A preliminary version of this paper appears in the proceedings of the 45th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT 2026). This is the full version.

Secure Cloud Storage: Modularization, Network Adversaries and Adaptive Corruptions

Jonas Janneck[1] and Doreen Riepel[2]

[1] Ruhr University Bochum, Germany
jonas.janneck@rub.de
[2] CISPA Helmholtz Center for Information Security, Germany
riepel@cispa.de

August 26, 2026

Abstract. End-to-end cloud storage solutions are deployed at large scale, yet recent works have demonstrated severe attacks against their confidentiality and integrity. Motivated by this, a first formal treatment of secure cloud storage was given at CRYPTO 2024 by Backendal, Davis, Günther, Haller and Paterson (BDGHP). They define syntax and security notions, capturing client-to-client security of cloud storage schemes with respect to a password distribution. They also give an efficient construction using the Two-Hash Diffie-Hellman (2HDH) OPRF and standard cryptographic building blocks, which they prove secure under selective corruptions in the random oracle model. However, several aspects of practical security guarantees remain open.

We extend and refine the work of BDGHP along multiple dimensions, advancing the analysis of secure cloud storage schemes. First, we prove that their construction can be proven secure against adaptive corruptions (with a slight modification), circumventing technical challenges posed by file sharing. Second, we modularize the scheme further by introducing an abstraction for the authentication procedure. This allows us to identify the concrete role of 2HDH and alternative instantiations. Third, we introduce a weaker model that captures adversaries who can arbitrarily control the network, except during registration. This allows us to prove concrete guarantees about online password guessing attacks, whereas the stronger model inherently allows for offline guessing. Finally, we formalize and prove explicit authentication, relying on the security of our new authentication abstraction and the MAC scheme, where the latter was previously not used in the security analysis.

Table of Contents

Changes from the published version [JR26]

Syntactical changes to the authentication primitive to allow for server authentication. Adding the Sig2HDH construction, which is required to achieve the offline guessing resistance in the network adversary setting. Updates to the related security assumptions, proofs and explanations. Additional minor changes throughout the paper to improve and clarify the security analysis.

1 Introduction

End-to-End Encrypted (E2EE) cloud storage solutions allow a user to have the control over cryptographically protecting their data stored in the cloud. In such solutions, the user typically generates symmetric encryption keys, often derived from a (possibly low-entropy) password, and then encrypts a file before uploading it to the server. Other than meta-data, the server should not learn anything about the content of the file nor compromise its integrity. E2EE cloud storage solutions, such as those from MEGA [MEG], Nextcloud [Nex] and Apple [App], are offered at large-scale, targeting both private and corporate customers. Files should be easily accessible from multiple devices, and a user may also share them in a secure way with other users, which makes these protocols generally highly interactive and the design of secure solutions non-trivial.

Attacks against cloud storage solutions. A recent line of works [BHP23, AHMP23, HR23, ABCP24, HT24] has shown that indeed many cloud storage solutions are not as secure as the users would probably have hoped for. Even though providers were promising strong privacy, the attacks are critical in that they are practical and affect millions of users. Motivated by this, Backendal, Davis, Günther, Haller and Paterson (BDGHP) [BDG+24] initiate the formal study of secure cloud storage solutions from a provable security perspective[1] to better understand the threat model and how to avoid such attacks in the first place. They present a formal security model for E2EE cloud storage that in particular captures the core of these attacks.

The model and scheme of [BDG+24]. BDGHP are the first to formally define the syntax and expected security guarantees that an E2EE cloud storage scheme should satisfy. They specify sub-protocols that capture registration, authentication, file upload, update and download, as well as file sharing. The main difference to prior work in related areas is that operations are all modeled in a non-atomic way, which allows for arbitrary interleavings of executions of the sub-protocol between multiple users.

Their multi-user security notions are defined using code-based games and parameterized by a password distribution. They capture two flavors of client-to-client (C2C) security:

- **Confidentiality:** An adversary should not learn anything about an honest user's file unless it is shared with another compromised user.
- **Integrity:** An adversary cannot modify an honest user's file or create new files for them, as long as they are not shared with another compromised user.

Both properties should hold even if the adversary can control the server, capturing what we intuitively refer to when talking about end-to-end encryption. At the same time, BDGHP also capture malicious users by allowing the adversary to corrupt users. Here, we mainly distinguish two settings:

- **Selective corruptions:** The adversary has to declare the set of users that it will corrupt throughout the game in advance.
- **Adaptive corruptions:** The adversary can decide during the execution and specifically dependent on other interactions which users it will corrupt.

For cloud storage schemes, corrupting a user not only gives the adversary access to their password, but also to their files and temporary states. This complicates security proofs, especially those for adaptive corruptions. In fact, BDGHP (whose contributions focus on many other things) only prove security under selective corruptions. We will reiterate some of the difficulties that arise in the adaptive setting below.

The construction from BDGHP uses prime-order groups, more explicitly, the Two-Hash Diffie-Hellman (2HDH) oblivious pseudorandom function (OPRF) [JKK14], an authenticated encryption scheme with associated data (AEAD), a pseudorandom function (PRF) and a message authentication code (MAC). It achieves confidentiality and integrity under selective corruptions in the random oracle model, relying on standard security properties of the AEAD scheme and the PRF, as well as the advantage of an adversary in a dedicated password guessing game.

[1] For a more detailed comparison to earlier and related work on cloud storage in cryptographic contexts, we refer the reader to the end of Section 1.2 and also [BDG+24, Section 1.3], especially Table 1.

On a high-level, the cloud storage scheme works as follows. In the registration and in the authentication procedure, the client computes a raw secret from their password which is then used to derive an encryption key and a MAC key. The encryption key is used to encrypt/decrypt the client's master key and the MAC key to authenticate to the server during the authentication procedure. On the next layer, the master key is used to encrypt file keys that can be potentially shared between clients and the file keys are eventually used to encrypt the files that are uploaded. The multiple layers are needed to enable client key rotation, sharing of files, and updating files without re-encryptions along the complete chain or for every user involved.

Selective vs. Adaptive Corruptions. For practical applications, security under adaptive corruptions is desired because it captures adversarial capabilities more accurately. It is the standard goal in many cryptographic contexts, like (password-based) key exchange [BR94, BPR00], and clearly preferred for practical applications, e.g., for threshold signatures [CKM23]. In some cases, however, it is not required to explicitly prove security under adaptive corruptions. The two notions are often asymptotically equivalent for "simple" primitives, e.g., signatures, public-key or secret-key encryption. In two-party interactive protocols, such as authenticated key exchange, security proofs for the multi-user setting often boil down to analyzing different cases, where in each case we can rely on two uncorrupted user secrets (either their long-term or their ephemeral secret key).

Unfortunately, this is not the case for cloud storage schemes. One key aspect why "naive" proof strategies fail lies in the fact that corrupting one user's state also leaks information about other users' states. In particular, this becomes clear when considering file sharing. Confidentiality is modeled by having the adversary distinguish which of two files was encrypted. However, a file can be shared with an arbitrary number of users. If we were to guess which set of users remains uncorrupted throughout the game, this would result in an exponential security loss.

Additional Desirable Security Properties. The work of BDGHP models the core properties of E2EE cloud storage. Interestingly, their construction makes use of building blocks that are not required to achieve those core properties. More specifically, the properties of 2HDH and the MAC scheme are never used in the proofs. As stated by the authors, they design the protocol in a way that already captures additional desirable properties which they only state intuitively. One of them is, since we are in the password-based setting, to capture online guessing attacks explicitly. In their model, the server is considered malicious, hence it can always brute-force passwords (offline). Ideally, we would want to capture a property similar to password-based key exchange, namely, that one online interaction is the best an (external) adversary can do to rule out one password. In this case, the (honest) server can limit attacks to a small number of tries. The MAC, on the other hand, can allow the server to explicitly authenticate a session. This feature allows the server to abort earlier for failed attempts but can also be used to notify users about a successful login allowing them to take action in case it was adversarial.

Post-Quantum Security. The construction of BDGHP is modular except for the authentication procedure, where 2HDH is used. It seems reasonable to simply replace 2HDH with any other (post-quantum) secure OPRF. Since the proof does not rely on any properties of 2HDH, it is however unclear what exact security properties are required for this replacement to be sound. Additionally, post-quantum secure OPRFs, e.g., those recently studied in [BDFH25, HKL+25], are rather cost-intense. If we require a weaker property, we can possibly construct a more efficient scheme.

1.1 Our Contributions

We build upon the work of BDGHP as follows:

- **Security under Adaptive Corruptions:** We prove that their construction achieves security under adaptive corruptions. While we make a slight modification which enables an easier proof, we also argue why our techniques should generally apply to the original construction.
- **Modularization:** To capture the properties of 2HDH, we provide an additional abstraction which we call an authentication protocol (Auth) and security notions. The name comes from the fact that it modularizes

Table 1. Overview of theorems for the cloud storage construction. The table shows which security notions are fulfilled under which assumptions on the underlying building blocks. All notions consider adaptive corruptions. OW-CPA is a new notion, its relation to standard notions is given in Appendix A.2.

Notion	Auth	AEAD	MAC
Theorem 10, Conf (Figure 9)	INS-OW	INT-CTXT + OW-CPA + IND-CCA	—
Theorem 11, Int (Figure 10)	INS-OW	INT-CTXT + OW-CPA	—
Theorem 12, Net-Conf (Figure 13)	OUT-OW	INT-CTXT + OW-CPA + IND-CCA	—
Theorem 13, Net-Int (Figure 14)	OUT-OW	INT-CTXT + OW-CPA	—
Theorem 14, Exp-Auth (Figure 15)	OUT-OW	—	SUF

the authentication procedure. This allows us to identify the exact role of 2HDH, and for which properties simpler constructions are sufficient, providing a path towards post-quantum secure cloud storage.

– **Security against Network Adversaries:** We formalize confidentiality and integrity for active adversaries that control the network arbitrarily, except that we assume secure channels in the registration process. This model is stronger than the notion of client-to-server (C2S) security sketched by BDGHP, but weaker than C2C security. It allows us to make a statement about online password guessing and crucially relies on the stronger authentication property, as achieved by 2HDH and signatures.
– **Explicit Authentication:** We give a security game that captures explicit (client) authentication which allows the server to detect malicious login attempts early, so it can abort in case of a failed attempt or notify the user in case of a successful attempt. We prove that the property is achieved by the construction of BDGHP, relying on the security of the MAC scheme, which explains its purpose in the design.

Along the way, we make some other minor contributions. On a more conceptual level, we make notational simplifications to the pseudocode description of the model. Finally, we obtain tighter bounds for the integrity proof.

1.2 Technical Overview

We now want to give a more detailed overview of our main contributions, in particular, about our proof techniques and challenges that we overcome, and the motivation for our new definitions. In Table 1, we provide an overview of the theorems that we prove and which assumptions we rely on.

Adaptive security. During account registration, the game needs to commit to a ciphertext that encrypts the user's master key. If the user is later compromised, the ciphertext can be decrypted by the adversary and the master key is revealed. If the user is not compromised and also owner of a challenge file (or a user that the challenge file is shared to), the master key must be replaced by a random key for the remainder of the proof. Therefore, the game must decide in the beginning, i.e., during the call to the account registration oracle, if they replace it by random or not. This is typically referred to as the commitment problem. In the setting with selective corruptions, the game (or the reduction) will have all the relevant information for a correct simulation. The problem arises in the adaptive setting and due to the fact that it is not enough to simply guess one uncorrupted user because of potentially shared files. Therefore, we need to address the problem differently.

Our approach borrows ideas from constructing and proving tight security of authenticated key exchange protocols, e.g., [GJ18, JKRS21]. Those works solve the commitment problem in the context of public-key encryption, whereas here, we are concerned with symmetric-key primitives. The ideas are, however, closely related when working in idealized models, such as the random oracle model (ROM), cf. also [Nie02, JT20, Jae23]. At this point it is worth emphasizing that the scheme from BDGHP already relies on the ROM. We minimally change the scheme, as we describe below, but we argue that the same proof technique can also work for the original construction.

Intuitively, the encryption scheme that we want to use to encrypt the master key needs to be non-committing. When the encryption happens, the game samples a random ciphertext. Then, later, the game

can "explain" how this ciphertext was produced by programming the random oracle accordingly. We can imagine two ways of constructing such an encryption scheme for the cloud storage scheme.

- By extending the message size of the encryption scheme, we can build a non-committing encryption scheme Enc from a scheme Enc' and a random oracle H as follows: We first split the message m into $m' \,||\, r$ where m' has the size of messages before the extension and some r some sufficiently large randomness. Then we compute $m^\star = m' \oplus \mathsf{H}(r)$ and encrypt $m^\star \,||\, r$ under Enc'. Note that this slightly increases the ciphertext size. The resulting scheme is non-committing because m can be chosen at the moment the random oracle is queried on r. For this to work, r and therefore the original message m must have sufficient entropy. This is satisfied by BDGHP because they encrypt uniformly random keys. Hence, we can instantiate each key encryption in BDGHP by the modified encryption described here and therefore still rely on the original scheme.
- There is a simpler way that also exploits the fact that the encryption operations in the cloud storage scheme do not handle arbitrary messages but mostly uniformly random messages.[2] Instead of choosing the master key uniformly at random, we choose a random seed and compute the master key as $\mathsf{H}(\mathsf{seed})$. Then we encrypt seed instead of the master key. This approach does not change the ciphertext size but slightly modifies the scheme.

In our analysis, we use the second approach. This way, the game does not need to commit to the exact value of $\mathsf{H}(\mathsf{seed})$ at the beginning, as long as seed is not queried to the random oracle. We decide against the first approach and instead change the BDGHP scheme slightly to avoid confusion for potential implementations. The first approach requires the AEAD to be non-committing which is usually not the case in practice[3] while the second approach works with an arbitrary (committing) AEAD and explicitly employs the non-committing technique in the scheme.

Authentication Protocols. Instead of directly using 2HDH over prime-order groups, we modularize the construction further. 2HDH is a concrete instantiation of an oblivious pseudorandom function (OPRF). Here, its main purpose is to authenticate the user: If the user holds the correct password, then by evaluating the OPRF, they can compute their raw secret and therefore their master key.

As far as we are aware, one cannot simply plug in "standard" properties of an OPRF[4], which is why we provide a new abstraction level which we call (two-message) authentication protocols, or short Auth. Such protocols are executed between a client and a server, each sending exactly one message. Both client and server hold a secret (whose distributions are part of the parameters of the scheme) and at the end, the client computes a key. In terms of security, we want this key to be hard to compute, which we formalize as a one-wayness property.

We consider two different notions of one-wayness, one against insider and one against outsider adversaries, which we call INS-OW and OUT-OW, respectively. We rely on INS-OW to prove the confidentiality notion of BDGHP. It turns out that a very simple construction using only a random oracle, which we call SHash, is already sufficient to achieve this notion[5]. We will however explain the role of 2HDH in more detail below, when talking about network adversaries.

