Cryptographic Security of the MLS RFC, Draft 11

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 provide a computational analysis of the MLS **key schedule, TreeKEM and their composition, as specified in Draft 11 of the** MLS **RFC. The analysis is carried out using the State Separating Proofs methodology [9], and showcases the flexibility of the approach, enabling us to provide a full computational analysis shortly after Draft 11 was published.**

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* [4], [23] ensures that the key is shared only with the intended recipient. *Key Indistinguishability* [16], [23], 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 [23].

With the introduction of *continuous* key-exchange, for example by protocols such as OTR [7] or ZRTP [28], 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. [11], [18].

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* [22] (formerly *TextSecure*) achieves strong FS and PCS guarantees [13], [11] due to the use of double ratcheting (formerly *Axolotl*).

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].

The first modern, continuous group key exchange protocol is based on an *Asynchronous Ratcheting Tree* (ART) and was proposed by Cohn-Gordon et al. [12]. Similar to an earlier protocol proposed by Steer et al. [26] (see Figure 1), it is based on a binary tree where each leaf represents a group member. The secrets of the members are recursively combined in lower nodes of the tree such that the secret in the root node is shared by all group members. In contrast to the protocol proposed by Steer et al., ART supports FS and PCS guarantees.

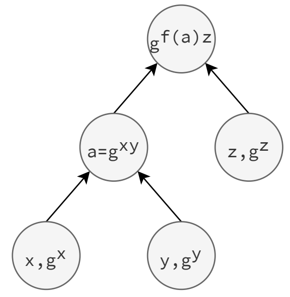


Figure 1: Key derivation in group key exchange protocol [26]. f encodes a as a valid exponent.

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 [3] that is efficient even for large groups [25]. To achieve this, the current draft makes use of the TreeKEM [6] protocol.

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 IV, 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. Earlier versions of the MLS RFC have been analyzed in several works [1], [2], [15], [27]. Alwen et al. [1] propose an alternative to TreeKEM, named *Tainted* TreeKEM, which replaces blank nodes (node without keys, see Section IV) by so-called *tainted* nodes. In contrast to a blank node, the keys of a tainted node are still used. A tainted node will be blank after the entity that tainted the node updates their keying material. In doing so, depending on the distribution of add and remove operations, updates are more efficient compared to TreeKEM.

Weidner [27] proposes an alternative version of TreeKEM, named *Causal* TreeKEM, that does not require a strict total order of group operations. Where the MLS RFC assumes a trusted server that can enforce ordering of messages because key updates overwrite existing keys, Causal TreeKEM does not have this requirement because it uses keying material that can be combined in an arbitrary order. However, this requires keying material that can be combined in an associative and commutative manner, such as Diffie-Hellman shares. MLS instead opts for higher modularity with a KEM approach, where the underlying encryption scheme can be changed if needed. Specifically, TreeKEM can achieve post-quantum security by choosing an appropriate KEM scheme, which is not possible in causal TreeKEM.

In [15], a comparison between MLS and a straightforward group messaging protocol made of 1:1 Signal sessions, reveals that in the 1:1 setting, authentication can be healed, while, in the group setting, MLS never heals authentication thus weakening PCS guarantees in the case of signature key corruptions.

Relatedly, Alwen et al. [2] show that the Forward Secrecy (FS) guarantees of MLS are relatively weak. An update to the keying material of one client will leave the group vulnerable w.r.t. FS from other corrupted clients. Therefore, in the worst case, FS is only achieved if every group member updates their keying material. To mitigate this shortcoming, Alwen et al. propose to use a PKE scheme in TreeKEM where keys are updated with every encryption and decryption operation. The proposal was not integrated into MLS due to concerns that supporting homomorphism is not compatible with considering the KEM as a black-box primitive.

Contributions. In this work, we study the security claims made in Draft 11 of the MLS RFC via a cryptographic analysis of TreeKEM, the MLS key schedule and their composition. Key schedule analysis was implicitly suggested in Krawczyk's study of Sigma protocols [19] and used for the analysis of TLS 1.3 [8]. 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 [9] which specifies security models via modular pseudocode and enables code reuse (see Section V for details).

Our model allows the adversary to corrupt all base secrets statically, and to decide to corrupt derived keys upon derivation (CUD). This corruption model simplifies the security analysis (in comparison to fully adaptive corruption) and models the intended security of MLS in a meaningful way. Firstly, adaptive corruption of long-term secrets is most relevant for signing keys, which are outside the scope of this work. Secondly, as session states are intended to be rather short-lived, dynamically corrupting old sessions should be difficult in practice. Thirdly, in light of the impossibility result of non-committing encryption [24], our model seems close to the strongest possible security which can be provably achieved without random oracles.

We establish that the keys produced by TreeKEM and the key schedule of MLS are pseudorandom in the CUD corruption model. 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 [20] is a dual pseudorandom function and thus, HKDF is a dual KDF, which has also been assumed in the analysis of Noise [17] and TLS 1.3 [8]. Additionally, we assume that all primitives are collision-resistant. Following the approach of [10], [8], we model the security of our symmetric primitives as multi-instance primitives with static corruptions. Jumping ahead, based on our analysis, we suggest to include the group context into the derivation of the joiner secret [14] in order to strengthen its uniqueness properties.

We rely on modular proofs and pseudocode-reuse as specified by the state-separating proofs (SSP) methodology [9]. 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 provides syntax for continuous key distribution and key schedule. Section IV explains the TreeKEM, the MLS key schedule and their composition. Section V introduces the state-separating proofs methodology and our assumptions. Section VI explains our security model, Section VII states our three main theorems, and Section VIII and the appendix contain the proofs.

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 $\mathsf{algorithms}$ 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 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. Continuous Group Key Distribution

A *continuous group key distribution* protocol (CGKD) allows a group of users 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 new clients 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. This way, we can model the important security property of post-compromise security (PCS), which demands that if key material of a user gets corrupted (and the adversary learns the session key), the security of the protocol can recover. This section introduces the syntax of a CGKD, the syntax of 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 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. We now first introduce the syntax of a CGKD, a wCGKD and a KS and then elaborate how this syntax reflects MLS.

Updates of the key material are initiated by one group member and are modeled in the (w)update algorithm 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

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.

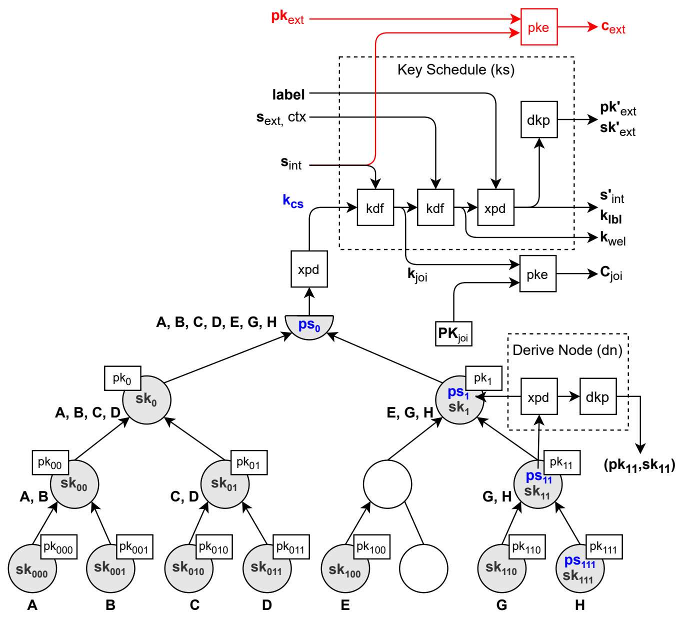


Figure 3: The tree represents a group with members A, B, C, D, E, G and H. Inside a node, we display TreeKEM secrets known to members in the corresponding sub-tree. The blue values are used in the derivation and not stored long-term.

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.

Definition III.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 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.

Definition III.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 III.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 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}.*

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

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

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

Figure 4: The MLS Key Schedule algorithm (ks) and the MLS Derive Node algorithm (dn).

IV. Messaging Layer Security (MLS)

In this section, we describe MLS and explain how CGKD, wCGKD and KS can be used to model MLS.

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 [20]. ks chains two full KDFs and then a single XPD function, see top of Figure 3. In OPTLS [21], 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 4 for the code of the MLS Key Schedule algorithm ks. It is split into a KDF call and a call to the subks procedure since, jumping ahead, the join algorithm of the MLS CGKD only uses subks rather than the full ks. The splitting is useful, since join only knows k_{joi}, but not k_{cs}. 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

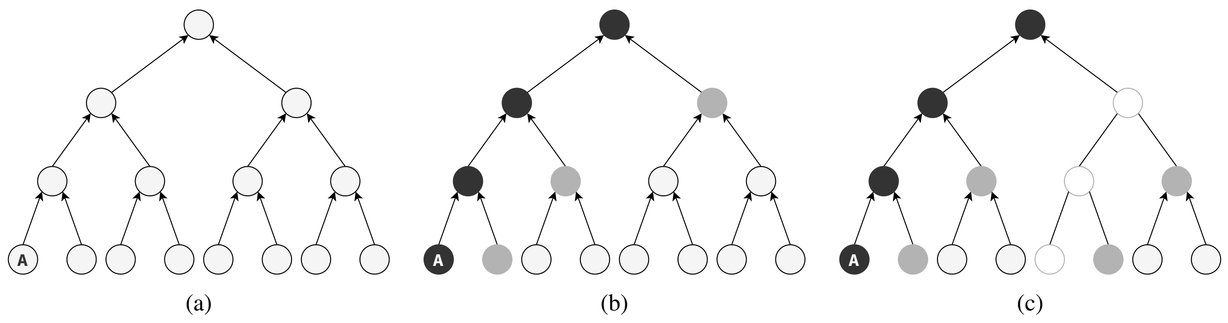


Figure 5: Visualisation of a member A updating their their key material. **(a)** shows the tree for the group. **(b)** shows the tree during the update, highlighting the *direct path* in black and the *co-path* in dark grey. **(c)** shows the tree during the update if there are *blank nodes* in the tree.

the group and indeed derived from the value ps which is stored at the root of the tree. Figure 3 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 3 and Figure 4. 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 3, 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 —see Figure 9 for the pseudocode of update— we describe the conceptual idea for each separately below.

Updating Key Material. Consider the group represented by the tree in Figure 5a. 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 in Figure 9—it is only executed if k_{upd} is non-empty. In the end, Alice has replaced all the path secrets, public-keys and secret-keys of the direct path, marked in black in Figure 5b.

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 5b, the co-path is marked in grey, i.e., for each grey node, Alice encrypts to the public-key associated with it.

Figure 5a and Figure 5b 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 5c depicts the case that the tree contains some blank nodes. In this case, Alice encrypts to the nodes marked in grey in Figure 5c. 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 list of sets, i.e., for each tree index i, $CP[i]$ contains a set of the indices to which Alice needs to encrypt.

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 6a. Here, to add Henry (H) to this group, the group

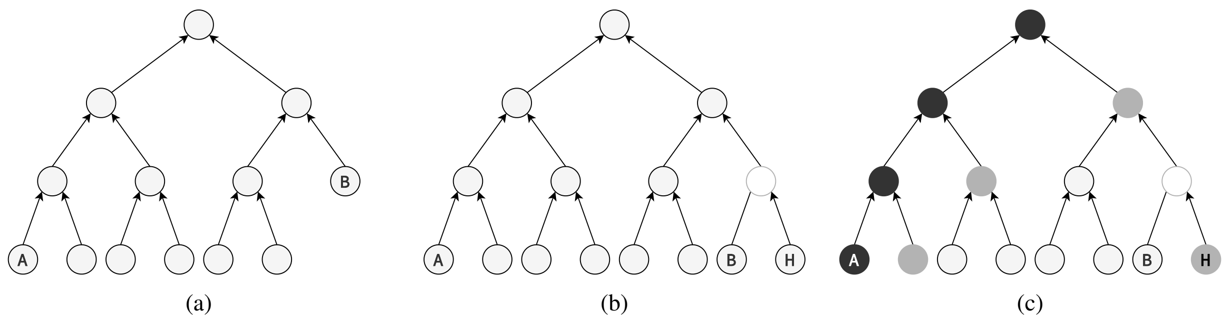


Figure 6: Visualisation of adding a new member to a group. **(a)** shows the tree for the original group. **(b)** shows the tree with *H* added to the group in an *add-only* operation, highlighting blanked nodes in white. **(c)** shows the tree for the update performed by *A*, highlighting the *direct path* in black and the *co-path* in dark grey. Note that H's unmerged leaf is part of the co-path.

