Tighter provable security for TreeKEM

Karen Azari[1]* and Andreas Ellison[2]

[1] University of Vienna, Faculty of Computer Science, Vienna, Austria
karen.azari@univie.ac.at
[2] ETH Zurich, Switzerland
andreas.ellison@inf.ethz.ch

Abstract. The Messaging Layer Security (MLS) protocol, recently standardized in RFC 9420, aims to provide efficient asynchronous group key establishment with strong security guarantees. The main component of MLS, which is the source of its key efficiency and security properties, is a protocol called TreeKEM. Given that a major vision for the MLS protocol is for it to become the new standard for messaging applications like WhatsApp, Facebook Messenger, Signal, etc., it has the potential to be used by a huge number of users. Thus, it is important to better understand the security of MLS and hence also of TreeKEM. In previous work, TreeKEM was proven adaptively secure in the Random Oracle Model (ROM) with a polynomial loss in security by proving a result about the security of an arbitrary IND-CPA secure public-key encryption scheme in a public-key version of a security game called GSD. Unfortunately, the concrete security guarantees implied by this line of work are not sufficient for parameters used in practice.

In this work, we prove a tighter bound for the security of TreeKEM when DHIES is used for public-key encryption; DHIES is currently the only standardized scheme in MLS. Our bound implies meaningful security guarantees even for small practical parameters. We follow the above approach and first introduce a modified version of the public-key GSD game better suited for analyzing TreeKEM. We then provide a simple and detailed proof of security for DHIES in this game in the ROM and achieve a smaller security loss compared to the previous best result. Finally, we state the result on the security of TreeKEM implied by this bound and give an interpretation of the result with protocol parameters used in practice.

Keywords: Messaging Layer Security · TreeKEM · Secure Messaging · Group Key-Agreement · Adaptive Security · DHIES

1 Introduction

We all rely on messaging applications like WhatsApp, Facebook Messenger, Signal, etc. in our daily lives and take it for granted that our messages will be

* Part of the work was done while the author was affiliated with ETH Zurich, Switzerland.

transmitted securely, for some definition of “secure”. A common security feature expected from the protocol employed in a messaging application is end-to-end encryption, i.e. that only the end users of a messaging session can read the messages being sent and the service provider or any party with access to the communication channel learns nothing of their contents. The protocol should work in an asynchronous setting: we would like to send messages even when the recipient is offline, and we expect them to receive the message once they come online. For this, we must rely on a delivery service to store and deliver the messages. Of course, also this delivery service should learn nothing from the messages.

There are two more advanced security features expected from messaging protocols today, both related to security in case a user is compromised:

– forward secrecy (FS): the compromise should not reveal the contents of old messages
– post-compromise security (PCS): after the user recovers from the compromise, new messages are secure once again

As a user may well not know that they have been compromised, ensuring PCS requires regularly updating the key material used for encryption (in a way that the information leaked in a compromise *before* the update does not suffice to compute encryption keys used *after* the update). The more often the key material is updated, the stronger the level of PCS that is achieved. Thus, updating the key material should be an efficient operation.

For messaging between two users, the Double Ratchet protocol [16], the main component of the Signal Protocol, is a widely adopted solution used by major messaging applications such as Signal, WhatsApp, Facebook Messenger and more. It is well-studied and achieves all of the above security guarantees [9]. For messaging in a group of more than two users, a straightforward solution is to maintain 1:1 communication channels using the Double Ratchet protocol between every pair of users and send messages to the group by sending them to every member individually. This achieves very strong security guarantees but requires a number of encryption operations linear in the group size to send a message.

Another common solution is to use sender keys [5]: every user creates a symmetric key, their *sender key*, and distributes this sender key to every other user using 1:1 channels as before. A user sending a message then derives a symmetric encryption key for the message from their sender key, while continually updating their sender key (with each sent message) to provide FS. However, achieving PCS is costly: if a user is compromised, the sender keys of all users are leaked and recovering from the compromise requires each user to send a new sender key to every other user over the respective 1:1 channels, resulting in a number of operations linear in the group size per user and a quadratic number of operations in total. Moreover, dynamic group membership introduces additional complexity:

– adding a new member involves the new member sharing their sender key with all other group members

– removing a member requires distributing new sender keys in the group, just like recovering from a compromise

The Messaging Layer Security (MLS) protocol, recently standardized in [6], proposes a solution for group messaging with better efficiency and the same strong security guarantees as for the two-party case. Updating key material and adding or removing members can be achieved with a logarithmic number of operations (although the complexity may still degrade to linear in certain scenarios). At the core of MLS is a fairly recent primitive called a *continuous group key agreement* (CGKA) scheme [2] (this primitive was introduced only *after* the first draft of the MLS protocol). In essence, a CGKA scheme enables a group of users to agree on a *group key*, which they can then use to derive symmetric message encryption keys. This key must be indistinguishable from a random key for anyone outside the group eavesdropping on all communication. However, a CGKA scheme must also achieve FS and PCS, and support dynamic group membership. Hence, it must provide mechanisms for members to update their key material, add new users to the group and remove members from the group. Moreover, the scheme must work in the asynchronous setting with an untrusted service to deliver protocol messages.

The CGKA scheme used in the MLS protocol is called TreeKEM (initially proposed in [8]) and the majority of the literature on MLS is dedicated to analyzing TreeKEM or proposing better CGKA schemes. We refer the reader to [15, Section 1.3] for an in-depth overview of the literature. The TreeKEM protocol has undergone multiple changes since its inception. In this work, we refer to the version documented in RFC 9420. Given that the vision for the MLS protocol is for it to become the new standard for messaging protocols and that it has support from several large companies [10,11], it has the potential to be used by a huge number of users. Thus, understanding the security of MLS and hence also of TreeKEM is of great importance. This means having formal security guarantees about the security provided by TreeKEM (based on appropriate hardness assumptions).

1.1 The TreeKEM protocol

Propose and commit syntax As a CGKA scheme, TreeKEM must support operations for updating the key material of a group member, adding a new user and removing a member. The syntax for these operations has changed over time. In the current version of MLS, the protocol uses so-called *proposals* and *commits*. Whenever a user would like to have their key material updated, add a new user or remove a group member, they create a corresponding *update*, *add* or *remove proposal*, respectively, and share this proposal with the group. Any group member can then create a *commit* to apply a set of proposals, create a new group key and update their key material in the process. The commit object includes (encrypted) information such that every group member can update their view of the group and compute the new group key.

TreeKEM dynamics TreeKEM uses a full binary tree to model the group and every user is associated with a leaf in the tree (the remaining leaves may be left empty). Each user maintains a synchronized view of the tree, though different users will know more about different parts of the tree. The group key is derived from the root of the tree. Every node n in the tree has an associated key pair (pk_n, sk_n) output by Π.Gen where Π is a public-key encryption scheme. All public keys are known to all users. Let the *direct path* of a leaf be the path from the leaf's parent to the root. A leaf's *path to the root* refers to the same path but including the leaf as well. Every user at a leaf knows the secret key of their leaf and, in the usual case, the secret keys of all nodes on their direct path, though there can be exceptions to this rule. To illustrate the scheme and how commit operations are performed, we will consider a group with users $A, B, \ldots, G$ and H, as depicted in Figure 1. In the following, we will use these labels for the users both to refer to the users themselves and to their nodes in the tree.

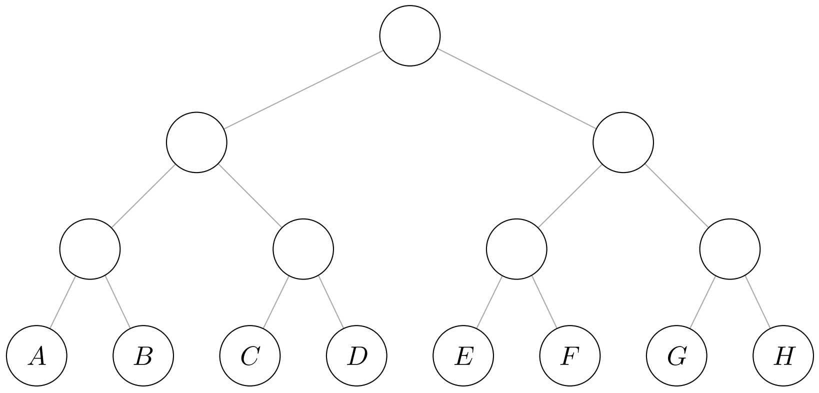

Fig. 1: Illustration of a group with users 8 users in the TreeKEM protocol.

Simple commits The idea behind this tree structure is that it allows for a user creating a commit with a new group key to share the new group key with the group using only a few encryptions, while still updating all the secrets the user knew in the tree in order to recover from a possible compromise (recall that in a CGKA scheme a commit also updates the committer's key material). To illustrate how a commit is performed and how the new group key is computed, say user A performs a commit. Let us consider commits without any proposals. TreeKEM specifies two hash functions $H_{\text{gen}}, H_{\text{dep}} \colon \{0,1\}^{\rho(\eta)} \to \{0,1\}^{\rho(\eta)}$ where $\rho(\eta)$ gives the number of random bits used by $\Pi.\text{Gen}(1^\eta)$. The depth of user A is $d = 3$. A will replace all the $d + 1$ nodes on their path to the root with new nodes $A, p_1, \ldots, p_d$. (In practice A would just replace the information stored in the original nodes.) The key pairs for the new nodes are sampled as follows. For the leaf node A, user A simply samples a key pair by running $\Pi.\mathrm{Gen}(1^\eta)$. For the remaining nodes, they first sample $s_1 \leftarrow \{0,1\}^{\rho(\eta)}$ and compute the key pair of the first parent p_1 as $\Pi.\mathrm{Gen}(1^\eta, H_{\mathrm{gen}}(s_1))$. For $i \in \{2, \dots, d\}$ they then compute $s_i \coloneqq H_{\mathrm{dep}}(s_{i-1})$ and set the key pair of p_i to be $\Pi.\mathrm{Gen}(1^\eta, H_{\mathrm{gen}}(s_i))$. The new group key is $k \coloneqq H_{\mathrm{dep}}(s_d)$.