Network Adversaries. In the C2C model from BDGHP, the adversary controls the server. While this is definitely an attack scenario that we want to capture, in practice, we may also be interested in a weaker setting, namely, what kind of security we can still achieve when the adversary (only) controls the network. One property that can be captured this way is impersonation attempts, e.g., the adversary could try to impersonate a client towards the server or vice versa. This allows us to distinguish between online and offline password guessing attacks. Note that in the malicious server model, offline password guessing cannot be avoided.

[2] This does not hold for the file encryption but this operation does not need to be non-committing.

[3] Additionally, there are good reasons to not rely on non-committing AEADs [DGRW18, LGR21]. We do not see how similar attacks could be applied to our second approach.

[4] Very recently, Friedrichs, Lehmann and Özbay [FL25] have provided a detailed study on game-based security notions for OPRFs. However, none of them seems directly applicable to our setting.

[5] This is not surprising since BDGHP does not rely on the security of 2HDH at all.

Table 2. Comparison between different cloud storage security models, which Auth notion is required to achieve them, and if for an Auth instantiation password guesses can be made offline (✗) or need an online query per guess (✓). THash refers to an intermediate construction using client-sided and server-sided hashing (see further Appendix D).

Model	Auth notion	Offline query resistance?			
		SHash	THash	2HDH	Sig2HDH
E2EE (C2C) (defined in [BDG+24])	INS-OW	✗	✗	✗	✗
Network Adversary (Section 6.1)	OUT-OW	✗	✗	✗	✓
External Adversary (C2S) (sketch in [BDG+24])	Weak-OW	✗	✓	✓	✓

We formalize such a model, which we call the network adversary model, and define the two properties of confidentiality (Net-Conf) and integrity (Net-Int). We only make one exception: namely, we need to assume that the registration is done via secure channels, similar to what asymmetric password-authenticated key exchange protocols [BM93] require. This seems generally unavoidable for such a model. BDGHP actually provides a sketch for (a variant of) such a model, generally assuming secure channels between clients and server. This is weaker than what we described for our network adversary. Informally, we can separate the different models instantiating the cloud storage scheme with different constructions of our Auth protocol. We give an overview in Table 2 including four possible instantiations for Auth, where Weak-OW is a weaker version of OUT-OW where the adversary does not get access to any client oracles, i.e. can only query the response oracle. It is even simpler to instantiate this notion to ensure that each password-guessing attempt requires an online query.

Eventually, we prove security of the cloud storage scheme relying on OUT-OW for Auth. When instantiated with 2HDH and a signature scheme, we get indeed better bounds regarding password guessing, additionally assuming a variant of the one-more CDH problem.

Explicit Authentication. We additionally formalize a third property in the network adversary setting, namely that of explicit (client) authentication. In the C2C setting, this notion does not make much sense because the server is malicious anyway. However, in combination with the previous property, this allows a server to actively detect online password guessing. This can be very useful in practice because the server can then limit the number of failed login attempts but can also notify clients about successful login attempts. To prove this property for the original cloud storage construction of BDGHP, we additionally require strong unforgeability of the MAC scheme.

Additional Related and Future Work. Earlier works have studied remote file sharing systems [GSMB03], availability [BCQ+13] and metadata hiding [CP20, CHGY22] for cloud services, compelled access security [TMRM18] or end-to-end encrypted web applications [PSV+14]. Importantly, their focus is more on advanced properties and none of these have looked at non-atomic operations. Useful in the context of secure cloud storage are also more fundamental cryptographic concepts like (oblivious) key management [JKR19, CLTY22, DHL22], generally password-based or password-hardened encryption [LESC17, LER+18, BS23] or updatable encryption [BLMR13].

Very recently and probably closest to our work and BDGHP, git services were also analyzed with respect to their practical E2EE [LSTY25]. They tackle additional problems because the effort of changing files should be proportional to the editing difference. Similar to BDGHP, they do not consider adaptive corruptions or stronger properties in weaker models.

For future work, we want to further study the relation of our authentication primitive to OPRFs. Since those are a well-studied cryptographic primitive, achieving post-quantum secure instantiations via OPRFs is the most straightforward path. Our scheme heavily relies on random oracles, so a natural question is whether one can achieve security in the quantum random oracle model. Here, we believe that techniques for tighter security for key exchange protocols [PWZ23, Jae25] may be applicable since they need to solve similar commitment problems.

2 Preliminaries

2.1 Notation

Sets and Algorithms. Unless explicitly stated otherwise, all algorithms are assumed to be probabilistic polynomial time (PPT). For such an algorithm $\mathcal{A}$ invoked on input $(x_1, \ldots)$, the notation $(y_1, \ldots) \xleftarrow{\$} \mathcal{A}(x_1, \ldots)$ denotes that the output tuple $(y_1, \ldots)$ is produced under the internal randomness of $\mathcal{A}$. Oracle access is denoted by $\mathcal{A}^{\mathsf{O}}$, signifying that $\mathcal{A}$ may query the oracle O as a subroutine. We write **return** to denote the return instruction of an oracle or internal subroutine, and **output** for the return of the main procedure, e.g., a game or an adversary's final output. For a probabilistic algorithm $\mathcal{A}$, the notation $x \in \mathcal{A}$ asserts that x belongs to the support of the output distribution of $\mathcal{A}$, i.e., there exists some random tape under which $\mathcal{A}$ outputs x with nonzero probability. If the special symbol $\perp$ is input to an algorithm, we generally define the output of the algorithm to be always $\perp$ as well. Finally, for any Boolean predicate $\mathbb{B}$, we define $[\![\mathbb{B}]\!] \in \{0,1\}$ to be the indicator of its truth value: it equals 1 if $\mathbb{B}$ evaluates to true, and 0 otherwise. For a vector $x = (x_1, \ldots, x_n)$ we use the function append which is defined as $x.\mathsf{append}(y) := (x_1, \ldots, x_n, y)$. The min-entropy of a distribution $\mathcal{D}$ is denoted by $H_\infty(\mathcal{D})$. We use $\mathsf{T}[x]$ to denote a table entry in table T for key x and assume that empty entries are $\perp$. For a set X, $\mathcal{U}(X)$ denotes the uniform distribution over X.

Security Games. We use standard code-based security games [BR06]. A *game* G is a probability experiment in which an adversary $\mathcal{A}$ interacts with an implicit challenger that answers oracle queries issued by $\mathcal{A}$. The game G has one *main procedure* and an arbitrary number of additional *oracle procedures* which describe how these oracle queries are answered. We denote the (binary) output b of game G between a challenger and an adversary $\mathcal{A}$ as $\mathsf{G}(\mathcal{A}) \Rightarrow b$. $\mathcal{A}$ is said to *win* G if $\mathsf{G}(\mathcal{A}) \Rightarrow 1$, or shortly $\mathsf{G} \Rightarrow 1$. Unless otherwise stated, the randomness in the probability term $\mathsf{Pr}[\mathsf{G}(\mathcal{A}) \Rightarrow 1]$ is over all the random coins in game G and adversary $\mathcal{A}$. To provide a cleaner description and avoid repetitions, we sometimes refer to procedures of different games. To call the oracle procedure Oracle of game G on input x, we shortly write $\mathsf{G}.\textsc{Oracle}(x)$. If a game is aborted, the output is 0. For our analysis we rely on the commonly used main difference lemma [BR06].

In the main body of the paper, we sometimes use informal statements of the form "if Scheme S is secure, then Scheme S' [built from S] is secure". Such statements are only meaningful when having an explicit security parameter and should be interpreted in that context. What we actually mean is "for any adversary $\mathcal{A}$ against S', there exists an adversary $\mathcal{B}$ against S such that the advantage of $\mathcal{A}$ is bounded by that of $\mathcal{B}$", specifying a concrete bound B and running time t. For every informal theorem we give such a concrete statement in the appendix.

Random Oracles. We use the random oracle model [BR93] and let a scheme S specify a set $\mathcal{OS}$ of functions, called the oracle space. Hence, we also need to define primitives with respect to an oracle space. A security game samples a function $\mathsf{H} \xleftarrow{\$} \mathcal{OS}$ at random and provides a random oracle RO to the adversary which on input x returns $\mathsf{H}(x)$. To support several oracles, the oracle space might be defined as a cartesian product, e.g. $\mathcal{OS} := \mathcal{OS}_1 \times \mathcal{OS}_2$. The adversary gets access to all sub oracles, e.g. RO_1 for $\mathsf{H}_1 \xleftarrow{\$} \mathcal{OS}_1$ and RO_2 for $\mathsf{H}_2 \xleftarrow{\$} \mathcal{OS}_2$. Since the random oracle is always defined as described here, we do not necessarily define it for each security notion individually. If an algorithm $A(x, \ldots)$ has access to random oracle H, we write $A[\mathsf{H}](x, \ldots)$. In case the oracle space is the empty set, we might ignore it.

2.2 AEAD and MAC Schemes

Definition 1 (AEAD). *A* (nonce-based) authenticated encryption scheme with associated data AEAD *is a tuple* $(\mathsf{Enc}, \mathsf{Dec})$ *along with* $\mathsf{kl}, \mathsf{nl} \in \mathbb{N}$ *defined as follows:*

$\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{k}, \mathsf{n}, \mathsf{m}, \mathsf{ad})$**:** *Given a key* $\mathsf{k} \in \{0,1\}^{\mathsf{kl}}$*, a nonce* $\mathsf{n} \in \{0,1\}^{\mathsf{nl}}$*, a message* $\mathsf{m} \in \{0,1\}^\star$*, and associated data* $\mathsf{ad} \in \{0,1\}^\star$*, the deterministic encryption algorithm outputs a ciphertext* $\mathsf{ct} \in \{0,1\}^\star$*.*

$\mathsf{m} \leftarrow \mathsf{Dec}(\mathsf{k}, \mathsf{n}, \mathsf{ct}, \mathsf{ad})$**:** *Given a key* $\mathsf{k} \in \{0,1\}^{\mathsf{kl}}$*, a nonce* $\mathsf{n} \in \{0,1\}^{\mathsf{nl}}$*, a ciphertext* $\mathsf{ct} \in \{0,1\}^\star$*, and associated data* $\mathsf{ad} \in \{0,1\}^\star$*, the deterministic decryption algorithm outputs a message* $\mathsf{m} \in \{0,1\}^{\mathsf{ml}}$*.*

An AEAD *is* correct *if for all* $\mathsf{k} \in \{0,1\}^{\mathsf{kl}}, \mathsf{n} \in \{0,1\}^{\mathsf{nl}}, \mathsf{m} \in \{0,1\}^{\star}, \mathsf{ad} \in \{0,1\}^{\star}$ *it holds that*

$$\mathsf{Dec}(\mathsf{k}, \mathsf{n}, \mathsf{Enc}(\mathsf{k}, \mathsf{n}, \mathsf{m}, \mathsf{ad}), \mathsf{ad}) = \mathsf{m}.$$

In Appendix A.1, we define several (multi-user) security notions for AEAD schemes that we will use throughout the paper. We also show their relation to standard notions in Appendix A.2. All the used notions can be reduced from standard notions [JSSOW17].

Definition 2 (MAC). *A* message authentication code MAC *is a tuple* $(\mathsf{Tag}, \mathsf{Ver})$ *along with* $\mathsf{kl}, \mathsf{tl} \in \mathbb{N}$ *defined as follows:*

$\mathsf{tag} \leftarrow \mathsf{Tag}(\mathsf{k}, \mathsf{m})$**:** *Given a key* $\mathsf{k} \in \{0,1\}^{\mathsf{kl}}$ *and a message* $\mathsf{m} \in \{0,1\}^{\star}$ *the deterministic tag algorithm outputs a tag* $\mathsf{tag} \in \{0,1\}^{\mathsf{tl}}$*.*

$b \leftarrow \mathsf{Ver}(\mathsf{k}, \mathsf{m}, \mathsf{tag})$**:** *Given a key* $\mathsf{k} \in \{0,1\}^{\mathsf{kl}}$*, a message* $\mathsf{m} \in \{0,1\}^{\star}$*, and a tag* $\mathsf{tag} \in \{0,1\}^{\mathsf{tl}}$ *the deterministic verification algorithm outputs a bit* $b \in \{0,1\}$*.*

A MAC *is* correct *if for all* $\mathsf{k} \in \{0,1\}^{\mathsf{kl}}$ *and* $\mathsf{m} \in \{0,1\}^{\star}$ *it holds that*

$$\mathsf{Ver}(\mathsf{k}, \mathsf{m}, \mathsf{Tag}(\mathsf{k}, \mathsf{m})) = 1.$$

Strong unforgeability for a MAC is defined in Appendix A.3.

Definition 3 (Signatures). *A* digital signature scheme Sig *is a tuple* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ *defined as follows:*

$(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{Gen}$**:** *The probabilistic key generation algorithm outputs a key pair consisting of a public (verification) key* pk *and a secret (signing) key* sk*.*

$\sigma \xleftarrow{\$} \mathsf{Sign}(\mathsf{sk}, \mathsf{m})$**:** *Given a secret key* sk *and a message* $\mathsf{m} \in \{0,1\}^{\star}$ *the signing algorithm outputs a signature* σ*.*

$b \leftarrow \mathsf{Ver}(\mathsf{pk}, \mathsf{m}, \sigma)$**:** *Given a public key* pk*, a message* $\mathsf{m} \in \{0,1\}^{\star}$*, and a signature* σ *the deterministic verification algorithm outputs a bit* $b \in \{0,1\}$*.*

A signature scheme Sig *is* correct *if for all* $(\mathsf{pk}, \mathsf{sk}) \in \mathsf{Gen}$ *and* $\mathsf{m} \in \{0,1\}^{\star}$ *it holds that*

$$\mathsf{Ver}(\mathsf{pk}, \mathsf{m}, \mathsf{Sign}(\mathsf{sk}, \mathsf{m})) = 1.$$

Unforgeability for a signature scheme is defined in Appendix A.3.

2.3 Groups and Diffie-Hellman Problems

Throughout the paper, let $(\mathbb{G}, q, g)$ be the description of a cyclic group of prime order q with generator g. We will use two assumptions to prove security of our group-based authentication scheme which is based on the 2HDH oblivious pseudorandom function. First, we will need (a variant of) the one-more computational Diffie-Hellman (OMCDH) problem, also called chosen-target OMCDH [Bol03]. We adopt this name and define the security game CT-OMCDH in the multi-user setting in Figure 1.