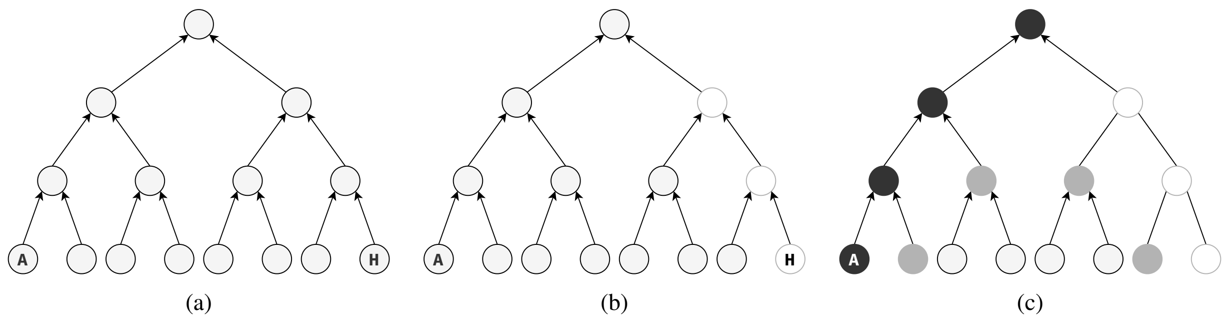


Figure 7: Visualisation of removing a member from a group. **(a)** shows the tree for the original group. **(b)** shows the tree with *H* removed. **(c)** shows the tree for the update after the removal, highlighting the *direct path* in black and the *co-path* in dark grey. Blanked nodes are highlighted in white in both **(b)** and **(c)**.

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 IV 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 3 and Figure 4) to *H*. Recall that the dopath procedure in Figure 9 is only executed if k_{upd} is a non-empty variable. If k_{upd} is empty and an add-only is performed, the computations are performed 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 red in Figure 9) to transmit s_{int}. Note that the red code 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 7a. 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 7b. 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 9. 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 7c.

MLS CGKD. Combining TreeKEM and ks as depicted in Figure 3 yields the MLS CGKD. Note that the MLS CGKD achieves significantly stronger security properties than the wCGKD. Namely, freshness in the wCGKD is violated if *any* of the group keys are compromised (and not yet updated), see left side of Figure 8. In turn, the MLS CGKD also yields security if the s_{ext} or the s_{int} from a previous epoch are fresh. See Figure 9 for the pseudocode of the update procedure and Figure 8 for its freshness.

```
$\mathsf{own\_keys}(PK_{mem}, PK_{rem}, PK_{joi}, SK_{own}, k_{\mathrm{upd}}, pk_{ext}, s_{ext}, s_{int})$
if $!\mathsf{fresh}(k_{\mathrm{upd}})$:
    return $all\ corrupt$
$PK'_{mem} \leftarrow PK_{mem}$
$PK'_{mem} \overset{\mathrm{rem}}{\leftarrow} PK_{rem}$
$PK'_{mem} \overset{\mathrm{merge}}{\leftarrow} PK_{joi}$
$F \leftarrow [\,]$
$fresh \leftarrow 1$
for $i_{cur}, \_ \in SK_{own}$:
    for $i_{nod} \in CP[i_{cur}]$:
        if $!\mathsf{fresh}(PK'_{mem}[i_{nod}])$:
            $fresh \leftarrow 0$
    $F[i_{cur}] \leftarrow fresh$
return $F$
```

```
$\mathsf{grp\_keys}(PK_{mem}, PK_{rem}, PK_{joi}, SK_{own}, k_{\mathrm{upd}}, pk_{ext}, s_{ext}, s_{int})$
if $\mathsf{fresh}(s_{int}) \wedge \mathsf{fresh}(pk_{ext})$:
    return $1$
if $\mathsf{fresh}(s_{ext})$:
    return $1$
if $0 \notin \mathsf{own\_keys}(\ldots)$:
    return $1$
return $0$
```

Figure 8: Freshness conditions of the MLS Key Distribution.

```
$\mathsf{update}(PK_{mem}, PK_{rem}, PK_{joi}, SK_{own}, i_{own}, k_{upd}, CP, pk_{ext}, s_{ext}, s_{int}, ctx)$
if $|PK_{rem}| > 0 \vee pk_{ext} \neq \perp$:
    assert $k_{upd} \neq \perp$
$(C_{cp}, PK'_{upd}, PK'_{mem}, SK'_{own}, k_{cs})$
    $\leftarrow \mathsf{wupdate}(PK_{mem}, PK_{rem}, PK_{joi}, SK_{own}, i_{own}, k_{upd}, CP)$
$c_{ext} \leftarrow\!\!\$\ \mathsf{pke.enc}(pk_{ext}, s_{int})$
$(s'_{int}, \overline{k}_{\overline{lbl}}, k_{joi}, pk'_{ext}, sk'_{ext})$
    $\leftarrow \mathsf{ks}(k_{cs}, s_{int}, s_{ext}, \overline{lbl}, ctx)$
$SK'_{own}[-1] \leftarrow sk'_{ext}$
$C_{joi} \leftarrow [\,]$
for $i_{nod}, pk \in PK_{joi}$:
    $C_{joi}[i_{nod}] \leftarrow\!\!\$\ \mathsf{pke.enc}(pk, k_{joi})$
return $(C_{cp}, C_{joi}, c_{ext}, PK'_{upd}, PK'_{mem}, SK'_{own}, \overline{k}_{\overline{lbl}}, s'_{int}, pk'_{ext})$
```

```
$\mathsf{dopath}(PK_{mem}, i_{cur}, k_{upd}, CP)$
$C_{cp} \leftarrow [\,]$
$PK'_{own}, SK'_{own} \leftarrow [\,], [\,]$
$ps \leftarrow k_{upd}$
while $i_{cur} \neq \perp$:
    $(ps', pk, sk) \leftarrow \mathsf{dn}(ps)$
    $PK'_{own}[i_{cur}] \leftarrow pk$
    $SK'_{own}[i_{cur}] \leftarrow sk$
    if $i_{cur} \neq i_{own}$:
        for $i_{nod} \in CP[i_{cur}]$:
            $pk_{nod} \leftarrow PK_{mem}[i_{nod}]$
            $c \leftarrow\!\!\$\ \mathsf{pke.enc}(pk_{nod}, ps')$
            $C_{cp}[i_{nod}] \leftarrow c$
    $i_{cur} \leftarrow \mathsf{parent}(i_{cur}, |PK_{mem}|)$
    $ps \leftarrow ps'$
return $(C_{cp}, PK'_{own}, SK'_{own}, ps)$
```

```
$\mathsf{process}(PK_{mem}, PK_{rem}, PK_{joi}, PK_{upd}, SK_{own}, i_{nod}, i_{upd}, s_{ext}, s_{int}, ctx, c, c_{ext})$
$(PK'_{mem}, SK'_{own}, k_{cs}) \leftarrow$
    $\mathsf{wprocess}(PK_{mem}, PK_{rem}, PK_{joi}, PK_{upd}, SK_{own}, i_{nod}, i_{upd}, c)$
$s_{int} \leftarrow \mathsf{pke.dec}(SK_{own}[-1], c_{ext})$
assert $s_{int} \neq \perp$
$(s'_{int}, \overline{k}_{\overline{lbl}}, \_, pk'_{ext}, sk'_{ext})$
    $\leftarrow \mathsf{ks}(k_{cs}, s_{int}, s_{ext}, \overline{lbl}, ctx)$
$SK'_{own}[-1] \leftarrow sk'_{ext}$
return $(PK'_{mem}, SK'_{own}, \overline{k}_{\overline{lbl}}, s'_{int}, pk'_{ext})$
```

```
$\mathsf{join}(PK_{mem}, SK_{own}, i_{own}, i_{upd}, s_{ext}, ctx, c)$
$SK'_{own} \leftarrow SK_{own}$
$sk_{own} \leftarrow SK'_{own}[i_{own}]$
$k_{joi} \leftarrow \mathsf{pke.dec}(sk_{own}, c)$
assert $k_{joi} \neq \perp$
$(s'_{int}, \overline{k}_{\overline{lbl}}, \_, pk'_{ext}, sk'_{ext})$
    $\leftarrow \mathsf{subks}(k_{joi}, s_{ext}, \overline{lbl}, ctx)$
$SK'_{own}[-1] \leftarrow sk'_{ext}$
return $(PK_{mem}, SK'_{own}, \overline{k}_{\overline{lbl}}, s'_{int}, pk'_{ext})$
```

```
$\mathsf{wupdate}(PK_{mem}, PK_{rem}, PK_{joi}, SK_{own}, i_{own}, k_{upd}, CP)$
$k_{cs} \leftarrow 0$
$PK'_{mem} \leftarrow PK_{mem}$
$PK'_{mem} \overset{\mathrm{rem}}{\leftarrow} PK_{rem}$
$PK'_{mem} \overset{\mathrm{merge}}{\leftarrow} PK_{joi}$
$C_{cp}, PK'_{upd} \leftarrow [\,], [\,]$
$SK'_{own} \leftarrow SK_{own}$
for $i \in \mathsf{directpath}(i_{own}, |PK_{mem}|)$:
    $PK_{upd}[i] \leftarrow PK'_{mem}[i]$
$(C_{cp}, PK'_{upd}, SK'_{own}, k_{cs})$
    $\leftarrow \mathsf{dopath}(PK'_{mem}, i_{own}, k_{upd}, CP)$
$PK'_{mem} \overset{\mathrm{merge}}{\leftarrow} PK'_{upd}$
return $(C_{cp}, PK'_{upd}, PK'_{mem}, SK'_{own}, k_{cs})$
```

Figure 9: Procedures update, process, join, and wupdate.

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_{add} and the view of the entire new tree PK_{mem} suffice to allow the join procedure to derive the same values. See Figure 9 for their code.

V. 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 [5], it has been popular to formulate games in pseudocode, and we follow this approach here. I.e., a game 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 $\mathcal{A} \circ \mathsf{G}$ that the adversary $\mathcal{A}$ interacts with the oracles of game G. I.e., the adversary is the *main procedure* that makes queries to the oracles of 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* $\mathsf{Adv}_{\mathsf{G}^0,\mathsf{G}^1}(\mathcal{A})$ measures an adversary $\mathcal{A}$'s ability to distinguish between G^0 (real) and G^1 (ideal).

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

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

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 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 10. Cryptographic 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 3). Now, if KDF computes keys and stores them in a KEY package using the SET oracle, then XPD can retrieve

```
KEY^{b,λ}
Package Parameters
  $b$:  idealization bit
  $\lambda$:  key length

Package State
  $K[h \mapsto k]$:  key table
  $H[h \mapsto b]$:  honesty table

SET($h, hon, s$)
  assert $|s| = \lambda$
  if $K[h] \neq \perp$:
    return $h$
  if $b \wedge hon$:
    $s \leftarrow\$ \{0,1\}^{\lambda}$
  UNQ($h, hon, s$)
  $K[h] \leftarrow s$
  $H[h] \leftarrow hon$
  return $h$

REG($hon, s$)
  assert $|s| = \lambda$
  $i \leftarrow$ index(KEY)
  $h \leftarrow \mathbf{reg}\langle |K|, i\rangle$
  if $K[h] \neq \perp$:
    return $h$
  if $hon$:
    $s \leftarrow\$ \{0,1\}^{\lambda}$
  UNQ($h, hon, s$)
  $K[h] \leftarrow s$
  $H[h] \leftarrow hon$
  return $h$

GET(h)
  assert $K[h] \neq \perp$
  return $K[h]$

CGET(h)
  assert $H[h] = 0$
  return $K[h]$

HON(h)
  assert $H[h] \neq \perp$
  return $H[h]$
```

```
LOG^{b}
Package Parameters
  $b$:  idealization bit

Package State
  $L$:  log table

UNQ($h, hon, s$)
  if $L[h] \neq \perp$ :
    $(h', hon', s') \leftarrow L[h]$
    assert $s' = s$
    if $hon' = hon$:
      return $h'$
  for $(h', hon', s') \in L$
          with $s = s'$:
    $r \leftarrow (level(h) = set)$
    $r' \leftarrow (level(h') = set)$
    if $r \neq r' \wedge M[s] = \perp$
        $\wedge\ hon = hon' = 0$:
      $M[s] \leftarrow 1$
      $L[h] \leftarrow (h', hon, s)$
      return $h'$
    if $hon = hon' = 0$
        $\wedge\ r = r'$: abort
    if $b \wedge r = r' = 1$:
      abort
  $Log[h] \leftarrow (h, hon, s)$
  return $h$
```

Figure 10: Definition of the KEY and LOG package, where $\mathbf{reg}\langle\cdot\rangle$ is an injective handle constructor, creating a tuple from the inputs.

them via the GET oracle.