User A only needs to share (encryptions of) the seeds s_i for the other users to update their view of the tree and compute the new group key:

- To share the group key with user B, A computes the ciphertext $c_1 \leftarrow \Pi.\mathrm{Enc}_{pk_B}(s_1)$. B can then compute the seed s_1, then use that to compute the seeds $s_2, \dots, s_d$, the key pairs of all new nodes on their path to the root and the group key k.
- To share the new group key with users C and D, A computes the ciphertext $c_2 \leftarrow \Pi.\mathrm{Enc}_{pk_X}(s_2)$, where X is the parent of the nodes C and D. Both C and D know the secret key sk_X of their parent and can decrypt c_2.
- To share the new group key with users E, F, G and H, A computes the ciphertext $c_3 \leftarrow \Pi.\mathrm{Enc}_{pk_Y}(s_3)$, where Y is the right child of the root node. Again, all users under Y know sk_Y and can thus decrypt c_3.

The commit c that A shares with all users includes the ciphertexts c_1, c_2 and c_3 and the public keys of all new nodes. Figure 2 illustrates the commit performed by A.

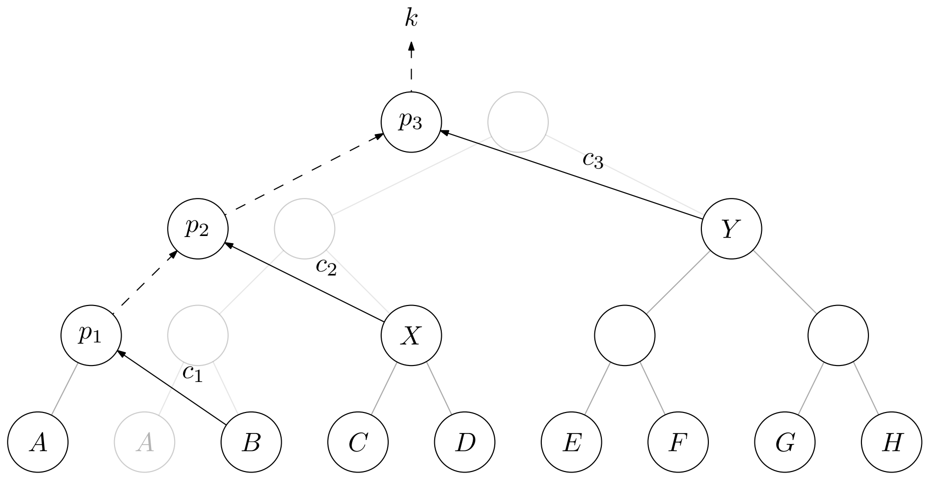

Fig. 2: The commit by user A described in the text. Dashed directed edges illustrate the fact that the target is related to the source via the hash function H_{dep}. The solid directed edges illustrate the fact that the seed of the target node is encrypted to the public key of the source node.

The nodes B, X and Y form the *copath* of A: the copath of a node v consists of the sibling of each node on v's path to the root, excluding the root itself. In the ideal case as above, a node performing a commit only has to compute one encryption for each node on its copath, i.e. logarithmically many encryptions in the total number of users.

The above concepts suffice to understand our main results. For a discussion of how proposals are incorporated into a commit, we refer the reader to Section C.1 of the appendix in the full version [4].

1.2 The GSD game

The Generalized Selective Decryption (GSD) security game [14] was introduced precisely to analyze adaptive security for protocols based on a graph-like structure, as is the case with TreeKEM. It was initially defined for the private-key setting and later adapted to the public-key setting in [13].

In the GSD security game, given an encryption scheme a graph, the *GSD graph*, is constructed by the challenger where every node in the graph is associated with a symmetric key in the private-key setting, or a public/private key pair in the public-key setting. The adversary can then request encryptions of a node's (secret) key under the (public) key of another node. In the public-key setting, such an *encryption query* also reveals the latter node's public key. This creates an *encryption edge* in the graph, directed from the node whose (public) key was used for encryption to the node whose key was encrypted. The adversary can also corrupt any node, which reveals its (secret) key and allows the adversary to compute the (secret) key of any other node reachable from the corrupted node in the graph by performing decryptions along the path to the other node. At the end of the game the adversary chooses a node to be challenged on, the *challenge node*. A coin is then tossed and the adversary is given either the (secret) key of the challenge node or a uniformly random (secret) key and it must guess which scenario it is in. The possible choices for the challenge node must of course be restricted to nodes whose keys were not compromised through a corruption, meaning that the challenge node should never be reachable from a corrupted node in the graph. Further restrictions are also necessary which we do not go into here and refer the reader to Section 4.1. Figure 3 illustrates what an example GSD graph may look like.

The graph constructed in the public-key GSD game and the tree structure behind the TreeKEM protocol clearly resemble each other. Let $\mathit{\Pi}$ the public-key encryption scheme in use, where $\mathit{\Pi}.\mathrm{Gen}(1^\eta)$ samples $\rho(\eta)$ random bits. We can make some small modifications to the public-key GSD game such that the operations performed in TreeKEM match the ones performed in this modified GSD game. Take the functions $H_{\mathrm{gen}}, H_{\mathrm{dep}}$ used in TreeKEM and first modify the game as follows:

- the key pair of a node v is generated by sampling a seed $s_v \in \{0,1\}^{\rho(\eta)}$ and computing $(pk_v, sk_v) = \mathit{\Pi}.\mathrm{Gen}(1^\eta, H_{\mathrm{gen}}(s_v))$
- encryption queries encrypt the seed of the target node instead of its secret key

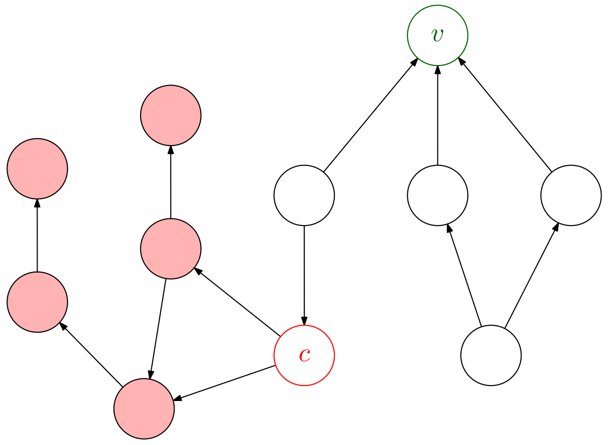

Fig. 3: An illustration of the GSD graph for an instance of the GSD game. The challenge node is v. The node c was corrupted, resulting in all nodes reachable from it being compromised, as marked with red color.

Now the generation of key pairs and the encryptions computed in TreeKEM match what is done in this adapted GSD game. To model the fact that in TreeKEM the seed of a node may depend on the seed of another node through H_{dep} (as in the new direct path computed in a commit), we can introduce a new type of edge which we call a *seed dependency*: a seed dependency (u, v) implies that $s_v = H_{\text{dep}}(s_u)$.

1.3 Existing security proofs

The work in [13] made use of the relationship between the (public-key) GSD game and TreeKEM described in the previous section. They proved a polynomial bound for the adaptive security of the public-key GSD game in the Random Oracle Model (ROM) [7] for an arbitrary IND-CPA secure public-key encryption scheme. The core idea behind their proof is simple. Intuitively, the only way for an adversary to learn about the seed of an uncorrupted node for the first time is through the encryption edges into the node. The authors show that in the ROM, one can reduce security in the GSD game to the IND-CPA security of the public-key encryption scheme with a moderate security loss. They apply standard guessing arguments and a clever, albeit complicated, hybrid argument to achieve a security loss in $\mathcal{O}(N^2)$, where N is the number of nodes in the GSD graph. The N^2 loss stems from the fact that two nodes in the GSD graph need to be guessed correctly for the reduction to succeed.

This result implies a polynomial bound for the security of TreeKEM as a CGKA scheme as outlined in [13, Theorem 4] and subsequently proven in more detail in [3, Theorem 12], and was the first proof of adaptive security for TreeKEM.

As far as we know, this is the tightest result on the security of TreeKEM in the literature.

1.4 Security model

We prove adaptive CGKA security of TreeKEM for a single group and a single challenge query when the DHIES scheme is used. Additionally, our result easily extends to adversaries performing multiple challenge queries with the same security loss (irrespective of the number of challenges). Our security definition is very close to the definition provided in [3, Section 4.1.2], but gives the adversary less power. We provide a weaker definition in order to simplify the definition and proof. However, we are confident that the exact same result can be proven for the CGKA security definition in [3]. The additional power given to the adversary in [3] should only affect the set of commits that the adversary can ask to be challenged on in the CGKA security game (defined in [4, Section C.2]).