Definition 4 (CT-OMCDH). *The chosen-target one-more Diffie-Hellman problem is defined via the game in Figure 1. We define the advantage of an adversary* $\mathcal{A}$ *as*

$$\mathsf{Adv}^{Q\text{-CT-OMCDH}}_{\mathbb{G},q,g}(\mathcal{A}) := \mathsf{Pr}[Q\text{-CT-OMCDH}_{\mathbb{G},q,g}(\mathcal{A}) \Rightarrow 1]$$

with $Q = (n, k, Q_{\textsc{Comp}}, Q_{\textsc{Exp}}, Q_{\textsc{Check}})$*.*

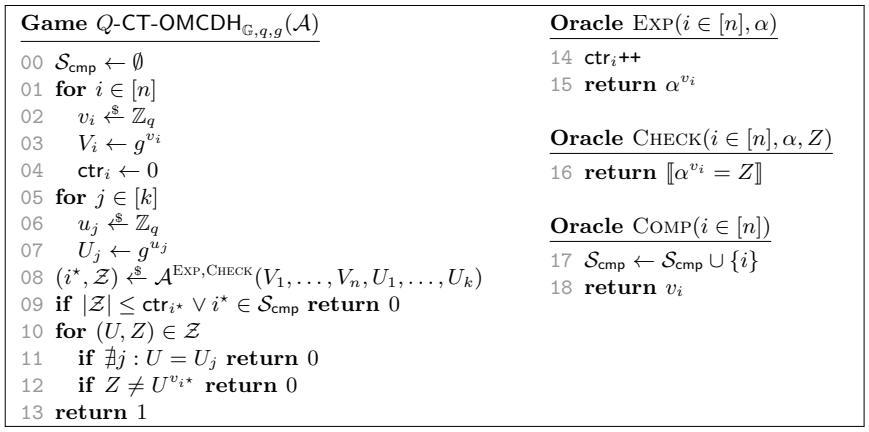

Fig. 1. Game defining the CT-OMCDH problem for a group $(\mathbb{G}, q, g)$ with $Q = (n, k, Q_{\textsc{Comp}}, Q_{\textsc{Exp}}, Q_{\textsc{Check}})$ and adversary $\mathcal{A}$ making at most Q_{O} queries to oracle O.

Remark 1 (Comparison to existing definitions). Compared to the more standard definition of a one-more assumption, e.g., the one analyzed in the generic group model by [BFP21], where the adversary gets an oracle that outputs the CDH solution to arbitrary group elements as input, the adversary in our game is more restricted. In particular, the Exp oracle is only defined with respect to exponents chosen by the game, making it falsifiable. On the other hand, our definition is multi-user, making it seemingly stronger. We note that our definition is similar to the original definition of chosen-target CDH in [Bol03] and also the one used in the characterization in [JX25], where it is simply called OMDH. Their definitions are captured as $(1, k, 0, Q_{\textsc{Exp}}, 0)$-CT-OMCDH, where there is only a single user (and therefore no Comp oracle) and no Check oracle, where the latter acts as a DDH oracle in our setting, the same way it is also used in the strong CDH assumption [ABR01]. Additionally, by guessing the index $i^\star$, we can show that the single-user variant (i.e., where $(n, Q_{\textsc{Comp}}) = (1, 0)$) non-tightly implies the multi-user one, therefore the main difference to previous assumption lies in the additional Check oracle, which typically makes no difference in the algebraic group model, see [BFL20].

2.4 Password Guessing Game

In the application of the cloud storage scheme, the client secret distribution will be a password distribution and the security should depend on the advantage of an adversary guessing passwords correctly. We capture this, similar to [BDG+24], via the game in Figure 2, except that we use the variant where the adversary can corrupt passwords adaptively.

Definition 5 (Password Guessing). *Password guessing for some distribution $\mathcal{PW}$ is defined via the game in Figure 2. We define the advantage of an adversary $\mathcal{A}$ against distribution $\mathcal{PW}$ as*

$$\mathsf{Adv}^{Q\text{-PW-Guess}}_{\mathcal{PW}}(\mathcal{A}) \coloneqq \mathsf{Pr}[Q\text{-PW-Guess}_{\mathcal{PW}}(\mathcal{A}) \Rightarrow 1]$$

with $Q = (n, Q_{\textsc{Test}}, Q_{\textsc{Comp}})$.

3 Authentication Protocols

In this section, we define authentication protocols as an abstract building block to construct cloud storage schemes. This allows us to take a modular approach analyzing different cloud storage security models and their implications for the authentication primitive. We also present different instantiations that can be chosen to plug into the generic cloud storage scheme depending on the security model one wants to achieve.

Game Q-PW-Guess$_{\mathcal{PW}}(\mathcal{A})$	**Oracle** $\textsc{Test}(i \in [n], \mathsf{pw})$
00 $\mathcal{S}_{\mathsf{cmp}} \leftarrow \emptyset$	05 **if** $\mathsf{pw} = \mathsf{pw}_i \wedge i \notin \mathcal{S}_{\mathsf{cmp}}$
01 $\mathsf{win} \leftarrow \mathsf{false}$	06 $\quad \mathsf{win} \leftarrow \mathsf{true}$
02 $(\mathsf{pw}_1, \ldots, \mathsf{pw}_n) \xleftarrow{\$} \mathcal{PW}$	07 **return** $[\![\mathsf{pw} = \mathsf{pw}_i]\!]$
03 $\mathcal{A}^{\textsc{Test},\textsc{Comp}}$	
04 **return** win	**Oracle** $\textsc{Comp}(i \in [n])$
	08 $\mathcal{S}_{\mathsf{cmp}} \leftarrow \mathcal{S}_{\mathsf{cmp}} \cup \{i\}$
	09 **return** pw_i

Fig. 2. Game defining password guessing for a distribution $\mathcal{PW}$ with $Q = (n, Q_{\textsc{Test}}, Q_{\textsc{Comp}})$ and adversary $\mathcal{A}$ making at most Q_{O} queries to oracle O.

3.1 Definitions

Definition 6 (Authentication Protocol). *A two-message* authentication protocol Auth *is a tuple* $(\mathcal{OS}, \mathcal{D}_{\mathsf{S}}, Y, \mathsf{Setup}, \mathsf{Init}, \mathsf{Resp}, \mathsf{Fin}, \mathsf{Full})$ *where* $\mathcal{OS}$ *is the oracle space,* $\mathcal{D}_{\mathsf{S}}$ *a distribution,* Y *(the output space) is a set. For* $\mathsf{H} \in \mathcal{OS}$*, the remaining algorithms are defined as follows:*

$(\mathsf{pk}_{\mathsf{S}}, \mathsf{sk}_{\mathsf{S}}) \xleftarrow{\$} \mathsf{Setup}[\mathsf{H}]$**:** *The randomized server setup algorithm outputs a server long-term public key* pk_{S} *and secret key* sk_{S}*. This algorithm is optional, i.e., it may simply be the empty string* ε*, if no server authentication is needed.*

$(\mathsf{auth}_1, \mathsf{st}) \xleftarrow{\$} \mathsf{Init}[\mathsf{H}](\mathsf{pk}_{\mathsf{S}}, x_{\mathsf{C}}, \mathsf{aux})$**:** *Given a server long-term public key, a client secret* $x_{\mathsf{C}} \in \{0,1\}^\star$ *and some auxiliary information* $\mathsf{aux} \in \{0,1\}^*$*, the initialization algorithm outputs a first authentication message* auth_1 *and a state* st*.*

$\mathsf{auth}_2 \leftarrow \mathsf{Resp}[\mathsf{H}](\mathsf{sk}_{\mathsf{S}}, \mathsf{auth}_1, x_{\mathsf{S}})$**:** *Given a server long-term secret key* sk_{S}*, a first authentication message* auth_1 *and a server's secret* $x_{\mathsf{S}} \in \sup(\mathcal{D}_{\mathsf{S}})$*, the deterministic*[6] *response algorithm outputs a second authentication message.*

$y \leftarrow \mathsf{Fin}[\mathsf{H}](\mathsf{pk}_{\mathsf{S}}, \mathsf{auth}_2, x_{\mathsf{C}}, \mathsf{st})$**:** *Given a server long-term public key* pk_{S}*, a second authentication message* auth_2*, a client secret* $x_{\mathsf{C}} \in \{0,1\}^\star$*, and a state* st*, the deterministic finalization algorithm outputs some* $y \in Y$*.*

$y \leftarrow \mathsf{Full}[\mathsf{H}](\mathsf{pk}_{\mathsf{S}}, x_{\mathsf{C}}, \mathsf{aux}, x_{\mathsf{S}})$**:** *Given a server public key* pk_{S}*, a client secret* $x_{\mathsf{C}} \in \{0,1\}^\star$*, auxiliary information* $\mathsf{aux} \in \{0,1\}^\star$*, and a server secret* $x_{\mathsf{S}} \in \sup(\mathcal{D}_{\mathsf{S}})$*, the deterministic full execution algorithm outputs some* $y \in Y$*.*

Auth *has correctness error*

$$\delta_{\mathsf{Auth}} := \max \Pr\left[\mathsf{Full}[\mathsf{H}](\mathsf{pk}_{\mathsf{S}}, x_{\mathsf{C}}, \mathsf{aux}, x_{\mathsf{S}}) \neq y \;\middle|\; \begin{array}{c} (\mathsf{pk}_{\mathsf{S}}, \mathsf{sk}_{\mathsf{S}}) \xleftarrow{\$} \mathsf{Setup}[\mathsf{H}] \\ (\mathsf{auth}_1, \mathsf{st}) \xleftarrow{\$} \mathsf{Init}[\mathsf{H}](\mathsf{pk}_{\mathsf{S}}, x_{\mathsf{C}}, \mathsf{aux}) \\ \mathsf{auth}_2 \leftarrow \mathsf{Resp}[\mathsf{H}](\mathsf{sk}_{\mathsf{S}}, \mathsf{auth}_1, x_{\mathsf{S}}) \\ y \leftarrow \mathsf{Fin}[\mathsf{H}](\mathsf{pk}_{\mathsf{S}}, \mathsf{auth}_2, x_{\mathsf{C}}, \mathsf{st}) \end{array} \right],$$

where the maximum is taken over $\mathsf{H} \in \mathcal{OS}$*,* $x_{\mathsf{C}} \in \{0,1\}^\star$*,* $x_{\mathsf{S}} \in \sup(\mathcal{D}_{\mathsf{S}})$ *and* $\mathsf{aux} \in \{0,1\}^*$*.*

We define two one-wayness security notions for the Auth primitive. It should be hard to recover the correct output of an Auth session between a client and a server but it should be easy to recognize it ($\textsc{Check}$ oracle). This property should be fulfilled under adaptive corruptions. In the stronger $\mathsf{INS\text{-}OW}$ notion, the adversary is an insider attacker having access to all the server secrets while in the $\mathsf{OUT\text{-}OW}$ they do not have any additional information. It is sufficient for the application to require security for unique auxiliary information per user since this will be the user identifier.

[6] For our instantiations, it is sufficient to have deterministic response algorithms. However, one could also generalize this to probabilistic algorithms.

Game Q-INS-OW$_{\mathsf{Auth},\mathcal{D}_\mathsf{C}}(\mathcal{A})$

00 $\mathsf{H} \xleftarrow{\$} \mathcal{OS}$
01 $\mathcal{S}_{\mathsf{cmp}} \leftarrow \emptyset$
02 $n_r \leftarrow 0$
03 $(\mathsf{pk}_\mathsf{S}, \mathsf{sk}_\mathsf{S}) \xleftarrow{\$} \mathsf{Setup}[\mathsf{H}]$
04 $(x_{\mathsf{C},1}, \ldots, x_{\mathsf{C},Q_{\mathrm{REG}}}) \xleftarrow{\$} \mathcal{D}_\mathsf{C}$
05 **for** $i \in [Q_{\mathrm{REG}}]$
06 $\quad x_{\mathsf{S},i} \xleftarrow{\$} \mathcal{D}_\mathsf{S}$
07 $(y^\star, \mathsf{aux}^\star) \xleftarrow{\$} \mathcal{A}^{\mathrm{REG},\mathrm{INIT},\mathrm{FIN},\mathrm{COMP},\mathrm{CHECK},\mathrm{RO}}(\mathsf{pk}_\mathsf{S}, \mathsf{sk}_\mathsf{S})$
08 **if** $\mathsf{T}_{\mathsf{aux}}[\mathsf{aux}^\star] = \bot$ **return** 0
09 $i^\star \leftarrow \mathsf{T}_{\mathsf{aux}}[\mathsf{aux}^\star]$
10 **return** $[\![y^\star = \mathsf{Full}[\mathsf{H}](\mathsf{pk}_\mathsf{S}, x_{\mathsf{C},i^\star}, \mathsf{aux}^\star, x_{\mathsf{S},i^\star}) \wedge i^\star \notin \mathcal{S}_{\mathsf{cmp}}]\!]$

Oracle REG(aux)

11 **require** $\mathsf{T}_{\mathsf{aux}}[\mathsf{aux}] = \bot$ \\ enforce unique aux
12 n_r++
13 $\mathsf{T}_{\mathsf{aux}}[\mathsf{aux}] \leftarrow n_r$
14 **return** x_{S,n_r}

Oracle INIT(aux, j)

15 **require** $\mathsf{T}_{\mathsf{aux}}[\mathsf{aux}] \neq \bot$
16 $i \leftarrow \mathsf{T}_{\mathsf{aux}}[\mathsf{aux}]$
17 $(\mathsf{auth}_1, \mathsf{st}) \xleftarrow{\$} \mathsf{Init}[\mathsf{H}](\mathsf{pk}_\mathsf{S}, x_{\mathsf{C},i}, \mathsf{aux})$
18 $\mathsf{T}[i,j] \leftarrow (\mathsf{st}, \mathsf{aux})$
19 **return** auth_1

Oracle FIN($\mathsf{auth}_2, i \in [n_r], j$)

20 **if** $\mathsf{T}[i,j] = \bot$
21 $\quad$ **return** $\bot$
22 $(\mathsf{st}, \mathsf{aux}) \leftarrow \mathsf{T}[i,j]$
23 $\mathsf{T}[i,j] \leftarrow \bot$ \\ no state reuse
24 $y \leftarrow \mathsf{Fin}[\mathsf{H}](\mathsf{pk}_\mathsf{S}, \mathsf{auth}_2, x_{\mathsf{C},i}, \mathsf{st})$
25 **if** $y = \mathsf{Full}[\mathsf{H}](\mathsf{pk}_\mathsf{S}, x_{\mathsf{C},i}, \mathsf{aux}, x_{\mathsf{S},i})$
26 $\quad$ **return** $\top$
27 **return** y

Oracle COMP($i \in [n_r]$)

28 $\mathcal{S}_{\mathsf{cmp}} \leftarrow \mathcal{S}_{\mathsf{cmp}} \cup \{i\}$
29 **return** $x_{\mathsf{C},i}$

Oracle CHECK(y, aux)

30 **if** $\mathsf{T}_{\mathsf{aux}}[\mathsf{aux}] = \bot$ **return** 0
31 $i \leftarrow \mathsf{T}_{\mathsf{aux}}[\mathsf{aux}]$
32 **return** $[\![y = \mathsf{Full}[\mathsf{H}](\mathsf{pk}_\mathsf{S}, x_{\mathsf{C},i}, \mathsf{aux}, x_{\mathsf{S},i})]\!]$

Oracle RO(x)

33 **return** $\mathsf{H}(x)$

Fig. 3. Game defining INS-OW security for a two-message authentication scheme $\mathsf{Auth} := (\mathcal{OS}, \mathcal{D}_\mathsf{S}, Y, \mathsf{Setup}, \mathsf{Init}, \mathsf{Resp}, \mathsf{Fin}, \mathsf{Full})$ with $Q = (Q_{\mathrm{REG}}, Q_{\mathrm{COMP}}, Q_{\mathrm{INIT}}, Q_{\mathrm{FIN}}, Q_{\mathrm{CHECK}}, Q_{\mathrm{RO}})$ and adversary $\mathcal{A}$ making at most Q_O queries to oracle O. Differences to OUT-OW (Figure 4) are highlighted in blue.

