsequent epoch can be honest if the commit secret k_{cs} is honest.

Freshness on Protocol-Level. MLS is a complex protocol and our work analyses a component of it. While we leave a full analysis based on our model for future work, we now extrapolate from the key distribution freshness to the protocol-level freshness.

For example, one can conclude from the honesty graph in Figure 11 that in MLS, a single honest input key (for example a leaf key) suffices to guarantee the security of an output key. To model a protocol operation such as `update` on top of our model, some derived keys now need to be encrypted under public-keys (marked as `dopath` in Figure 1). If they are encrypted under a dishonest public-key, they become known to the adversary. Thus, when deriving them, they need to be derived with an *hon* $= 0$ value so they are marked as dishonest, since our model only allows dishonest keys to be encrypted to corrupted public-keys (see the description of HPKE.ENC in Figure 7).

This procedure allows protocol-level models to encode many protocol-level attack scenarios such as active insider attacks or the *double join* scenario, where a party has access to secrets that are not in its direct path. Namely, the corruption mechanism of the protocol-level model would then corrupt these intermediate secrets and provide the adversary with access.

This means that a protocol-level freshness condition can't rely on a fine-grained honesty graph such as the one in Figure 11, (which would show the corruption upon derivation of a secret, but not the reason for the corruption, e.g. a corrupt public key to which it is encrypted). Instead, it would more likely encode freshness in terms of the corruption of parties, which is what one would expect and what is the case, e.g. in the work of AJM (Figure 15 in [4]).

In summary, while our model doesn't consider protocol-level attacks directly, we can still show that any protocol-level model built on CGKD will give a number of guarantees, for example, that the output keys of the key schedule are fresh only if either the previous epoch's init secret is fresh or the injected PSK is fresh or the new input secret after an update operation, as well as any public key it is encrypted to.

Adversarial Model. The models introduced in Section V allow the adversary to corrupt all base secrets statically, and to decide to corrupt derived keys upon derivation. This corruption-upon-derivation (CUD) model simplifies the security analysis (in comparison to fully adaptive corruption) and captures the security aimed for by MLS in a meaningful way.

We chose this limitation in adversarial capabilities, as our model can't allow the adversary to corrupt keys after they are used, as it would allow them to trivially win the distinguishing game. This problem is known as the *commitment problem*, initially uncovered by Nielsen [28]. Still, there is a gap between the minimal restriction required by avoiding the commitment problem and forcing the adversary to decide on corruption in the moment of derivation. It seems an interesting direction to explore whether this gap can be partially bridged by (1) postponing the onset of idealized behaviour and (2) delaying a derivation query to the point where a key is used, since else, the model does not produce any key-dependent information (and everything else can be simulated).

Comparison to AJM As the security model used in AJM is strictly stronger than that of ACDT, ACCK+, as well as ACJM, we now compare our model with AJM.

The first difference to AJM is that we analyze the key distribution of MLS rather than the protocol itself. Specifically, the adversary doesn't have access to MLS protocol operations such as `update`, `add`, etc, but can instead control the inputs to the various key generation and derivation structures of the protocol directly. This means that in contrast to AJM (and traditional (key exchange) protocol security models in general), our model does not consider parties, but instead presents the adversary with access to each part of a more global protocol state (across sessions).

This allows adversaries to inject or generate key material at the various interfaces of the components of MLS and perform key derivation operations based on these keys without the strict separation of state into parties, groups or even epochs. For example, an adversary could generate an (honest) key for injection into the MLS key schedule as a PSK. That key could then be injected into the key schedule in epoch of any group and of any party and in conjuntion with any other key material. The only restriction we pose is that given by the structure of the protocol: An honest key generated for injection at a given point in the key schedule cannot be used, for example, for injection at another point in the key schedule. Of course, the adversary can still inject the same (dishonest) key in multiple places.

Additionally, our work differs from AJM in that we don't model the authentication properties of MLS and in particular don't consider the compromise of any signature keys. Instead, our model focuses on providing the adversary with fine-

grained capabilities to compromise symmetric key material. We argue that overall, this gives the adversary more state compromise capabilities compared to AJM (with the exception of the signature key). This is because firstly, any compromise an adversary would make on a protocol level can be simulated by compromising the individual keys on a more granular level and secondly, the fine-grained access our model provides allows it to capture additional scenarios such as cross-group and cross-epoch attacks.

Finally, the findings of our analysis as detailed in this section correlate closely with those of AJM: Given our adversarial model, we establish that the keys produced by TreeKEM and the key schedule of MLS are pseudorandom if the output key is fresh. Extrapolating our freshness condition to that of a protocol-level model such as that of AJM (Figure 15 in [4]) is not straightforward, but we have shown examples earlier in this section, of how our CGKD can allow a protocol-level model to express fine-grained security guarantees comparable to those proven by AJM.

VII. Theorems

This section relates the security of the MLS CGKD to the security of the underlying primitives XPD, KDF, and HPKE. We first make separate security claims for the wCGKD and the key schedule and then bound the security of their composition.

Theorem 1. *(wCGKD Security) For all polynomials d and all adversaries $\mathcal{A}$ which generate trees of depth at most d, it holds that*

$$\mathsf{Adv}^{\mathsf{xpd,pke}}_{wCGKD}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{xpd}}_{EXP}(\mathcal{A} \circ \mathcal{R}_{exp}) + \sum_{i=0}^{d-1} (\mathsf{Adv}^{\mathsf{xpd}}_{EXP}(\mathcal{A} \circ \mathcal{R}_i \circ \mathcal{R}_{exp,i})$$
$$+ \mathsf{Adv}^{\mathsf{pke}}_{DKP}(\mathcal{A} \circ \mathcal{R}_i \circ \mathcal{R}_{dkp,i})) + \mathsf{Adv}^{\mathsf{pke}}_{HPKE}(\mathcal{A} \circ \mathcal{R}_i \circ \mathcal{R}_{hpke,i}))$$

where $\text{GWCGKD}^{b,d}_{\mathsf{xpd,pke}}$ *is provided in Figure 8b.*

Theorem 2. *(Key Schedule Security) For all polynomials d and t and all adversaries $\mathcal{A}$ which generate trees of depth at most d and run groups for at most t epochs, it holds that*

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{xpd,kdf,pke}}_{KS}(\mathcal{A}) \leq\; & \mathsf{Adv}^{\mathsf{xpd,kdf,dkp}}_{CR}(\mathcal{A} \circ \mathcal{R}_{cr}) \\
& + \mathsf{Adv}^{\mathsf{kdf}}_{KDFL}(\mathcal{A} \circ \mathcal{R}_{int}) + \mathsf{Adv}^{\mathsf{kdf}}_{KDFL}(\mathcal{A} \circ \mathcal{R}_{joi}) \\
& + \sum_{i=0}^{t} \mathsf{Adv}^{\mathsf{kdf}}_{KDFR}(\mathcal{A} \circ \mathcal{R}_{hyb,i} \circ \mathcal{R}^{ks}_{1,i}) + \mathsf{Adv}^{\mathsf{pke}}_{HPKE}(\mathcal{A} \circ \mathcal{R}_{hyb,i} \circ \mathcal{R}^{ks}_{2,i}) \\
& \quad + \mathsf{Adv}^{\mathsf{kdf}}_{KDFR}(\mathcal{A} \circ \mathcal{R}_{hyb,i} \circ \mathcal{R}^{ks}_{3,i}) + \mathsf{Adv}^{\mathsf{xpd}}_{EXP}(\mathcal{A} \circ \mathcal{R}_{hyb,i} \circ \mathcal{R}^{ks}_{4,i}) \\
& \quad + \mathsf{Adv}^{\mathsf{pke}}_{DKP}(\mathcal{A} \circ \mathcal{R}_{hyb,i} \circ \mathcal{R}^{ks}_{5,i}), + \mathsf{Adv}^{\mathsf{pke}}_{HPKE}(\mathcal{A} \circ \mathcal{R}_{hyb,i} \circ \mathcal{R}^{ks}_{6,i}),
\end{aligned}$$

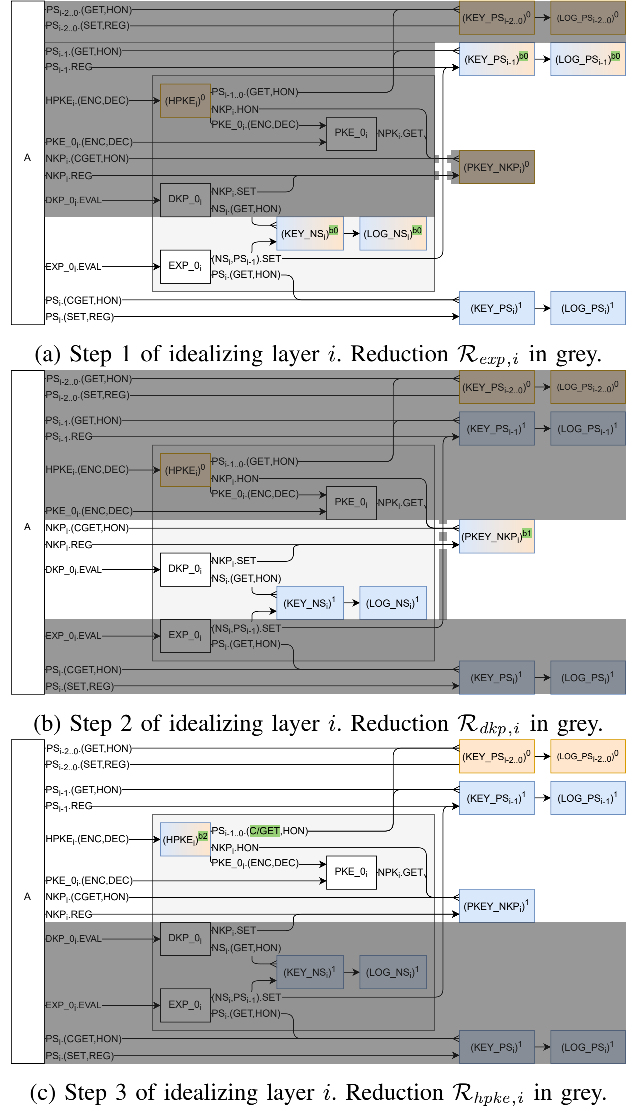
where $\text{GKS}^{b,d,t}_{\mathsf{xpd,kdf,pke}}$ *is defined in Figure 8c.*

Theorem 3. *(CGKD) For all polynomials d and t and all adversaries $\mathcal{A}$ which generate trees of depth at most d and run groups for at most t epochs, it holds that* $\mathsf{Adv}^{\mathsf{xpd,kdf,pke}}_{CGKD}(\mathcal{A})$

$$\leq \mathsf{Adv}^{\mathsf{xpd,pke}}_{wCGKD}(\mathcal{A} \circ \mathcal{R}_{wCGKD}) + \mathsf{Adv}^{\mathsf{xpd,kdf,pke}}_{KS}(\mathcal{A} \circ \mathcal{R}_{KS}),$$

where $\text{GCGKD}^{b,d,t}_{\mathsf{xpd,kdf,pke}}$ *is given in Figure 8a.*

We prove Theorem 1 in Appendix VIII-A, provide the proof of Theorem 3 in Appendix B and the proof of Theorem 2 in Appendix C, D and E.