1.5 Contributions

Tighter GSD security for DHIES In this work, we formally prove the adaptive security of a specific public-key encryption scheme, the DHIES scheme [1], in the adapted public-key GSD game described above, in the ROM. Focusing on the DHIES scheme allows us to achieve a tighter bound than the one in [13]. Each encryption of a seed s_v with DHIES is a tuple of the form $\langle g^x, c\rangle$ with $c \leftarrow \Pi_s.\mathrm{Enc}_k(s_v)$, where $k = H_{\mathrm{DH}}(g^{xy})$, g is a generator in the Diffie-Hellman group, Π_s is the private-key encryption scheme used in DHIES, H_{DH} is a hash function and y is the secret key of the source node of the encryption edge. By modelling H_{DH} as a random oracle, we know that in order for an adversary to learn anything from this encryption, intuitively, they must have either learned k by querying H_{DH} and then decrypting the ciphertext c, or they must have gleaned information from c without knowing the key k. We can reduce the first case to the hardness of the Diffie-Hellman problem with a security loss in $\mathcal{O}(N)$. To achieve a linear loss, we first guess the node v for which the Diffie-Hellman problem was solved on an outgoing encryption edge and exploit the self-reducibility property of the Diffie-Hellman problem in order to embed a single Diffie-Hellman challenge into all outgoing edges. The second case can be reduced to the security of the private-key encryption scheme with a security loss in $\mathcal{O}(\delta \cdot N)$ where δ is the maximum in-degree in the GSD graph. This follows from guessing the target node s_v and then applying a standard hybrid argument to the up to δ encryption edges ending in v.[3] In [4, Section B.4] we show that the security loss can be reduced to $\mathcal{O}(N)$ for a very specific set of schemes.

Besides providing a tighter bound, our proof has further significant advantages. Arguably, it is more intuitive and less complex than the ones in [3,13]. Furthermore, our approach can easily be adapted to prove the same security loss in a GSD

[3] The same hybrid argument does not work for directly reducing to the security of the public-key encryption scheme due to a technicality.

game with multiple challenge queries. Both of these advantages stem from how we compute the challenge in our GSD definition. See Section 4.1 for details.

The security of TreeKEM Our final result for TreeKEM has a security loss in $\mathcal{O}(u \cdot (c \cdot \log(u) + p))$ reducing to EAV security[4] of the private-key encryption scheme and a loss in $\mathcal{O}(c \cdot \log(u) + p)$ reducing to the hardness of the Diffie-Hellman problem, where c is the number of commits, p the number of add or update proposals and u the number of users. As with GSD, our result holds irrespective of the number of challenge queries. The result in [13] implies a security loss of $\mathcal{O}((c \cdot \log(u) + p)^2)$ for a single challenge query, reducing to IND-CPA security of the public-key encryption scheme. This can be generalized to multiple challenges by a hybrid argument, involving an additional multiplicative loss in the number of challenges. When p is small such that $p \leq c \cdot \log(u)$, the losses simplify to $\mathcal{O}(c \cdot u \cdot \log(u))$ and $\mathcal{O}(c \cdot \log(u))$, respectively, as opposed to $\mathcal{O}((c \cdot \log(u))^2)$ in [13]. In this setting, our result guarantees 90 bits of security with 128-bit parameters and 210 bits with 256-bit parameters for large groups (under standard assumptions). When p is large, e.g. when many updates are applied in each commit, the losses simplify to $\mathcal{O}(c \cdot u^2)$ and $\mathcal{O}(c \cdot u)$ as opposed to $\mathcal{O}((c \cdot u)^2)$ in [13]. Our result guarantees 95 bits of security when a 128-bit Diffie-Hellman group is combined with 256-bit AES, while the result in [13] only guarantees 64 bits. Using 256-bit parameters gives a 209-bit security level.

1.6 Relation to MLS security

The work in [3] provides a comprehensive security definition for group messaging protocols and reduces the security of MLS to the security of its underlying primitives, including the CGKA scheme. Since our proof can be adapted to go through with the CGKA definition used in this work, our result implies a tighter result for the security of MLS as a whole. However, the security proof of MLS in [3] has a very large (but polynomial) loss and even with our improved bound for TreeKEM we don't expect this to provide a meaningful level of security. Therefore, finding a tighter reduction for the MLS protocol as a whole is a crucial open problem on the path to proving meaningful security guarantees for the protocol.

2 Preliminaries

Our definitions were adapted from [12]. A summary of our notation, some simple lemmas, definitions for private-key and public-key encryption, IND-CPA security, EAV security, group-generation algorithms and the Decisional Diffie-Hellman (DDH) problem, and an explanation of the Random Oracle Model (ROM) can be found in [4, Section A].

[4] EAV security is implied by IND-CPA security. See [4, Section A.3] for details.

2.1 DHIES

In the following definition we will refer to "key-derivation functions". This is only meant as a hint to the reader. We do not provide a definition here, as we will always model such a function as a random oracle.

Definition 1 (DHIES [12, Construction 12.19]). *Let $\mathcal{G}$ a group-generation algorithm (for Diffie-Hellman groups). Let Π_s a private-key encryption scheme where $\Pi_s.\mathrm{Gen}(1^\eta)$ samples a key u.a.r. from $\{0,1\}^\eta$. Let $\mathcal{H}_{\mathrm{DH}} = \{H_{\mathrm{DH}}^{(\eta)} \mid \eta \in \mathbb{N}\}$ a family of key-derivation functions where $H_{\mathrm{DH}}^{(\eta)} \colon \{0,1\}^* \to \{0,1\}^\eta$. We write $H_{\mathrm{DH}} \coloneqq H_{\mathrm{DH}}^{(\eta)}$ when η is clear from the context. Define the algorithms* Gen, Enc *and* Dec *as follows:*

- Gen*: on input 1^η run $\mathcal{G}(1^\eta)$ to obtain $(\mathbb{G}, q, g)$. Sample $x \leftarrow [q]$ and set $h_1 \coloneqq g^x$. Set $pk \coloneqq \langle \mathbb{G}, q, g, h_1 \rangle$ and $sk \coloneqq \langle \mathbb{G}, q, g, x \rangle$, and output (pk, sk). The message space is the message space of Π_s.*
- Enc*: on input a public key $\langle \mathbb{G}, q, g, h_1 \rangle$ and a message m, sample $y \leftarrow [q]$, set $h_2 \coloneqq g^y, k \coloneqq H_{\mathrm{DH}}(h_1^y)$,[5] compute $c' \leftarrow \Pi_s.\mathrm{Enc}_k(m)$ and output the ciphertext $\langle h_2, c' \rangle$.*
- Dec*: on input a private key $\langle \mathbb{G}, q, g, x \rangle$ and a ciphertext $\langle h_2, c' \rangle$, compute $k \coloneqq H(h_2^x)$ and output $\Pi_s.\mathrm{Dec}_k(c')$. If the ciphertext is not of the right form or Π_s.*Dec *outputs $\bot$, output $\bot$.*

The public-key encryption scheme $\Pi_{\mathrm{DH}} \coloneqq (\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec})$ is called the Diffie-Hellman Integrated Encryption Scheme (DHIES).

When using the DHIES scheme later on, we will set $pk \coloneqq h$ and $sk \coloneqq x$ in Gen *for simplicity. In practice $\mathbb{G}, q, g$ and H_{DH} will be known.*

Under the DDH assumption (i.e. the assumption that the DDH problem is hard relative to $\mathcal{G}$), using DHIES with an EAV secure private-key scheme gives an IND-CPA secure public-key encryption scheme in the ROM, as proven in [12, Theorem 12.12].

3 Our concrete result for TreeKEM

The following theorem describing the security of TreeKEM, stated informally here, is our main practical result. It bounds the advantage of any adversary creating c commits and p add or update proposals in a group with at most u users in distinguishing the group key of any uncompromised commit from a random bit string.

Theorem 1 (Informal). *If the DHIES scheme is used in TreeKEM, the private-key encryption scheme in DHIES is $(t, \varepsilon_{\mathrm{EAV}})$-EAV-secure and the DDH problem*

[5] Where for $h \in \mathbb{G}$, $H_{\mathrm{DH}}(h)$ denotes the output of H_{DH} with the binary representation of h given as input.

is $(t, \varepsilon_{\mathrm{DDH}})$*-hard in the Diffie-Hellman group, then for all* c, p, u*, TreeKEM is* $(\tilde{t}, \tilde{\varepsilon}, c, p, u)$*-secure in the ROM with* $\tilde{t} \approx t$ *and*

$$\begin{aligned}\tilde{\varepsilon} = {} & 2 \cdot u \cdot (3 \cdot c \cdot \log(u) + p) \cdot \varepsilon_{\mathrm{EAV}} \\ & + 2 \cdot (3 \cdot c \cdot \log(u) + p) \cdot \varepsilon_{\mathrm{DDH}} \\ & + \mathrm{negl}\end{aligned}$$

where

- *c is the number of commits created*
- *p is the number of add or update proposals created*
- *u is the maximum number of users*
- *the term* $\mathrm{negl} = \mathrm{negl}(\eta)$ *is negligible and much smaller than the other terms*

In [4, Section C] we restate the above theorem formally as Theorem 4. To this end, we also provide formal definitions for the syntax and security of propose and commit CGKA schemes and a high-level description of how to instantiate (the essence of) the TreeKEM protocol with our definitions.

Security against multiple challenges As noted in the introduction, although our security definitions only allow the adversary to make a single challenge query, it is straightforward to adapt our proof to show the same security loss with multiple challenge queries. See the note after [4, Theorem 4] for details.