Definition 7 (One-Wayness). *Output one-wayness is defined via the game in Figure 3 for insider adversaries and in Figure 4 for outsider adversaries. We define the advantage of an adversary $\mathcal{A}$ and a client's secrets distribution $\mathcal{D}_\mathsf{C}$ as*

$$\mathsf{Adv}^{Q\text{-INS-OW}}_{\mathsf{Auth},\mathcal{D}_\mathsf{C}}(\mathcal{A}) := \Pr\left[Q\text{-INS-OW}_{\mathsf{Auth},\mathcal{D}_\mathsf{C}}(\mathcal{A}) \Rightarrow 1\right]$$

with $Q = (Q_{\mathrm{REG}}, Q_{\mathrm{COMP}}, Q_{\mathrm{INIT}}, Q_{\mathrm{FIN}}, Q_{\mathrm{CHECK}}, Q_{\mathrm{RO}})$ *and*

$$\mathsf{Adv}^{Q\text{-OUT-OW}}_{\mathsf{Auth},\mathcal{D}_\mathsf{C}}(\mathcal{A}) := \Pr\left[Q\text{-OUT-OW}_{\mathsf{Auth},\mathcal{D}_\mathsf{C}}(\mathcal{A}) \Rightarrow 1\right]$$

with $Q = (Q_{\mathrm{REG}}, Q_{\mathrm{COMP}}, Q_{\mathrm{INIT}}, Q_{\mathrm{RESP}}, Q_{\mathrm{FIN}}, Q_{\mathrm{CHECK}}, Q_{\mathrm{RO}})$.

Remark 2 (Relation to OPRFs). There is an obvious syntactic similarity to oblivious pseudorandom functions (OPRFs). However, a formal proof that relates the two primitives requires more care. In this work, we are primarily concerned with the exact requirements for the application and its (tight) security proof, which is why we consider (one-way) multi-user definitions with corruptions. We are not aware that such notions exist for OPRFs. Therefore, we think that identifying the exact relation is an important open question, especially for post-quantum secure instantiations.

3.2 Instantiations

We provide three instantiations here, and a fourth one for demonstrative purposes (cf. Table 2) in Appendix D.

Game Q-OUT-OW$_{\mathsf{Auth},\mathcal{D}_\mathsf{C}}(\mathcal{A})$

00 $\mathsf{H} \xleftarrow{\$} \mathcal{OS}$
01 $\mathcal{S}_{\mathsf{cmp}} \leftarrow \emptyset$
02 $n_r \leftarrow 0$
03 $(\mathsf{pk}_\mathsf{S}, \mathsf{sk}_\mathsf{S}) \xleftarrow{\$} \mathsf{Setup}[\mathsf{H}]$
04 $(x_{\mathsf{C},1}, \ldots, x_{\mathsf{C},Q_{\mathrm{REG}}}) \xleftarrow{\$} \mathcal{D}_\mathsf{C}$
05 **for** $i \in [Q_{\mathrm{REG}}]$
06 $\quad x_{\mathsf{S},i} \xleftarrow{\$} \mathcal{D}_\mathsf{S}$
07 $(y^\star, \mathsf{aux}^\star) \xleftarrow{\$} \mathcal{A}^{\mathrm{REG},\mathrm{INIT},\mathrm{RESP},\mathrm{FIN},\mathrm{COMP},\mathrm{CHECK},\mathrm{RO}}(\mathsf{pk}_\mathsf{S})$
08 **if** $\mathsf{T}_{\mathsf{aux}}[\mathsf{aux}^\star] = \bot$ **return** 0
09 $i^\star \leftarrow \mathsf{T}_{\mathsf{aux}}[\mathsf{aux}^\star]$
10 **return** $[\![y^\star = \mathsf{Full}[\mathsf{H}](\mathsf{pk}_\mathsf{S}, x_{\mathsf{C},i^\star}, \mathsf{aux}^\star, x_{\mathsf{S},i^\star}) \wedge i^\star \notin \mathcal{S}_{\mathsf{cmp}}]\!]$

Oracle $\mathrm{REG}(\mathsf{aux})$

11 **require** $\mathsf{T}_{\mathsf{aux}}[\mathsf{aux}] = \bot$ \\ enforce unique aux
12 n_r++
13 $\mathsf{T}_{\mathsf{aux}}[\mathsf{aux}] \leftarrow n_r$

Oracle $\mathrm{INIT}(\mathsf{aux}, j)$

14 **require** $\mathsf{T}_{\mathsf{aux}}[\mathsf{aux}] \neq \bot$
15 $i \leftarrow \mathsf{T}_{\mathsf{aux}}[\mathsf{aux}]$
16 $(\mathsf{auth}_1, \mathsf{st}) \xleftarrow{\$} \mathsf{Init}[\mathsf{H}](\mathsf{pk}_\mathsf{S}, x_{\mathsf{C},i}, \mathsf{aux})$
17 $\mathsf{T}[i,j] \leftarrow (\mathsf{st}, \mathsf{aux})$
18 **return** auth_1

Oracle $\mathrm{RESP}(\mathsf{auth}_1, i \in [n_r])$

19 $\mathsf{auth}_2 \leftarrow \mathsf{Resp}[\mathsf{H}](\mathsf{sk}_\mathsf{S}, \mathsf{auth}_1, x_{\mathsf{S},i})$
20 **return** auth_2

Oracle $\mathrm{FIN}(\mathsf{auth}_2, i \in [n_r], j)$

21 **if** $\mathsf{T}[i,j] = \bot$
22 $\quad$ **return** $\bot$
23 $(\mathsf{st}, \mathsf{aux}) \leftarrow \mathsf{T}[i,j]$
24 $\mathsf{T}[i,j] \leftarrow \bot$ \\ no state reuse
25 $y \leftarrow \mathsf{Fin}[\mathsf{H}](\mathsf{pk}_\mathsf{S}, \mathsf{auth}_2, x_{\mathsf{C},i}, \mathsf{st})$
26 **if** $y = \mathsf{Full}[\mathsf{H}](\mathsf{pk}_\mathsf{S}, x_{\mathsf{C},i}, \mathsf{aux}, x_{\mathsf{S},i})$
27 $\quad$ **return** $\top$
28 **return** y

Oracle $\mathrm{COMP}(i \in [n_r])$

29 $\mathcal{S}_{\mathsf{cmp}} \leftarrow \mathcal{S}_{\mathsf{cmp}} \cup \{i\}$
30 **return** $(x_{\mathsf{C},i}, x_{\mathsf{S},i})$

Oracle $\mathrm{CHECK}(y, \mathsf{aux})$

31 **if** $\mathsf{T}_{\mathsf{aux}}[\mathsf{aux}] = \bot$ **return** 0
32 $i \leftarrow \mathsf{T}_{\mathsf{aux}}[\mathsf{aux}]$
33 **return** $[\![y = \mathsf{Full}[\mathsf{H}](\mathsf{pk}_\mathsf{S}, x_{\mathsf{C},i}, \mathsf{aux}, x_{\mathsf{S},i})]\!]$

Oracle $\mathrm{RO}(x)$

34 **return** $\mathsf{H}(x)$

Fig. 4. Game defining OUT-OW security for a two-message authentication scheme $\mathsf{Auth} := (\mathcal{OS}, \mathcal{D}_\mathsf{S}, Y, \mathsf{Setup}, \mathsf{Init}, \mathsf{Resp}, \mathsf{Fin}, \mathsf{Full})$ with $Q = (Q_{\mathrm{REG}}, Q_{\mathrm{COMP}}, Q_{\mathrm{INIT}}, Q_{\mathrm{RESP}}, Q_{\mathrm{FIN}}, Q_{\mathrm{CHECK}}, Q_{\mathrm{RO}})$ and adversary $\mathcal{A}$ making at most Q_O queries to oracle O. Differences to INS-OW (Figure 3) are highlighted in blue.

Simple Hash. We start with a naive construction that only uses a hash function (modeled as a random oracle). This construction is parametrized by a distribution $\mathcal{D}_\mathsf{S}$ and an output size kl. We then define

$$\mathsf{SHash}[\mathcal{D}_\mathsf{S}, \mathsf{kl}] := (\mathcal{OS}, \mathcal{D}_\mathsf{S}, \{0,1\}^\mathsf{kl}, \varepsilon, \mathsf{Init}, \mathsf{Resp}, \mathsf{Fin}, \mathsf{Full}),$$

for oracle space $\mathcal{OS} := \{\{0,1\}^* \to \{0,1\}^\mathsf{kl}\}$ and the remaining algorithms as defined in Figure 5. Note that it does not have a server setup.

Lemma 1 (Correctness). *It holds that* $\delta_{\mathsf{SHash}[\mathcal{D}_\mathsf{S},\mathsf{kl}]} = 0$.

Theorem 1 (INS-OW of SHash). *For any distribution* $\mathcal{D}_\mathsf{C}$ *and adversary* $\mathcal{A}$ *against* INS-OW *of* $\mathsf{SHash} := \mathsf{SHash}[\mathcal{D}_\mathsf{S}, \mathsf{kl}]$*, there exists a* PW-Guess *adversary* $\mathcal{B}$ *against* $\mathcal{D}_\mathsf{C}$ *with* $t_\mathcal{A} \approx t_\mathcal{B}$ *such that*

$$\mathsf{Adv}^{Q\text{-INS-OW}}_{\mathsf{SHash},\mathcal{D}_\mathsf{C}}(\mathcal{A}) \leq \mathsf{Adv}^{(Q_{\mathrm{REG}},Q_{\mathrm{RO}},Q_{\mathrm{COMP}})\text{-PW-Guess}}_{\mathcal{D}_\mathsf{C}}(\mathcal{B}) + \frac{Q_{\mathrm{CHECK}} + 1}{2^\mathsf{kl} - Q_{\mathrm{CHECK}}}$$

with $Q = (Q_{\mathrm{REG}}, Q_{\mathrm{COMP}}, Q_{\mathrm{INIT}}, Q_{\mathrm{FIN}}, Q_{\mathrm{CHECK}}, Q_{\mathrm{RO}})$.

The proof is given in Appendix B.1.

2HDH. This construction is parametrized by a group $\mathbb{G}$ of prime order q and generator g, and an output size kl. We then define

$$\mathsf{2HDH}[\mathbb{G}, q, g, \mathsf{kl}] := (\mathcal{OS}, \mathcal{U}(\mathbb{Z}_q), \{0,1\}^\mathsf{kl}, \varepsilon, \mathsf{Init}, \mathsf{Resp}, \mathsf{Fin}, \mathsf{Full}),$$

$\underline{\mathsf{SHash.Init}(\varepsilon, x_C, \mathsf{aux})}$
00 $\mathsf{auth}_1 \leftarrow \mathsf{aux}$
01 $\mathsf{st} \leftarrow \mathsf{aux}$
02 **return** $(\mathsf{auth}_1, \mathsf{st})$

$\underline{\mathsf{SHash.Fin}(\varepsilon, \mathsf{auth}_2, x_C, \mathsf{st})}$
03 $y \leftarrow \mathsf{H}(x_C, \mathsf{st}, \mathsf{auth}_2)$
04 **return** y

$\underline{\mathsf{SHash.Resp}(\varepsilon, \mathsf{auth}_1, x_S)}$
05 **return** x_S

$\underline{\mathsf{SHash.Full}(\varepsilon, x_C, \mathsf{aux}, x_S)}$
06 $y \leftarrow \mathsf{H}(x_C, \mathsf{aux}, x_S)$
07 **return** y

Fig. 5. Hash-based scheme $\mathsf{SHash} := (\mathcal{OS}, \mathcal{D}_S, \{0,1\}^{\mathsf{kl}}, \varepsilon, \mathsf{Init}, \mathsf{Resp}, \mathsf{Fin}, \mathsf{Full})$, where $\mathcal{D}_S$ is an arbitrary distribution and the setup is empty.

$\underline{\mathsf{2HDH.Init}(\varepsilon, x_C, \mathsf{aux})}$
00 $r \xleftarrow{\$} \mathbb{Z}_q$
01 $\alpha \leftarrow \mathsf{H}_1(x_C, \mathsf{aux})^r$
02 **return** $(\mathsf{auth}_1 = \alpha, (r, \mathsf{aux}, \alpha))$

$\underline{\mathsf{2HDH.Fin}(\varepsilon, \mathsf{auth}_2, x_C, (r, \mathsf{aux}, \alpha))}$
03 $\beta \leftarrow \mathsf{auth}_2$
04 **if** $\beta \notin \mathbb{G}$ **return** $\perp$
05 $\gamma \leftarrow \beta^{1/r}$
06 $y \leftarrow \mathsf{H}_2(x_C, \mathsf{aux}, \gamma)$
07 **return** y

$\underline{\mathsf{2HDH.Resp}(\varepsilon, \mathsf{auth}_1, x_S)}$
08 **if** $\mathsf{auth}_1 \notin \mathbb{G}$
09 **return** $\perp$
10 $\beta \leftarrow \mathsf{auth}_1^{x_S}$
11 **return** β

$\underline{\mathsf{2HDH.Full}(\varepsilon, x_C, \mathsf{aux}, x_S)}$
12 $y \leftarrow \mathsf{H}_2(x_C, \mathsf{aux}, \mathsf{H}_1(x_C, \mathsf{aux})^{x_S})$
13 **return** y

$\underline{\mathsf{Sig2HDH.Setup}}$
14 $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{Gen}$
15 **return** $(\mathsf{pk}, \mathsf{sk})$

$\underline{\mathsf{Sig2HDH.Fin}(\mathsf{pk}, \mathsf{auth}_2, x_C, (r, \mathsf{aux}, \alpha))}$
16 $(\beta, \sigma) \leftarrow \mathsf{auth}_2$
17 **if** $\mathsf{Ver}(\mathsf{pk}, (\alpha, \beta), \sigma) \neq 1$ **return** $\perp$
18 **return** $\mathsf{2HDH.Fin}(\varepsilon, \beta, x_C, (r, \mathsf{aux}, \alpha))$

$\underline{\mathsf{Sig2HDH.Resp}(\mathsf{sk}, \mathsf{auth}_1, x_S)}$
19 $\beta \leftarrow \mathsf{2HDH.Resp}(\varepsilon, \mathsf{auth}_1, x_S)$
20 **if** $\beta = \perp$ **return** $\perp$
21 $\sigma \xleftarrow{\$} \mathsf{Sign}(\mathsf{sk}, (\alpha, \beta))$
22 **return** $\mathsf{auth}_2 = (\beta, \sigma)$

Fig. 6. Diffie-Hellman based authentication scheme $\mathsf{2HDH}[\mathbb{G}, q, g, \mathsf{kl}] := (\mathcal{OS}, \mathcal{U}(\mathbb{Z}_q), \{0,1\}^{\mathsf{kl}}, \varepsilon, \mathsf{Init}, \mathsf{Resp}, \mathsf{Fin}, \mathsf{Full})$ and its variant $\mathsf{Sig2HDH}[\mathsf{Sig}, \mathbb{G}, q, g, \mathsf{kl}] := (\mathcal{OS}, \mathcal{U}(\mathbb{Z}_q), \{0,1\}^{\mathsf{kl}}, \mathsf{Setup}, \mathsf{2HDH.Init}, \mathsf{Resp}, \mathsf{Fin}, \mathsf{2HDH.Full})$ that additionally uses a signature scheme $\mathsf{Sig} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$.

for oracle space $\mathcal{OS} := \{\{0,1\}^* \to \mathbb{G}\} \times \{\{0,1\}^* \to \{0,1\}^{\mathsf{kl}}\}$ and the remaining algorithms as defined in Figure 6.