We now explain how to use KEY packages to model security. Figure 12 and Figure 13 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 12d) 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 $b = 0$. Concretely, we model multi-instance security with static corruptions on the input keys and corrupt-

```
PKEY^{b,pke}
Package Parameters
  $b$:  idealization bit
  pke: pub. key enc. sch.

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 $Q[h] \neq \perp$:
    return $Q[h]$
  if $b \wedge hon$:
    $(pk, sk) \leftarrow\$$ pke.kgen()
  $h' \leftarrow (h, pk)$
  UNQ($h', hon, sk$)
  $K[h'] \leftarrow sk$
  $H[h'] \leftarrow hon$
  $Q[h] \leftarrow h'$
  return $h'$

REG($hon, pk, sk$)
  assert pke.valid($pk, sk$)
  if $hon$:
    $(pk, sk) \leftarrow\$$ pke.kgen()
  $h' \leftarrow \mathbf{reg}\langle(|K|, pk)\rangle$
  UNQ($h', hon, sk$)
  $K[h'] \leftarrow sk$
  $H[h'] \leftarrow hon$
  $Q[h] \leftarrow h'$
  return $h'$

GET($h$)
  assert $K[h] \neq \perp$
  return $K[h]$

CGET($h$)
  assert $H[h] = 0$
  return $K[h]$

HON($h$)
  assert $H[h] \neq \perp$
  return $H[h]$
```

Figure 11: Definition of the PKEY package. $\mathbf{reg}\langle\cdot\rangle$ is an injective handle constructor, creating a tuple from the inputs.

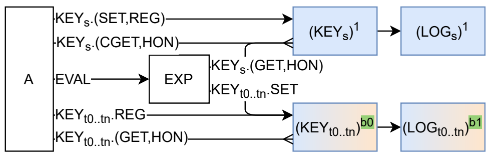
upon-derivation (CUD) on the output keys. If all input keys are corrupt, then the output value is marked as corrupt as well. Moreover, if the adversary uses $hon = 0$ as a parameter in its oracle query to EVAL, then the key will be marked as corrupt, too. Else, keys are marked as honest. Note, that in 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 $\mathbf{reg}\langle\cdot\rangle$ is composed of the index of the KEY (denoted by index(KEY)) and a counter, implemented as $|K|$ where K is the table of keys already stored in the KEY package. We denote by **reg** the corresponding an injective handle constructor, see Figure 10.

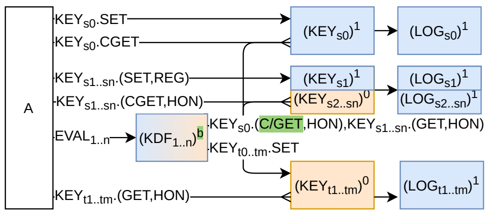
If a key is derived from one or more other keys, the handle of the output key is constructed from the handles of the input key(s), as well as any other inputs to the key derivation. See, e.g., the description of the EVAL oracles in Figure 12d. This guarantees that the handle-to-key mapping is *compuitationally 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 13). Note that pairs of secret keys and public keys are stored in the PKEY package, see Figure 11 and Figure 13.

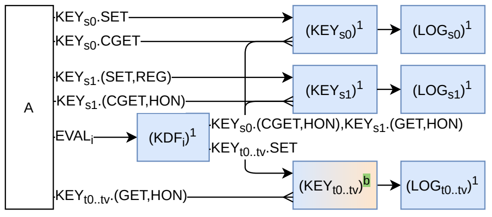
Logging. We now turn to how we model collision resistance and key uniqueness. We call two handles h and h' such that they correspond to the same key in the K table in KEY a *key collision*. For honest, uniformly random keys, such collisions



(a) Game $\text{GEXP}_f^{b_0 b_1}$.



(b) Game GKDFL_g^{b}.



(c) Game GKDFR_g^{b}.

```
EXP

Package Parameters
$f$ : a PRF
$\overline{lbl}$ : labels
$s$ : input name
$\overline{t}$ : output names

Package State
no state

EVAL($h_s, \overline{ctx}, \overline{hon}$)
$s \leftarrow \texttt{KEY}_s.\text{GET}(h_s)$
$\overline{hon'} \overset{\text{vec}}{\leftarrow} \texttt{KEY}_s.\text{HON}(h_s) \wedge \overline{hon}$
$\overline{t} \overset{\text{vec}}{\leftarrow} f(s, (\overline{lbl}, \overline{ctx}))$
$\overline{h_t} \overset{\text{vec}}{\leftarrow} (h_s, (\overline{lbl}, \overline{ctx}), \overline{t})$
$\overline{h_t'} \overset{\text{vec}}{\leftarrow} \texttt{KEY}_{\overline{t}}.\text{SET}(\overline{h_t}, \overline{hon'}, \overline{t})$
return $\overline{h_t}$
```

```
KDF$_i^b$

Package Parameters
$g$ : a KDF              $s0$ : left name
$\overline{lbl}$ : labels            $s1$ : right name
$b$ : idealization bit   $\overline{t}$ : output names
$i$ : epoch index

Package State
no state

EVAL($h_{s0}, h_{s1}, \overline{ctx}, \overline{hon}$)
$hon_0 \leftarrow \texttt{KEY}_{s0}.\mathsf{HON}(h_{s0})$
$hon_1 \leftarrow \texttt{KEY}_{s1}.\mathsf{HON}(h_{s1})$
if $b \wedge hon_0$:
    $\overline{t} \overset{\text{vec}\,\$}{\leftarrow} \{0,1\}^{\lambda'}$
else $s_0 \leftarrow \texttt{KEY}_{s0}.\mathsf{GET}(h_{s0})$:
    $s_1 \leftarrow \texttt{KEY}_{s1}.\mathsf{GET}(h_{s1})$
    $\overline{ctx}' \overset{\text{vec}}{\leftarrow} (\overline{ctx}, i)$
    $\overline{t} \overset{\text{vec}}{\leftarrow} g(s_0, s_1, (\overline{lbl}, \overline{ctx}'))$
$\overline{hon'} \overset{\text{vec}}{\leftarrow} (hon_0 \vee hon_1) \wedge \overline{hon}$
$\overline{h_t} \overset{\text{vec}}{\leftarrow} (h_{s0}, h_{s1}, (\overline{lbl}, \overline{ctx}'), \overline{t})$
$\overline{h_t'} \overset{\text{vec}}{\leftarrow} \texttt{KEY}_{\overline{t}}.\text{SET}(\overline{h_t}, \overline{hon'}, \overline{t})$
return $\overline{h_t}$
```

(d) Code of packages EXP and KDF$_i$.

Figure 12: Games $\text{GEXP}_f^{b_0 b_1}$, GKDFL_g^{b} and GKDFR_g^{b}.

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 [9]) 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.

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 $(\mathit{lbl}, \mathit{ctx})$, where in some cases, ctx might be empty. We parametrize our security game for XPD with a list of labels $\overline{\mathit{lbl}}$ and measure the security of the XPD function by the advantage which adversaries $\mathcal{A}$ have in distinguishing the real game $\text{GEXP}_f^{0,0}$ and the ideal game $\text{GEXP}_f^{1,1}$, defined in Figure 12a, for $f = \mathsf{xpd}$. For an adversary $\mathcal{A}$, we define the advantage function $\mathsf{Adv}_{\text{EXP}}^{f}(\mathcal{A}) :=$

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

In asymptotic terminology, if the advantage $\mathsf{Adv}_{\text{EXP}}^{\mathsf{xpd}}(\mathcal{A})$ is negligible for all PPT adversaries, then xpd is a secure pseudorandom function. $\mathsf{Adv}_{\text{EXP}}^{f}(\mathcal{A})$ 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 $\mathsf{Adv}_{\text{PREXP}}^{f}(\mathcal{A}) :=$

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

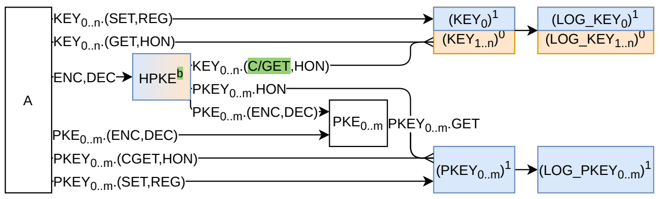
KDF. We model Krawczyk's HKDF as a function $\mathsf{kdf} : (\{0,1\}^\lambda \times \{0,1\}^\lambda \times \{0,1\}^*) \mapsto \{0,1\}^{\lambda'}$ for some hard-coded $\lambda' = \lambda'(\lambda)$ such that a triple $(k, \mathit{lbl}, \mathit{ctx})$ is mapped to a string of length λ'. We assume that 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 $\mathsf{Adv}_{\text{KDFL}}^{g}(\mathcal{A}) :=$

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

where GKDFL_g^{b} is defined in Figure 12b. 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 $\mathsf{Adv}_{\text{KDFR}}^{g}(\mathcal{A}) :=$

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

where GKDFR_g^{b} is defined in Figure 12c.



(a) Game $\text{GHPKE}^{b}_{\mathsf{pke}}$.

```
PKE
Package Parameters
pke : pub. key enc. sch.

Package State
$H[c \mapsto b]$ : hon. ciph.

ENC($h_{kp}, h_k$)
$pk \leftarrow$ parse($h_{kp}$)
$c \leftarrow\!\!\$\ $ pke.enc($pk, m$)
$H[c] \leftarrow 1$
return $c$

DEC($h_{kp}, c$)
$j \leftarrow$ parse($h_{kp}$)
if $H[c] = 1$:
    return $\perp$
$sk \leftarrow$ PKEY$_j$.GET($h_{kp}$)
$m \leftarrow$ pke.dec($sk, c$)
return $m$
```

```
HPKE
Package Parameters
$b$ : idealisation bit
$\lambda$ : encrypted key length

Package State
$Q[c \mapsto h]$ : handle table

ENC($h_{kp}, h_k$)
$j \leftarrow$ parse($h_{kp}$)
$i \leftarrow$ parse($h_k$)
assert PKEY$_j$.HON($h_{kp}$)
    $\vee \neg$KEY$_i$.HON($h_k$)
if $b \wedge$ PKEY$_j$.HON($h_{pk}$):
    $m \leftarrow 0^{\lambda}$
else if $b$:
    $m \leftarrow$ KEY$_i$.CGET($h_k$)
else :
    $m \leftarrow$ KEY$_i$.GET($h_k$)
$c \leftarrow\!\!\$\ $ PKE$_j$.ENC($h_{kp}, m$)
$Q[c] \leftarrow h_k$
return $c$

DEC($h_{kp}, c$)
$j \leftarrow$ parse($h_{kp}$)
if $Q[c] \neq \perp$:
    return $Q[c]$
$m \leftarrow$ DEC($h_{kp}, c$)
return $m$
```

(b) Packages PKE and HPKE

Figure 13: Game $\text{GHPKE}^{b}_{\mathsf{pke}}$.

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 $\mathsf{Adv}^{\mathsf{pke}}_{\mathrm{HPKE}}(\mathcal{A}) :=$

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

where the game $\text{GHPKE}^{b}_{\mathsf{pke}}$ is defined in Figure 13 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. Since we will need to run pke.kgen with explicit randomness, we introduce a *derive key pair function* (DKP), which maps the random coins to the output of pke.kgen.

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 these advantages by $\mathsf{Adv}^{\mathsf{xpd}}_{\mathrm{CRXPD}}(\mathcal{A})$ and $\mathsf{Adv}^{\mathsf{xpd}}_{\mathrm{CRKDF}}(\mathcal{A})$, respectively, and we refer by $\mathsf{Adv}^{\mathsf{xpd}}_{\mathrm{CRDKP}}(\mathcal{A})$ to the collision-resistance advantage of the derive key pair function dkp.