(a) Step 1 of idealizing layer i. Reduction $\mathcal{R}_{exp,i}$ in grey.

(b) Step 2 of idealizing layer i. Reduction $\mathcal{R}_{dkp,i}$ in grey.

(c) Step 3 of idealizing layer i. Reduction $\mathcal{R}_{hpke,i}$ in grey.

Figure 12: Proof of Lemma 1.

VIII. Proof Methodology

Our proofs rely on central concept of the state-separating proofs methodology: (1) Code composition & code re-use, (2) Precise reductions and (3) Indices.

Code Composition and code re-use We use packages similar to module systems in programming languages where a bigger program can be built out of smaller modules/packages. I.e., we only define the code of each base package once and then define all games and reductions by (call-)graphs whose nodes consist of copies of the base packages, cf. Section IV and Section V, where we defined our security models.

Code re-use for explicit reductions. Since we define the composition of packages visually via a graph we can check that a reduction works via graph *pattern-matching*. For example, Figure 12a reduces the difference between two games to EXP security. The gray area is the reduction $\mathcal{R}_{exp,i}$ and the non-gray area is the $\text{GEXP}^{b_0,b_0}_{\mathsf{xpd}}$ which was defined in Figure 6a (with $b_1 = b_0$). Denoting the two games defined by Figure 24

Package	in	$\overline{out}$	$\overline{lbl}$	Fig.
$\mathtt{EXP}_{-1}$	$\mathtt{PS}_i$	$(\mathtt{CS})$	(“path”)	9a
$\mathtt{EXP}_0_i$	$\mathtt{PS}_i$	$(\mathtt{NS}_i, \mathtt{PS}_{i-1})$	(“node”, “path”)	9b
$\mathtt{EXP}_1_i$	$\mathtt{ES}_i$	$(\mathtt{EX}'_i, \mathtt{IS}_i, \mathtt{GR}_i)$	(“external”, “init”, ...)	10b
Package	(L, R)	$\overline{out}$	$\overline{lbl}$	Fig.
$\mathtt{KDF}_0_i$	$(\mathtt{CS}, \mathtt{IS}_{i-1})$	$(\mathtt{JS}_i)$	(“joiner”)	10b
$\mathtt{KDF}_1_i$	$(\mathtt{PSK}, \mathtt{JS}_i)$	$(\mathtt{ES}_i, \mathtt{WS}_i)$	(“epoch”, “welcome”)	10b
Package	$\overline{keys}$	$\overline{pkeys}$	$\overline{pkes}$	Fig.
$\mathtt{HPKE}_i$	$(\mathtt{PS}_{i-1}, .., \mathtt{PS}_0)$	$(\mathtt{NKP}_i)$	$(\mathtt{PKE}_0_i)$	9b
$\mathtt{HPKE}_0_i$	$(\mathtt{JS}_i)$	$(\mathtt{NKP}_0, .., \mathtt{NKP}_d)$	$(\mathtt{PKE}_0_0, .., \mathtt{PKE}_0_d)$	10b
$\mathtt{HPKE}_1_i$	$(\mathtt{IS}_i)$	$(\mathtt{EX}_i)$	$(\mathtt{PKE}_1_i)$	10b

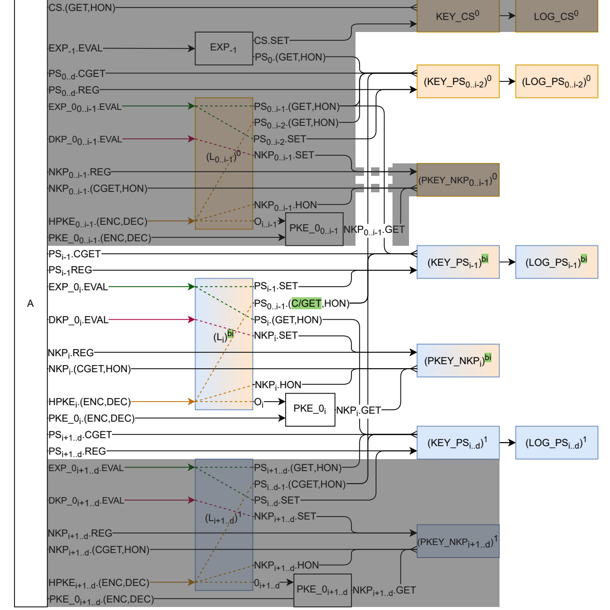
Figure 13: Values of package parameters.

(one for $b_0 = 0$ and one for $b_0 = 1$) as $\mathtt{G}^0$ and $\mathtt{G}^1$, if an adversary $\mathcal{A}$ can distinguish between $\mathtt{G}^0$ and $\mathtt{G}^1$, then the adversary $\mathcal{A} \circ \mathcal{R}_{exp,i}$ can distinguish between $\mathtt{GEXP}^{0,0}_{\mathsf{xpd}}$ and $\mathtt{GEXP}^{1,1}_{\mathsf{xpd}}$. I.e., the reduction $\mathcal{R}_{exp,i}$ is a part of code of the games $\mathtt{G}^0$ and $\mathtt{G}^1$ which we *move into the adversary*. This way of reasoning gives (a) precise definitions of reductions and (b) straightforward arguments that the simulation of the reduction is perfect. This not only makes proofs easier to verify, it also speeds up our own work since we write less reductions and changes in the code of a package immediately propagate through the entire proof and article. Publishing the code of this article should allow others to make small changes to the code without affecting the validity of the proof.

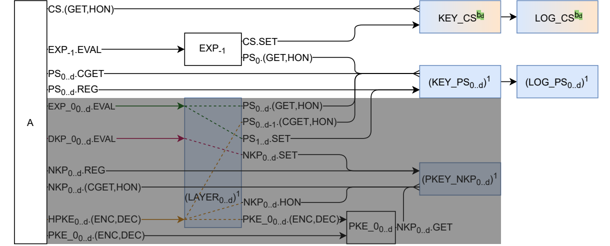
Package Indices. Composing different packages requires us to specify their interfaces and match them. In particular, we often compose multiple packages of the same kind in parallel, e.g., the $\mathtt{GKDFR}^b_{\mathsf{kdf}}$ game in Figure 25 composes $(\mathtt{KEY_IS}_{i-1})^1$, $(\mathtt{KEY_JS}_i)^{b0}$ and $(\mathtt{KEY_CS})^1$ packages in parallel which are all a variant of the $\mathtt{KEY}^b$ package. This is analogous to classes and objects in object-oriented programming languages. A package definition is akin to a *class*, describing a number of method (oracles) and member variables (package state). Following that analogy, a package (or package instance), such as $(\mathtt{KEY_IS}_{i-1})^1$ is an *object* of that class, with the index (in subscript) distinguishing it from other objects/package instances, and with the superscript denoting member variable values (for brevity values clear from the context are omitted). Additionally, a package name suffix may be used to distinguish instances as well. If a package does not have a subscript this means there is only one instance of the package. If a package has a range (or vector) of indices (e.g. $(0..x)$) this means there are x parallel instances of the package.

Bit Indices. The proofs of lemmas and theorems consist of game hops each of which flips a bit which we indicate by highlighting the bit (and related queries) in green in the diagrams accompanying a proof. The index of the highlighted bit corresponds to the *index* of the game hop. That is, we state all our theorems and lemmas in terms of flipping a *single* bit.

Scoping. Adversary and bit variables are *local* to a definition, theorem or diagram and unrelated to the use of a variable with the same name in other theorems and diagrams. I.e., in theorems and proofs, we split a single bit b into several bits and construct multiple adversaries by composing an adversary with a reduction into $\mathcal{A} \circ \mathcal{R}$. Note that reduction names are *globally unique* in this article.



(a) Hybrid argument. Reduction $\mathcal{R}_i$ in grey.



(b) Step to idealize the root layer and the *commit secret*.

Figure 14: Proof of Theorem 1.

A. Proof of Theorem 1 (wCGKD security)

The proof of Theorem 1 consists of Lemma 1 and a hybrid argument. We start with the former which shows that a TreeKEM layer with an ideal path secret can be completely idealized such that the path secret of its parent is ideal as well as the encryptions it performs.

Lemma 1. *(Layer Security) Let $\mathcal{B}$ be an adversary and $\mathrm{GLAYER}^{b,i}_{\mathsf{xpd,pke}}$ the indistinguishability game defined by Figure 9b. Then it holds that*

$$\mathsf{Adv}^{\mathsf{xpd,pke},i}_{LAYER}(\mathcal{B}) \leq \mathsf{Adv}^{\mathsf{xpd}}_{EXP}(\mathcal{B} \circ \mathcal{R}_{exp,i}) + \mathsf{Adv}^{\mathsf{pke}}_{DKP}(\mathcal{B} \circ \mathcal{R}_{dkp,i}) + \mathsf{Adv}^{\mathsf{pke}}_{HPKE}(\mathcal{B} \circ \mathcal{R}_{hpke,i}),$$

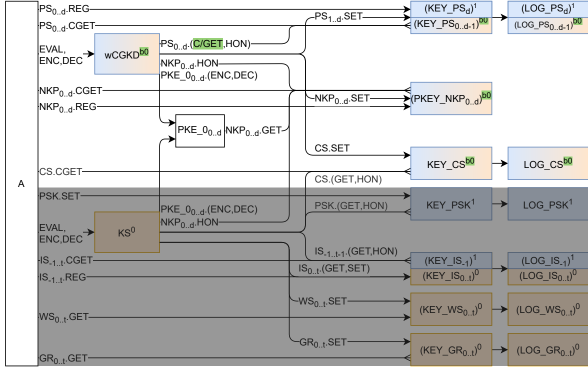
where $\mathcal{R}_{exp,i}$ is defined as the grey area in Figure 12a where $\mathcal{R}_{dkp,i}$ as the grey area in Figure 12b, and $\mathcal{R}_{hpke,i}$ as the grey area in Figure 12c.