Lemma 2 (Correctness). *It holds that* $\delta_{\mathsf{2HDH}[\mathbb{G},q,g,\mathsf{kl}]} = 0$.

Theorem 2 (INS-OW of 2HDH). *For any distribution* $\mathcal{D}_C$ *and adversary* $\mathcal{A}$ *against* INS-OW *of* $\mathsf{2HDH} := \mathsf{2HDH}[\mathbb{G}, q, g, \mathsf{kl}]$ *there exists a* PW-Guess *adversary* $\mathcal{B}$ *against* $\mathcal{D}_C$ *with* $t_{\mathcal{A}} \approx t_{\mathcal{B}}$ *such that*

$$\mathsf{Adv}^{Q\text{-INS-OW}}_{\mathsf{2HDH},\mathcal{D}_C}(\mathcal{A}) \leq \mathsf{Adv}^{(Q_{\mathrm{REG}},Q_{\mathrm{RO}},Q_{\mathrm{COMP}})\text{-PW-Guess}}_{\mathcal{D}_C}(\mathcal{B}) + \frac{Q_{\mathrm{CHECK}}+1}{2^{\mathsf{kl}} - Q_{\mathrm{CHECK}}}$$

with $Q = (Q_{\mathrm{REG}}, Q_{\mathrm{COMP}}, Q_{\mathrm{INIT}}, Q_{\mathrm{FIN}}, Q_{\mathrm{CHECK}}, Q_{\mathrm{RO}})$.

The proof is given in Appendix B.2.

2HDH WITH SIGNATURE. To the previous construction we add a server setup that generates a signature key pair and we let the server sign its message. Therefore, this construction is parametrized by a signature scheme Sig, a group $\mathbb{G}$ of prime order q and generator g, and an output size kl. We then define

$$\mathsf{Sig2HDH}[\mathsf{Sig}, \mathbb{G}, q, g, \mathsf{kl}] := (\mathcal{OS}, \mathcal{U}(\mathbb{Z}_q), \{0,1\}^{\mathsf{kl}}, \mathsf{Setup}, \mathsf{2HDH.Init}, \mathsf{Resp}, \mathsf{Fin}, \mathsf{2HDH.Full}),$$

for oracle space $\mathcal{OS} := \{\{0,1\}^* \to \mathbb{G}\} \times \{\{0,1\}^* \to \{0,1\}^{\mathsf{kl}}\}$ and the remaining algorithms as defined in Figure 6.

Lemma 3 (Correctness). *Let* Sig *be correct. Then it holds that* $\delta_{\mathsf{Sig2HDH}[\mathsf{Sig},\mathbb{G},q,g,\mathsf{kl}]} = 0$.

Theorem 3 (OUT-OW of Sig2HDH). *For any distribution* $\mathcal{D}_{\mathsf{C}}$ *and adversary* $\mathcal{A}$ *against* OUT-OW *of* $\mathsf{Sig2HDH} := \mathsf{Sig2HDH}[\mathsf{Sig},\mathbb{G},q,g,\mathsf{kl}]$*, there exist a* UF *adversary* $\mathcal{B}$ *against* Sig*, a* CT-OMCDH *adversary* $\mathcal{C}$ *against* $(\mathbb{G},q,g)$*, and a* PW-Guess *adversary* $\mathcal{D}$ *against* $\mathcal{D}_{\mathsf{C}}$ *with* $t_{\mathcal{A}} \approx t_{\mathcal{B}} \approx t_{\mathcal{C}} \approx t_{\mathcal{D}}$ *such that*

$$\begin{aligned}\mathsf{Adv}^{Q\text{-}\mathsf{OUT}\text{-}\mathsf{OW}}_{\mathsf{Sig2HDH},\mathcal{D}_{\mathsf{C}}}(\mathcal{A}) \leq\ & \mathsf{Adv}^{Q_{\mathrm{RESP}}\text{-}\mathsf{UF}}_{\mathsf{Sig}}(\mathcal{B}) + \mathsf{Adv}^{(Q_{\mathrm{REG}},Q_{\mathrm{RO}},Q_{\mathrm{COMP}},Q_{\mathrm{RESP}},Q_{\mathrm{RO}}+Q_{\mathrm{CHECK}})\text{-}\mathsf{CT}\text{-}\mathsf{OMCDH}}_{\mathbb{G},q,g}(\mathcal{C}) \\ & + \mathsf{Adv}^{(Q_{\mathrm{REG}},Q_{\mathrm{RESP}},Q_{\mathrm{COMP}})\text{-}\mathsf{PW}\text{-}\mathsf{Guess}}_{\mathcal{D}_{\mathsf{C}}}(\mathcal{D}) + \frac{Q_{\mathrm{CHECK}}+1}{2^{\mathsf{kl}}-Q_{\mathrm{CHECK}}} + \frac{Q_{\mathrm{RO}}^2}{2^{\mathsf{kl}}}\end{aligned}$$

with $Q = (Q_{\mathrm{REG}}, Q_{\mathrm{COMP}}, Q_{\mathrm{INIT}}, Q_{\mathrm{RESP}}, Q_{\mathrm{FIN}}, Q_{\mathrm{CHECK}}, Q_{\mathrm{RO}})$.

The proof is given in Appendix B.3.

Remark 3. 2HDH without the signature scheme is actually not OUT-OW secure in the same sense as Sig2HDH. The issue is that 2HDH always allows for offline guessing attacks since the client has no way to authenticate the server and the FIN oracle will leak information about the client secret. More concretely, the adversary can simply pick a server secret and send β. The client will output some y with respect to this server secret. Now the adversary can check offline which client secret produces the value y. By adding server authentication, we protect against this type of attack. We still expect that the adversary can simply make an online guess of the client secret, as reflected in the password guessing advantage of the bound that now depends on RESP queries. The CT-OMCDH assumption then ensures that no further guess can be made.

4 Cloud Storage Schemes

In this section we define a cloud storage following [BDG+24]. The notation is mainly taken from the same work but we introduce small conceptual changes in the security notion to clarify the model.

4.1 Syntax

The definitions in this section are taken from [BDG+24].

CLOUD STORAGE SCHEME. A cloud storage scheme CS is a tuple of an oracle space $\mathcal{OS}$, a server setup algorithm, and seven two-party protocols:

$$\mathsf{CS} = (\mathcal{OS}, \mathsf{Setup}, \Pi_{\mathsf{areg}}, \Pi_{\mathsf{auth}}, \Pi_{\mathsf{put}}, \Pi_{\mathsf{upd}}, \Pi_{\mathsf{get}}, \Pi_{\mathsf{share}}, \Pi_{\mathsf{accept}}) \ .$$

In contrast to [BDG+24], we add a server setup algorithm Setup. This probabilistic algorithm outputs a public key pk_{S} and a secret key sk_{S}. The input and output interfaces of the interactive protocols are given in Table 3.

PROTOCOL EXECUTION. The execution of a two-party protocol Π proceeds in $\Pi.\mathsf{SN} \in \mathbb{N}$ sequential steps, alternating between the client and the server. Each step r is denoted as $\Pi^{\mathsf{C}:\mathsf{r}}$ or $\Pi^{\mathsf{S}:\mathsf{r}}$ depending on the client (C) or the server (S) executing it. The syntax for a protocol step is

$$(\mathsf{st}_{\mathsf{P}}, \mathsf{st}^{\mathsf{tmp}}_{\mathsf{P}}, \mathsf{out}_{\mathsf{P}}, \mathsf{m}_{\mathsf{out}}) \leftarrow \Pi^{\mathsf{P}:\mathsf{r}}[\mathsf{H}](\mathsf{st}_{\mathsf{P}}, \mathsf{st}^{\mathsf{tmp}}_{\mathsf{P}}, [\mathsf{in}_{\mathsf{P}},]\mathsf{m}_{\mathsf{in}}),$$

where $\mathsf{H} \in \mathcal{OS}$, st_{P} is the persistent state of party P, $\mathsf{st}^{\mathsf{tmp}}_{\mathsf{P}}$ is their temporary state (only for that particular protocol execution), in_{P} is the local input of party P which is only used for the first step $\mathsf{r} = 1$[7], m_{in} and

[7] Brackets "[...]" indicate that the input is optional; in this case, only for the first step.

Table 3. Input-output semantics of the subprotocols in the cloud storage system CS from a client's perspective.

Protocol	Input	Output	Description
Π_{areg}	$\mathsf{aid}, \mathsf{pw}, \mathsf{pk_S}$	ε	Account registration
Π_{auth}	$\mathsf{aid}, \mathsf{pw}, \mathsf{pk_S}$	ε	Authentication and start of a new session
Π_{put}	fid, f	ε	File upload
Π_{upd}	fid, f	ε	File update
Π_{get}	fid	f	File download
Π_{share}	$\mathsf{fid}, \mathsf{raid}$	oob	Out-of-band file sharing
Π_{accept}	$\mathsf{fid}, \mathsf{oob}$	ε	Accept shared file

Procedure $\mathsf{Exec}_{\Pi[\mathsf{H}]}(\mathsf{st_C}, \mathsf{st_S}, \mathsf{in_C}, \mathsf{in_S})$:

00 $(\mathsf{st_C}, \mathsf{st_C^{tmp}}, \mathsf{out_C}, \mathsf{m}[1]) \xleftarrow{\$} \Pi^{(\mathsf{C}:1)}[\mathsf{H}](\mathsf{st_C}, \varepsilon, \mathsf{in_C}, \varepsilon)$
01 $(\mathsf{st_S}, \mathsf{st_S^{tmp}}, \mathsf{out_S}, \mathsf{m}[2]) \xleftarrow{\$} \Pi^{(\mathsf{S}:1)}[\mathsf{H}](\mathsf{st_S}, \varepsilon, \mathsf{in_S}, \mathsf{m}[1])$
02 for $i = 3, \ldots, \Pi.\mathsf{SN}$:
03 $\quad r \leftarrow \lceil i/2 \rceil$
04 $\quad$ **if** i odd: $\mathsf{P} \leftarrow \mathsf{C}$ **else** $\mathsf{P} \leftarrow \mathsf{S}$
05 $\quad$ **if** $\mathsf{out_P}.\mathsf{decision} = \bot$
06 $\quad\quad (\mathsf{st_P}, \mathsf{st_P^{tmp}}, \mathsf{out_P}, \mathsf{m}[i]) \xleftarrow{\$} \Pi^{(\mathsf{P}:r)}[\mathsf{H}](\mathsf{st_P}, \mathsf{st_P^{tmp}}, \mathsf{m}[i-1])$
07 **return** $(\mathsf{m}, \mathsf{st_C}, \mathsf{st_S}, \mathsf{out_C}, \mathsf{out_S})$

Fig. 7. Execution of protocol $\Pi[\mathsf{H}]$ between client C and server S.

$\mathsf{m_{out}}$ denote the input and output message of the protocol, and $\mathsf{out_P}$ is the local output of party P. If inputs/outputs are not used, they are set to the empty string ε. This is for example the case for $\mathsf{m_{in}}$ for the first client step of an execution or for $\mathsf{out_P}$ for every non-terminating step of the protocol. By assumption, a client C always initiates a protocol execution, i.e. sends the first message.

Termination. Return statements of a party $\mathsf{P} \in \{\mathsf{C}, \mathsf{S}\}$ are captured by the two operations $\mathsf{success_P}$ and $\mathsf{fail_P}$ defined as

$$\mathsf{success_P}(\mathsf{m_P}) : \mathsf{out_P}.\mathsf{decision} \leftarrow \mathsf{true};\ \textbf{return}\ (\mathsf{st_P}, \mathsf{st_P^{tmp}}, \mathsf{out_P}, \mathsf{m_P}),$$
$$\mathsf{fail_P} : \mathsf{out_P}.\mathsf{decision} \leftarrow \mathsf{false};\ \textbf{return}\ (\mathsf{st_P}, \mathsf{st_P^{tmp}}, \mathsf{out_P}, \bot).$$

Further, $\mathsf{success_P}(\top)$ returns $\top$ and failing sub procedures lead to a failing main procedure.

Full Execution and Correctness. In Figure 7, we recall the full, i.e. atomic, execution of a two-party protocol where the protocol between client and server is executed with the corrected messages in the correct order from [BDG+24]. This in particular used to define correctness.

Definition 8 (Corr). *Correctness for a cloud storage scheme* CS *is defined via the game in Figure 8. We define the advantage of an adversary* $\mathcal{A}$ *as*

$$\mathsf{Adv}^{Q\text{-}\mathsf{Corr}}_{\mathsf{CS}}(\mathcal{A}) := \mathsf{Pr}\left[Q\text{-}\mathsf{Corr}_{\mathsf{CS}}(\mathcal{A}) \Rightarrow 1\right],$$

for $Q = (Q_{\mathrm{Reg}}, Q_{\mathrm{Auth}}, Q_{\mathrm{Put}}, Q_{\mathrm{Upd}}, Q_{\mathrm{Get}}, Q_{\mathrm{Share}}, Q_{\mathrm{Acpt}}, Q_{\mathrm{RO}})$.

4.2 Security Definitions

We define confidentiality and integrity for cloud storage schemes against *adaptive* compromises. Compared to [BDG+24], we make small changes in notation. Our games are given in Figures 9 and 10.

Instead of using indices for users, we only act on the level of account identifiers aid which were also used in [BDG+24]. This is closer to practice since in practice only the account identifier is used as a handle, e.g.