VI. Security Models

We now introduce our security model for the MLS CGKD. We start with a high-level overview which considers the 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.

A. CGKD

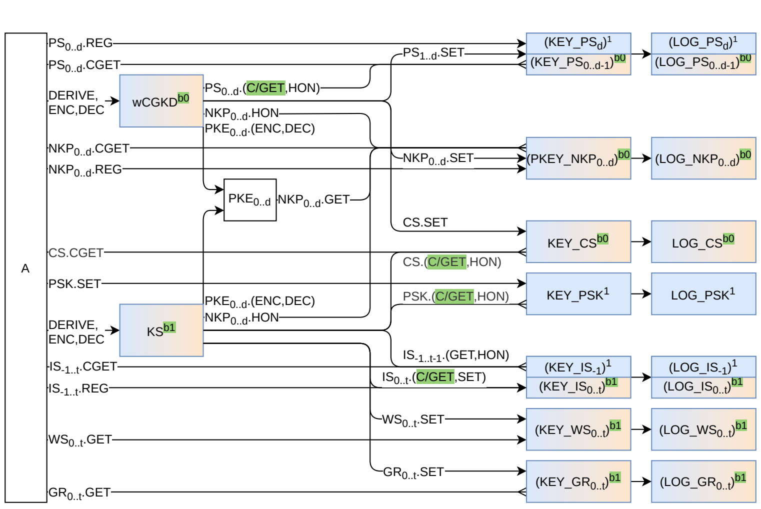
A crucial component in the modeling are different keys, inputs and outputs as described in Figure 2. Based on the rationale outlined in Section V, storing keys in separate packages is useful for composition, and so, our model stores each type of key in a separate KEY package, i.e., there are two main packages wCGKD (modeling TreeKEM) and KS, each reading and writing keys from/into the different KEY and PKEY packages, see Figure 14a for the composed game GWCGK and Figure 14c for GKS. Figure 14b and 14c 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 2 to the KEY packages.

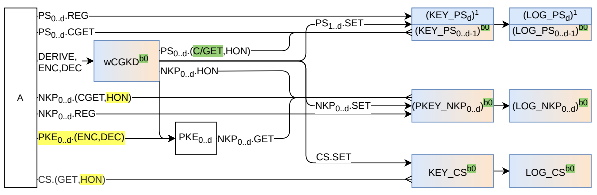
The values $(pk,\ sk)$ of the KEMs associated with different nodes in the tree are stored in PKEY_NKP packages, the k_{upd} variable, conceptually, corresponds to a path secret and thus is stored in the KEY_PS packages with other path secrets. Finally, values for the s_{ext} variable are stored in KEY_PSK packages, values for the s_{int} variable are stored in KEY_IS packages, and values of each of the k variables in $\overline{k}_{\overline{lbl}}$ 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 V 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 VI.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*

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


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


(b) Game $\text{GWCGKD}^{b_0,d}_{\text{xpd,pke}}$.

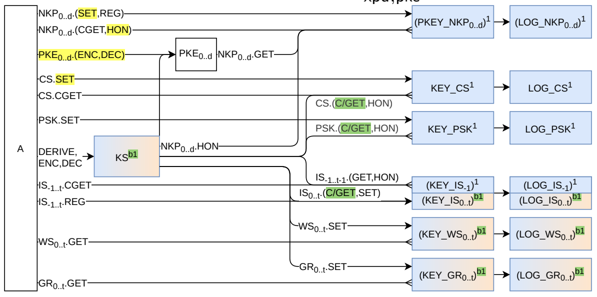
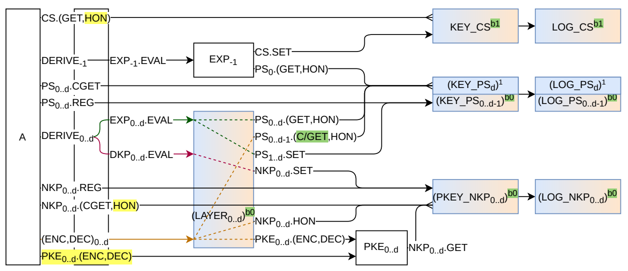

(c) Game $\text{GKS}^{b_1,d,t}_{\text{xpd,kdf,pke}}$.

Figure 14: Composed (left) and individual (right) TreeKEM and Key Schedule games.


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

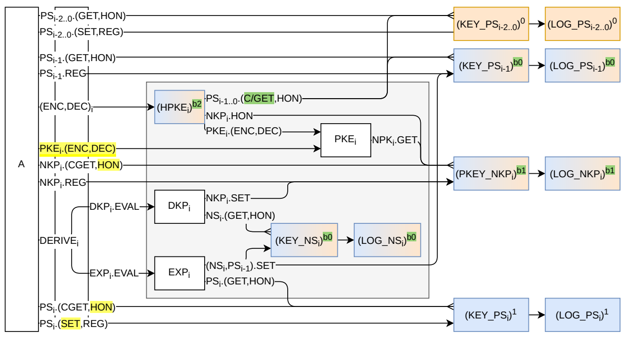

(b) Game of $\text{GLAYER}^{b_0 b_1 b_2,i}_{\text{xpd,pke}}$. We write $\text{GLAYER}^{b,i}_{\text{xpd,pke}}$ if $b_0 = b_1 = b_2$.

Figure 15: TreeKEM subgames.

B. TreeKEM model

In Figure 14b, the wCGKD package represents the composition of packages shown in the center of Figure 15a ($\text{LAYER}_{0..d}$ and EXP). Note that the KEY, PKEY, and PKE packages correspond to those in Figure 14b. 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 15b. Again, the KEY, PKEY_NKP, and PKE packages correspond to those in Figure 14b. Figure 15b encodes both the dn function (Figure 4) through the EXP and DKP 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.

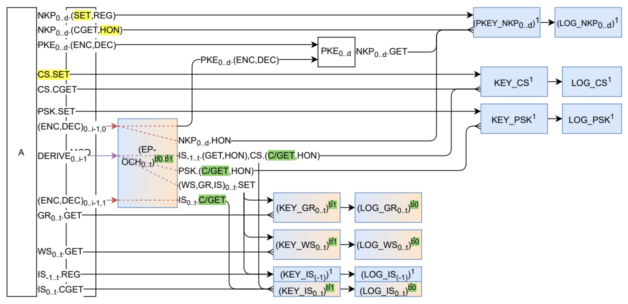
Definition VI.2 (wCGKD advantage)**.** *For an expand function* xpd*, a PKE* pke*, for all polynomials d and all adversaries $\mathcal{A}$ which generate trees of depth at most d, we denote by $\text{Adv}^{\text{xpd,pke}}_{wCGKD}(\mathcal{A})$ the advantage*

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

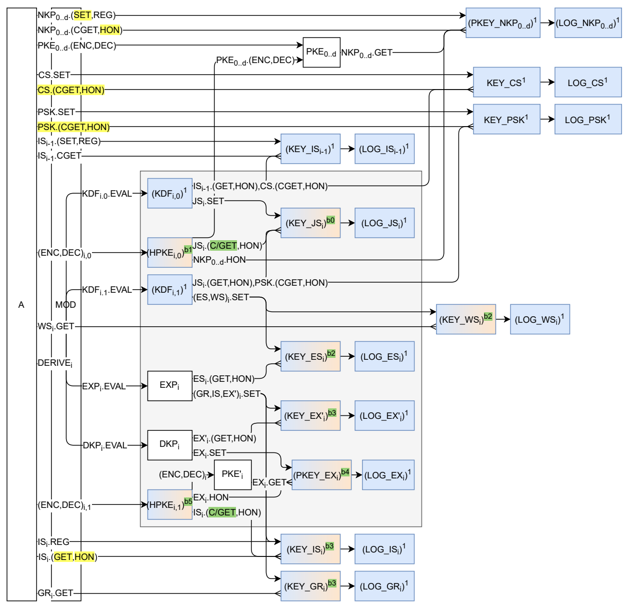
C. Key Schedule model

We define the KS package in Figure 14c by the composition of packages shown in the center of Figure 16a ($\text{EPOCH}_{0..t}$). The KEY, PKEY, and PKE packages correspond to those in Figure 14c. 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 16b. Recall that the KEY, PKEY, and PKE packages correspond to those in Figure 14c.

Figure 16b encodes the ks and subks functions (Figure 4). The KDF and EXP packages correspond to kdf and xpd calls



(a) Game $\text{GGKS}^{b_0' b_1', d, t}_{\text{xpd,kdf,pke}}$. We write $\text{GGKS}^{b,d,t}_{\text{xpd,kdf,pke}}$ if $b_0' = b_1'$.



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

Figure 16: Key Schedule subgames.

in ks and subks, 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 VI.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,* $\mathsf{Adv}^{\mathsf{xpd,kdf,pke}}_{KS}(\mathcal{A})$ *denotes*

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

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. Our security statements are concrete and constructive, i.e., they transform an adversary against the CGKD into an adversary against the underlying primitives.

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

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

where $\text{GWCGKD}^{b,d}_{\text{xpd,pke}}$ *is provided in Figure 14b.*

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*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{xpd,kdf,pke}}_{KS}(\mathcal{A}) \leq\ & \mathsf{Adv}^{\mathsf{xpd,kdf,dkp}}_{CR}(\mathcal{A} \circ \mathcal{R}_{cr}) \\ &+ \mathsf{Adv}^{\mathsf{kdf}}_{KDFL}(\mathcal{A} \circ \mathcal{R}_{int}) + \mathsf{Adv}^{\mathsf{kdf}}_{KDFL}(\mathcal{A} \circ \mathcal{R}_{joi}) \\ &+ \sum_{i=0}^{t} \mathsf{Adv}^{\mathsf{kdf}}_{KDFR}(\mathcal{A} \circ \mathcal{R}_{hyb,i} \circ \mathcal{R}_{a,i}) + \mathsf{Adv}^{\mathsf{pke}}_{HPKE}(\mathcal{A} \circ \mathcal{R}_{hyb,i} \circ \mathcal{R}_{b,i}) \\ &\quad + \mathsf{Adv}^{\mathsf{kdf}}_{KDFR}(\mathcal{A} \circ \mathcal{R}_{hyb,i} \circ \mathcal{R}_{c,i}) + \mathsf{Adv}^{\mathsf{xpd}}_{EXP}(\mathcal{A} \circ \mathcal{R}_{hyb,i} \circ \mathcal{R}_{d,i}) \\ &\quad + \mathsf{Adv}^{\mathsf{pke}}_{HPKE}(\mathcal{A} \circ \mathcal{R}_{hyb,i} \circ \mathcal{R}_{e,i}),\end{aligned}$$

where $\text{GKS}^{b,d,t}_{\text{xpd,kdf,pke}}$ *is defined in Figure 14c.*

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*

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

where $\text{GCGKD}^{b,d,t}_{\text{xpd,kdf,pke}}$ *is given in Figure 14a.*

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

VIII. 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* $\text{GLAYER}^{b,i}_{\text{xpd,pke}}$ *the indistinguishability game defined by Figure 15b. Then it holds that*

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

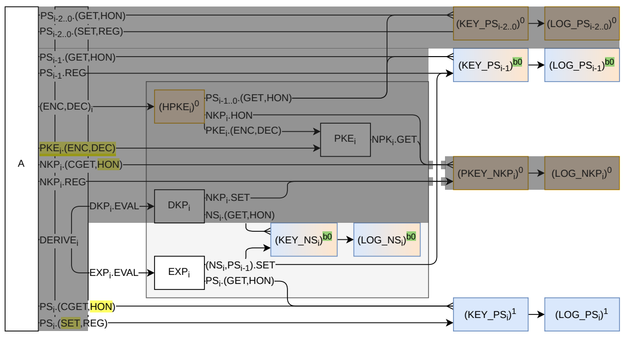
where $\mathcal{R}_{exp,i}$ is defined as the grey area in Figure 17a and $\mathcal{R}_{hpke,i}$ is defined as the grey area in Figure 17c.