3.1 Interpreting the result

In the following, we will go through some concrete examples to see what level of security our proof guarantees for TreeKEM with different parameter choices. We will look at the `MLS_128_DHKEMX25519_AES128GCM_SHA256_Ed25519` cipher suite [6, Section 17.1] for 128-bit parameters, which uses Curve25519 as the Diffie-Hellman group and AES with a 128-bit key size for private-key encryption. We will assume that both Curve25519 and AES have a 128-bit security level and we will set $(t, \varepsilon_{\mathrm{EAV}}) = (t, \varepsilon_{\mathrm{DDH}}) = (2^{48}, 2^{-80})$.

For 256-bit parameters, we will look at the `MLS_256_DHKEMP521_AES256GCM_SHA512_P521` cipher suite, which uses curve P-521 and 256-bit AES. We will assume that P-521 and 256-bit AES have a 256-bit security level and set $(t, \varepsilon_{\mathrm{EAV}}) = (t, \varepsilon_{\mathrm{DDH}}) = (2^{128}, 2^{-128})$.

Large groups with hourly commits and frequent updates In this example we consider a large group of about 10'000 users, existing for 5 years and making one commit every hour. Then $u \leq 2^{14}$ and $c \leq 2^{16}$. We also assume that a significant fraction of the users will want to update with every commit. Then, assuming that add proposals are relatively rare, we can bound $p \leq c \cdot u = 2^{30}$. This implies $3 \cdot c \cdot \log(u) + p \leq 2^{31}$, dominated by p.

Then with 128-bit parameters we get

$$\tilde{\varepsilon} \leq 2^{46} \cdot \varepsilon_{\text{EAV}} + 2^{32} \cdot \varepsilon_{\text{DDH}} + \text{negl} \leq \frac{1}{2^{33}}$$

with the ε_{EAV} term dominating the result. This only gives a security level of $\tilde{t}/\tilde{\varepsilon} \geq 2^{81}$. Since private-key encryption is relatively cheap, using 256-bit AES would have a small impact on the performance and would increase the security level to 95 bits (with the ε_{DDH} term now dominating).[6] Finally, using full 256-bit parameters yields 209 bits of security for TreeKEM.

The previous best result in [13, Theorem 3] proved the bound

$$\tilde{\varepsilon} \leq 2 \cdot (3 \cdot c \cdot \log(u) + p)^2 \cdot \varepsilon + \text{negl}$$

where ε is the IND-CPA security of the underlying public-key encryption scheme. If we assume that DHIES has an x-bit security level as a public-key encryption scheme with x-bit parameters, the result implies 64 bits of security with 128-bit parameters (with no change when using 256-bit AES) and 192 bits with 256-bit parameters.

Few updates In this example we use the same number of users and commits, but assume that the number of proposals is small such that $p \leq c \cdot \log(u)$. In this case, we have $3 \cdot c \cdot \log(u) + p \leq 2^{22}$. Then our result guarantees 90 bits of security with 128-bit parameters, 104 bits with 256-bit AES and 218 bits with 256-bit parameters.

In contrast, the bound in [13] implies 82 bits with 128-bit parameters and 210 bits with 256-bit parameters.

Very large groups with one commit every minute and frequent updates In this example we consider more extreme values for c and u to highlight the gap between our result and the one in [13]. We assume a group of about 1 million users, existing for 50 years and making one commit every minute. Furthermore, we will again use $p \leq c \cdot u$. This means that $u \leq 2^{20}$, $c \leq 2^{25}$ and $3 \cdot c \cdot \log(u) + p \leq 2^{46}$.

These values imply 60 bits of security with 128-bit parameters, 80 bits with 256-bit AES and 188 bits with 256-bit parameters using our result. The result in [13] implies 34 bits of security with 128-bit parameters and 162 bits with 256-bit parameters.

4 Tighter GSD security

In this section, we present the main theoretical result behind Theorem 1 in Theorem 2. To this end, we first define a modified version of the public-key GSD game from [13] and then proceed to prove Theorem 2 in detail.

[6] For this we set $(t, \varepsilon_{\text{EAV}}) = (2^{48}, 2^{-208})$.

4.1 Seeded GSD with Dependencies

The GSD game defined here is inspired by the definition of the public-key GSD game (Definition 7) and the proof of Theorem 3 in [13]. We have already motivated the differences in Section 1.2. We will call our adapted game *Seeded GSD with Dependencies* (SD-GSD). A very similar definition appears in [3], providing essentially the same abstraction over TreeKEM and also allowing for an adversary to provide the randomness used for encryption and key generation. However, our definition has one key difference: when asking to be challenged on a node with seed s, the adversary must distinguish $H_{\text{dep}}(s)$ from random as opposed to s. This stays true to how the group key is computed in TreeKEM and provides two significant advantages:

- it allow us to significantly simplify our proof
- it enables one to easily prove the same bound when the adversary can make multiple challenge queries

On the other hand, the security implied by our definition is weaker (at least in the ROM), as it only guarantees that an adversary cannot compute the seed of the challenge node, whereas the other definitions guarantee that this seed cannot be distinguished from random.

Definition 2 (The SD-GSD game). *Let* $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$ *a public-key encryption scheme, where* $\text{Gen}(1^\eta)$ *uses* $\rho(\eta)$ *random bits and* $\{0,1\}^{\rho(\eta)}$ *is a subset of the message space. Let* $\mathcal{H}_{\text{gen}} = \{H_{\text{gen}}^{(\eta)} \mid \eta \in \mathbb{N}\}, \mathcal{H}_{\text{dep}} = \{H_{\text{DH}}^{(\eta)} \mid \eta \in \mathbb{N}\}$ *families of functions with* $H_{\text{gen}}^{(\eta)}, H_{\text{dep}}^{(\eta)} \colon \{0,1\}^{\rho(\eta)} \to \{0,1\}^{\rho(\eta)}$. *We will write* $H_{\text{gen}} := H_{\text{gen}}^{(\eta)}, H_{\text{dep}} := H_{\text{dep}}^{(\eta)}$ *and* $\rho := \rho(\eta)$ *if* η *is clear from the context. Define the game* $\text{Game}^{\text{SD-GSD}}_{(\Pi,\mathcal{H}_{\text{gen}},\mathcal{H}_{\text{dep}}),\eta}(\mathcal{A})$ *for an adversary* $\mathcal{A}$*:*

1. *The adversary* $\mathcal{A}$ *outputs* $n \in \mathbb{N}$ *and a list of dependencies* $D = \{(a_i, b_i)\}_{i=1}^m \subseteq [n]^2$. *For each* $v \in [n]$*:*
   (i) – ***Case*** $v = b_i$ ***for some*** i ***(***v ***is the target of some dependency):*** *set* $s_v = H_{\text{dep}}(s_{a_i})$.
   – ***Otherwise:*** *sample* $s_v \leftarrow \{0,1\}^\rho$.
   We call s_v *the* seed *of the node* v *and a tuple* $(a, b) \in D$ *a* seed dependency.
   (ii) Compute $(pk_v, sk_v) = \text{Gen}(1^\eta, H_{\text{gen}}(s_v))$.
   Set $\mathcal{C} = E = \varnothing$. *We call the directed graph* $([n], E)$ *a* GSD graph *of* size n.
2. $\mathcal{A}$ *may adaptively make the following queries:*
   - reveal(v) *for* $v \in [n]$*:* $\mathcal{A}$ *is given* pk_v.
   - encrypt(u, v) *for* $u, v \in [n], u \neq v, (u, v) \notin E$*:* (u, v) *is added to* E *and* $\mathcal{A}$ *is given* $c \leftarrow \text{Enc}_{pk_u}(s_v)$.
   - corrupt(v) *for* $v \in [n], v \notin \mathcal{C}$*:* $\mathcal{A}$ *is given* s_v *and* v *is added to* $\mathcal{C}$. *We call such a node* $v \in \mathcal{C}$ corrupted. *All nodes not reachable from any corrupted node in the graph* $([n], E \cup D)$ *are* safe *(while all other nodes are* unsafe*) and we call their seeds* hidden *(even if an unsafe node happens to have the same seed).*

3. *$\mathcal{A}$ outputs a node $v \in [n]$. We call v the* challenge node. *A bit $b \leftarrow \{0,1\}$ is sampled and $\mathcal{A}$ is given*

$$r = \begin{cases} H_{\text{dep}}(s_v) & b = 0 \\ s & b = 1 \end{cases},$$

where $s \leftarrow \{0,1\}^\rho$.[7] *$\mathcal{A}$ may continue to do queries as before.*
4. *$\mathcal{A}$ outputs a bit b'. The output of the game is defined to be 1 if $b' = b$, and 0 otherwise.*

We require an adversary playing the above game to adhere to the following:

– *The challenge node is safe*
– *The challenge node is not the source of a seed dependency*[8]
– *Every node is the source and target of at most one seed dependency*[9]
– *The graph $([n], E \cup D)$ is acyclic and without self-loops*

It is interesting to note that previous definitions of the GSD game also included the following restrictions to the adversary:

– The challenge node always remains a sink in $([n], E)$
– reveal is never queried on the challenge node

Our proof in the ROM does not require these restrictions. If H_{dep} is modelled as a random oracle, then an encryption edge outgoing from the challenge node, or knowing its public key gives no advantage to $\mathcal{A}$, as by the assumption of H_{dep} being a random oracle the only way to learn information about $H_{\text{dep}}(s)$ is by querying s.