Game $Q\text{-Corr}_{\mathsf{CS}}(\mathcal{A})$

00 $\mathsf{H} \xleftarrow{\$} \mathcal{OS}$
01 $\mathsf{win} \leftarrow \mathsf{false}$
02 $n_c \leftarrow 0$ \\ session counter
03 $(\mathsf{pk_S}, \mathsf{sk_S}) \xleftarrow{\$} \mathsf{Setup}$
04 $\mathsf{st_S.sk} \leftarrow \mathsf{sk_S}$
05 $\mathcal{A}^{\mathcal{O}}(\mathsf{pk_S}, \mathsf{sk_S})$
06 **return** win

Procedure $\mathsf{RunProt}(\mathsf{prot}, \mathsf{i_c}, \mathsf{in_C}, \mathsf{in_S})$

07 **if** $\mathsf{prot} \in \{\mathsf{areg}, \mathsf{auth}\}$
08 $\quad \mathsf{st_C} \leftarrow \varepsilon$
09 **else**
10 $\quad (\mathsf{aid}, \mathsf{st_C}) \leftarrow \mathsf{T_{st_C}}[\mathsf{i_c}]$
11 $(\mathsf{m}, \mathsf{st_C}, \mathsf{st_S}, \mathsf{out_C}, \mathsf{out_S}) \xleftarrow{\$} \mathsf{Exec}_{\Pi_{\mathsf{prot}}}(\mathsf{st_C}, \mathsf{st_S}, \mathsf{in_C}, \mathsf{in_S})$
12 **if** $\mathsf{out_C.decision} \wedge \mathsf{out_S.decision}$
13 $\quad$ **if** $\mathsf{prot} = \mathsf{areg}$
14 $\quad\quad \mathsf{T_{uaid}}[\mathsf{in_C.aid}] \leftarrow \mathsf{true}$
15 $\quad$ **if** $\mathsf{prot} = \mathsf{auth}$
16 $\quad\quad \mathsf{T_{st_C}}[\mathsf{i_c}] \leftarrow (\mathsf{in_C.aid}, \mathsf{st_C})$
17 **if** $\mathsf{prot} \notin \{\mathsf{areg}, \mathsf{auth}\}$
18 $\quad \mathsf{T_{st_C}}[\mathsf{i_c}] \leftarrow (\mathsf{aid}, \mathsf{st_C})$
19 **return** $(\mathsf{m}, \mathsf{st_C}, \mathsf{st_S}, \mathsf{out_C}, \mathsf{out_S})$

Oracle $\textsc{Reg}(\mathsf{in_C}, \mathsf{in_S})$

20 **require** $\mathsf{T_{uaid}}[\mathsf{in_C.aid}] = \bot$
21 $\mathsf{T_{pw}}[\mathsf{in_C.aid}] \leftarrow \mathsf{in_C.pw}$
22 **return** $\mathsf{RunProt}(\mathsf{areg}, \varepsilon, \mathsf{in_C}, \mathsf{in_S})$

Oracle $\textsc{Auth}(\mathsf{in_C}, \mathsf{in_S})$

23 **require** $\mathsf{T_{uaid}}[\mathsf{in_C.aid}] = \mathsf{true}$
24 n_c++
25 $\mathsf{i_c} \leftarrow n_c$
26 **return** $\mathsf{RunProt}(\mathsf{auth}, \mathsf{i_c}, \mathsf{in_C}, \mathsf{in_S})$

Oracle $\textsc{Put}(\mathsf{i_c} \in [n_c], \mathsf{in_C}, \mathsf{in_S})$

27 **require** $\mathsf{T_f}[\mathsf{in_C.fid}] = \bot$
28 $(\mathsf{m}, \mathsf{st_C}, \mathsf{st_S}, \mathsf{out_C}, \mathsf{out_S}) \xleftarrow{\$} \mathsf{RunProt}(\mathsf{put}, \mathsf{i_c}, \mathsf{in_C}, \mathsf{in_S})$
29 **if** $\mathsf{out_S.decision}$
30 $\quad \mathsf{U} \leftarrow \{\mathsf{T_{st_C}}[\mathsf{i_c}].\mathsf{aid}\}$
31 $\quad \mathsf{F} \leftarrow \mathsf{in_C.file}$
32 $\quad \mathsf{T_f}[\mathsf{in_C.fid}] \leftarrow (\mathsf{U}, \mathsf{F})$
33 **return** $(\mathsf{m}, \mathsf{st_C}, \mathsf{st_S}, \mathsf{out_C}, \mathsf{out_S})$

Oracle $\textsc{Upd}(\mathsf{i_c} \in [n_c], \mathsf{fid}, \mathsf{file})$

34 **require** $\mathsf{i_c} \in \mathsf{T_{st_C}} \wedge \mathsf{T_{st_C}}[\mathsf{in_C.aid}] \in \mathsf{T_f}[\mathsf{in_C.fid}].\mathsf{U}$
35 $(\mathsf{m}, \mathsf{st_C}, \mathsf{st_S}, \mathsf{out_C}, \mathsf{out_S}) \xleftarrow{\$} \mathsf{RunProt}(\mathsf{update}, \mathsf{i_c}, \mathsf{in_C}, \mathsf{in_S})$
36 **if** $\mathsf{out_S.decision}$
37 $\quad \mathsf{T_f}[\mathsf{in_C.fid}].\mathsf{F} \leftarrow \mathsf{in_C.file}$
38 **return** $(\mathsf{m}, \mathsf{st_C}, \mathsf{st_S}, \mathsf{out_C}, \mathsf{out_S})$

Oracle $\textsc{Get}(\mathsf{i_c} \in [n_c], \mathsf{in_C}, \mathsf{in_S})$

39 **require** $\mathsf{T_{st_C}}[\mathsf{i_c}].\mathsf{aid} \in \mathsf{T_f}[\mathsf{in_C.fid}].\mathsf{U}$
40 $(\mathsf{m}, \mathsf{st_C}, \mathsf{st_S}, \mathsf{out_C}, \mathsf{out_S}) \xleftarrow{\$} \mathsf{RunProt}(\mathsf{get}, \mathsf{i_c}, \mathsf{in_C}, \mathsf{in_S})$
41 **if** $\mathsf{out_C.decision} \wedge \mathsf{out_C.file} \neq \mathsf{T_f}[\mathsf{in_C.fid}].\mathsf{F}$
42 $\quad \mathsf{win} \leftarrow \mathsf{true}$ \\ get wrong file
43 **if** $\mathsf{out_C.decision} = \mathsf{false} \wedge \mathsf{in_C.fid} \in \mathsf{T_f}[\mathsf{in_C.fid}]$
44 $\quad \mathsf{win} \leftarrow \mathsf{true}$ \\ do not get file
45 **return** $(\mathsf{m}, \mathsf{st_C}, \mathsf{st_S}, \mathsf{out_C}, \mathsf{out_S})$

Oracle $\textsc{Share}(\mathsf{i_c} \in [n_c], \mathsf{in_C}, \mathsf{in_S})$

46 **require** $\mathsf{T_{st_C}}[\mathsf{i_c}].\mathsf{aid} \in \mathsf{T_f}[\mathsf{in_C.fid}].\mathsf{U}$
47 $(\mathsf{m}, \mathsf{st_C}, \mathsf{st_S}, \mathsf{out_C}, \mathsf{out_S}) \xleftarrow{\$} \mathsf{RunProt}(\mathsf{share}, \mathsf{i_c}, \mathsf{in_C}, \mathsf{in_S})$
48 **if** $\mathsf{out_C.decision}$
49 $\quad \mathcal{S}_{\mathsf{shr}} \leftarrow \mathcal{S}_{\mathsf{shr}} \cup \{(\mathsf{in_C.raid}, \mathsf{in_C.fid})\}$
50 $\quad \mathsf{T_{oob}}[(\mathsf{in_C.raid}, \mathsf{in_C.fid})] \leftarrow \mathsf{out_C.oob}$
51 **return** $(\mathsf{m}, \mathsf{st_C}, \mathsf{st_S}, \mathsf{out_C}, \mathsf{out_S})$

Oracle $\textsc{Acpt}(\mathsf{i_c} \in [n_c], \mathsf{in_C}, \mathsf{in_S})$

52 **require** $\mathsf{T_{oob}}[(\mathsf{T_{st_C}}[\mathsf{i_c}].\mathsf{aid}, \mathsf{in_C.fid})] \neq \bot$
53 $\mathsf{in_C.oob} \leftarrow \mathsf{T_{oob}}[(\mathsf{T_{st_C}}[\mathsf{i_c}].\mathsf{aid}, \mathsf{in_C.fid})]$
54 $(\mathsf{m}, \mathsf{st_C}, \mathsf{st_S}, \mathsf{out_C}, \mathsf{out_S}) \xleftarrow{\$} \mathsf{RunProt}(\mathsf{accept}, \mathsf{i_c}, \mathsf{in_C}, \mathsf{in_S})$
55 **if** $\mathsf{out_S.decision}$
56 $\quad \mathsf{T_f}[\mathsf{in_C.fid}].\mathsf{U} \leftarrow \mathsf{T_f}[\mathsf{in_C.fid}].\mathsf{U} \cup \{\mathsf{T_{st_C}}[\mathsf{i_c}].\mathsf{in_C.aid}\}$
57 **if** $\mathsf{out_C.decision}$
58 $\quad \mathsf{T_{oob}}[(\mathsf{T_{st_C}}[\mathsf{i_c}].\mathsf{aid}, \mathsf{in_C.fid})] \leftarrow \bot$
59 **if** $\mathsf{out_C.decision} = \mathsf{false} \wedge (\mathsf{T_{st_C}}[\mathsf{i_c}].\mathsf{in_C.aid}, \mathsf{in_C.fid}) \in \mathcal{S}_{\mathsf{shr}}$
60 $\quad \mathsf{win} \leftarrow \mathsf{true}$ \\ does not accept shared file
61 **return** $(\mathsf{m}, \mathsf{st_C}, \mathsf{st_S}, \mathsf{out_C}, \mathsf{out_S})$

Oracle $\mathrm{RO}(x)$

62 **return** $\mathsf{H}(x)$

Fig. 8. Correctness game for a cloud storage scheme CS. Adversary $\mathcal{A}$ has access to oracles $\mathcal{O} = \{\textsc{Reg}, \textsc{Auth}, \textsc{Put}, \textsc{Upd}, \textsc{Get}, \textsc{Share}, \textsc{Acpt}, \mathrm{RO}\}$ making at most Q_{O} queries to oracle O.

the e-mail address or the username of a client. Note that we still allow for several registration attempts but after the registration succeeds the account identifier must be unique. Not using indices allows us to drop the set $\mathcal{S}_{\mathsf{hon}}$ since we track correctly registered aids and therefore it is sufficient to keep track of the accounts that were compromised.

The input interface of oracles is also changed compared to [BDG+24]. Instead of using a generic input $\mathsf{in_C}$, we clarify what the input should include by making the components explicit which provides an easier understanding of the notion.

We also need to take care of the server setup procedure we introduced. In all security notions, we assume that the server setup is executed honestly. However, in case the server is compromised (E2EE notions), the adversary learns the secret component of the server's setup.[8]

We change the confidentiality from [BDG+24] by marking all challenge file owners as challenge owners in the accept oracle (Line 75).[9] In BDGHP, no owners are added in the accept oracle but only in the share oracle which allows an adversary to first share a (non-challenge) file with a compromised user, then register it as a challenge file by updating the content, and finally accept it by the compromised user. In such a case, the compromised user was not registered as a challenge file owner and the adversary would win the game.

Lastly, for the integrity notion we capture the winning condition explicitly by requiring the output file to be not $\perp$. In the notion of [BDG+24], this is only implicitly captured via the decision property. We note that this may be problematic since the property is specified by the protocol itself. A badly designed protocol can always set the variable to true, bypassing the intentions of the security notion.[10]

Later, in Section 6, we also define a new security notion that captures security guarantees for honest servers.

Definition 9 (Conf). *Confidentiality for a cloud storage scheme* CS *is defined via the game in Figure 9. We define the advantage of an adversary* $\mathcal{A}$ *and a password distribution* $\mathcal{PW}$ *as*

$$\mathsf{Adv}^{Q\text{-}\mathsf{Conf}}_{\mathsf{CS},\mathcal{PW}}(\mathcal{A}) \coloneqq 2\,\mathsf{Pr}\left[Q\text{-}\mathsf{Conf}_{\mathsf{CS},\mathcal{PW}}(\mathcal{A}) \Rightarrow 1\right] - 1,$$

for $Q = (Q_{\textsc{Step}}, Q_{\textsc{Reg}_1}, Q_{\textsc{Auth}_1}, Q_{\textsc{Put}_1}, Q_{\textsc{Upd}_1}, Q_{\textsc{Get}_1}, Q_{\textsc{Share}_1}, Q_{\textsc{Acpt}_1}, Q_{\textsc{Chall}}, Q_{\textsc{Comp}}, Q_{\textsc{RO}})$.

Confidentiality captures that an adversary does not learn anything about honest users' files unless they are shared with a compromised party. This needs to hold even if the adversary can control the server and adaptively compromise clients. Further, protocols can be executed non-atomic and each step of the protocols can be interleaved arbitrarily.

Definition 10 (Int). *Integrity for a cloud storage scheme* CS *is defined via the game in Figure 10. We define the advantage of an adversary* $\mathcal{A}$ *as*

$$\mathsf{Adv}^{Q\text{-}\mathsf{Int}}_{\mathsf{CS},\mathcal{PW}}(\mathcal{A}) \coloneqq \mathsf{Pr}\left[Q\text{-}\mathsf{Int}_{\mathsf{CS},\mathcal{PW}}(\mathcal{A}) \Rightarrow 1\right],$$

for $Q = (Q_{\textsc{Step}}, Q_{\textsc{Reg}_1}, Q_{\textsc{Auth}_1}, Q_{\textsc{Put}_1}, Q_{\textsc{Upd}_1}, Q_{\textsc{Get}_1}, Q_{\textsc{Share}_1}, Q_{\textsc{Acpt}_1}, Q_{\textsc{Comp}}, Q_{\textsc{RO}})$.

The adversary model for integrity is the same as for confidentiality. Integrity captures that an attacker cannot modify honest client's files or create new valid files as long as the files are not shared with compromised parties.

5 Construction

We define the scheme $\mathsf{CS}[\mathsf{Auth}, \mathsf{AEAD}, \mathsf{MAC}]$ based on three primitives: $\mathsf{Auth}, \mathsf{AEAD}$, and MAC. Let $\mathsf{Auth} \coloneqq (\mathcal{OS}_{\mathsf{Auth}}, \mathcal{D}_{\mathsf{S}}, Y, \mathsf{Setup}, \mathsf{Init}, \mathsf{Resp}, \mathsf{Fin}, \mathsf{Full})$. Then we define the oracle space as

$$\mathcal{OS} \coloneqq \{f : \{0,1\}^\star \to \{0,1\}^{\mathsf{kl}}\} \times \mathcal{OS}_{\mathsf{Auth}}$$

[8] We could strengthen this notion to have the adversary pick the server's long-term secret. In this case, the adversary would provide a secret key to the game, which then simulates the communication using the potentially maliciously created secret key.