The proof of Lemma 1 consists of three steps, each of which modifies one or two bits in the overall game. We denote

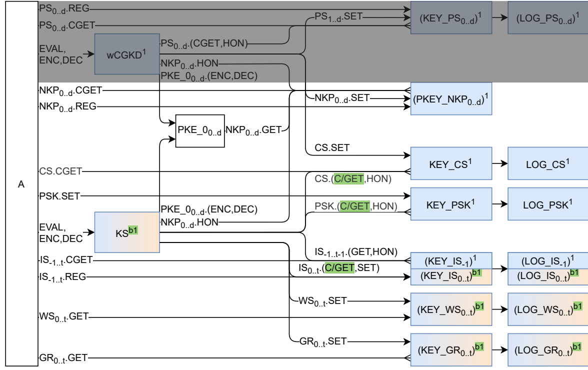
$$\mathrm{GLAYER}^{b,i}_{\mathsf{xpd,pke}} := \mathrm{GLAYER}^{b,b,b,i}_{\mathsf{xpd,pke}},$$

i.e., $\mathrm{GLAYER}^{b_0,b_1,b_2,i}_{\mathsf{xpd,pke}}$ with $b = b_0 = b_1 = b_2$.

In the first proof step, we apply the expand assumption for $\overline{lbl} = \{path, node\}$ to idealize the node secret of layer i and the path secret of layer $p(i)$. We obtain the following bound:



(a) First idealization step of composition theorem.



(b) Second idealization step of composition theorem.

Figure 15: Theorem 3.

$$\left|\Pr\left[1 \leftarrow \mathcal{B} \circ \mathrm{GLAYER}^{0,0,0,i}_{\mathsf{xpd,pke}}\right] - \Pr\left[1 \leftarrow \mathcal{B} \circ \mathrm{GLAYER}^{1,0,0,i}_{\mathsf{xpd,pke}}\right]\right|$$
$$\leq \mathsf{Adv}^{\mathsf{xpd}}_{\mathrm{EXP}}(\mathcal{B} \circ \mathcal{R}_{exp,i}),$$

where $\mathcal{R}_{exp,i}$ is defined as the grey area in Figure 12a.

In the second step of this proof we apply the DKP assumption to idealize the $\mathtt{PKEY}$ package of layer i, and we obtain that

$$\left|\Pr\left[1 \leftarrow \mathcal{B} \circ \mathrm{GLAYER}^{1,0,0,i}_{\mathsf{xpd,pke}}\right] - \Pr\left[1 \leftarrow \mathcal{B} \circ \mathrm{GLAYER}^{1,1,0,i}_{\mathsf{xpd,pke}}\right]\right|$$

$\leq \mathsf{Adv}^{\mathsf{pke}}_{\mathrm{DKP}}(\mathcal{B} \circ \mathcal{R}_{dkp,i})$. See Figure 12b for $\mathcal{R}_{dkp,i}$.

In the third and last step of this proof we apply the HPKE assumption for $n = i - 1$ and $m = 1$ to idealize the $\mathtt{HPKE}$ package of layer i. This gives us the following bound w.r.t. $\mathrm{GLAYER}^{b,i}_{\mathsf{xpd,pke}}$

$$\left|\Pr\left[1 \leftarrow \mathcal{B} \circ \mathrm{GLAYER}^{1,1,0,i}_{\mathsf{xpd,pke}}\right] - \Pr\left[1 \leftarrow \mathcal{B} \circ \mathrm{GLAYER}^{1,1,1,i}_{\mathsf{xpd,pke}}\right]\right|$$
$$\leq \mathsf{Adv}^{\mathsf{pke}}_{\mathrm{HPKE}}(\mathcal{B} \circ \mathcal{R}_{hpke,i}),$$

where $\mathcal{R}_{hpke,i}$ is defined as the grey area in Figure 12c. Hence,

$$\mathsf{Adv}^{\mathsf{xpd,pke},i}_{\mathrm{LAYER}}(\mathcal{B}) \leq \mathsf{Adv}^{\mathsf{xpd}}_{\mathrm{EXP}}(\mathcal{B} \circ \mathcal{R}_{exp,i}) + \leq \mathsf{Adv}^{\mathsf{pke}}_{\mathrm{DKP}}(\mathcal{B} \circ \mathcal{R}_{dkp,i}) + \mathsf{Adv}^{\mathsf{pke}}_{\mathrm{HPKE}}(\mathcal{B} \circ \mathcal{R}_{hpke,i}),$$

which concludes the proof of Lemma 1.

Hybrid argument. We apply a hybrid over Lemma 1 to the game GWCGKD defined by Figure 9a, which, as explained in Section V, is equivalent to Figure 8b.

For convenience we add several bits into the superscript of GWCGKD to model security of the different layers, i.e., if $b = b_0 = ... = b_d$, then

$$\mathrm{GWCGKD}^{b_0,\ldots,b_d,d}_{\mathsf{xpd,pke}} = \mathrm{GWCGKD}^{b,d}_{\mathsf{xpd,pke}}.$$

Recall that the $\mathtt{KEY_PS}_d$ package represents base secrets at the lowest possible leaf layer. Thus, the package is idealized so that its values are either corrupt and chosen by the adversary or honest, randomly generated and never shared.

Due to Lemma 1, we can idealize any layer with an ideal path secret, resulting in an ideal path secret for the parent layer. Hence, we use a hybrid argument upwards through the tree, see Figure 14a. Hence, for all PPT adversaries $\mathcal{A}$,

$$\left|\Pr\left[1 \leftarrow \mathcal{A} \circ \mathrm{GWCGKD}^{1,0,\ldots,0,d}_{\mathsf{xpd,pke}}\right] - \Pr\left[1 \leftarrow \mathcal{A} \circ \mathrm{GWCGKD}^{1,\ldots,1,0,d}_{\mathsf{xpd,pke}}\right]\right|$$
$$\leq \mathsf{Adv}^{\mathsf{xpd,pke},i}_{\mathrm{LAYER}}(\mathcal{A} \circ \mathcal{R}_i),$$

where $\mathcal{R}_i$ is defined as the grey area in Figure 14a. We apply this hybrid argument until $\mathtt{KEY_PS}_0$ is ideal. At this point the entire TreeKEM tree is ideal, except for the commit secret derived at the root. Therefore, as visualized in Figure 14b, we apply the expand assumption and obtain

$$\left|\Pr\left[1 \leftarrow \mathcal{A} \circ \mathrm{GWCGKD}^{1,\ldots,1,0,d}_{\mathsf{xpd,pke}}\right] - \Pr\left[1 \leftarrow \mathcal{A} \circ \mathrm{GWCGKD}^{1,\ldots,1,1,d}_{\mathsf{xpd,pke}}\right]\right|$$
$$\leq \mathsf{Adv}^{\mathsf{xpd}}_{\mathrm{EXP}}(\mathcal{A} \circ \mathcal{R}_{exp}),$$

where $\mathcal{R}_{exp}$ is defined as the grey area in Figure 14b. Using $\mathcal{B} = \mathcal{A} \circ \mathcal{R}_i$, we derive the desired bound for Theorem 1:

$$\begin{aligned}\mathsf{Adv}^{\mathsf{xpd,pke}}_{\mathrm{wCGKD}}(\mathcal{A}) \leq\ & \mathsf{Adv}^{\mathsf{xpd}}_{\mathrm{EXP}}(\mathcal{A} \circ \mathcal{R}_{exp}) + \sum_{i=0}^{d-1} \mathsf{Adv}^{\mathsf{xpd,pke},i}_{\mathrm{LAYER}}(\mathcal{A} \circ \mathcal{R}_i) \\ =\ & \mathsf{Adv}^{\mathsf{xpd}}_{\mathrm{EXP}}(\mathcal{A} \circ \mathcal{R}_{exp}) + \sum_{i=0}^{d-1}(\mathsf{Adv}^{\mathsf{xpd}}_{\mathrm{EXP}}(\mathcal{A} \circ \mathcal{R}_i \circ \mathcal{R}_{exp,i}) \\ & + \mathsf{Adv}^{\mathsf{pke}}_{\mathrm{DKP}}(\mathcal{A} \circ \mathcal{R}_i \circ \mathcal{R}_{dkp,i}) + \mathsf{Adv}^{\mathsf{pke}}_{\mathrm{HPKE}}(\mathcal{A} \circ \mathcal{R}_i \circ \mathcal{R}_{hpke,i})).\end{aligned}$$

REFERENCES

[1] Joël Alwen, Margarita Capretto, Miguel Cueto, Chethan Kamath, Karen Klein, Ilia Markov, Guillermo Pascual-Perez, Krzysztof Pietrzak, Michael Walter, and Michelle Yeo. Keep the dirt: Tainted TreeKEM, adaptively and actively secure continuous group key agreement. Cryptology ePrint Archive, Report 2019/1489, 2019. https://eprint.iacr.org/2019/1489.

[2] Joël Alwen, Sandro Coretti, Yevgeniy Dodis, and Yiannis Tselekounis. Security analysis and improvements for the IETF MLS standard for group messaging. Cryptology ePrint Archive, Report 2019/1189, 2019. https://eprint.iacr.org/2019/1189.

[3] Joël Alwen, Sandro Coretti, Daniel Jost, and Marta Mularczyk. Continuous group key agreement with active security. Cryptology ePrint Archive, Report 2020/752, 2020. https://eprint.iacr.org/2020/752.

[4] Joël Alwen, Daniel Jost, and Marta Mularczyk. On the insider security of mls. Cryptology ePrint Archive, Report 2020/1327, 2020. https://eprint.iacr.org/2020/1327.

[5] Richard Barnes, Benjamin Beurdouche, Jon Millican, Emad Omara, Katriel Cohn-Gordon, and Raphael Robert. The Messaging Layer Security (MLS) Protocol. Internet-Draft draft-ietf-mls-protocol-11, Internet Engineering Task Force, December 2020. Work in Progress.

[6] Richard Barnes, Karthikeyan Bhargavan, Benjamin Lipp, and Christopher A Wood. Hybrid public key encryption. Technical report, IRTF Internet-Draft draft-irtf-cfrg-hpke-02, 2019.

[7] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, CRYPTO '93, volume 773 of *LNCS*, pages 232–249, Santa Barbara, CA, USA, August 22–26, 1994. Springer, Heidelberg, Germany.

[8] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In Serge Vaudenay, editor, *E*UROCRYPT 2006, volume 4004 of *LNCS*, pages 409–426, St. Petersburg, Russia, May 28 – June 1, 2006. Springer, Heidelberg, Germany.

[9] Karthikeyan Bhargavan, Richard Barnes, and Eric Rescorla. Treekem: Asynchronous decentralized key management for large dynamic groups a protocol proposal for messaging layer security (mls). Preprint, 2018. https://prosecco.gforge.inria.fr/personal/karthik/pubs/treekem.pdf.