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

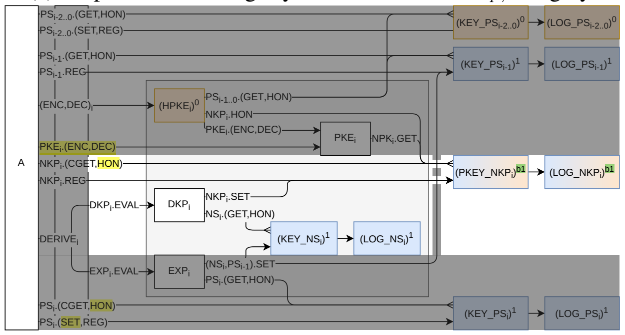
$$\text{GLAYER}^{b,i}_{\text{xpd,pke}} := \text{GLAYER}^{bbb,i}_{\text{xpd,pke}},$$

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

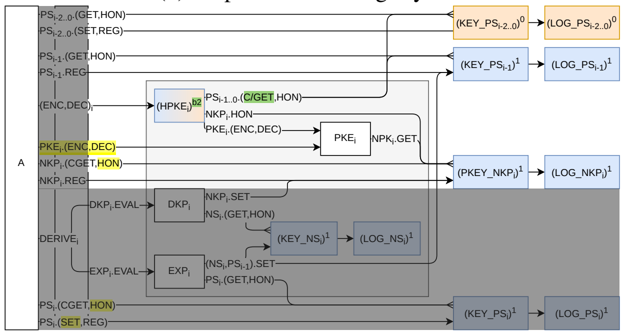
In the first proof step, we apply the expand assumption for $\overline{lbl} = \{path, node\}$ to idealize the node secret of layer i and the path secret of layer $p(i)$. We obtain the following bound:



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



(b) Step 2 of idealizing layer i.



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

Figure 17: Proof of Lemma 1.

$$\left|\Pr\left[1 \leftarrow \mathcal{B} \circ \text{GLAYER}^{000,i}_{\text{xpd,pke}}\right] - \Pr\left[1 \leftarrow \mathcal{B} \circ \text{GLAYER}^{100,i}_{\text{xpd,pke}}\right]\right| \le \text{Adv}^{\text{xpd}}_{\text{EXP}}(\mathcal{B} \circ \mathcal{R}_{exp,i}),$$

where $\mathcal{R}_{exp,i}$ is defined as the grey area in Figure 17a.

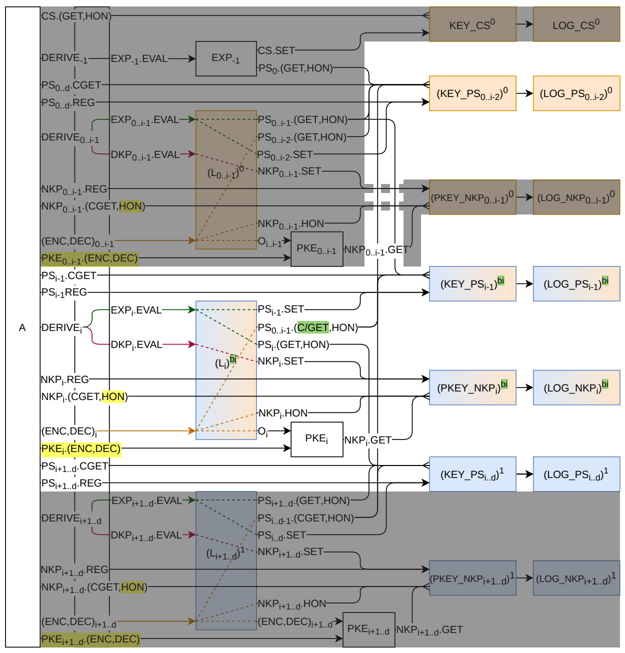
In the second step of this proof we use the definition of a DKP function to idealize the PKEY package of layer i. By the perfect correctness of the DKP, we obtain that

$$\left|\Pr\left[1 \leftarrow \mathcal{B} \circ \text{GLAYER}^{100,i}_{\text{xpd,pke}}\right] - \Pr\left[1 \leftarrow \mathcal{B} \circ \text{GLAYER}^{110,i}_{\text{xpd,pke}}\right]\right| = 0.$$

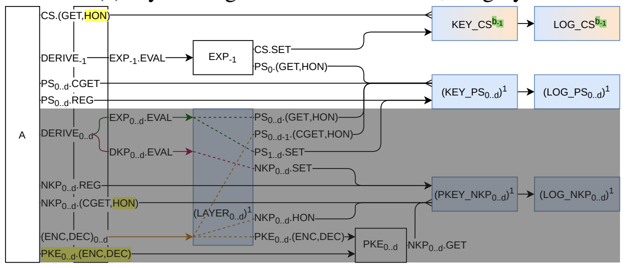
In the third and last step of this proof we apply the HPKE assumption for $n = i - 1$ and $m = 1$ to idealize the HPKE package of layer i. This gives us the following bound w.r.t. $\text{GLAYER}^{b,i}_{\text{xpd,pke}}$

$$\left|\Pr\left[1 \leftarrow \mathcal{B} \circ \text{GLAYER}^{110,i}_{\text{xpd,pke}}\right] - \Pr\left[1 \leftarrow \mathcal{B} \circ \text{GLAYER}^{111,i}_{\text{xpd,pke}}\right]\right| \le \text{Adv}^{\text{pke}}_{\text{HPKE}}(\mathcal{B} \circ \mathcal{R}_{hpke,i}),$$

where $\mathcal{R}_{hpke,i}$ is defined as the grey area in Figure 17c. Hence,



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



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

Figure 18: Proof of Lemma 1.

$$\begin{aligned}\text{Adv}^{\text{xpd,pke},i}_{\text{LAYER}}(\mathcal{B}) &\le \text{Adv}^{\text{xpd}}_{\text{EXP}}(\mathcal{B} \circ \mathcal{R}_{exp,i}) + 0 + \text{Adv}^{\text{pke}}_{\text{HPKE}}(\mathcal{B} \circ \mathcal{R}_{hpke,i}) \\ &= \text{Adv}^{\text{xpd}}_{\text{EXP}}(\mathcal{B} \circ \mathcal{R}_{exp,i}) + \text{Adv}^{\text{pke}}_{\text{HPKE}}(\mathcal{B} \circ \mathcal{R}_{hpke,i}),\end{aligned}$$

which concludes the proof of Lemma 1.

Hybrid argument. We apply a hybrid over Lemma 1 to the game GWCGKD defined by Figure 15a, which, as explained in Section VI, is equivalent to Figure 14b.

For convenience we add several bits into the superscript of GWCGKD to model security of the different layers, i.e., if $b = b_{-1} = \ldots = b_{d-1}$, then

$$\text{GWCGKD}^{b_{-1}\ldots b_{d-1},d}_{\text{xpd,pke}} = \text{GWCGKD}^{b\ldots b,d}_{\text{xpd,pke}}.$$

Recall that the 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 18a. Hence, for all PPT adversaries $\mathcal{A}$, we have

$$\left|\Pr\left[1 \leftarrow \mathcal{A} \circ \text{GWCGKD}^{10\ldots0,d}_{\text{xpd,pke}}\right] - \Pr\left[1 \leftarrow \mathcal{A} \circ \text{GWCGKD}^{1\ldots10,d}_{\text{xpd,pke}}\right]\right| \le \text{Adv}^{\text{xpd,pke},i}_{\text{LAYER}}(\mathcal{A} \circ \mathcal{R}_i),$$

```
$\mathsf{wprocess}(PK_{mem}, PK_{rem},$
        $PK_{joi}, PK_{upd},$
        $SK_{own}, i_{nod}, i_{upd}, c)$
$k_{cs} \leftarrow 0$
$PK'_{mem} \leftarrow PK_{mem}$
$PK'_{mem} \overset{\text{rem}}{\leftarrow} PK_{rem}$
$PK'_{mem} \overset{\text{merge}}{\leftarrow} PK_{joi}$
$PK'_{mem} \overset{\text{vec}}{\leftarrow} PK_{upd}$
$SK'_{own} \leftarrow SK_{own}$
$sk_{nod} \leftarrow SK_{own}[i_{nod}]$
$ps \leftarrow \mathsf{pke.dec}(sk_{nod}, c)$
$\mathbf{assert}\ ps \neq \perp$
$i_{cur} \leftarrow \mathsf{lcp}(i_{own}, i_{upd}, |PK'_{mem}|)$
$(\_, \_, SK''_{own}, k_{cs})$
    $\leftarrow \mathsf{dopath}(PK'_{mem}, i_{cur}, ps, \perp)$
$SK'_{own} \overset{\text{vec}}{\leftarrow} SK''_{own}$
$\mathbf{return}\ (PK'_{mem}, SK'_{own}, k_{cs})$
```

```
$\mathsf{wjoin}(PK_{mem}, SK_{own}, i_{own},$
    $i_{upd}, c)$
$k_{cs} \leftarrow 0$
$sk_{own} \leftarrow SK_{own}[i_{own}]$
$ps \leftarrow \mathsf{pke.dec}(sk_{own}, c)$
$\mathbf{assert}\ ps \neq \perp$
$i_{cur} \leftarrow \mathsf{lcp}(i_{own}, i_{upd}, |PK_{mem}|)$
$(\_, \_, \_, k_{cs})$
    $\leftarrow \mathsf{dopath}(PK_{mem}, i_{cur}, ps, \perp)$
$\mathbf{return}\ (PK_{mem}, SK_{own}, k_{cs})$
```

Figure 19: Procedures wprocess and wjoin.

where $\mathcal{R}_i$ is defined as the grey area in Figure 18a. 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 18b, we apply the expand assumption and obtain

$$\left|\Pr\left[1 \leftarrow \mathcal{A} \circ \mathrm{GWCGKD}^{1\ldots10,d}_{\mathsf{xpd,pke}}\right] - \Pr\left[1 \leftarrow \mathcal{A} \circ \mathrm{GWCGKD}^{1\ldots11,d}_{\mathsf{xpd,pke}}\right]\right|$$
$$\leq \mathsf{Adv}^{\mathsf{xpd}}_{\mathsf{EXP}}(\mathcal{A} \circ \mathcal{R}_{exp}),$$

where $\mathcal{R}_{exp}$ is defined as the grey area in Figure 18b. Using $\mathcal{B} = \mathcal{A} \circ \mathcal{R}_i$, we derive the desired bound for Theorem 1:

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

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] Richard Barnes, Benjamin Beurdouche, Jon Millican, Katriel Cohn-Gordon Emad Omara, and Raphael Robert. Message layer security rfc draft 11. 2019. https://datatracker.ietf.org/doc/draft-ietf-mls-protocol/.

[4] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In *Annual international cryptology conference*, pages 232–249. Springer, 1993.

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

[6] Karthikeyan Bhargavan, Richard Barnes, and Eric Rescorla. *TreeKEM: Asynchronous Decentralized Key Management for Large Dynamic Groups A protocol proposal for Messaging Layer Security (MLS)*. PhD thesis, Inria Paris, 2018. https://prosecco.gforge.inria.fr/personal/karthik/pubs/treekem.pdf.

[7] Nikita Borisov, Ian Goldberg, and Eric Brewer. Off-the-record communication, or, why not to use pgp. In *Proceedings 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.

[8] Chris Brzuska, Antoine Delignat-Lavaud, Cédric Fournet, Konrad Kohbrok, and Markulf Kohlweiss. Handshake security for the TLS 1.3 standard. Preprint.

[9] 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, *ASIACRYPT 2018, Part III*, volume 11274 of *LNCS*, pages 222–249, Brisbane, Queensland, Australia, December 2–6, 2018. Springer, Heidelberg, Germany.

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

[11] 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.

[12] 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, *ACM CCS 2018*, pages 1802–1819, Toronto, ON, Canada, October 15–19, 2018. ACM Press.

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

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

[15] 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.

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

[17] 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, *PKC 2020, Part I*, volume 12110 of *LNCS*, pages 341–373, Edinburgh, UK, May 4–7, 2020. Springer, Heidelberg, Germany.

[18] Yongdae Kim, Adrian Perrig, and Gene Tsudik. Communication-efficient group key agreement. In *Trusted 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 *IFIP Conference Proceedings*, pages 229–244. Kluwer, 2001.

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

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

[21] 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.