Definition 3 (SD-GSD security). *Let $\Pi, \mathcal{H}_{\text{gen}}$ and $\mathcal{H}_{\text{dep}}$ as in Definition 2 above and let $t, \varepsilon, N, \delta$ functions in η. The triple $(\Pi, H_{\text{gen}}, H_{\text{dep}})$ is $(t, \varepsilon, N, \delta)$-*SD-GSD-secure *if for all η, for any adversary $\mathcal{A}$ constructing a GSD graph of size at most $N(\eta)$ and indegree at most $\delta(\eta)$ and running in time $t(\eta)$ we have*

$$\text{Adv}^{\text{SD-GSD}}_{(\Pi,\mathcal{H}_{\text{gen}},\mathcal{H}_{\text{dep}}),\eta}(\mathcal{A}) := 2 \cdot \left(\Pr\left[\text{Game}^{\text{SD-GSD}}_{(\Pi,\mathcal{H}_{\text{gen}},\mathcal{H}_{\text{dep}}),\eta}(\mathcal{A}) = 1\right] - \frac{1}{2} \right) \leq \varepsilon(\eta).$$

Since in this work we are interested in SD-GSD security for the case where H_{gen} and H_{dep} are modelled as random oracles and our focus is on the encryption scheme being used, we introduce the following definition for convenience.

[7] Note that (in general) r is not a hidden seed, as (with overwhelming probability) it is not the seed of any node.

[8] Otherwise the adversary could learn the value of H_{dep} on the seed of the challenge node by creating a seed dependency with the challenge node as the source and corrupting the target node.

[9] If a node were the source node of multiple seed dependencies, then corrupting one target node would reveal the seeds of all target nodes. Additionally, the computation of seeds is not well-defined if a node is the target of multiple dependencies.

Definition 4 (SD-GSD security in the ROM). *A public-key encryption scheme Π is $(t, \varepsilon, N, \delta)$*-SD-GSD-secure in the ROM *if the triple $(\Pi, \mathcal{H}_{\text{gen}}, \mathcal{H}_{\text{dep}})$ is $(t, \varepsilon, N, \delta)$-SD-GSD-secure when H_{gen} and H_{dep} are modelled as random oracles. For security parameter η and an adversary $\mathcal{A}$, we write $\text{Game}^{\text{SD-GSD}}_{\Pi,\eta}(\mathcal{A})$ to denote the game where H_{gen} and H_{dep} are modelled as random oracles and $\text{Adv}^{\text{SD-GSD}}_{\Pi,\eta}(\mathcal{A})$ for $\mathcal{A}$'s advantage in this game.*

4.2 Proving SD-GSD security for DHIES in the ROM

Theorem 2. *Let Π_{DH} denote the DHIES scheme instantiated with a group-generation algorithm $\mathcal{G}$ and a private-key encryption scheme Π_s. If Π_s is $(t, \varepsilon_{\text{EAV}})$-EAV-secure, the DDH problem is $(t, \varepsilon_{\text{DDH}})$-hard relative to $\mathcal{G}$ and the function H_{DH} in Π_{DH} is modelled as a random oracle, then for any δ, N with $\delta \leq N$, Π_{DH} is $(\tilde{t}, \tilde{\varepsilon}, N, \delta)$-SD-GSD-secure in the ROM with*[10]

$$\tilde{\varepsilon} = 2 \cdot \delta \cdot N \cdot \varepsilon_{\text{EAV}} + 2 \cdot N \cdot \varepsilon_{\text{DDH}} + \frac{2 \cdot m_{\text{DH}} \cdot N^2}{q} + \frac{m_{\text{s}} \cdot N}{2^{\rho-1}} + \frac{N^2}{2^{\rho-3}},$$

*where m_{s} is an upper bound on the number of queries made to either H_{gen} or H_{dep}, m_{DH} is an upper bound on the number of queries made to H_{DH}, q is a lower bound on the size of the group output by $\mathcal{G}$ and ρ is the number of random bits sampled by Π_{DH}.*Gen, *and with $\tilde{t} \approx t$.*[11]

In contrast, the result in [13] achieves a security loss in $\mathcal{O}(N^2)$ and reduces to the IND-CPA security of the public-key encryption scheme.

For ease of exposition, we will assume that $\mathcal{G}(1^\eta)$ is deterministic, as is the case in practice. We will therefore set $pk := h_1, sk := x$ in Π_{DH}.Gen, as $\mathbb{G}, q, g$ are implied by η. The results nevertheless hold also for the general case.

Intuition Consider an arbitrary SD-GSD adversary $\mathcal{A}$. For an execution of $\text{Game}^{\text{SD-GSD}}_{\Pi_{\text{DH}},\eta}(\mathcal{A})$ we say "*$\mathcal{A}$ wins*" to denote the event $\text{Game}^{\text{SD-GSD}}_{\Pi_{\text{DH}},\eta}(\mathcal{A}) = 1$. For one argument, we will need to assume that all seeds sampled in the SD-GSD game are distinct, which happens with overwhelming probability. To this end, let C denote the event that there are two nodes with the same seed.

As usual with random oracles we proceed by a case distinction on whether they were queried on some interesting value. Accordingly, let Q_x denote the event

[10] Note that in the following equality we have omitted writing the argument η to the various functions and are implying that the equality holds for all η.

[11] We provide a more precise expression of the runtime in [4, Section B.1].

that $\mathcal{A}$ queries H_x on a hidden seed for $x \in \{\text{gen}, \text{dep}\}$. Then we can write

$$\begin{aligned}
\Pr\big[\mathcal{A} \text{ wins} \mid \overline{C}\,\big] &= \Pr\big[\mathcal{A} \text{ wins} \wedge Q_{\text{dep}} \mid \overline{C}\,\big] + \Pr\big[\mathcal{A} \text{ wins} \wedge \overline{Q_{\text{dep}}} \mid \overline{C}\,\big] \\
&\leq \Pr\big[\mathcal{A} \text{ wins} \wedge Q_{\text{dep}} \mid \overline{C}\,\big] + \Pr\big[\mathcal{A} \text{ wins} \mid \overline{C}, \overline{Q_{\text{dep}}}\,\big] \\
&\overset{(\dagger)}{=} \Pr\big[\mathcal{A} \text{ wins} \wedge Q_{\text{dep}} \mid \overline{C}\,\big] + \frac{1}{2} \\
&\leq \Pr\big[Q_{\text{s}} \mid \overline{C}\,\big] + \frac{1}{2} \\
&\overset{(\spadesuit)}{\leq} \Pr[Q_{\text{s}}] + \Pr[C] + \frac{1}{2},
\end{aligned} \tag{1}$$

where $Q_{\text{s}} \coloneqq Q_{\text{gen}} \cup Q_{\text{dep}}$ (s for *seed*). Step (†) intuitively holds because without having queried H_{dep} for any hidden seed, in particular the seed s_v of the challenge node v, $H_{\text{dep}}(s_v)$ is a uniformly random value from $\mathcal{A}$'s perspective. Therefore, it can do no better than guessing to distinguish $H_{\text{dep}}(s_v)$ from $s \leftarrow \{0,1\}^\rho$. Step ($\spadesuit$) follows from simple manipulations (see the proof for details).

The heart of the proof is to bound $\Pr[Q_{\text{s}}]$. When the adversary first triggers Q_{s} by querying the seed of some safe node w, it can only have learned the seed through encryptions $c_1 \leftarrow \Pi_{\text{DH}}.\text{Enc}_{pk_{u_1}}(s_w), \ldots, c_d \leftarrow \Pi_{\text{DH}}.\text{Enc}_{pk_{u_d}}(s_w)$ where $(u_1, w), \ldots, (u_d, w)$ are edges in the GSD graph (obtained through corresponding queries $\text{encrypt}(u_1, w), \ldots, \text{encrypt}(u_d, w)$). The only other potential source of information about s_w would be a seed dependency (p, w), but this tells $\mathcal{A}$ nothing: Since w is safe, p would also be safe and $H_{\text{dep}}(s_p)$ cannot have been queried due to the assumption that w was the first node to trigger Q_{s}. Without having queried $H_{\text{dep}}(s_p)$, by virtue of H_{dep} being a random oracle s_w has the same distribution as a seed without a dependency from $\mathcal{A}$'s perspective (uniformly random). See Figure 4 for an illustration of node w in the GSD graph.

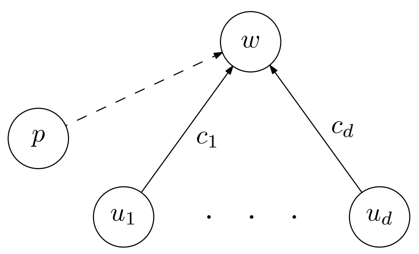

Fig. 4: Illustration of the GSD graph when Q_{s} is triggered at a node w. The dashed edge represents a seed dependency (p, w) and the remaining edges represent encryption queries $c_i \leftarrow \text{encrypt}(u_i, w)$.