[10] Karthikeyan Bhargavan, Benjamin Beurdouche, and Prasad Naldurg. Formal Models and Verified Protocols for Group Messaging: Attacks and Proofs for IETF MLS. Research report, Inria Paris, December 2019.

[11] Nikita Borisov, Ian Goldberg, and Eric Brewer. Off-the-record communication, or, why not to use pgp. In *P*roceedings of the 2004 ACM Workshop on Privacy in the Electronic Society, WPES '04, page 77–84, New York, NY, USA, 2004. Association for Computing Machinery.

[12] Chris Brzuska, Antoine Delignat-Lavaud, Christoph Egger, Cédric Fournet, Konrad Kohbrok, and Markulf Kohlweiss. Handshake security for the TLS 1.3 standard. https://eprint.iacr.org/2021/467.

[13] Chris Brzuska, Antoine Delignat-Lavaud, Cédric Fournet, Konrad Kohbrok, and Markulf Kohlweiss. State separation for code-based game-playing proofs. In Thomas Peyrin and Steven Galbraith, editors, *A*SIACRYPT 2018, Part III, volume 11274 of *LNCS*, pages 222–249, Brisbane, Queensland, Australia, December 2–6, 2018. Springer, Heidelberg, Germany.

[14] Chris Brzuska and Jan Winkelmann. Nprfs and their application to message layer security. Preprint. http://chrisbrzuska.de/2020-NPRF.html.

[15] Katriel Cohn-Gordon, Cas Cremers, and Luke Garratt. On post-compromise security. Cryptology ePrint Archive, Report 2016/221, 2016. http://eprint.iacr.org/2016/221.

[16] Katriel Cohn-Gordon, Cas Cremers, Luke Garratt, Jon Millican, and Kevin Milner. On ends-to-ends encryption: Asynchronous group messaging with strong security guarantees. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *A*CM CCS 2018, pages 1802–1819, Toronto, ON, Canada, October 15–19, 2018. ACM Press.

[17] Katriel Cohn-Gordon, Cas J. F. Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. A formal security analysis of the signal messaging protocol. In *2*017 IEEE European Symposium on Security and Privacy, EuroS&P 2017, Paris, France, April 26-28, 2017, pages 451–466. IEEE, 2017.

[18] Eric Cornelissen. Pull request 453: Use the GroupContext to derive the joiner_secret. https://github.com/mlswg/mls-protocol/pull/453.

[19] Cas Cremers, Britta Hale, and Konrad Kohbrok. Efficient post-compromise security beyond one group. Cryptology ePrint Archive, Report 2019/477, 2019. https://eprint.iacr.org/2019/477.

[20] Cyprien Delpech de Saint Guilhem, Marc Fischlin, and Bogdan Warinschi. Authentication in key-exchange: Definitions, relations and composition. 2019.

[21] Benjamin Dowling, Paul Rösler, and Jörg Schwenk. Flexible authenticated and confidential channel establishment (fACCE): Analyzing the noise protocol framework. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *P*KC 2020, Part I, volume 12110 of *LNCS*, pages 341–373, Edinburgh, UK, May 4–7, 2020. Springer, Heidelberg, Germany.

[22] Yongdae Kim, Adrian Perrig, and Gene Tsudik. Communication-efficient group key agreement. In *T*rusted Information: The New Decade Challenge, IFIP TC11 Sixteenth Annual Working Conference on Information Security (IFIP/Sec'01), June 11-13, 2001, Paris, France, volume 193 of *I*FIP Conference Proceedings, pages 229–244. Kluwer, 2001.

[23] Hugo Krawczyk. SIGMA: The "SIGn-and-MAc" approach to authenticated Diffie-Hellman and its use in the IKE protocols. In *C*RYPTO 2003, volume 2729 of *LNCS*, pages 400–425, Santa Barbara, CA, USA, August 17–21, 2003. Springer, Heidelberg, Germany.

[24] Hugo Krawczyk. Cryptographic extraction and key derivation: The HKDF scheme. In Tal Rabin, editor, *C*RYPTO 2010, volume 6223 of *LNCS*, pages 631–648, Santa Barbara, CA, USA, August 15–19, 2010. Springer, Heidelberg, Germany.

[25] Hugo Krawczyk and Hoeteck Wee. The OPTLS protocol and TLS 1.3. Cryptology ePrint Archive, Report 2015/978, 2015. http://eprint.iacr.org/2015/978.

[26] M. Marlinspike and T. Perrin. Signal specifications. Technical report, 2016. https://signal.org/docs/.

[27] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *H*andbook of Applied Cryptography. CRC Press, Boca Raton, Florida, 1996.

[28] Jesper Buus Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In Moti Yung, editor, *C*RYPTO 2002, volume 2442 of *LNCS*, pages 111–126, Santa Barbara, CA, USA, August 18–22, 2002. Springer, Heidelberg, Germany.

$\mathsf{ks}(k_{cs}, s_{int}, s_{ext}, \overline{lbl}, ctx)$

$k_{\mathrm{joi}} \leftarrow \mathsf{kdf}(k_{\mathrm{cs}}, s_{int}, (\text{“joiner”}, ctx))$
$(s'_{int}, \overline{k}_{\overline{lbl}}, k_{wel}, pk'_{ext}, sk'_{ext})$
$\quad \leftarrow \mathsf{subks}(k_{\mathrm{joi}}, s_{ext}, (\overline{lbl}, ctx))$
$\mathbf{ret}\ (s'_{int}, \overline{k}_{\overline{lbl}}, k_{wel}, k_{joi}, pk_{ext}, sk_{ext})$

$\mathsf{subks}(k_{\mathrm{joi}}, s_{ext}, \overline{lbl}, ctx)$

$k_{epc} \leftarrow \mathsf{kdf}(s_{ext}, k_{\mathrm{joi}}, (\text{“epoch”}, ctx))$
$k_{wel} \leftarrow \mathsf{kdf}(s_{ext}, k_{\mathrm{joi}}, (\text{“welcome”}, \text{“}\bot\text{”}))$
$s_{int'} \leftarrow \mathsf{xpd}(k_{epc}, (\text{“init”}, \text{“}\bot\text{”}))$
$\overline{k}_{\overline{lbl}} \overset{\text{vec}}{\leftarrow} \mathsf{xpd}(k_{epc}, (\overline{lbl}, \text{“}\bot\text{”}))$
$k_{exs} \leftarrow \mathsf{xpd}(k_{epc}, (\text{“external”}, \text{“}\bot\text{”}))$
$(pk'_{ext}, sk'_{ext}) \leftarrow \mathsf{dkp}(k_{exs})$
$\mathbf{ret}\ (s'_{int}, \overline{k}_{\overline{lbl}}, k_{wel}, k_{joi}, pk_{ext}, sk_{ext})$

$\mathsf{dn}(ps)$

$ps' \leftarrow \mathsf{xpd}(ps, \text{“path”})$
$ns \leftarrow \mathsf{xpd}(ps, \text{“node”})$
$(pk, sk) \leftarrow \mathsf{dkp}(ns)$
$\mathbf{ret}\ (ps', pk, sk)$

Figure 16: The MLS Key Schedule algorithm (ks) and the MLS Derive Node algorithm (dn).

[29] Emad Omara, Benjamin Beurdouche, Eric Rescorla, Srinivas Inguva, Albert Kwon, and Alan Duric. The messaging layer security (mls) architecture rfc draft 04. 2019. https://tools.ietf.org/html/draft-ietf-mls-architecture-04.

[30] Matthew Weidner. *G*roup messaging for secure asynchronous collaboration. PhD thesis, MPhil Dissertation, 2019. Advisors: A. Beresford and M. Kleppmann, 2019.

[31] P. Zimmermann, A. Johnston, and J. Callas. Zrtp: Media path key agreement for unicast secure rtp. RFC 6189, RFC Editor, April 2011. http://www.rfc-editor.org/rfc/rfc6189.txt.

Appendix

A. Protocol

In this appendix, we describe the algorithms update, process, and join of the MLS CGKD as well as the algorithms wupdate, wprocess, and wjoin. We first describe their syntax and then provide the algorithms.

Definition A.1 (CGKD). *A continuous group key distribution (CGKD)* cgkd *consists of four PPT algorithms* (pke.kgen, update, process, join) *with inputs and outputs as specified by Figure 2 that satisfy the correctness criteria Valid Keys, Consistent Public Views and Consistent Secret Views.*

Valid Keys demands that all (sk, pk) returned by the algorithms are in the range of pke.kgen. *Consistent Public Views* captures that different group members have consistent views on the public-keys PK_{mem} of each node as well as of the external public-key pk_{ext}. *Consistent Secret Views* requires that each pair of group members agrees on 1) the values of k and s'_{int}, 2) the value of the external secret-key sk_{ext} and 3) the secret keys associated with the nodes the direct path of group members, i.e., the intersection of the lists SK_{own} of two nodes agree.

Definition A.2 (wCGKD). *A weak continuous group key distribution (wCGKD)* wcgkd *consists of four PPT algorithms* (pke.kgen, wupdate, wprocess, wjoin) *with inputs and outputs as specified by Figure 2 such that the correctness criteria Valid Keys, Consistent Public Views and Consistent Secret Views are satisfied.*

The correctness criteria are the same, except that they are restricted to the inputs and outputs of wcgkd (Figure 2).