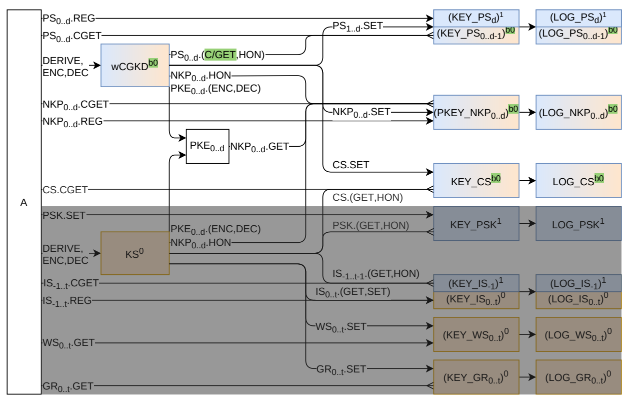
[22] M. Marlinspike and T. Perrin. Signal specifications. Technical report, 2016. https://signal.org/docs/.

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

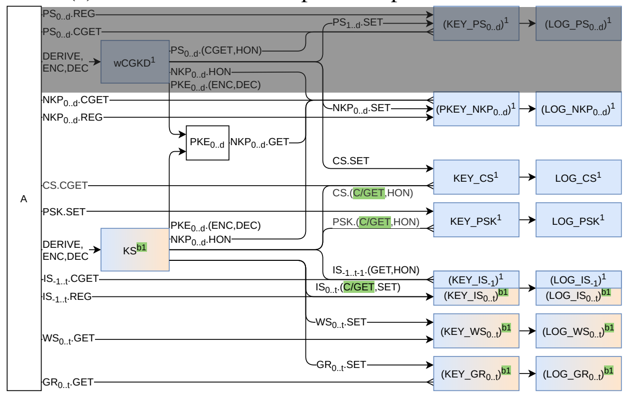
[24] Jesper Buus Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 111–126, Santa Barbara, CA, USA, August 18–22, 2002. Springer, Heidelberg, Germany.

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

[26] D. G. Steer, L. Strawczynski, Whitfield Diffie, and Michael J. Wiener. A secure audio teleconference system. In Shafi Goldwasser, editor,

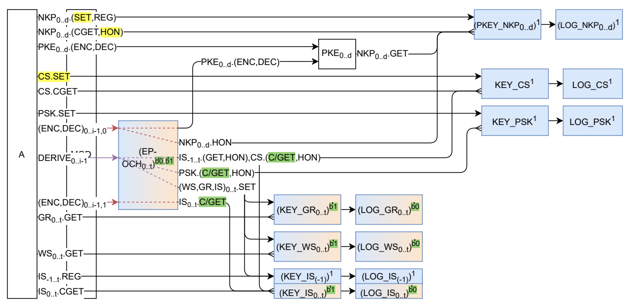


(a) First idealization step of composition theorem.

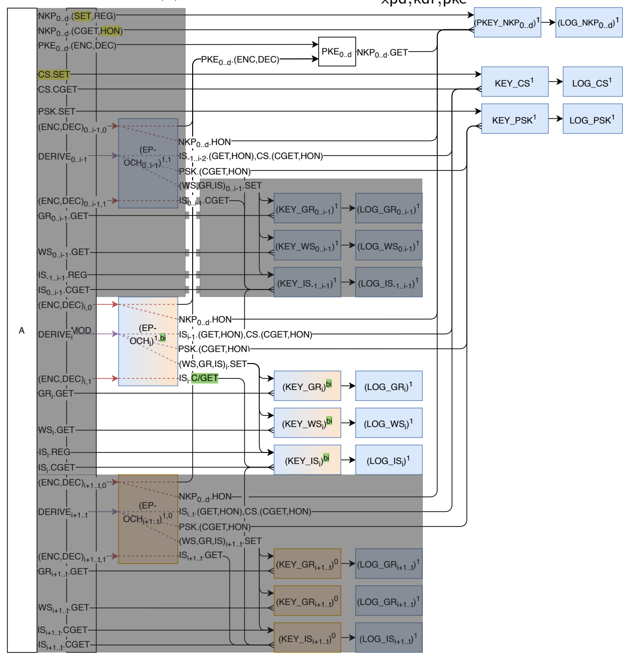


(b) Second idealization step of composition theorem.

Figure 20: Theorem 3.



(a) Definition of $\mathrm{GGKS}^{b_0' b_1',d,t}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}$.



(b) Reducing a hybrid step in the key schedule proof to the security of a single epoch. Reduction $\mathcal{R}_{\mathrm{hyb,i}}$ is marked in grey.

Figure 21: Proof of Theorem 2.

CRYPTO '88, volume 403 of *LNCS*, pages 520–528, Santa Barbara, CA, USA, August 21–25, 1990. Springer, Heidelberg, Germany.

[27] Matthew Weidner. Group messaging for secure asynchronous collaboration. PhD thesis, MPhil Dissertation, 2019. Advisors: A. Beresford and M. Kleppmann, 2019.

[28] 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. *Proof of Theorem 3 (CGKD security)*

We prove Theorem 3 using Theorem 1 and Theorem 2. Recall (see Figure 14a) that, if $b = b_0 = b_1$, then

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

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

$$\left|\Pr\left[1 \leftarrow \mathcal{A} \circ \mathrm{GCGKD}^{00,d,t}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}\right] - \Pr\left[1 \leftarrow \mathcal{A} \circ \mathrm{GCGKD}^{10,d,t}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}\right]\right|$$
$$\leq \mathsf{Adv}^{\mathsf{xpd},\mathsf{pke}}_{\mathrm{wCGKD}}(\mathcal{A} \circ \mathcal{R}_p),$$

where $\mathcal{R}_p$ is defined as the grey area in Figure 20a. Next, we modify b_1 from 0 to 1 and reduce to GKS. We obtain

$$\left|\Pr\left[1 \leftarrow \mathcal{A} \circ \mathrm{GCGKD}^{10,d,t}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}\right] - \Pr\left[1 \leftarrow \mathcal{A} \circ \mathrm{GCGKD}^{11,d,t}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}\right]\right|$$
$$\leq \mathsf{Adv}^{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}_{\mathrm{KS}}(\mathcal{A} \circ \mathcal{R}_q),$$

where $\mathcal{R}_q$ is defined as the grey area in Figure 20b. Hence, we obtain the desired bound for Theorem 3:

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

B. *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.

$\mathtt{KEY\_IS}_0$ 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 register 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 3.

Figure 21a depicts $\mathrm{GGKS}^{b_0' b_1',d,t}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}$ for trees up to depth d and groups that run for up to t epochs. For $b = b_0' = b_1'$, we define

$$\mathrm{GKS}^{b,d,t}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}} := \mathrm{GGKS}^{b_0' b_1',d,t}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}.$$

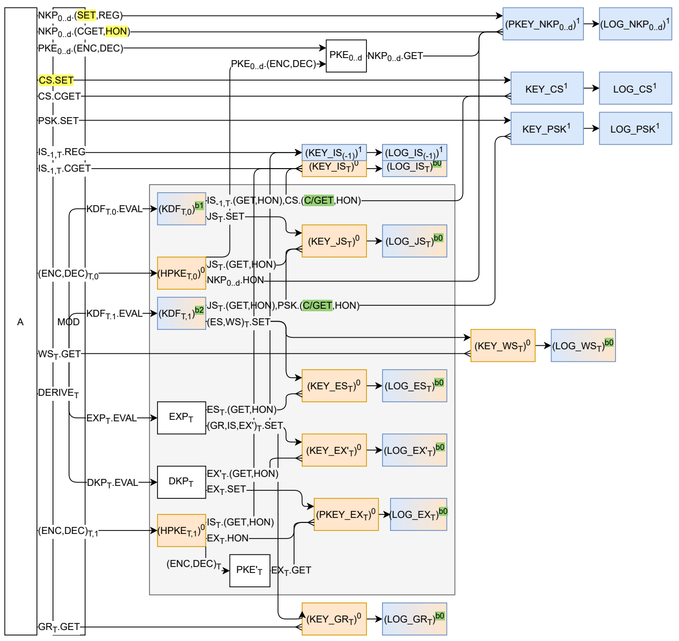


Figure 22: $\text{GPGKS}^{b_0 b_1 b_2 0,d,t}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}$.

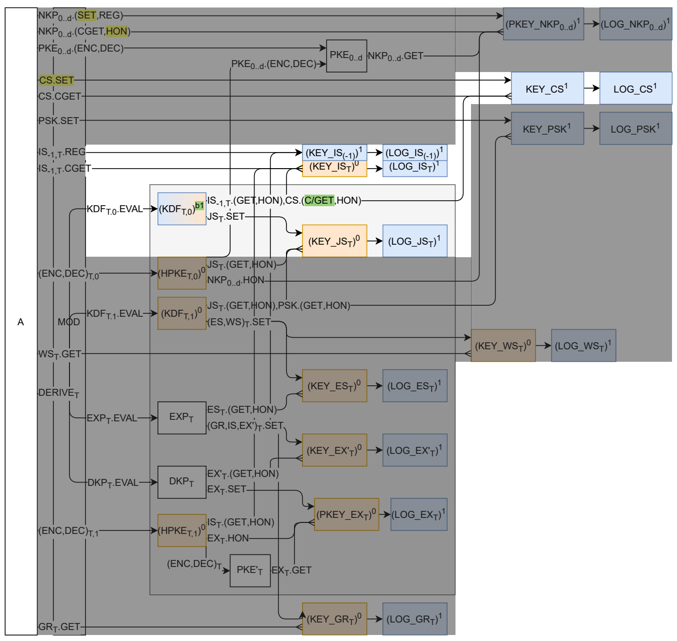


Figure 23: Reducing to KDFL security based on ideal CS. Reduction $\mathcal{R}_{\text{cs}}$ is marked in grey.

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* $\mathsf{Adv}^{\mathsf{xpd},\mathsf{kdf},\mathsf{pke},b0}_{GGKS}(\mathcal{A}) :=$

$$\left|\Pr\left[1 \leftarrow \mathcal{A} \circ \text{GGKS}^{00,d,t}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}\right] - \Pr\left[1 \leftarrow \mathcal{A} \circ \text{GGKS}^{10,d,t}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}\right]\right| \leq \mathsf{Adv}^{\mathsf{xpd},\mathsf{kdf},\mathsf{dkp}}_{CR}(\mathcal{A} \circ \mathcal{R}_{cr}) + \mathsf{Adv}^{\mathsf{kdf}}_{KDFL}(\mathcal{A} \circ \mathcal{R}_{cs}) + \mathsf{Adv}^{\mathsf{kdf}}_{KDFL}(\mathcal{A} \circ \mathcal{R}_{psk})$$

Recall that $\text{GGKS}^{b'_0 b'_1,d,t}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}$ is defined in Figure 21a. Analogously to $\mathsf{Adv}^{\mathsf{xpd},\mathsf{kdf},\mathsf{pke},b0}_{\text{GGKS}}(\mathcal{A})$, we use the notation

$$\mathsf{Adv}^{\mathsf{xpd},\mathsf{kdf},\mathsf{pke},1b}_{\text{GGKS}}(\mathcal{A}) := \left|\Pr\left[1 \leftarrow \mathcal{A} \circ \text{GGKS}^{10,d,t}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}\right] - \Pr\left[1 \leftarrow \mathcal{A} \circ \text{GGKS}^{11,d,t}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}\right]\right|.$$

In order to state Lemma 3, we need to split b'_1 into t different bits b_0 to b_t. We write $\text{GGKS}^{0b_0\ldots b_t,d,t}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}$ and note that

$$\text{GGKS}^{0b_0\ldots b_t,d,t}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}} = \text{GGKS}^{0b,d,t}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}$$

for $b'_1 = b_0 = \ldots = b_t$.