The proof in [13] simply argued that this is not too likely if these encryptions were made with an IND-CPA secure scheme. In the context of the DHIES scheme we can say more about these encryptions and achieve a better reduction loss. Let

$(\mathbb{G}, q, g) = \mathcal{G}(1^\eta)$. Let $x_i = \log_g(pk_{u_i})$. Each encryption c_i is a tuple of the form $\langle g^{y_i}, \Pi_s.\mathrm{Enc}_{k_i}(s_w)\rangle$ where $y_i \leftarrow [q]$ and $k_i = H_{\mathrm{DH}}\,(g^{x_i \cdot y_i})$. Now we can again do a case distinction on whether H_{DH} was queried for (the encoding of) some group element $g^{x_j \cdot y_j}$ or not:

(i) If such a query was made, then $\mathcal{A}$ solved the Diffie-Hellman challenge (g^{x_j}, g^{y_j}). (Remember that we assumed that w is the first node for which Q_{s} is triggered and as before if w is safe, then so are the nodes u_i. Thus the adversary has not learned the exponent x_i through querying $H_{\mathrm{gen}}(s_{u_i})$ for any i.)
(ii) If no such query was made, then from $\mathcal{A}$'s perspective all the k_i are independent, uniformly random keys and it still was able to learn s_w from the EAV secure encryptions $\Pi_s.\mathrm{Enc}_{k_1}(s_w), \ldots, \Pi_s.\mathrm{Enc}_{k_d}(s_w)$.

We can bound the probability of either of these events occurring using the hardness of the DDH problem relative to $\mathcal{G}$ and EAV security of Π_s, respectively.

To this end, we call a group element $h \in \mathbb{G}$ a *hidden Diffie-Hellman key* if $h = pk_u^{y_{u,v}}$, where (u, v) is an edge in the GSD graph, u is safe and $y_{u,v}$ is the exponent chosen in the DHIES encryption of s_v (i.e. $\mathcal{A}$ was given a ciphertext of the form $\langle g^{y_{u,v}}, \ldots\rangle$ when it queried encrypt(u, v)). Now analogously to above let Q_{DH} the event that $\mathcal{A}$ queries H_{DH} on a hidden Diffie-Hellman key, and let F_{DH} the event that $\mathcal{A}$ triggers Q_{DH} when Q_{s} has not (yet) been triggered. Then we can split the event Q_{s} into two cases as motivated above:

$$\Pr[Q_{\mathrm{s}}] = \Pr[Q_{\mathrm{s}} \wedge F_{\mathrm{DH}}] + \Pr\big[Q_{\mathrm{s}} \wedge \overline{F_{\mathrm{DH}}}\,\big].$$

We bound $\Pr[Q_{\mathrm{s}} \wedge F_{\mathrm{DH}}]$ and $\Pr\big[Q_{\mathrm{s}} \wedge \overline{F_{\mathrm{DH}}}\,\big]$ in Lemma 3 and Lemma 2, respectively.[12] Overall this gives us a bound on the advantage of $\mathcal{A}$ using (1).

Proof (of Theorem 2). Let δ, N functions in η (mapping to $\mathbb{N}$) with $\delta \leq N$. Let η arbitrary and let $\mathcal{A}$ an arbitrary SD-GSD adversary constructing a GSD graph of size at most $N(\eta)$ and indegree at most $\delta(\eta)$, making at most $m_{\mathrm{s}}(\eta)$ queries to H_{gen} or H_{dep} and at most $m_{\mathrm{DH}}(\eta)$ queries to H_{DH}, each of length at most $\gamma(\eta)$, and running in time $\tilde{t}(\eta)$. We will use the events defined above.

We have

$$\begin{aligned}\Pr[\mathcal{A} \text{ wins}] &= \Pr[\mathcal{A} \text{ wins} \wedge C] + \Pr\big[\mathcal{A} \text{ wins} \wedge \overline{C}\,\big] \\ &\leq \Pr[C] + \Pr\big[\mathcal{A} \text{ wins} \mid \overline{C}\,\big].\end{aligned} \tag{2}$$

Let us first bound the probability of a seed collision. Let C_g the event that there is a collision among nodes that are not the source of a seed dependency. The seeds of these up to N nodes are sampled independently of each other u.a.r. from $\{0, 1\}^\rho$, so by the birthday bound we have

$$\Pr[C_g] \leq \frac{N(N-1)}{2 \cdot 2^\rho}.$$

[12] To be precise, the event $Q_{\mathrm{s}} \wedge F_{\mathrm{DH}}$ is a superset of case (i) above. However, the argument applied in Lemma 3 gives the same bound for either event and this more general event has the advantage of being simpler.

Now consider $\Pr\left[C \mid \overline{C_g}\right]$. The source seed of every seed dependency is distinct if $\overline{C_g}$. Therefore, given $\overline{C_g}$ every additional seed is sampled u.a.r. from $\{0,1\}^\rho$, with up to N seeds already being taken. Similar to proving the birthday bound, the probability that the i-th seed derived from a seed dependency collides with one of the already sampled seeds is at most $\frac{N+i}{2^\rho}$. Therefore,

$$\begin{aligned}\Pr\left[C \mid \overline{C_g}\right] &\leq \frac{N+0}{2^\rho} + \ldots + \frac{N+(N-1)}{2^\rho} \\ &= \frac{N^2}{2^\rho} + \frac{N(N-1)}{2 \cdot 2^\rho}.\end{aligned}$$

Then

$$\begin{aligned}\Pr[C] &= \Pr[C \wedge C_g] + \Pr\left[C \wedge \overline{C_g}\right] \\ &\leq \Pr[C_g] + \Pr\left[C \mid \overline{C_g}\right] \\ &\leq \frac{N^2}{2^\rho} + \frac{N(N-1)}{2^\rho} \\ &\leq \frac{N^2}{2^{\rho-1}}.\end{aligned} \tag{3}$$

Next we justify step (†) in (1). Note that by the rules imposed on the adversary in the SD-GSD game, the challenge node v is safe and its seed is thus indeed hidden. If Q_{dep} does not hold, then $\mathcal{A}$ has not queried H_{dep} for s_v. Moreover, since v is the only node with seed s_v by $\overline{C}$, the SD-GSD challenger only ever queries H_{dep} for s_v when computing the challenge, so the challenger reveals no information about $H_{\text{dep}}(s_v)$ through its responses to other queries.[13] Thus, by virtue of H_{dep} being a random oracle, $H_{\text{dep}}(s_v)$ is a uniformly distributed value in $\{0,1\}^\rho$ from $\mathcal{A}$'s perspective. The value s follows the same distribution. Thus, $\mathcal{A}$ behaves the same when given either $r = s$ or $r = H_{\text{dep}}(s_v)$ and

$$\begin{aligned}\Pr\left[1 \leftarrow \mathcal{A} \mid \overline{C}, \overline{Q_{\text{dep}}}, b=1\right] &= \Pr\left[1 \leftarrow \mathcal{A} \mid \overline{C}, \overline{Q_{\text{dep}}}, r=s\right] \\ &= \Pr\left[1 \leftarrow \mathcal{A} \mid \overline{C}, \overline{Q_{\text{dep}}}, r=H_{\text{dep}}(s_v)\right] \\ &= \Pr\left[1 \leftarrow \mathcal{A} \mid \overline{C}, \overline{Q_{\text{dep}}}, b=0\right].\end{aligned} \tag{4}$$

[13] If collisions were possible, then another node u in the GSD graph could by chance have the same seed and the challenger could leak $H_{\text{dep}}(s_u) = H_{\text{dep}}(s_v)$, e.g. if u is the source of a seed dependency and the target node of the dependency is corrupted.

Therefore

$$\begin{aligned}\Pr\bigl[\mathcal{A} \text{ wins} \mid \overline{C}, \overline{Q_{\text{dep}}}\,\bigr] &= \Pr\bigl[1 \leftarrow \mathcal{A} \mid \overline{C}, \overline{Q_{\text{dep}}}, b=1\bigr] \cdot \frac{1}{2} \\ &\quad + \Pr\bigl[0 \leftarrow \mathcal{A} \mid \overline{C}, \overline{Q_{\text{dep}}}, b=0\bigr] \cdot \frac{1}{2} \\ &\overset{(4)}{=} \Pr\bigl[1 \leftarrow \mathcal{A} \mid \overline{C}, \overline{Q_{\text{dep}}}, b=0\bigr] \cdot \frac{1}{2} \\ &\quad + \Pr\bigl[0 \leftarrow \mathcal{A} \mid \overline{C}, \overline{Q_{\text{dep}}}, b=0\bigr] \cdot \frac{1}{2} \\ &= \frac{1}{2}.\end{aligned}$$

Next, (♠) is easy to show

$$\begin{gathered}\Pr\bigl[Q_{\text{s}} \mid \overline{C}\,\bigr] \cdot (1 - \Pr[C]) = \Pr\bigl[Q_{\text{s}} \mid \overline{C}\,\bigr] \cdot \Pr\bigl[\,\overline{C}\,\bigr] = \Pr[Q_{\text{s}}] - \Pr[Q_{\text{s}} \wedge C] \\ \Longrightarrow \\ \Pr\bigl[Q_{\text{s}} \mid \overline{C}\,\bigr] \le \Pr[Q_{\text{s}}] + \Pr[C].\end{gathered}$$

By Lemma 3 we have[14]

$$\Pr[Q_{\text{s}} \wedge F_{\text{DH}}\,] \le N \cdot \varepsilon_{\text{DDH}} + \frac{m_{\text{DH}} \cdot N^2}{q}.$$

and by Lemma 2 we have

$$\Pr\bigl[Q_{\text{s}} \wedge \overline{F_{\text{DH}}}\,\bigr] \le \delta \cdot N \cdot \varepsilon_{\text{EAV}} + \frac{m_{\text{s}} \cdot N}{2^{\rho}},$$

so we know that

$$\Pr[Q_{\text{s}}] \le N \cdot \varepsilon_{\text{DDH}} + \delta \cdot N \cdot \varepsilon_{\text{EAV}} + \frac{m_{\text{DH}} \cdot N^2}{q} + \frac{m_{\text{s}} \cdot N}{2^{\rho}}. \tag{5}$$

Then

$$\begin{aligned}\mathrm{Adv}^{\text{SD-GSD}}_{\Pi,\eta}(\mathcal{A}) &= 2 \cdot \left(\Pr[\mathcal{A} \text{ wins}] - \frac{1}{2}\right) \\ &\overset{(2),(1)}{\le} 4 \cdot \Pr[C] + 2 \cdot \Pr[Q_{\text{s}}] \\ &\overset{(3),(5)}{\le} \tilde{\varepsilon}(\eta).\end{aligned}$$

Security against multiple challenges As noted already in the introduction, it is straightforward to prove the same result for an SD-GSD game with multiple challenge queries. In this SD-GSD game, the adversary would have access to an

[14] Note that we are again omitting the argument η from the functions on the right-hand side ($N, \varepsilon_{\text{DDH}}$ and m_{DH} in this case).

additional *challenge* oracle to perform a challenge on any node as in step 3. of the original game, with the bit b sampled once at the beginning of the game, and the seed s sampled fresh for every query. Of course, the adversary must ensure that the restrictions that apply to the challenge node in the original game apply to all challenge nodes in this new game.