Definition A.3 (KS). *A key schedule* ks *takes as input three symmetric keys k_{cs}, s_{int}, and s_{ext}, a vector of labels $\overline{lbl}$ and a*

```
update(PK_mem, PK_rem, PK_joi,
       SK_own, i_own, k_upd, CP,
       pk_ext, s_ext, s_int, ctx)
-----------------------------------------
if |PK_rem| > 0 ∨ pk_ext ≠⊥:
  assert k_upd ≠⊥
(C_cp, PK'_upd, PK'_mem, SK'_own, k_cs)
  ← wupdate(PK_mem, PK_rem,
     PK_joi, SK_own, i_own, k_upd, CP)
c_ext ←$ pke.enc(pk_ext, s_int)
(s'_int, k̄_lbl, k_joi, pk'_ext, sk'_ext)
  ← ks(k_cs, s_int, s_ext, lbl̄, ctx)
SK'_own[−1] ← sk'_ext
C_joi ← [ ]
for i_nod, pk ∈ PK_joi:
  C_joi[i_nod] ←$ pke.enc(pk, k_joi)
ret (C_cp, C_joi, c_ext, PK'_upd,
  PK'_mem, SK'_own, k̄_lbl, s'_int, pk'_ext)
```

```
dopath(PK_mem, i_cur, k_upd, CP)
-----------------------------------------
C_cp ← [ ]
PK'_own, SK'_own ← [ ], [ ]
ps ← k_upd
while i_cur ≠⊥:
  (ps', pk, sk) ← dn(ps)
  PK'_own[i_cur] ← pk
  SK'_own[i_cur] ← sk
  if i_cur ≠ i_own:
    for i_nod ∈ CP[i_cur]:
      pk_nod ← PK_mem[i_nod]
      c ←$ pke.enc(pk_nod, ps')
      C_cp[i_nod] ← c
  i_cur ← parent(i_cur, |PK_mem|)
  ps ← ps'
ret (C_cp, PK'_own, SK'_own, ps)
```

```
process(PK_mem, PK_rem, PK_joi,
        PK_upd, SK_own, i_nod, i_upd,
        s_ext, s_int, ctx, c, c_ext)
-----------------------------------------
(PK'_mem, SK'_own, k_cs) ←
  wprocess(PK_mem, PK_rem, PK_joi,
           PK_upd, SK_own, i_nod, i_upd, c)
s_int ← pke.dec(SK_own[−1], c_ext)
assert s_int ≠⊥
(s'_int, k̄_lbl, _, pk'_ext, sk'_ext)
  ← ks(k_cs, s_int, s_ext, lbl̄, ctx)
SK'_own[−1] ← sk'_ext
ret (PK'_mem, SK'_own, k̄_lbl, s'_int, pk'_ext)
```

```
join(PK_mem, SK_own, i_own, i_upd, s_ext,
     ctx, c)
-----------------------------------------
SK'_own ← SK_own
sk_own ← SK'_own[i_own]
k_joi ← pke.dec(sk_own, c)
assert k_joi ≠⊥
(s'_int, k̄_lbl, _, pk'_ext, sk'_ext)
  ← subks(k_joi, s_ext, lbl̄, ctx)
SK'_own[−1] ← sk'_ext
ret (PK_mem, SK'_own, k̄_lbl, s'_int, pk'_ext)
```

```
wupdate(PK_mem, PK_rem, PK_joi,
        SK_own, i_own, k_upd, CP)
-----------------------------------------
k_cs ← 0
PK'_mem ← PK_mem
PK'_mem ←rem− PK_rem
PK'_mem ←merge− PK_joi
C_cp, PK'_upd ← [ ], [ ]
SK'_own ← SK_own
for i ∈ directpath(i_own, |PK_mem|):
  PK_upd[i] ← PK'_mem[i]
(C_cp, PK'_upd, SK'_own, k_cs)
  ← dopath(PK'_mem, i_own, k_upd, CP)
PK'_mem ←merge− PK'_upd
ret (C_cp, PK'_upd, PK'_mem, SK'_own, k_cs)
```

Figure 17: Procedures update, process, join, and wupdate.

context value ctx. It returns a secret key k_{lbl} for each $lbl \in \overline{lbl}$, a new key s_{int}, a pair (sk, pk), key k_{joi} and key k_{wel}.

For a more general definition, one might replace the subscript joi with a more general subscript.

We now provide pseudocode for MLS CGKD, MLS wCGKD and MLS KS. Recall Section III for a high-level description of update and see Figure 17 for its code. The blue and pink inputs are optional. The code colored in the corresponding color is only executed if the corresponding input is provided. I.e., encryption to the external public-key pk_{ext} is only performed if there is an external public-key pk_{ext} in the input, modeling and external add operation. Note that there is an if condition which requires that if pk_{ext} is given, then also k_{upd} must be given. This is because an external add operation requires adding new key material as the external updater does not know the current key material. In turn, for an internal add operation, if is it performed as add-only operation, then k_{upd} would be empty. In this case, the wupdate procedure (Figure 17) does not run the encryption to the co-path, encoded

```
wprocess(PK_mem, PK_rem,
         PK_joi, PK_upd,
         SK_own, i_nod, i_upd, c)
-----------------------------------------
k_cs ← 0
PK'_mem ← PK_mem
PK'_mem ←rem− PK_rem
PK'_mem ←merge− PK_joi
PK'_mem ←vec− PK_upd
SK'_own ← SK_own
sk_nod ← SK_own[i_nod]
ps ← pke.dec(sk_nod, c)
assert ps ≠⊥
i_cur ← lcp(i_own, i_upd, |PK'_mem|)
(_, _, SK''_own, k_cs)
  ← dopath(PK'_mem, i_cur, ps, ⊥)
SK'_own ←vec− SK''_own
ret (PK'_mem, SK'_own, k_cs)
```

```
wjoin(PK_mem, SK_own, i_own,
      i_upd, c)
-----------------------------------------
k_cs ← 0
sk_own ← SK_own[i_own]
ps ← pke.dec(sk_own, c)
assert ps ≠⊥
i_cur ← lcp(i_own, i_upd, |PK_mem|)
(_, _, _, k_cs)
  ← dopath(PK_mem, i_cur, ps, ⊥)
ret (PK_mem, SK_own, k_cs)
```

Figure 18: Procedures wprocess and wjoin.

as the dopath procedure.

The algorithms process and join update their key material consistently with update. I.e., the ciphertexts C_{cp} and the list of changes to the public-key information PK_{rem}, PK_{joi} and PK_{upd} suffice for process to derive the same values as update. Similarly, the ciphertexts C_{joi} and the view of the entire new tree PK_{mem} suffice to allow the join procedure to derive the same values. See Figure 17 for their code.

B. Proof of Theorem 3 (CGKD security)

We prove Theorem 3 using Theorem 1 and Theorem 2. Recall (see Figure 8a) that, if $b = b_0 = b_1$, then

$$\mathrm{GCGKD}^{b_0,b_1,d,t}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}} = \mathrm{GCGKD}^{b,d,t}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}.$$

In the first step of this proof we modify b_0 from 0 to 1 and reduce to GWCGKD. For all PPT adversaries $\mathcal{A}$, we obtain

$$\left|\Pr\left[1 \leftarrow \mathcal{A} \circ \mathrm{GCGKD}^{0,0,d,t}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}\right] - \Pr\left[1 \leftarrow \mathcal{A} \circ \mathrm{GCGKD}^{1,0,d,t}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}\right]\right|$$
$$\leq \mathsf{Adv}^{\mathsf{xpd},\mathsf{pke}}_{\mathrm{wCGKD}}(\mathcal{A} \circ \mathcal{R}_{\mathrm{wCGKD}}),$$

where $\mathcal{R}_{\mathrm{wCGKD}}$ is defined as the grey area in Figure 15a. Next, we modify b_1 from 0 to 1 and reduce to GKS. We obtain

$$\left|\Pr\left[1 \leftarrow \mathcal{A} \circ \mathrm{GCGKD}^{1,0,d,t}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}\right] - \Pr\left[1 \leftarrow \mathcal{A} \circ \mathrm{GCGKD}^{1,1,d,t}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}\right]\right|$$
$$\leq \mathsf{Adv}^{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}_{\mathrm{KS}}(\mathcal{A} \circ \mathcal{R}_{\mathrm{KS}}),$$

where $\mathcal{R}_{\mathrm{KS}}$ is defined as the grey area in Figure 15b. Hence, we obtain the desired bound for Theorem 3:

$$\mathsf{Adv}^{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}_{\mathrm{CGKD}}(\mathcal{A})$$
$$\leq \mathsf{Adv}^{\mathsf{xpd},\mathsf{pke}}_{\mathrm{wCGKD}}(\mathcal{A} \circ \mathcal{R}_{\mathrm{wCGKD}}) + \mathsf{Adv}^{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}_{\mathrm{KS}}(\mathcal{A} \circ \mathcal{R}_{\mathrm{KS}}).$$

C. Proof of Theorem 2 (KS security)

This appendix proves Theorem 2. We first idealize collision-resistance for all primitives (unique inputs translate into unique outputs), then idealize the two KDFs of the key schedule based on the commit secret and the external secret globally for all epochs (since the commit secret and the external secret do not have an epoch) and then finally apply a hybrid argument over the epochs which is captured by Lemma 3.

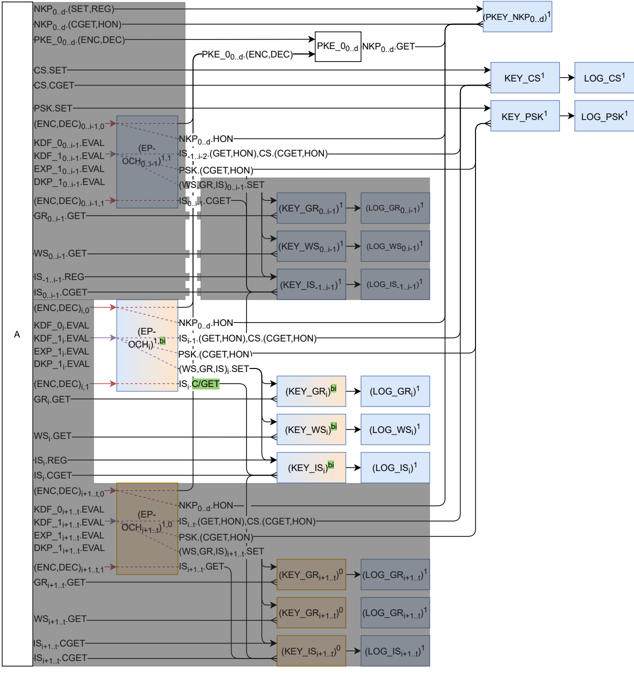


Figure 19: Proof of Theorem 2. Hybrid step which reduces to a single epoch. Reduction $\mathcal{R}_{\text{hyb,i}}$ is marked in grey.

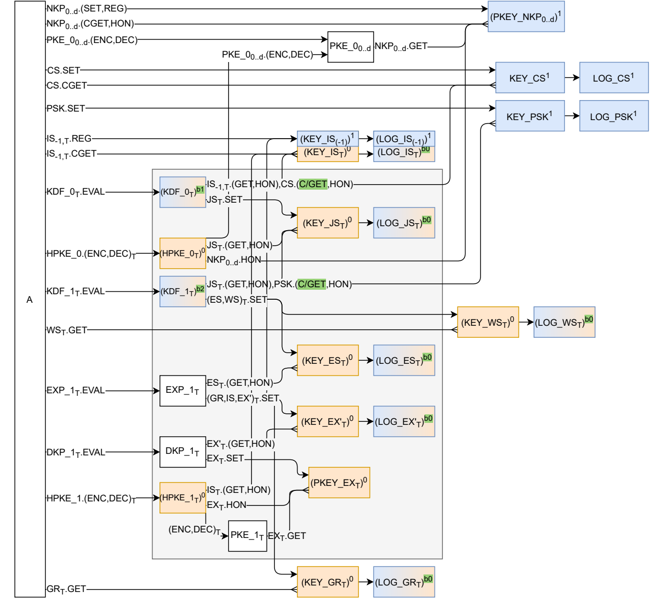


Figure 20: $\text{GPGKS}^{b_0,b_1,b_2,d,t}_{\text{xpd,kdf,pke}}$ where $T = (0..t)$.