Lemma 3. *(Epoch Security) Let $\mathcal{B}$ be an adversary and $\text{GEPOCH}^{b,d,t,i}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}$ the indistinguishability game defined by Figure 25. Then $\forall d,t,i \in \mathbb{N} : 0 \leq i \leq t$*

$$\mathsf{Adv}^{\mathsf{xpd},\mathsf{kdf},\mathsf{pke},i}_{EPOCH}(\mathcal{B}) := \left|\Pr\left[1 \leftarrow \mathcal{B} \circ \text{GEPOCH}^{0,d,t,i}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}\right] - \Pr\left[1 \leftarrow \mathcal{B} \circ \text{GEPOCH}^{1,d,t,i}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}\right]\right| \leq \mathsf{Adv}^{\mathsf{kdf}}_{KDFR}(\mathcal{B} \circ \mathcal{R}_{a,i}) + \mathsf{Adv}^{\mathsf{pke}}_{HPKE}(\mathcal{B} \circ \mathcal{R}_{b,i}) + \mathsf{Adv}^{\mathsf{kdf}}_{KDFR}(\mathcal{B} \circ \mathcal{R}_{c,i}) + \mathsf{Adv}^{\mathsf{xpd}}_{EXP}(\mathcal{B} \circ \mathcal{R}_{d,i}) + \mathsf{Adv}^{\mathsf{pke}}_{HPKE}(\mathcal{B} \circ \mathcal{R}_{e,i}),$$

where $\mathcal{R}_{a,i}$ is defined as the grey area in Figure 26, $\mathcal{R}_{b,i}$ as the grey area in Figure 27, $\mathcal{R}_{c,i}$ as the grey area in Figure 28, $\mathcal{R}_{d,i}$ as the grey area in Figure 29, and $\mathcal{R}_{e,i}$ as the grey area in Figure 30.

Proof of Theorem 2. We first apply Lemma 2 and then apply a hybrid argument over the t epochs using that for all $\mathcal{B}$,

$$\mathsf{Adv}^{\mathsf{xpd},\mathsf{kdf},\mathsf{pke},i}_{\text{EPOCH}}(\mathcal{B} \circ \mathcal{R}_{\text{hyb,i}}) = \left| \Pr\left[1 \leftarrow \mathcal{B} \circ \text{GGKS}^{1,1_0,..,1_{i-1},0_i,0_{i+1},..,0_t,d,t}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}\right] - \Pr\left[1 \leftarrow \mathcal{B} \circ \text{GGKS}^{1,1_0,..,1_{i-1},1_i,0_{i+1},..,0_t,d,t}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}\right] \right|, \quad (1)$$

where $\mathcal{R}_{\text{hyb,i}}$ is defined in Figure 21b. We obtain Theorem 2 as follows:

$$\begin{aligned}\mathsf{Adv}^{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}_{\text{KS}}(\mathcal{A}) &\leq \mathsf{Adv}^{\mathsf{xpd},\mathsf{kdf},\mathsf{pke},b0}_{\text{GKS}}(\mathcal{A}) + \mathsf{Adv}^{\mathsf{xpd},\mathsf{kdf},\mathsf{pke},1b}_{\text{GKS}}(\mathcal{A})\\ &\leq \mathsf{Adv}^{\mathsf{xpd},\mathsf{kdf},\mathsf{pke},b0}_{\text{GKS}}(\mathcal{A}) + \sum_{i=0}^{t-1} \mathsf{Adv}^{\mathsf{xpd},\mathsf{kdf},\mathsf{pke},i}_{\text{EPOCH}}(\mathcal{A} \circ \mathcal{R}_{\text{hyb},i}).\end{aligned}$$

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

C. Proof of Lemma 2

In order to prove Lemma 2, we split b'_0 into 3 different bits b_0, b_1, and b_2 and write $\text{GPGKS}^{b_0 b_1 b_2,0,d,t}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}$ for the game depicted in Figure 22, where the P in GPGKS stands for *proof*. Note that when $b_0 = b_1 = b_2 = b'_0$, then

$$\text{GPGKS}^{b_0 b_1 b_2 0,d,t}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}} = \text{GGKS}^{b'_0 0,d,t}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}.$$

For collision-resistance, we can construct a reduction $\mathcal{R}_{\text{cr}}$ such that the distinguishing advantage between $\text{GPGKS}^{0000,d,t}_{\mathsf{xpd},\mathsf{kdf},\mathsf{pke}}$ and

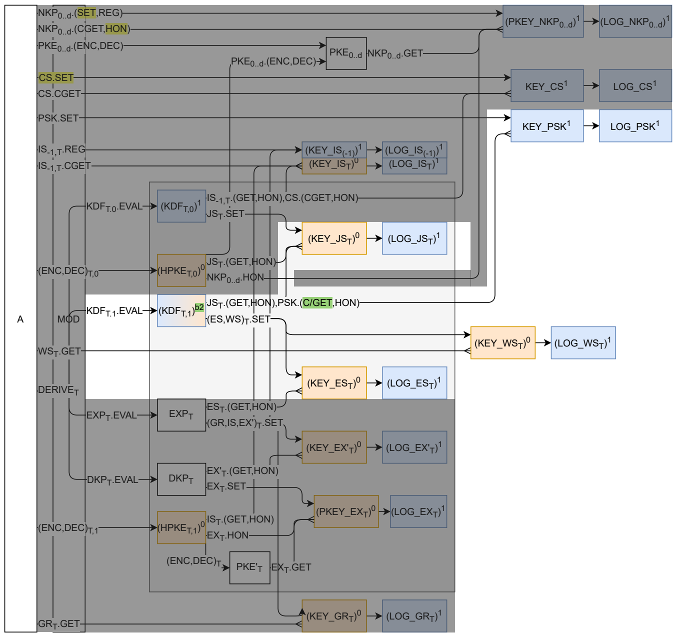


Figure 24: Reducing to KDFL security based on ideal PSK. Reduction $\mathcal{R}_{\text{psk}}$ is marked in grey.

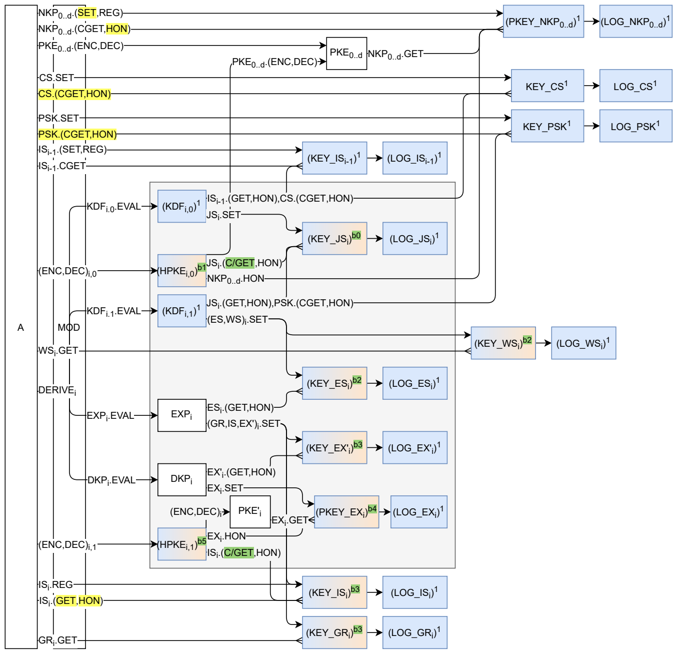


Figure 25: Definition of $\text{GEPOCH}^{b_0 b_1 b_2 b_3 b_4 b_5,d,t,i}_{\text{xpd,kdf,pke}}$. We write $\text{GEPOCH}^{b,d,t,i}_{\text{xpd,kdf,pke}}$ if $b_0 = b_1 = b_2 = b_3 = b_4 = b_5$. Note that $\texttt{KEY\_IS}_{-1}$ should technically have a REG query, but for an ideal $\texttt{KEY}$ package REG and SET are functionally equivalent.

$\text{GPGKS}^{1000,d,t}_{\text{xpd,kdf,pke}}$ 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

$$\left|\Pr\left[1 \leftarrow \mathcal{A} \circ \text{GPGKS}^{0000,d,t}_{\text{xpd,kdf,pke}}\right] - \Pr\left[1 \leftarrow \mathcal{A} \circ \text{GPGKS}^{1000,d,t}_{\text{xpd,kdf,pke}}\right]\right| \le \mathsf{Adv}^{\text{xpd,kdf,dkp}}_{\text{CR}}(\mathcal{A} \circ \mathcal{R}_{cr}).$$

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

$$\left|\Pr\left[1 \leftarrow \mathcal{A} \circ \text{GPGKS}^{1000,d,t}_{\text{xpd,kdf,pke}}\right] - \Pr\left[1 \leftarrow \mathcal{A} \circ \text{GPGKS}^{1100,d,t}_{\text{xpd,kdf,pke}}\right]\right| \le \mathsf{Adv}^{\text{kdf}}_{\text{KDFL}}(\mathcal{A} \circ \mathcal{R}_{cs}),$$

where $\mathcal{R}_{cs}$ is defined in Figure 23. Analogously,

$$\left|\Pr\left[1 \leftarrow \mathcal{A} \circ \text{GPGKS}^{1100,d,t}_{\text{xpd,kdf,pke}}\right] - \Pr\left[1 \leftarrow \mathcal{A} \circ \text{GPGKS}^{1110,d,t}_{\text{xpd,kdf,pke}}\right]\right| \le \mathsf{Adv}^{\text{kdf}}_{\text{KDFL}}(\mathcal{A} \circ \mathcal{R}_{\text{psk}}),$$

where $\mathcal{R}_{psk}$ is defined in Figure 24. We obtain

$$\begin{aligned}\mathsf{Adv}^{\text{xpd,kdf,pke},b0}_{\text{GKS}}(\mathcal{A}) &:= \\ \left|\Pr\left[1 \leftarrow \mathcal{A} \circ \text{GKS}^{00,d,t}_{\text{xpd,kdf,pke}}\right] - \Pr\left[1 \leftarrow \mathcal{A} \circ \text{GKS}^{10,d,t}_{\text{xpd,kdf,pke}}\right]\right| &\le \mathsf{Adv}^{\text{xpd,kdf,dkp}}_{\text{CR}}(\mathcal{A} \circ \mathcal{R}_{cr}) + \mathsf{Adv}^{\text{kdf}}_{\text{KDFL}}(\mathcal{A} \circ \mathcal{R}_{\text{cs}}) \\ &\quad + \mathsf{Adv}^{\text{kdf}}_{\text{KDFL}}(\mathcal{A} \circ \mathcal{R}_{\text{psk}}),\end{aligned}$$

which concludes the proof of Lemma 2.

D. 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.

The proof of Lemma 3 consists of six steps, each of which flips one or more bits in the overall game. See Figure 25 for the definition of $\text{GEPOCH}^{b_0 b_1 b_2 b_3 b_4 b_5,d,t,i}_{\text{xpd,kdf,pke}}$ and note that

$$\text{GEPOCH}^{b,d,t,i}_{\text{xpd,kdf,pke}} = \text{GEPOCH}^{b_0 b_1 b_2 b_3 b_4 b_5,d,t,i}_{\text{xpd,kdf,pke}}$$

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 $\texttt{KEY\_JS}$ package of epoch i. We obtain

$$\left|\Pr\left[1 \leftarrow \mathcal{A} \circ \text{GEPOCH}^{000000,d,t,i}_{\text{xpd,kdf,pke}}\right] - \Pr\left[1 \leftarrow \mathcal{A} \circ \text{GEPOCH}^{100000,d,t,i}_{\text{xpd,kdf,pke}}\right]\right| \le \mathsf{Adv}^{\text{kdf}}_{\text{KDFR}}(\mathcal{A} \circ \mathcal{R}_{a,i}).$$

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

$$\left|\Pr\left[1 \leftarrow \mathcal{A} \circ \text{GEPOCH}^{100000,d,t,i}_{\text{xpd,kdf,pke}}\right] - \Pr\left[1 \leftarrow \mathcal{A} \circ \text{GEPOCH}^{110000,d,t,i}_{\text{xpd,kdf,pke}}\right]\right| \le \mathsf{Adv}^{\text{pke}}_{\text{HPKE}}(\mathcal{A} \circ \mathcal{R}_{b,i}).$$

In the third step of this proof we use the KDFR assumption to idealize the $\texttt{KEY\_ES}$ package of epoch i. This is possible since there is no GET query on the joiner secret anymore. We obtain the bound

$$\left|\Pr\left[1 \leftarrow \mathcal{A} \circ \text{GEPOCH}^{110000,d,t,i}_{\text{xpd,kdf,pke}}\right] - \Pr\left[1 \leftarrow \mathcal{A} \circ \text{GEPOCH}^{111000,d,t,i}_{\text{xpd,kdf,pke}}\right]\right| \le \mathsf{Adv}^{\text{kdf}}_{\text{KDFR}}(\mathcal{A} \circ \mathcal{R}_{c,i}).$$

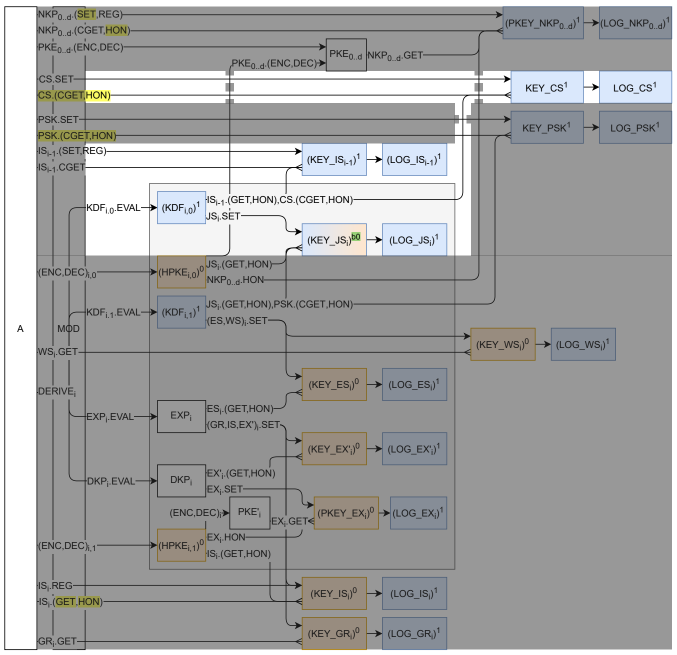


Figure 26: Step 1 of idealizing epoch i. We mark $\mathcal{R}_{a,i}$ in grey.