[9] This change as well as the following description of the possible attack against BDGHP's security model was pointed out by a EUROCRYPT'26 reviewer.

[10] While such a protocol would not be considered correct, we still think that it makes definitions less intuitive. More concretely, the correctness game of [BDG+24] would never check whether an uploaded file cannot be retrieved (line 20 in Figure 3 in the ePrint version), but only whether the user retrieved an unexpected file (line 21). Since $\perp$ counts as an unexpected file, this works. However, we believe that such a subtlety could be avoided by making the decision bit dependent on some event, as we do for the integrity notion by requiring the file to be not $\perp$.

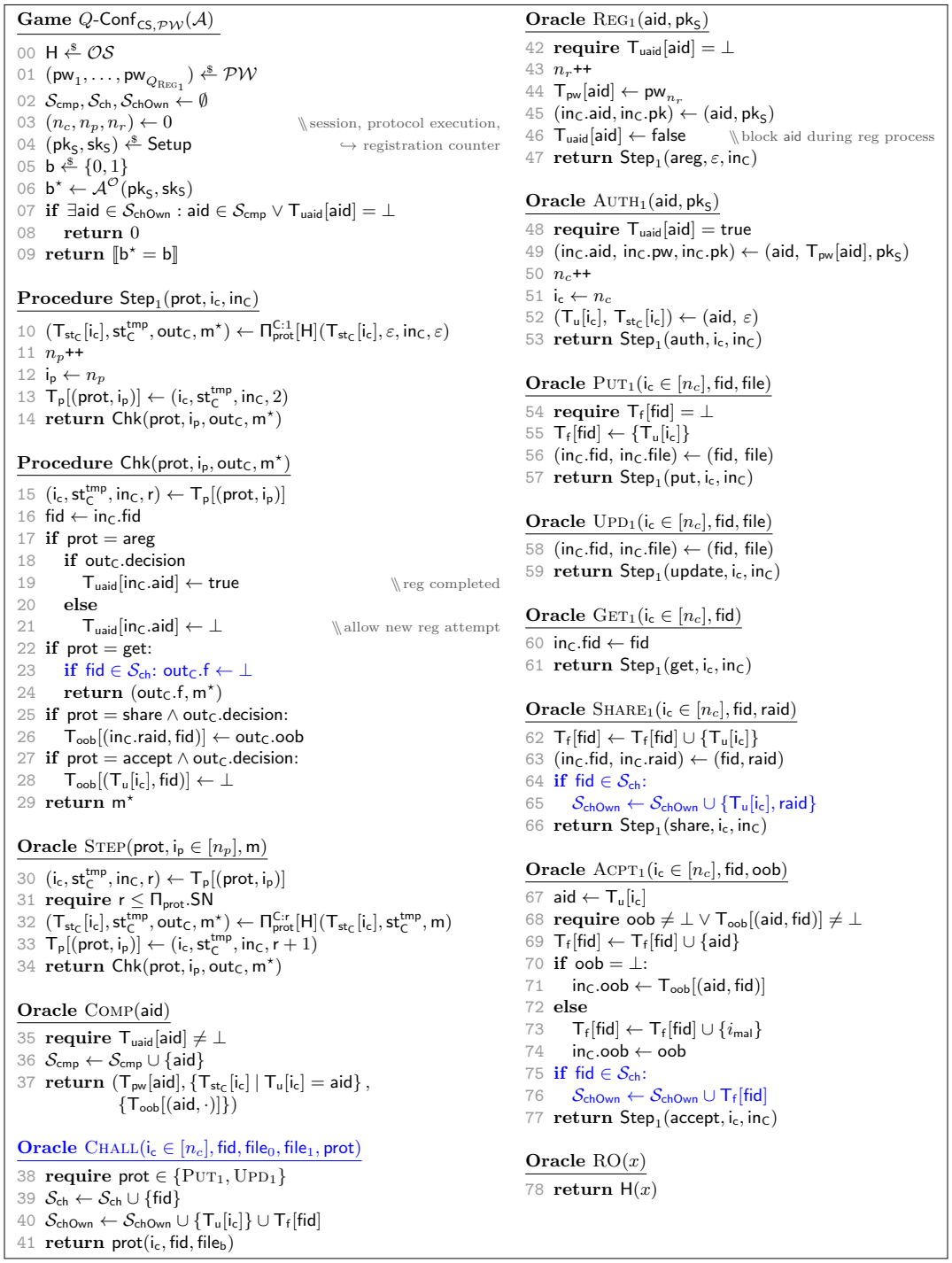

Fig. 9. Confidentiality game for a cloud storage scheme CS. Adversary $\mathcal{A}$ has access to oracles $\mathcal{O} = \{\mathrm{STEP}, \mathrm{REG}_1, \mathrm{AUTH}_1, \mathrm{PUT}_1, \mathrm{UPD}_1, \mathrm{GET}_1, \mathrm{SHARE}_1, \mathrm{ACPT}_1, \mathrm{CHALL}, \mathrm{COMP}, \mathrm{RO}\}$ making at most Q_{O} queries to oracle O. Differences to Int (Figure 10) are highlighted in blue.

where kl is the key length of AEAD and the cloud storage scheme as

$$\mathsf{CS}[\mathsf{Auth}, \mathsf{AEAD}, \mathsf{MAC}] := (\mathcal{OS}, \mathsf{Auth.Setup}, \Pi_{\mathsf{areg}}, \Pi_{\mathsf{auth}}, \Pi_{\mathsf{put}}, \Pi_{\mathsf{upd}}, \Pi_{\mathsf{get}}, \Pi_{\mathsf{share}}, \Pi_{\mathsf{accept}}),$$

Game $Q\text{-Int}^{\mathsf{CS},\mathcal{PW}}_{\mathsf{CS}}(\mathcal{A})$

$\mathsf{H} \xleftarrow{\$} \mathcal{OS}$
$(\mathsf{pw}_1, \ldots, \mathsf{pw}_{Q_{\mathrm{REG}_1}}) \xleftarrow{\$} \mathcal{PW}$
$\mathcal{S}_{\mathsf{cmp}} \leftarrow \{i_{\mathsf{mal}}\}$ \\ i_{mal} is a special symbol not in $\mathbb{N}$
$(n_c, n_p, n_r) \leftarrow 0$ \\ session, protocol execution, registration counter
$(\mathsf{pk_S}, \mathsf{sk_S}) \xleftarrow{\$} \mathsf{Setup}$
$\mathsf{win} \leftarrow \mathsf{false}$
$\mathcal{A}^{\mathcal{O}}(\mathsf{pk_S}, \mathsf{sk_S})$
return win

Procedure $\mathrm{Step}_1(\mathsf{prot}, \mathsf{i_c}, \mathsf{in_C})$

$(\mathsf{T_{st_C}}[\mathsf{i_c}], \mathsf{st_C^{tmp}}, \mathsf{out_C}, \mathsf{m}^\star) \leftarrow \Pi^{\mathsf{C}:1}_{\mathsf{prot}}[\mathsf{H}](\mathsf{T_{st_C}}[\mathsf{i_c}], \varepsilon, \mathsf{in_C}, \varepsilon)$
n_p++
$\mathsf{i_p} \leftarrow n_p$
$\mathsf{T_p}[(\mathsf{prot}, \mathsf{i_p})] \leftarrow (\mathsf{i_c}, \mathsf{st_C^{tmp}}, \mathsf{in_C}, 2)$
return $\mathsf{Chk}(\mathsf{prot}, \mathsf{i_p}, \mathsf{out_C}, \mathsf{m}^\star)$

Procedure $\mathsf{Chk}(\mathsf{prot}, \mathsf{i_p}, \mathsf{out_C}, \mathsf{m}^\star)$

$(\mathsf{i_c}, \mathsf{st_C^{tmp}}, \mathsf{in_C}, \mathsf{r}) \leftarrow \mathsf{T_p}[(\mathsf{prot}, \mathsf{i_p})]$
$\mathsf{fid} \leftarrow \mathsf{in_C}.\mathsf{fid}$
if $\mathsf{prot} = \mathsf{areg}$
  if $\mathsf{out_C}.\mathsf{decision}$
    $\mathsf{T_{uaid}}[\mathsf{in_C}.\mathsf{aid}] \leftarrow \mathsf{true}$ \\ reg completed
  else
    $\mathsf{T_{uaid}}[\mathsf{in_C}.\mathsf{aid}] \leftarrow \bot$ \\ allow new reg attempt
if $\mathsf{prot} = \mathsf{get}$:
  if $\mathsf{T_u}[\mathsf{i_c}] \notin \mathcal{S}_{\mathsf{cmp}} \wedge \mathsf{T_f}[\mathsf{fid}].\mathsf{U} \cap \mathcal{S}_{\mathsf{cmp}} = \emptyset \wedge \neg\mathsf{win}$
    $\mathsf{file} \leftarrow \mathsf{out_C}.\mathsf{file}$
    $\mathsf{win} \leftarrow [\![\mathsf{out_C}.\mathsf{decision} \wedge \mathsf{file} \neq \bot]\!] \wedge [\![\mathsf{file} \notin \mathsf{T_f}[\mathsf{fid}].\mathsf{F}]\!]$
  return $(\mathsf{out_C}.\mathsf{f}, \mathsf{m}^\star)$
if $\mathsf{prot} = \mathsf{share} \wedge \mathsf{out_C}.\mathsf{decision}$:
  $\mathsf{T_{oob}}[(\mathsf{in_C}.\mathsf{raid}, \mathsf{fid})] \leftarrow \mathsf{out_C}.\mathsf{oob}$
if $\mathsf{prot} = \mathsf{accept} \wedge \mathsf{out_C}.\mathsf{decision}$:
  $\mathsf{T_{oob}}[(\mathsf{T_u}[\mathsf{i_c}], \mathsf{fid})] \leftarrow \bot$
return $\mathsf{m}^\star$

Oracle $\mathrm{Step}(\mathsf{prot}, \mathsf{i_p} \in [n_p], \mathsf{m})$

$(\mathsf{i_c}, \mathsf{st_C^{tmp}}, \mathsf{in_C}, \mathsf{r}) \leftarrow \mathsf{T_p}[(\mathsf{prot}, \mathsf{i_p})]$
require $\mathsf{r} \leq \Pi_{\mathsf{prot}}.\mathsf{SN}$
$(\mathsf{T_{st_C}}[\mathsf{i_c}], \mathsf{st_C^{tmp}}, \mathsf{out_C}, \mathsf{m}^\star) \leftarrow \Pi^{\mathsf{C}:\mathsf{r}}_{\mathsf{prot}}[\mathsf{H}](\mathsf{T_{st_C}}[\mathsf{i_c}], \mathsf{st_C^{tmp}}, \mathsf{m})$
$\mathsf{T_p}[(\mathsf{prot}, \mathsf{i_p})] \leftarrow (\mathsf{i_c}, \mathsf{st_C^{tmp}}, \mathsf{in_C}, \mathsf{r}+1)$
return $\mathsf{Chk}(\mathsf{prot}, \mathsf{i_p}, \mathsf{out_C}, \mathsf{m}^\star)$

Oracle $\mathrm{Comp}(\mathsf{aid})$

require $\mathsf{T_{uaid}}[\mathsf{aid}] \neq \bot$
$\mathcal{S}_{\mathsf{cmp}} \leftarrow \mathcal{S}_{\mathsf{cmp}} \cup \{\mathsf{aid}\}$
return $(\mathsf{T_{pw}}[\mathsf{aid}], \{\mathsf{T_{st_C}}[\mathsf{i_c}] \mid \mathsf{T_u}[\mathsf{i_c}] = \mathsf{aid}\}, \{\mathsf{T_{oob}}[(\mathsf{aid}, \cdot)]\})$

Oracle $\mathrm{Reg}_1(\mathsf{aid}, \mathsf{pk_S})$

require $\mathsf{T_{uaid}}[\mathsf{aid}] = \bot$
n_r++
$\mathsf{T_{pw}}[\mathsf{aid}] \leftarrow \mathsf{pw}_{n_r}$
$(\mathsf{in_C}.\mathsf{aid}, \mathsf{in_C}.\mathsf{pk}) \leftarrow (\mathsf{aid}, \mathsf{pk_S})$
$\mathsf{T_{uaid}}[\mathsf{aid}] \leftarrow \mathsf{false}$ \\ block aid during reg process
return $\mathrm{Step}_1(\mathsf{areg}, \varepsilon, \mathsf{in_C})$

Oracle $\mathrm{Auth}_1(\mathsf{aid}, \mathsf{pk_S})$

require $\mathsf{T_{uaid}}[\mathsf{aid}] = \mathsf{true}$
$(\mathsf{in_C}.\mathsf{aid}, \mathsf{in_C}.\mathsf{pw}, \mathsf{in_C}.\mathsf{pk}) \leftarrow (\mathsf{aid}, \mathsf{T_{pw}}[\mathsf{aid}], \mathsf{pk_S})$
n_c++
$\mathsf{i_c} \leftarrow n_c$
$(\mathsf{T_u}[\mathsf{i_c}], \mathsf{T_{st_C}}[\mathsf{i_c}]) \leftarrow (\mathsf{aid}, \varepsilon)$
return $\mathrm{Step}_1(\mathsf{auth}, \mathsf{i_c}, \mathsf{in_C})$

Oracle $\mathrm{Put}_1(\mathsf{i_c} \in [n_c], \mathsf{fid}, \mathsf{file})$

require $\mathsf{T_f}[\mathsf{fid}] = \bot$
$\mathsf{U} \leftarrow \{\mathsf{T_u}[\mathsf{i_c}]\}$
$\mathsf{F} \leftarrow \{\mathsf{file}\}$
$\mathsf{T_f}[\mathsf{fid}] \leftarrow (\mathsf{U}, \mathsf{F})$
$(\mathsf{in_C}.\mathsf{fid}, \mathsf{in_C}.\mathsf{file}) \leftarrow (\mathsf{fid}, \mathsf{file})$
return $\mathrm{Step}_1(\mathsf{put}, \mathsf{i_c}, \mathsf{in_C})$

Oracle $\mathrm{Upd}_1(\mathsf{i_c} \in [n_c], \mathsf{fid}, \mathsf{file})$

if $\mathsf{T_u}[\mathsf{i_c}] \in \mathsf{T_f}[\mathsf{fid}].\mathsf{U}$
  $\mathsf{T_f}[\mathsf{fid}].\mathsf{F} \leftarrow \mathsf{T_f}[\mathsf{fid}].\mathsf{F} \cup \{\mathsf{file}\}$
$(\mathsf{in_C}.\mathsf{fid}, \mathsf{in_C}.\mathsf{file}) \leftarrow (\mathsf{fid}, \mathsf{file})$
return $\mathrm{Step}_1(\mathsf{update}, \mathsf{i_c}, \mathsf{in_C})$