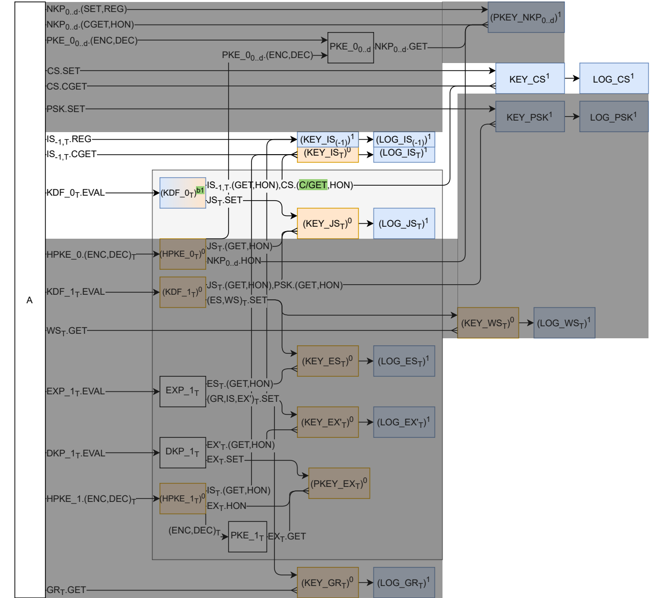


Figure 21: Reducing to KDFL security based on ideal CS. Reduction $\mathcal{R}_{\text{cs}}$ is marked in grey, where $T = (0..t)$.

$\texttt{KEY_IS}_0$ represents the initial init secret and is ideal from the start, i.e., it samples honest values independently at random and allows the adversary to set corrupt values. Similarly, the pre-shared keys (KEY_PSK) are ideal from the start as they are exchanged out-of-bound. Lastly, note that the public-private key-pairs (PKEY_NKP) and commit secrets (KEY_CS) are ideal form the start, which follows from Theorem 1.

Figure 10a depicts $\text{GGKS}^{b_0,b_1,d,t}_{\text{xpd,kdf,pke}}$ for trees up to depth d and groups that run for up to t epochs. For $b = b_0 = b_1$, we define $\text{GKS}^{b,d,t}_{\text{xpd,kdf,pke}} := \text{GGKS}^{b_0,b_1,d,t}_{\text{xpd,kdf,pke}}$. Here, b_0 corresponds to *global* proof steps (collision-resistance and KDFs), and b_1 corresponds to *epoch-wise* proof steps. Note that the second G in GGKS stands for *global*.

We now state Lemma 2 and Lemma 3, then show that they imply Theorem 2 and then prove Lemma 2 and Lemma 3.

Lemma 2. *For all polynomials d and t and all adversaries $\mathcal{A}$ which generate trees of depth at most d and run groups for at most t epochs, it holds that* $\mathsf{Adv}^{\mathsf{xpd,kdf,pke},(0,0),(1,0)}_{GGKS}(\mathcal{A}) :=$

$$\left|\Pr\left[1 \leftarrow \mathcal{A} \circ \text{GGKS}^{0,0,d,t}_{\text{xpd,kdf,pke}}\right] - \Pr\left[1 \leftarrow \mathcal{A} \circ \text{GGKS}^{1,0,d,t}_{\text{xpd,kdf,pke}}\right]\right| \\ \leq \mathsf{Adv}^{\mathsf{xpd,kdf}}_{CR}(\mathcal{A} \circ \mathcal{R}_{cr}) + \mathsf{Adv}^{\mathsf{kdf}}_{KDFL}(\mathcal{A} \circ \mathcal{R}_{cs}) \\ + \mathsf{Adv}^{\mathsf{kdf}}_{KDFL}(\mathcal{A} \circ \mathcal{R}_{psk})$$

Recall that $\text{GGKS}^{b_0,b_1,d,t}_{\text{xpd,kdf,pke}}$ is defined in Figure 10a. Analogously, we also use the notation $\mathsf{Adv}^{\mathsf{xpd,kdf,pke},(1,0),(1,1)}_{\text{GGKS}}(\mathcal{A}) :=$

$$\left|\Pr\left[1 \leftarrow \mathcal{A} \circ \text{GGKS}^{1,0,d,t}_{\text{xpd,kdf,pke}}\right] - \Pr\left[1 \leftarrow \mathcal{A} \circ \text{GGKS}^{1,1,d,t}_{\text{xpd,kdf,pke}}\right]\right|.$$

Lemma 3. *(Epoch Security) Let $\mathcal{B}$ be an adversary and $\text{GEPOCH}^{b,d,t,i}_{\text{xpd,kdf,pke}}$ the indistinguishability game defined by Figure 23. Then $\forall d, t, i \in \mathbb{N} : 0 \leq i \leq t$,* $\mathsf{Adv}^{\mathsf{xpd,kdf,pke},i}_{EPOCH}(\mathcal{B}) :=$

$$\left|\Pr\left[1 \leftarrow \mathcal{B} \circ \text{GEPOCH}^{0,d,t,i}_{\text{xpd,kdf,pke}}\right] - \Pr\left[1 \leftarrow \mathcal{B} \circ \text{GEPOCH}^{1,d,t,i}_{\text{xpd,kdf,pke}}\right]\right| \\ \leq \mathsf{Adv}^{\mathsf{kdf}}_{KDFR}(\mathcal{B} \circ \mathcal{R}^{ks}_{1,i}) + \mathsf{Adv}^{\mathsf{pke}}_{HPKE}(\mathcal{B} \circ \mathcal{R}^{ks}_{2,i}) + \mathsf{Adv}^{\mathsf{kdf}}_{KDFR}(\mathcal{B} \circ \mathcal{R}^{ks}_{3,i}) \\ + \mathsf{Adv}^{\mathsf{xpd}}_{EXP}(\mathcal{B} \circ \mathcal{R}^{ks}_{4,i}) + \mathsf{Adv}^{\mathsf{pke}}_{DKP}(\mathcal{B} \circ \mathcal{R}^{ks}_{5,i}) + \mathsf{Adv}^{\mathsf{pke}}_{HPKE}(\mathcal{B} \circ \mathcal{R}^{ks}_{6,i}),$$

where $\mathcal{R}^{ks}_{1,i}$ is defined as the grey area in Figure 24, $\mathcal{R}^{ks}_{2,i}$ as the grey area in Figure 25, $\mathcal{R}^{ks}_{3,i}$ as the grey area in Figure 26, $\mathcal{R}^{ks}_{4,i}$ as the grey area in Figure 27, and $\mathcal{R}^{ks}_{5,i}$ as the grey area in Figure 28.

Proof of Theorem 2. We first apply Lemma 2 and then apply a hybrid argument over the t epochs using that for $0 \leq i \leq t-1$,

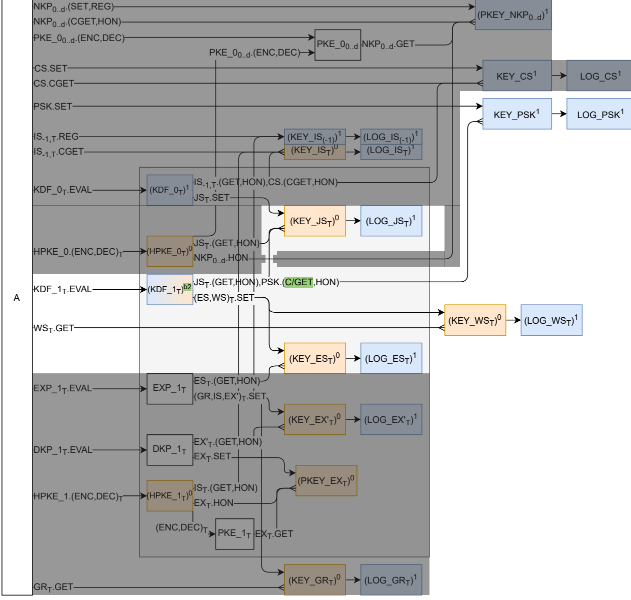


Figure 22: Reducing to KDFL security based on ideal PSK. Reduction $\mathcal{R}_{\mathsf{psk}}$ is marked in grey, where $T = (0..t)$.

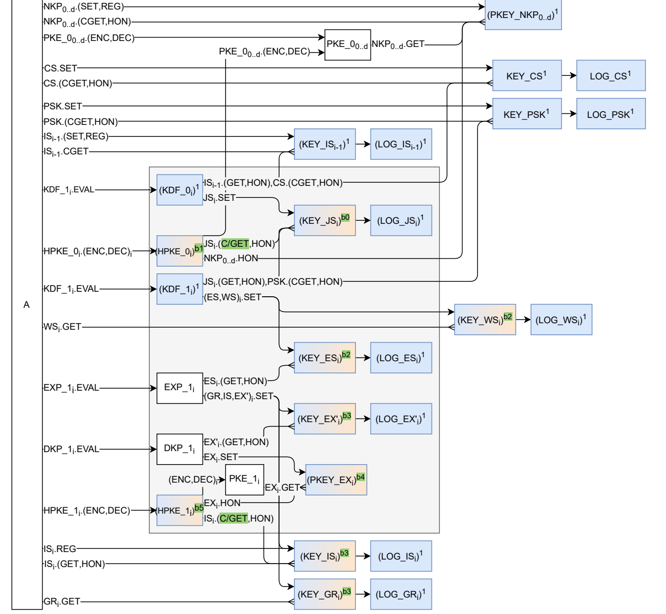


Figure 23: Definition of $\text{GEPOCH}^{b_0,b_1,b_2,b_3,b_4 b_5,d,t,i}_{\mathsf{xpd,kdf,pke}}$. We write $\text{GEPOCH}^{b,d,t,i}_{\mathsf{xpd,kdf,pke}}$ if $b = b_0 = b_1 = b_2 = b_3 = b_4 = b_5$. Technically, `KEY_IS_-1` should have a REG query, but for an ideal `KEY` package REG and SET are functionally equivalent.