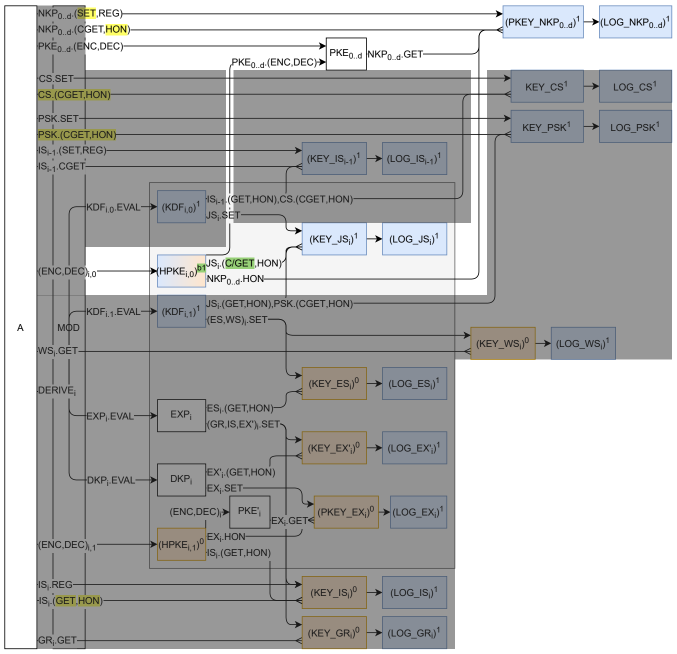


Figure 27: Step 2 of idealizing epoch i. We mark $\mathcal{R}_{b,i}$ in grey.

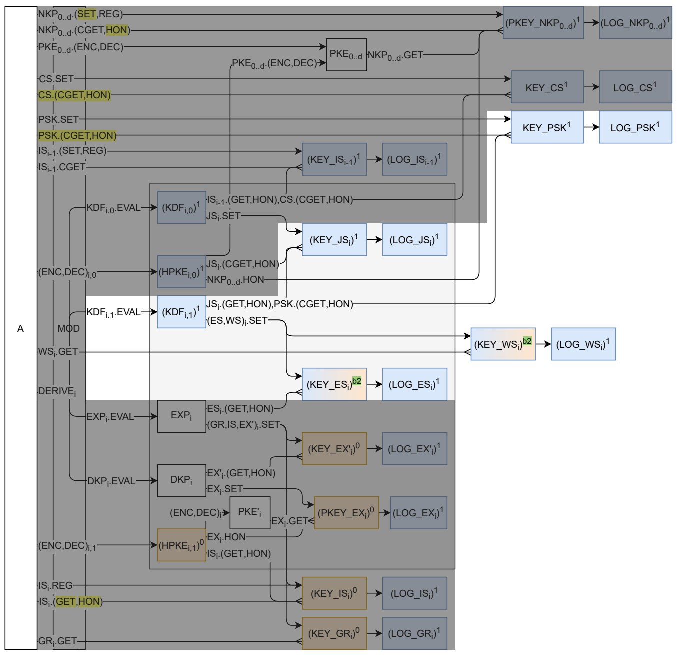


Figure 28: Step 3 of idealizing epoch i. We mark $\mathcal{R}_{c,i}$ in grey.

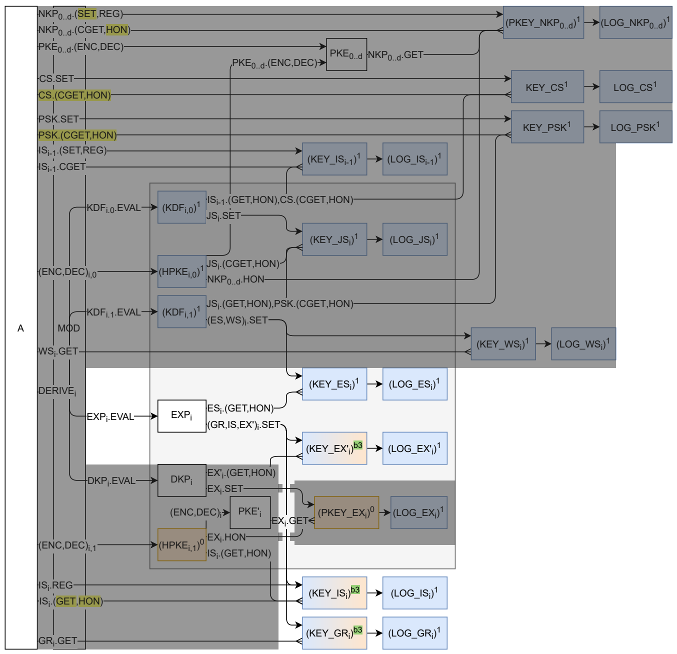


Figure 29: Step 4 of idealizing epoch i. We mark $\mathcal{R}_{d,i}$ in grey.

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

$$\left|\Pr\left[1 \leftarrow \mathcal{A} \circ \text{GEPOCH}^{111000,d,t,i}_{\mathsf{xpd,kdf,pke}}\right] - \Pr\left[1 \leftarrow \mathcal{A} \circ \text{GEPOCH}^{111100,d,t,i}_{\mathsf{xpd,kdf,pke}}\right]\right|$$
$$\leq \mathsf{Adv}^{\mathsf{xpd}}_{\mathsf{EXP}}(\mathcal{A} \circ \mathcal{R}_{d,i}).$$

In the fifth step of this proof, we use the definition of a DKP function to idealize the `PKEY_EX` package of epoch i. Due to the perfect correctness of the DKP, we have that $0 =$

$$\left|\Pr\left[1 \leftarrow \mathcal{A} \circ \text{GEPOCH}^{111100,d,t,i}_{\mathsf{xpd,kdf,pke}}\right] - \Pr\left[1 \leftarrow \mathcal{A} \circ \text{GEPOCH}^{111110,d,t,i}_{\mathsf{xpd,kdf,pke}}\right]\right|$$

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

$$\left|\Pr\left[1 \leftarrow \mathcal{A} \circ \text{GEPOCH}^{111110,d,t,i}_{\mathsf{xpd,kdf,pke}}\right] - \Pr\left[1 \leftarrow \mathcal{A} \circ \text{GEPOCH}^{111111,d,t,i}_{\mathsf{xpd,kdf,pke}}\right]\right|$$
$$\leq \mathsf{Adv}^{\mathsf{pke}}_{\mathsf{HPKE}}(\mathcal{A} \circ \mathcal{R}_{e,i}).$$

Hence,

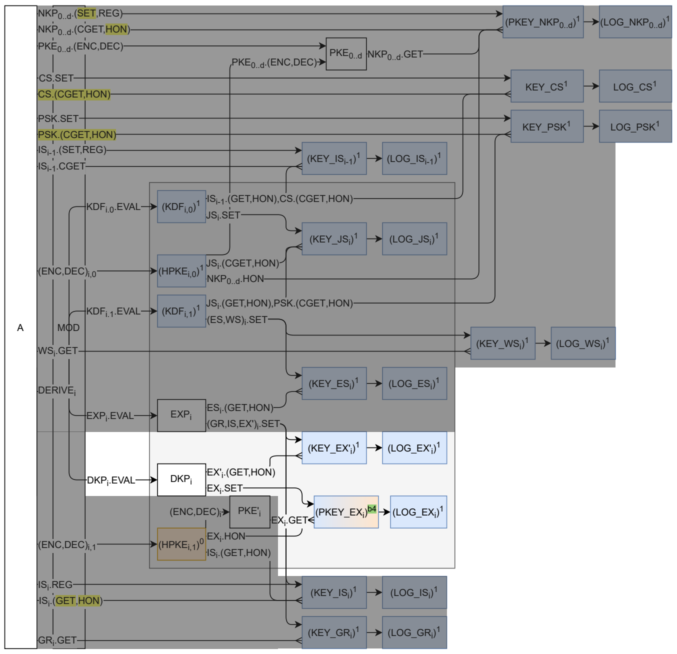


Figure 30: Step 5 of idealizing epoch i. We mark $\mathcal{R}_{\text{dkp}}$ in grey.

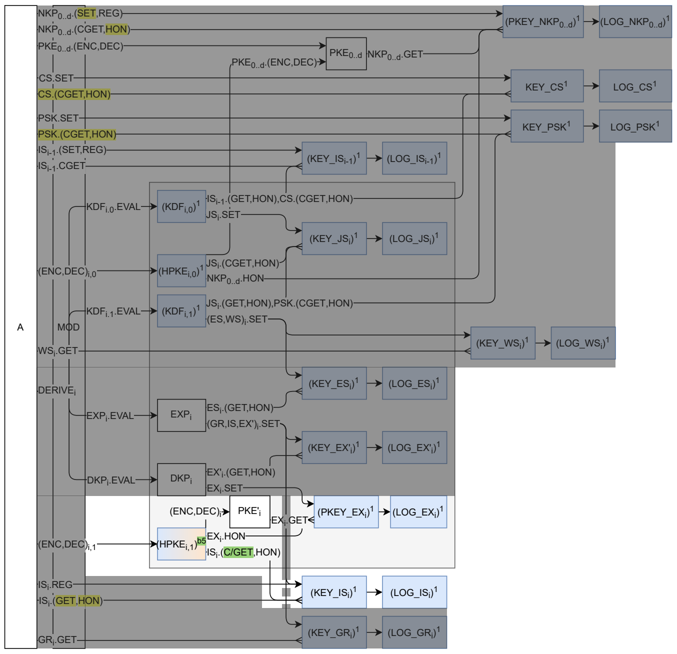


Figure 31: Step 6 of idealizing epoch i. We mark $\mathcal{R}_{e,i}$ in grey.

$$\begin{aligned}\mathsf{Adv}^{\mathsf{xpd},\mathsf{kdf},\mathsf{pke},i}_{\text{EPOCH}}(\mathcal{A}) \leq{} & \mathsf{Adv}^{\mathsf{kdf}}_{\text{KDFR}}(\mathcal{A} \circ \mathcal{R}_{a,i}) + \mathsf{Adv}^{\mathsf{pke}}_{\text{HPKE}}(\mathcal{A} \circ \mathcal{R}_{b,i}) \\ & + \mathsf{Adv}^{\mathsf{kdf}}_{\text{KDFR}}(\mathcal{A} \circ \mathcal{R}_{c,i}) + \mathsf{Adv}^{\mathsf{xpd}}_{\text{EXP}}(\mathcal{A} \circ \mathcal{R}_{d,i}) \\ & + 0 + \mathsf{Adv}^{\mathsf{pke}}_{\text{HPKE}}(\mathcal{A} \circ \mathcal{R}_{e,i}),\end{aligned}$$

and Lemma 3 holds.