To prove security in this multi-challenge SD-GSD game, the same case distinction can be performed as in (1) and, concerning step (†), it can be shown through a sequence of statistically indistinguishable hybrids that the adversary has no advantage if they did not trigger Q_{dep}.

Reducing to EAV security Recall case (ii) in the high-level discussion of Theorem 2: the adversary $\mathcal{A}$ was able to learn the seed s_w given the EAV secure encryptions $\mathit{\Pi}_s.\text{Enc}_{k_1}(s_w), \dots, \mathit{\Pi}_s.\text{Enc}_{k_d}(s_w)$. We can see $\mathcal{A}$ as an adversary in a security game where $\mathcal{A}$ is given d EAV secure encryptions $c_1 \leftarrow \mathit{\Pi}_s.\text{Enc}_{k_1}(m), \dots, c_d \leftarrow \mathit{\Pi}_s.\text{Enc}_{k_d}(m)$ of a message m with $k_i \leftarrow \mathit{\Pi}_s.\text{Gen}(1^\eta)$ and must compute m. If we can prove that beating such a game is hard, then we can bound the probability of $\mathcal{A}$ actually learning s_w in this way.

This is exactly how we proceed in this section. Instead of asking the adversary to compute an encrypted message m, we turn to a more familiar decisional formulation as in the IND-CPA game (where the adversary may choose a pair m_0, m_1 and must distinguish whether the d ciphertexts encrypt m_0 or m_1). We call this security notion *EAV security under multiple (M) independent (I) encryptions of a single (S) pair of messages* (MIS-EAV).

Definition 5 (The MIS-EAV game). *Let κ denote the security parameter and let $\mathit{\Pi}$ a private-key encryption scheme. Define the game* $\text{Game}^{\text{MIS-EAV}}_{\mathit{\Pi},\kappa}(\mathcal{A})$ *for an adversary $\mathcal{A}$:*

1. *The adversary $\mathcal{A}$ outputs $q \in \mathbb{N}$ and a pair of messages m_0, m_1 of the same length. We refer to q as the number of* queries *made by $\mathcal{A}$.*
2. *A bit $b \leftarrow \{0, 1\}$ is sampled. For each $i \in [q]$, $\mathcal{A}$ is given an encryption $c_i \leftarrow \mathit{\Pi}.\text{Enc}_{k_i}(m_b)$ where $k_i \leftarrow \mathit{\Pi}.\text{Gen}(1^\kappa)$ is generated independently of the other keys.*
3. *$\mathcal{A}$ outputs a bit b'. The output of the game is defined to be 1 if $b' = b$, and 0 otherwise.*

Definition 6 (MIS-EAV security). *A private-key encryption scheme $\mathit{\Pi}$ is* (t, ε, q)-MIS-EAV-secure *if for all κ, for any adversary $\mathcal{A}$ making at most $q(\kappa)$ queries and running in time $t(\kappa)$ we have*

$$\text{Adv}^{\text{MIS-EAV}}_{\mathit{\Pi},\kappa}(\mathcal{A}) := 2 \cdot \left(\Pr\Big[\text{Game}^{\text{MIS}-\text{EAV}}_{\mathit{\Pi},\kappa}(\mathcal{A}) = 1\Big] - \frac{1}{2} \right) \le \varepsilon(\kappa).$$

Similar to how IND-CPA security for a single encryption query implies IND-CPA security for q queries with a security loss of q by a standard hybrid argument, one can show that EAV security implies MIS-EAV security with the same loss. To see why, recall the hybrid argument for IND-CPA security (as discussed in e.g.

[12, Theorem 12.6]): We define the sequence of hybrid games $G_0, \dots, G_q$ where in the game G_i the first i encryption queries encrypt the second message and the remaining $q - i$ queries encrypt the first message. Then given an IND-CPA adversary $\mathcal{A}$ for multiple encryptions, an IND-CPA adversary $\mathcal{A}'$ is constructed to bound

$$|\Pr[\mathcal{A} \text{ outputs } 0 \text{ in game } G_{i-1}] - \Pr[\mathcal{A} \text{ outputs } 0 \text{ in game } G_i]|$$

for arbitrary i. The adversary $\mathcal{A}'$ simulates G_{i-1} or G_i to $\mathcal{A}$ depending on whether the ciphertext received from the (single-query) IND-CPA challenger, which gets passed on as the response to the i-th query, encrypts the first or the second message from the i-th pair of messages. $\mathcal{A}'$ then uses the encryption oracle to pass on the right encryptions to $\mathcal{A}$ for all other queries. Now notice that if we wanted to simulate to an MIS-EAV adversary we would not need access to an encryption oracle since for the MIS-EAV security game all the other encryptions can easily be generated by $\mathcal{A}'$ sampling the new keys itself.

The argument would of course also work without restricting the adversary to a single pair of messages. However, we make use of this restriction to provide a tighter reduction for a certain class of schemes in [4, Section B.4].

Lemma 1. *Let Π a private-key encryption scheme with finite message space. Let $t_{\text{Gen}}, t_{\text{Enc}}$ functions in κ that upper bound the runtime of Π.*Gen *and Π.*Enc*, respectively. If Π is (t, ε)-EAV-secure, then for any function q, Π is $(\tilde{t}, q \cdot \varepsilon, q)$-MIS-EAV-secure with $\tilde{t} = t - \mathcal{O}(q \cdot (t_{\text{Gen}} + t_{\text{Enc}}))$.*

The details of the proof can be found in [4, Section B.2].

Lemma 2. *Recall the assumptions, variables and events from the statement and proof of Theorem 2. In particular, assume that Π_s is $(t, \varepsilon_{\text{EAV}})$-EAV-secure. Let η arbitrary and let $\mathcal{A}$ an SD-GSD adversary constructing a GSD graph of size at most $N(\eta)$ and indegree at most $\delta(\eta)$, making at most $m_{\text{s}}(\eta)$ queries to H_{gen} or H_{dep} and at most $m_{\text{DH}}(\eta)$ queries to H_{DH}, and running in time $\tilde{t}(\eta)$. Then*

$$\Pr\left[Q_{\text{s}} \wedge \overline{F_{\text{DH}}}\,\right] \leq \delta \cdot N \cdot \varepsilon_{\text{EAV}} + \frac{m_{\text{s}} \cdot N}{2^{\rho}}.$$

Intuition By Lemma 1 we know that Π_s is MIS-EAV secure. Continuing the high-level argument before the proof of Theorem 2, consider the first moment that $\mathcal{A}$ triggers $Q_{\text{s}} \wedge \overline{F_{\text{DH}}}$ by querying the seed of some safe node w. As intended, it follows from the definition of the event F_{DH} that from $\mathcal{A}$'s perspective all DHIES ciphertexts it got from queries encrypt(u, w) for any u contain encryptions of s_w under independent, uniformly random keys using Π_s. Moreover, as already argued once, $\mathcal{A}$ has learned nothing from a potential seed dependency (p, w), so these encryptions are everything $\mathcal{A}$ had at its proposal to learn s_w.

We can use $\mathcal{A}$'s ability to compute the seed s_w of a safe node w from encryptions of s_w to construct an MIS-EAV adversary: We first guess a node w whose seed $\mathcal{A}$ may query first. Next, we give the MIS-EAV challenger s_w and some other independent seed s. We simulate the SD-GSD game to $\mathcal{A}$ and embed the encryptions from the MIS-EAV challenger when answering queries of the form $\text{encrypt}(u, w)$ for any u. Now consider the behavior of $\mathcal{A}$ depending on which seed the challenger chooses to encrypt:

- If the challenger chooses to encrypt s_w, then $\mathcal{A}$ will trigger the event $Q_\text{s} \wedge \overline{F_\text{DH}}$ with the same probability as before. We can detect whether $Q_\text{s} \wedge \overline{F_\text{DH}}$ gets triggered since all seeds in the simulation are known. If $Q_\text{s} \wedge \overline{F_\text{DH}}$ occurs and we guessed w correctly, the event will be triggered at w and $\mathcal{A}$ will query s_w, telling us that the challenger encrypted s_w.
- If the challenger chooses to encrypt s, then $\mathcal{A}$ receives no information about s_w and has a negligible probability of querying it.