the following games are equal:

$$\begin{aligned} \text{GGKS}^{1,0,d,t}_{\mathsf{xpd,kdf,pke}} &= \mathcal{R}_{\mathsf{hyb,0}} \circ \text{GEPOCH}^{0,d,t,i}_{\mathsf{xpd,kdf,pke}} \qquad (1)\\ \mathcal{R}_{\mathsf{hyb,i}} \circ \text{GEPOCH}^{1,d,t,i}_{\mathsf{xpd,kdf,pke}} &= \mathcal{R}_{\mathsf{hyb,i+1}} \circ \text{GEPOCH}^{0,d,t,i}_{\mathsf{xpd,kdf,pke}} \\ \mathcal{R}_{\mathsf{hyb,t}} \circ \text{GEPOCH}^{1,d,t,i}_{\mathsf{xpd,kdf,pke}} &= \text{GGKS}^{1,1,d,t}_{\mathsf{xpd,kdf,pke}}, \end{aligned}$$

where $\mathcal{R}_{\mathsf{hyb,i}}$ is defined in Figure 19. We obtain Theorem 2 as follows:

$$\begin{aligned} &\mathsf{Adv}^{\mathsf{xpd,kdf,pke}}_{\mathrm{KS}}(\mathcal{A}) \\ \leq\ &\mathsf{Adv}^{\mathsf{xpd,kdf,pke},(0,0),(1,0)}_{\mathrm{GKS}}(\mathcal{A}) + \mathsf{Adv}^{\mathsf{xpd,kdf,pke},(1,1),(1,1)}_{\mathrm{GKS}}(\mathcal{A}) \\ \overset{\text{Equation (1)}}{\leq}\ &\mathsf{Adv}^{\mathsf{xpd,kdf,pke},(0,0),(1,0)}_{\mathrm{GKS}}(\mathcal{A}) + \sum_{i=0}^{t-1} \mathsf{Adv}^{\mathsf{xpd,kdf,pke},i}_{\mathrm{EPOCH}}(\mathcal{A} \circ \mathcal{R}_{\mathsf{hyb},i}). \end{aligned}$$

Plugging in the bounds from Lemma 2 and Lemma 3 with $\mathcal{B} = \mathcal{A} \circ \mathcal{R}_{\mathsf{hyb},i}$ yields Theorem 2. We now turn to the proof of Lemma 2.

D. Proof of Lemma 2

In order to prove Lemma 2, we introduce game $\text{GPGKS}^{b_0,b_1,b_2,d,t}_{\mathsf{xpd,kdf,pke}}$, depicted in Figure 20, where the P in GPGKS stands for *proof*. The following equalities hold:

$$\begin{aligned} \text{GPGKS}^{0,0,0,d,t}_{\mathsf{xpd,kdf,pke}} &= \text{GGKS}^{0,0,d,t}_{\mathsf{xpd,kdf,pke}} \\ \text{GPGKS}^{1,1,1,d,t}_{\mathsf{xpd,kdf,pke}} &= \text{GGKS}^{1,0,d,t}_{\mathsf{xpd,kdf,pke}} \end{aligned}$$

For collision-resistance, we can construct a reduction $\mathcal{R}_{\mathsf{cr}}$ such that the distinguishing advantage between $\text{GPGKS}^{0,0,0,d,t}_{\mathsf{xpd,kdf,pke}}$ and $\text{GPGKS}^{1,0,0,d,t}_{\mathsf{xpd,kdf,pke}}$ can be turned into a collision against one of the underlying primitives. This is possible because (a) the initial commit secret, PSK, and level 0 init secrets are unique and (b) we have domain separation between the different epochs due to the inclusion of the group context in the KDFs. Thus, we can argue inductively that

$$\begin{aligned} &\left|\Pr\left[1 \leftarrow \mathcal{A} \circ \text{GPGKS}^{0,0,0,d,t}_{\mathsf{xpd,kdf,pke}}\right] - \Pr\left[1 \leftarrow \mathcal{A} \circ \text{GPGKS}^{1,0,0,d,t}_{\mathsf{xpd,kdf,pke}}\right]\right| \\ &\leq \mathsf{Adv}^{\mathsf{xpd,kdf,dkp}}_{\mathrm{CR}}(\mathcal{A} \circ \mathcal{R}_{cr}). \end{aligned}$$

After idealizing collision-resistance, we can now reduce to the LKDF security of the commit secret, since the respective second inputs are unique. We thus obtain that

$$\begin{aligned} &\left|\Pr\left[1 \leftarrow \mathcal{A} \circ \text{GPGKS}^{1,0,0,d,t}_{\mathsf{xpd,kdf,pke}}\right] - \Pr\left[1 \leftarrow \mathcal{A} \circ \text{GPGKS}^{1,1,0,d,t}_{\mathsf{xpd,kdf,pke}}\right]\right| \\ &\leq \mathsf{Adv}^{\mathsf{kdf}}_{\mathrm{KDFL}}(\mathcal{A} \circ \mathcal{R}_{cs}), \end{aligned}$$

where $\mathcal{R}_{cs}$ is defined in Figure 21. Analogously,

$$\begin{aligned} &\left|\Pr\left[1 \leftarrow \mathcal{A} \circ \text{GPGKS}^{1,1,0,d,t}_{\mathsf{xpd,kdf,pke}}\right] - \Pr\left[1 \leftarrow \mathcal{A} \circ \text{GPGKS}^{1,1,1,d,t}_{\mathsf{xpd,kdf,pke}}\right]\right| \\ &\leq \mathsf{Adv}^{\mathsf{kdf}}_{\mathrm{KDFL}}(\mathcal{A} \circ \mathcal{R}_{\mathsf{psk}}), \end{aligned}$$

where $\mathcal{R}_{psk}$ is defined in Figure 22. We obtain

$$\begin{aligned} &\left|\Pr\left[1 \leftarrow \mathcal{A} \circ \text{GGKS}^{0,0,d,t}_{\mathsf{xpd,kdf,pke}}\right] - \Pr\left[1 \leftarrow \mathcal{A} \circ \text{GGKS}^{1,0,d,t}_{\mathsf{xpd,kdf,pke}}\right]\right| \\ &\qquad \leq \mathsf{Adv}^{\mathsf{xpd,kdf,dkp}}_{\mathrm{CR}}(\mathcal{A} \circ \mathcal{R}_{cr}) + \mathsf{Adv}^{\mathsf{kdf}}_{\mathrm{KDFL}}(\mathcal{A} \circ \mathcal{R}_{\mathsf{cs}}) \\ &\qquad\qquad + \mathsf{Adv}^{\mathsf{kdf}}_{\mathrm{KDFL}}(\mathcal{A} \circ \mathcal{R}_{\mathsf{psk}}), \end{aligned}$$

which concludes the proof of Lemma 2.

E. Key Schedule Epoch

We now prove Lemma 3 which captures the security of a single epoch of the key schedule, a core argument in the proof of Theorem 2. We show that when an MLS epoch relies on an ideal internal secret s_{int} derived in a previous epoch or an ideal PSK s_{ext} or an ideal commit secret k_{cs}, then the current epoch provides security guarantees, too, for the new s'_{int} derived in the current epoch, all group secrets for the current epoch and for the current external public-key encryption keys.

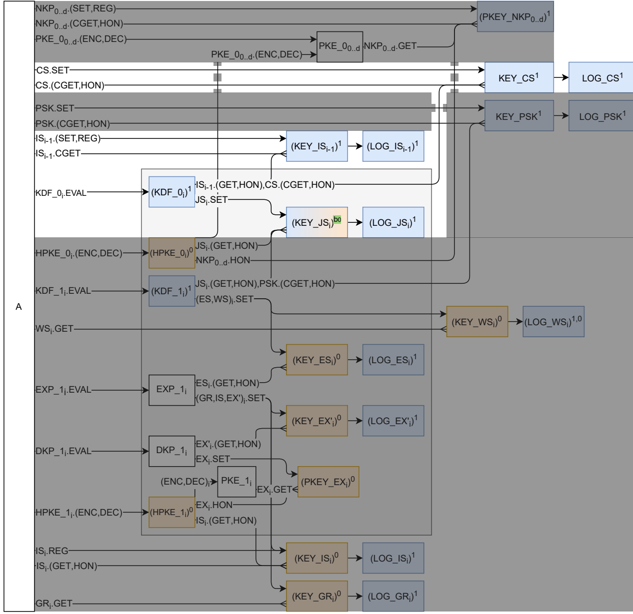


Figure 24: Step 1 of idealizing epoch i. We mark $\mathcal{R}^{\mathsf{ks}}_{1,i}$ in grey.

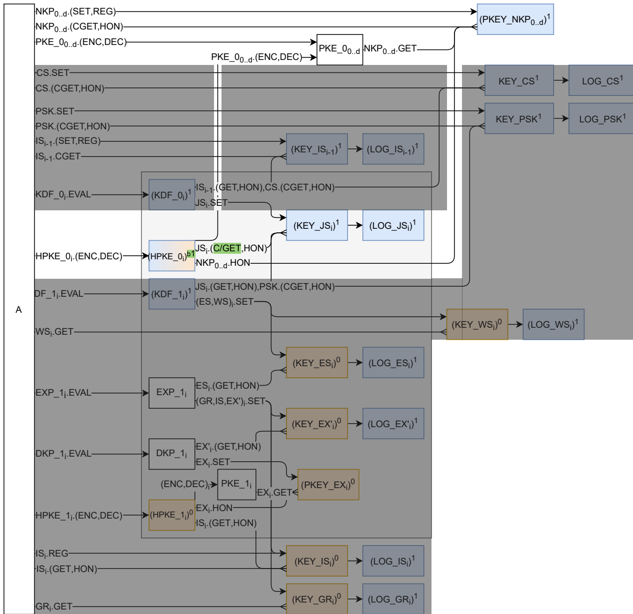


Figure 25: Step 2 of idealizing epoch i. We mark $\mathcal{R}^{\mathsf{ks}}_{2,i}$ in grey.

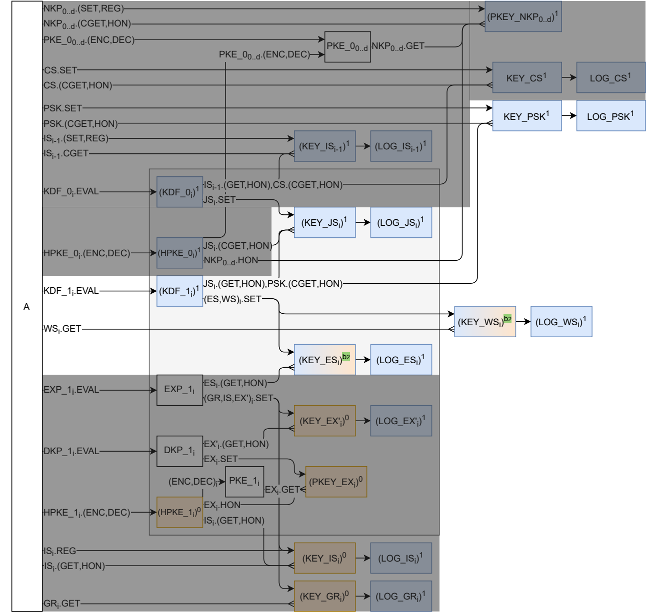


Figure 26: Step 3 of idealizing epoch i. We mark $\mathcal{R}^{\mathsf{ks}}_{3,i}$ in grey.

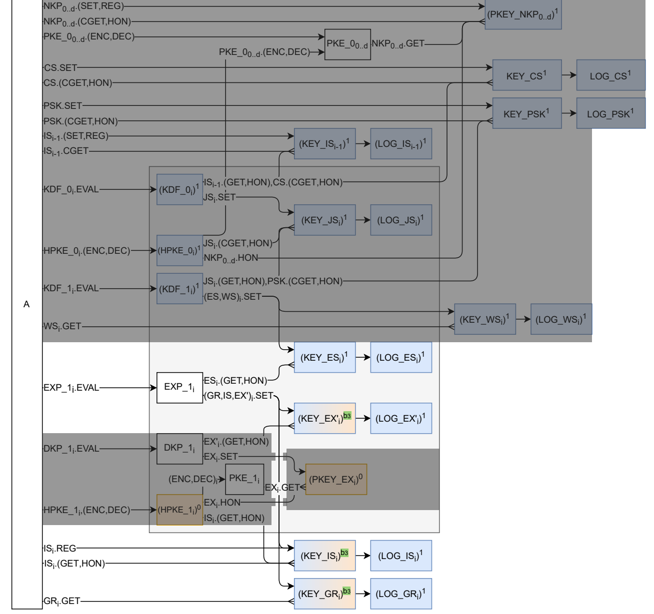


Figure 27: Step 4 of idealizing epoch i. We mark $\mathcal{R}^{\mathsf{ks}}_{4,i}$ in grey.

The proof of Lemma 3 consists of six steps, each of which flips one or more bits in the overall game. See Figure 23 for the definition of $\text{GEPOCH}^{b_0,b_1,b_2,b_3,b_4 b_5,d,t,i}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}$ and note that $\text{GEPOCH}^{b,d,t,i}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}} = \text{GEPOCH}^{b_0,b_1,b_2,b_3,b_4,b_5,d,t,i}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}$ if $b = b_0 = b_1 = b_2 = b_3 = b_4 = b_5$.

In the first step of this proof we use the KDFR assumption to idealize the `KEY_JS` package of epoch i. We obtain

$$\Big| \Pr\left[1 \leftarrow \mathcal{A} \circ \text{GEPOCH}^{0,0,0,0,0,0,d,t,i}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}\right] - \Pr\left[1 \leftarrow \mathcal{A} \circ \text{GEPOCH}^{1,0,0,0,0,0,d,t,i}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}\right] \Big| \leq \mathsf{Adv}^{\mathsf{kdf}}_{\mathsf{KDFR}}(\mathcal{A} \circ \mathcal{R}^{\mathsf{ks}}_{1,i}).$$

Now, in the second game hop, we use the HPKE assumption for $n = 1$ and $m = d$ to idealize the first `HPKE` package of epoch i and to remove access to the joiner secret. We obtain

$$\Big| \Pr\left[1 \leftarrow \mathcal{A} \circ \text{GEPOCH}^{1,0,0,0,0,0,d,t,i}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}\right] - \Pr\left[1 \leftarrow \mathcal{A} \circ \text{GEPOCH}^{1,1,0,0,0,0,d,t,i}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}\right] \Big| \leq \mathsf{Adv}^{\mathsf{pke}}_{\mathsf{HPKE}}(\mathcal{A} \circ \mathcal{R}^{\mathsf{ks}}_{2,i}).$$

In the third step of this proof we use the KDFR assumption to idealize the `KEY_ES` package of epoch i. This is possible since there is no GET query on the joiner secret anymore. We obtain the bound

$$\Big| \Pr\left[1 \leftarrow \mathcal{A} \circ \text{GEPOCH}^{1,1,0,0,0,0,d,t,i}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}\right] - \Pr\left[1 \leftarrow \mathcal{A} \circ \text{GEPOCH}^{1,1,1,0,0,0,d,t,i}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}\right] \Big| \leq \mathsf{Adv}^{\mathsf{kdf}}_{\mathsf{KDFR}}(\mathcal{A} \circ \mathcal{R}^{\mathsf{ks}}_{3,i}).$$

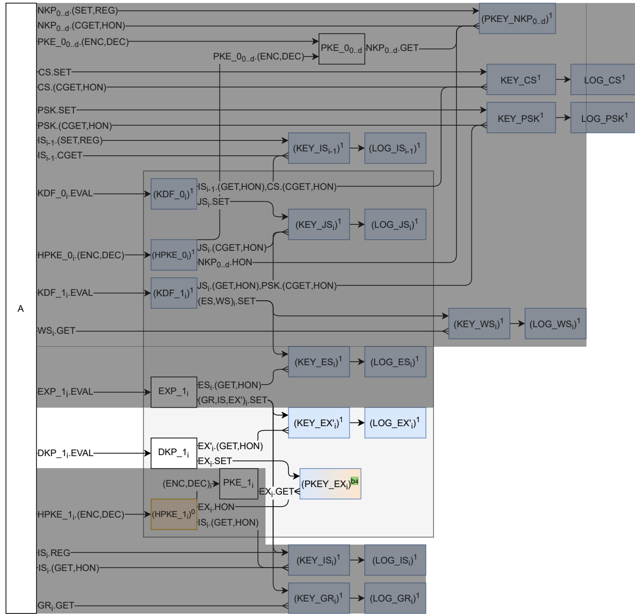


Figure 28: Step 5 of idealizing epoch i. We mark $\mathcal{R}^{\text{ks}}_{5,i}$ in grey.

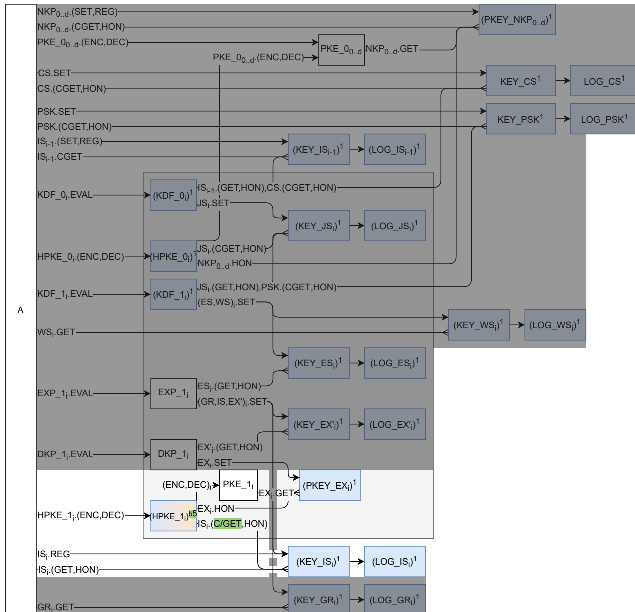


Figure 29: Step 6 of idealizing epoch i. We mark $\mathcal{R}^{\text{ks}}_{6,i}$ in grey.

In the fourth step of this proof we use the XPD assumption with $n = 2 + |GR|$ to idealize all current group secrets in `KEY_GR`, the init secret in `KEY_IN` and the seed in `KEY_EX` (later used for computing the external public-key). We obtain

$$\Big| \Pr\Big[1 \leftarrow \mathcal{A} \circ \text{GEPOCH}^{1,1,1,0,0,0,d,t,i}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}\Big] - \Pr\Big[1 \leftarrow \mathcal{A} \circ \text{GEPOCH}^{1,1,1,1,0,0,d,t,i}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}\Big] \Big| \leq \mathsf{Adv}^{\mathsf{xpd}}_{\text{EXP}}(\mathcal{A} \circ \mathcal{R}^{\text{ks}}_{4,i}).$$

In the fifth step of this proof, we lose the DKP advantage to idealize the `PKEY_EX` package of epoch i, and we obtain

$$\Big| \Pr\Big[1 \leftarrow \mathcal{A} \circ \text{GEPOCH}^{1,1,1,1,0,0,d,t,i}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}\Big] - \Pr\Big[1 \leftarrow \mathcal{A} \circ \text{GEPOCH}^{1,1,1,1,1,0,d,t,i}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}\Big] \Big| \leq \mathsf{Adv}^{\mathsf{xpd}}_{\text{DKP}}(\mathcal{A} \circ \mathcal{R}^{\text{ks}}_{5,i}).$$

In the sixth and final step of the epoch proof we use the HPKE assumption for $n = 1$ and $m = 1$ to idealize the second `HPKE` package of epoch i. We obtain

$$\Big| \Pr\Big[1 \leftarrow \mathcal{A} \circ \text{GEPOCH}^{1,1,1,1,1,0,d,t,i}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}\Big] - \Pr\Big[1 \leftarrow \mathcal{A} \circ \text{GEPOCH}^{1,1,1,1,1,1,d,t,i}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}\Big] \Big| \leq \mathsf{Adv}^{\mathsf{pke}}_{\text{HPKE}}(\mathcal{A} \circ \mathcal{R}^{\text{ks}}_{6,i}).$$

Hence,

$$\begin{aligned}\mathsf{Adv}^{\mathsf{xpd},\mathsf{kdf},\mathsf{pke},i}_{\text{EPOCH}}(\mathcal{A}) \leq\ & \mathsf{Adv}^{\mathsf{kdf}}_{\text{KDFR}}(\mathcal{A} \circ \mathcal{R}^{\text{ks}}_{1,i}) + \mathsf{Adv}^{\mathsf{pke}}_{\text{HPKE}}(\mathcal{A} \circ \mathcal{R}^{\text{ks}}_{2,i}) \\ &+ \mathsf{Adv}^{\mathsf{kdf}}_{\text{KDFR}}(\mathcal{A} \circ \mathcal{R}^{\text{ks}}_{3,i}) + \mathsf{Adv}^{\mathsf{xpd}}_{\text{EXP}}(\mathcal{A} \circ \mathcal{R}^{\text{ks}}_{4,i}) \\ &+ \mathsf{Adv}^{\mathsf{pke}}_{\text{DKP}}(\mathcal{A} \circ \mathcal{R}^{\text{ks}}_{5,i}) + \mathsf{Adv}^{\mathsf{pke}}_{\text{HPKE}}(\mathcal{A} \circ \mathcal{R}^{\text{ks}}_{6,i}),\end{aligned}$$

and Lemma 3 holds.