Thus, the advantage of the MIS-EAV adversary is about $\Pr\left[Q_\text{s} \wedge \overline{F_\text{DH}}\,\right]/N$, where the factor $1/N$ arises from guessing w, and using that Π_s is MIS-EAV secure we can bound this probability. Since we are only interested in checking whether the event was triggered for w, the adversary can abort when this is no longer possible (w is corrupted, some other hidden seed is queried, etc.). The details of the proof can be found in [4, Section B.3].

For a certain class of schemes, we can improve on Lemma 1 and achieve a tight reduction. This allows us to get rid of the factor δ in Lemma 2. However, we do not use this in our main result and refer the interested reader to [4, Section B.4].

Reducing to the DDH problem

Lemma 3. *Recall the assumptions, variables and events from the statement and proof of Theorem 2. In particular, assume that the DDH problem is $(t, \varepsilon_\text{DDH})$-hard relative to $\mathcal{G}$. Let η arbitrary and let $\mathcal{A}$ an SD-GSD adversary constructing a GSD graph of size at most $N(\eta)$ and indegree at most $\delta(\eta)$, making at most $m_\text{s}(\eta)$ queries to H_gen or H_dep and at most $m_\text{DH}(\eta)$ queries to H_DH, and running in time $\tilde{t}(\eta)$. Then*

$$\Pr[Q_\text{s} \wedge F_\text{DH}] \leq N \cdot \varepsilon_\text{DDH} + \frac{m_\text{DH} \cdot N^2}{q}.$$

Intuition We will bound the simpler event F_DH. This event tells us that there is some safe node a in the GSD graph with encryption edges to nodes $u_1, \ldots, u_d$, where the query $\text{encrypt}(a, u_i)$ returned the ciphertext $\langle g^{y_i}, \Pi_s.\text{Enc}_{k_i}(s_{u_i})\rangle$ with $k_i = H_\text{DH}(g^{sk_a \cdot y_i})$, such that $g^{sk_a \cdot y_j}$ was the first hidden Diffie-Hellman key queried by $\mathcal{A}$ for some j. Moreover, at the time $g^{sk_a \cdot y_j}$ was queried, no hidden seed had yet been queried by $\mathcal{A}$, implying that $\mathcal{A}$ had not queried $H_\text{gen}(s_a)$ and thus had no information about sk_a besides pk_a (recall that $(pk_a, sk_a) = \Pi_\text{DH}.\text{Gen}(1^\eta, H_\text{gen}(s_a))$).

It is interesting to note that our approach does not require that $\mathcal{A}$ has not queried H_dep for a hidden seed (i.e. that Q_dep was not triggered) as is implied by the event F_DH, because knowing $H_\text{gen}(s_a)$ is the only way to learn about sk_a. Regardless, we still want to have our definition of F_DH include this information, as the bound on $\Pr\left[Q_\text{s} \wedge \overline{F_\text{DH}}\,\right]$ in Lemma 2 relies on the fact that in the event of

$Q_\mathrm{s} \wedge \overline{F_\mathrm{DH}}$ happening, Q_DH was not yet triggered when the event Q_s was triggered, i.e. when either the event Q_gen *or* the event Q_dep was triggered.

The intuition is clear that this means that $\mathcal{A}$ solved the Diffie-Hellman challenge (g^{sk_a}, g^{y_j}). What is not immediately clear is how to embed a *given* Diffie-Hellman challenge (g^x, g^y) from an instance of the DDH game and use $\mathcal{A}$ to tell whether the key k chosen by the challenger is the real key $g^{x \cdot y}$ or a uniformly random group element. An intuitive strategy would be to embed the challenge by setting $pk_a = g^x$ and $g^{y_j} = g^y$, which involves guessing u_j, and simply checking whether for any of the queries q_i to H_DH by $\mathcal{A}$, such that q_i encodes a group element in $\mathbb{G}$, it holds that $q_i = k$. Now:

– If $k = g^{x \cdot y}$, $\mathcal{A}$ triggers F_DH and we guessed a and u_j correctly, then indeed as described above $q_i = g^{sk_a \cdot y_j} = g^{x \cdot y} = k$ will hold for some i.
– If k is a random group element, then $\mathcal{A}$ has negligible probability of querying k, as no information about k is ever leaked to $\mathcal{A}$.

If we make sure not to change $\mathcal{A}$'s view of the game in the case $k = g^{x \cdot y}$ in this process, we can achieve an advantage of about $\Pr[F_\mathrm{DH}]/N^2$, where one factor $1/N$ arises from guessing a and another from guessing u_j. Unfortunately, this would yield no improvement over the result from [13].

To avoid this issue, we can use the random self-reducibility of the DDH problem and avoid guessing u_j. Instead of embedding g^y into a single encryption edge, we embed it into all d encryption edges. To get a uniformly random exponent from y we set $y_j = y + r_j \mod q$ with $r_j \leftarrow [q]$. Given $g^{x \cdot y_j}$, we can easily compute $g^{x \cdot y}$:

$$g^{x \cdot y_j} = g^{x \cdot (y + r_j)} = g^{x \cdot y} \cdot g^{x \cdot r_j} \iff g^{x \cdot y} = g^{x \cdot y_j} \cdot \underbrace{((g^x)^{r_j})^{-1}}_{=: R_j}.$$

Now to determine whether k is the real Diffie-Hellman key, we check whether $q_i \cdot R_j = k$ for some i, j. This yields an advantage of about $\Pr[F_\mathrm{DH}]/N$ (and a slightly larger runtime). The details of the proof can be found in [4, Section B.5].

References

1. Abdalla, M., Bellare, M., Rogaway, P.: DHIES: An encryption scheme based on the Diffie-Hellman Problem (1998), https://cseweb.ucsd.edu/~mihir/papers/dhies.pdf
2. Alwen, J., Coretti, S., Dodis, Y., Tselekounis, Y.: Security analysis and improvements for the IETF MLS standard for group messaging. In: Micciancio, D., Ristenpart, T. (eds.) Advances in Cryptology – CRYPTO 2020. pp. 248–277. Springer International Publishing, Cham (2020)
3. Alwen, J., Coretti, S., Dodis, Y., Tselekounis, Y.: Modular design of secure group messaging protocols and the security of MLS. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. p. 1463–1483. CCS '21, Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3460120.3484820
4. Azari, K., Ellison, A.: Tighter provable security for TreeKEM. Cryptology ePrint Archive, Paper 2024/1878 (2024), https://eprint.iacr.org/2024/1878
5. Balbás, D., Collins, D., Gajland, P.: WhatsUpp with sender keys? Analysis, improvements and security proofs. In: Guo, J., Steinfeld, R. (eds.) Advances in Cryptology – ASIACRYPT 2023. pp. 307–341. Springer Nature Singapore, Singapore (2023)
6. Barnes, R., Beurdouche, B., Robert, R., Millican, J., Omara, E., Cohn-Gordon, K.: The Messaging Layer Security (MLS) Protocol. RFC 9420 (Jul 2023). https://doi.org/10.17487/RFC9420, https://www.rfc-editor.org/info/rfc9420
7. Bellare, M., Rogaway, P.: Random oracles are practical: a paradigm for designing efficient protocols. In: Proceedings of the 1st ACM Conference on Computer and Communications Security. p. 62–73. CCS '93, Association for Computing Machinery, New York, NY, USA (1993). https://doi.org/10.1145/168588.168596
8. Bhargavan, K., Barnes, R., Rescorla, E.: TreeKEM: Asynchronous Decentralized Key Management for Large Dynamic Groups A protocol proposal for Messaging Layer Security (MLS). Research report, Inria Paris (May 2018), https://inria.hal.science/hal-02425247
9. Cohn-Gordon, K., Cremers, C., Dowling, B., Garratt, L., Stebila, D.: A formal security analysis of the signal messaging protocol. In: 2017 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 451–466 (2017). https://doi.org/10.1109/EuroSP.2017.27
10. Hogben, G.: An important step towards secure and interoperable messaging. https://security.googleblog.com/2023/07/an-important-step-towards-secure-and.html (2023), accessed: 2023-11-01
11. IETF: Support for MLS. https://www.ietf.org/blog/support-for-mls-2023/ (2023), accessed: 2023-11-01
12. Katz, J., Lindell, Y.: Introduction to Modern Cryptography, Third Edition. Chapman & Hall/CRC, 3rd edn. (2020)
13. Klein, K., Pascual-Perez, G., Walter, M., Kamath, C., Capretto, M., Cueto, M., Markov, I., Yeo, M., Alwen, J., Pietrzak, K.: Keep the dirt: Tainted TreeKEM, adaptively and actively secure continuous group key agreement. In: 2021 IEEE Symposium on Security and Privacy (SP). pp. 268–284 (2021). https://doi.org/10.1109/SP40001.2021.00035
14. Panjwani, S.: Tackling adaptive corruptions in multicast encryption protocols. In: Proceedings of the 4th Conference on Theory of Cryptography. p. 21–40. TCC'07, Springer-Verlag, Berlin, Heidelberg (2007)

15. Pascual Perez, G.: On the efficiency and security of secure group messaging. Ph.D. thesis, Institute of Science and Technology Austria (2024). https://doi.org/10.15479/at:ista:18088
16. Perrin, T., Marlinspike, M.: The Double Ratchet algorithm. https://signal.org/docs/specifications/doubleratchet/ (2016), accessed: 2024-03-05