Oracle $\mathrm{Get}_1(\mathsf{i_c} \in [n_c], \mathsf{fid})$

$\mathsf{in_C}.\mathsf{fid} \leftarrow \mathsf{fid}$
return $\mathrm{Step}_1(\mathsf{get}, \mathsf{i_c}, \mathsf{in_C})$

Oracle $\mathrm{Share}_1(\mathsf{i_c} \in [n_c], \mathsf{fid}, \mathsf{raid})$

$\mathsf{T_f}[\mathsf{fid}].\mathsf{U} \leftarrow \mathsf{T_f}[\mathsf{fid}].\mathsf{U} \cup \{\mathsf{T_u}[\mathsf{i_c}]\}$
$(\mathsf{in_C}.\mathsf{fid}, \mathsf{in_C}.\mathsf{raid}) \leftarrow (\mathsf{fid}, \mathsf{raid})$
return $\mathrm{Step}_1(\mathsf{share}, \mathsf{i_c}, \mathsf{in_C})$

Oracle $\mathrm{Acpt}_1(\mathsf{i_c} \in [n_c], \mathsf{fid}, \mathsf{oob})$

$\mathsf{aid} \leftarrow \mathsf{T_u}[\mathsf{i_c}]$
require $\mathsf{oob} \neq \bot \vee \mathsf{T_{oob}}[(\mathsf{aid}, \mathsf{fid})] \neq \bot$
$\mathsf{T_f}[\mathsf{fid}].\mathsf{U} \leftarrow \mathsf{T_f}[\mathsf{fid}].\mathsf{U} \cup \{\mathsf{aid}\}$
if $\mathsf{oob} = \bot$:
  $\mathsf{in_C}.\mathsf{oob} \leftarrow \mathsf{T_{oob}}[(\mathsf{aid}, \mathsf{fid})]$
else
  $\mathsf{T_f}[\mathsf{fid}].\mathsf{U} \leftarrow \mathsf{T_f}[\mathsf{fid}].\mathsf{U} \cup \{i_{\mathsf{mal}}\}$
  $\mathsf{in_C}.\mathsf{oob} \leftarrow \mathsf{oob}$
return $\mathrm{Step}_1(\mathsf{accept}, \mathsf{i_c}, \mathsf{in_C})$

Oracle $\mathrm{RO}(x)$

return $\mathsf{H}(x)$

Fig. 10. Integrity game for a cloud storage scheme CS. Adversary $\mathcal{A}$ has access to oracles $\mathcal{O} = \{\mathrm{Step}, \mathrm{Reg}_1, \mathrm{Auth}_1, \mathrm{Put}_1, \mathrm{Upd}_1, \mathrm{Get}_1, \mathrm{Share}_1, \mathrm{Acpt}_1, \mathrm{Comp}, \mathrm{RO}\}$ making at most Q_{O} queries to oracle O. Differences to Conf (Figure 9) are highlighted in blue.

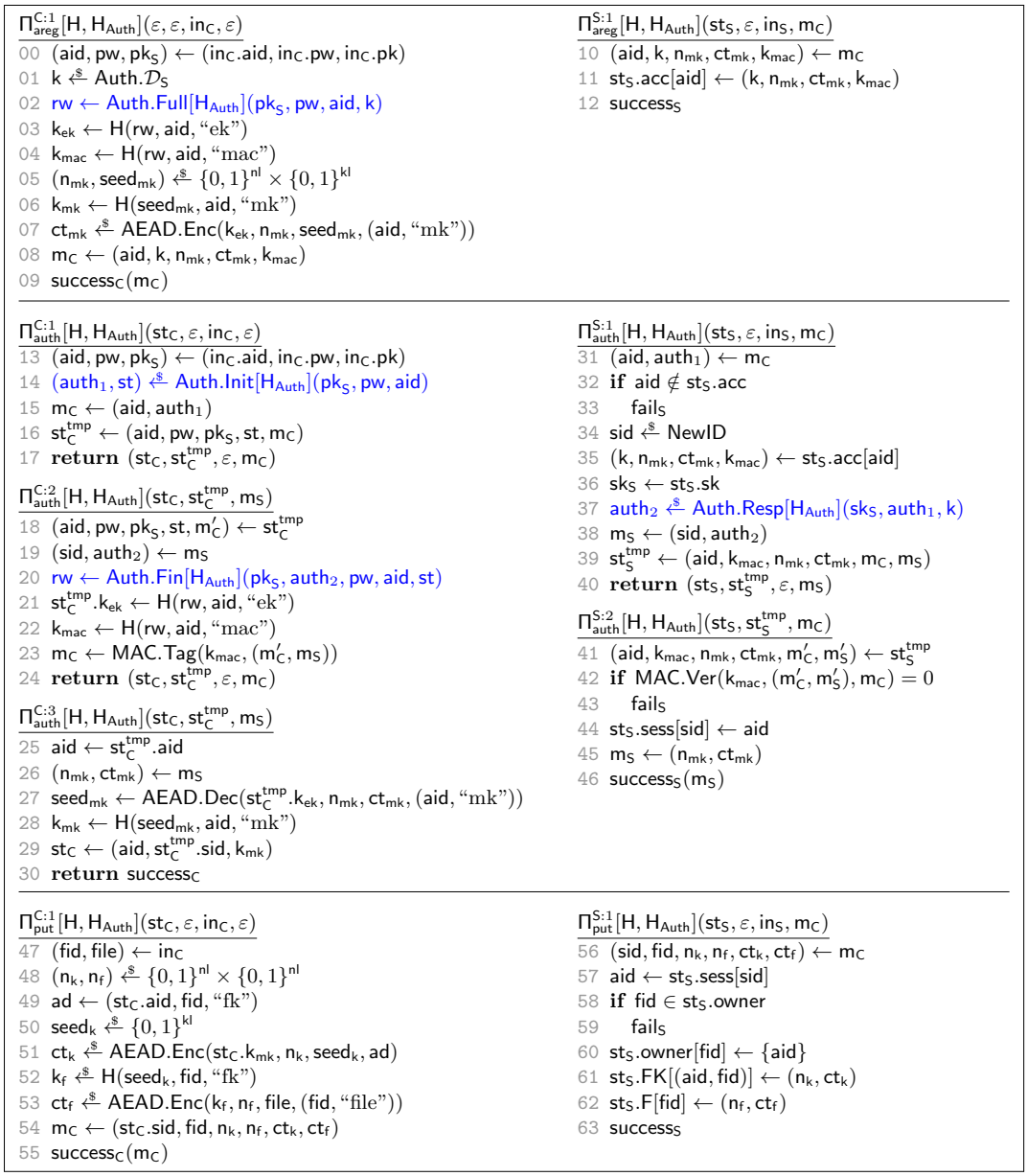

Fig. 11. Protocols for account registration (areg), authentication (auth), and file upload (put). Our new authentication abstraction is highlighted in blue.

where the seven protocols are defined in Figures 11 and 12. By NewID, we denote a procedure which samples a new ID that was never output before.

Correctness. Correctness follows from the correctness of all underlying primitives. We only the state the theorem.

Theorem 4 (Correctness). *Let* AEAD *and* MAC *be correct and* Auth *be* δ_{Auth}*-correct. Then for any adversary* $\mathcal{A}$ *against* Corr *of* CS[Auth, AEAD, MAC] *it holds that*

$$\mathsf{Adv}^{Q_{\mathcal{A}}\text{-Corr}}_{\mathsf{CS}[\mathsf{Auth},\mathsf{AEAD},\mathsf{MAC}]}(\mathcal{A}) \leq Q_{\text{Auth}} \cdot \delta_{\mathsf{Auth}}$$

$\underline{\Pi^{C:1}_{upd}[H, H_{Auth}](st_C, \varepsilon, in_C, \varepsilon)}$
00 $m_C \leftarrow \text{getHeader}^{C:1}(st_C, in_C.fid)$
01 $(st^{tmp}_C.fid, st^{tmp}_C.file) \leftarrow (in_C.fid, in_C.file)$
02 **return** $(st_C, st^{tmp}_C, \varepsilon, m_C)$

$\underline{\Pi^{C:2}_{upd}[H, H_{Auth}](st_C, st^{tmp}_C, m_S)}$
03 $seed_k \leftarrow \text{getHeader}^{C:2}(st_C, st^{tmp}_C.fid, m_S)$
04 $k_f \leftarrow H(seed_k, st^{tmp}_C.fid, \text{“fk”})$
05 $n \xleftarrow{\$} \{0,1\}^{nl}$
06 $ct_f \xleftarrow{\$} \text{AEAD.Enc}(k_f, n, st^{tmp}_C.file, (st^{tmp}_C.fid, \text{“file”}))$
07 $m_C \leftarrow (st_C.sid, st^{tmp}_C.fid, n, ct_f)$
08 $\text{success}_C(m_C)$

$\underline{\Pi^{S:1}_{upd}[H, H_{Auth}](st_S, \varepsilon, in_S, m_C)}$
09 $m_S \leftarrow \text{getHeader}^{S}(st_S, m_C)$
10 **return** $(st_S, \varepsilon, \varepsilon, m_S)$

$\underline{\Pi^{S:2}_{upd}[H, H_{Auth}](st_S, st^{tmp}_S, m_C)}$
11 $(sid, fid, n, ct_f) \leftarrow m_C$
12 $aid \leftarrow st_S.sess[sid]$
13 **if** $aid \notin st_S.owner[fid]$
14 **return** $\perp$
15 $F[fid] \leftarrow (n, ct_f)$
16 success_S

$\underline{\Pi^{C:1}_{get}[H, H_{Auth}](st_C, \varepsilon, in_C, \varepsilon)}$
17 $m_C \leftarrow (st_C.sid, in_C.fid)$
18 $st^{tmp}_C.fid \leftarrow in_C.fid$
19 **return** $(st_C, \varepsilon, \varepsilon, m_C)$

$\underline{\Pi^{C:2}_{get}[H, H_{Auth}](st_C, st^{tmp}_C, m_S)}$
20 $((n_k, ct_k), (n_f, ct_f)) \leftarrow m_S$
21 $ad \leftarrow (st_C.aid, st^{tmp}_C.fid, \text{“fk”})$
22 $seed_k \leftarrow \text{AEAD.Dec}(st_C.k_{mk}, n_k, ct_k, ad)$
23 $k_f \leftarrow H(seed_k, st_C.fid, \text{“fk”})$
24 $out_C.file \leftarrow \text{AEAD.Dec}(k_f, n_f, ct_f, (st^{tmp}_C.fid, \text{“file”}))$
25 success_C

$\underline{\Pi^{S:1}_{get}[H, H_{Auth}](st_S, \varepsilon, in_S, m_C)}$
26 $(sid, fid) \leftarrow m_C$
27 $aid \leftarrow st_S.sess[sid]$
28 **if** $aid \notin st_S.owner[fid]$
29 **return** $\perp$
30 $m_S \leftarrow (\text{FK}[(aid, fid)], F[fid])$
31 $\text{success}_S(m_S)$

$\underline{\Pi^{C:1}_{share}[H, H_{Auth}](st_C, \varepsilon, in_C, \varepsilon)}$
32 $fid \leftarrow in_C.fid$
33 $m' \leftarrow \text{getHeader}^{C:1}(st_C, fid)$
34 $st^{tmp}_C.fid \leftarrow fid$
35 $m_C \leftarrow (m', in_C.raid, fid)$
36 **return** $(st_C, st^{tmp}_C, \varepsilon, m_C)$

$\underline{\Pi^{C:2}_{share}[H, H_{Auth}](st_C, st^{tmp}_C, m_S)}$
37 $seed_k \leftarrow \text{getHeader}^{C:2}(st_C, st^{tmp}_C.fid, m_S)$
38 $out_C.oob \leftarrow seed_k$
39 success_C

$\underline{\Pi^{S:1}_{share}[H, H_{Auth}](st_S, \varepsilon, in_S, m_C)}$
40 $(m', raid, fid) \leftarrow m_C$
41 $m_S \leftarrow \text{getHeader}^{S}(st_S, m')$
42 $st_S.owner[fid] \leftarrow st_S.owner[fid] \cup \{raid\}$
43 $\text{success}_S(m_S)$

$\underline{\Pi^{C:1}_{accept}[H, H_{Auth}](st_C, \varepsilon, in_C, \varepsilon)}$
44 $seed_k \leftarrow in_C.oob$
45 $n \xleftarrow{\$} \{0,1\}^{nl}$
46 $ad \leftarrow (st_C.aid, in_C.fid, \text{“fk”})$
47 $ct_k \xleftarrow{\$} \text{AEAD.Enc}(st_C.k_{mk}, n, seed_k, ad)$
48 $m_C \leftarrow (st_C.sid, in_C.fid, n, ct_k)$
49 $\text{success}_C(m_C)$

$\underline{\Pi^{S:1}_{accept}[H, H_{Auth}](st_S, \varepsilon, in_S, m_C)}$
50 $(sid, fid, n, ct_k) \leftarrow m_C$
51 $aid \leftarrow st_S.sess[sid]$
52 **if** $aid \notin st_S.owner[fid]$
53 **return** $\perp$
54 $\text{FK}[(aid, fid)] \leftarrow (n, ct_k)$
55 success_S

$\underline{\text{getHeader}^{C:1}(st_C, fid)}$
56 $m_C \leftarrow (st_C.sid, fid)$
57 **return** m_C

$\underline{\text{getHeader}^{C:2}(st_C, fid, m_S)}$
58 $(n_k, ct_k) \leftarrow m_S$
59 $ad \leftarrow (st_C.aid, fid, \text{“fk”})$
60 $seed_k \leftarrow \text{AEAD.Dec}(st_C.k_{mk}, n_k, ct_k, ad)$
61 **return** $seed_k$

$\underline{\text{getHeader}^{S}(st_S, m_C)}$
62 $(sid, fid) \leftarrow m_C$
63 $aid \leftarrow st_S.sess[sid]$
64 **if** $aid \notin st_S.owner[fid]$
65 **return** $\perp$
66 $m_S \leftarrow (\text{FK}[(aid, fid)])$
67 **return** m_S

Fig. 12. Protocols for file update (upd), download (get), sharing (share), accepting (accept), and helper functions.

with $Q_{\mathcal{A}} = (Q_{\text{Reg}}, Q_{\text{Auth}}, Q_{\text{Put}}, Q_{\text{Upd}}, Q_{\text{Get}}, Q_{\text{Share}}, Q_{\text{Acpt}}, Q_{\text{RO}})$.

Confidentiality and Integrity. We give informal statements below. The formal theorems with concrete bounds can be found as Theorems 10 and 11 in Appendices C.1 and C.2, respectively, where we also give the full proofs. Additionally, we discuss and provide bounds for the scheme instantiated with 2HDH in Section 7.

Theorem 5 (Confidentiality, Informal). *Let* Auth *be an authentication scheme that is* δ_{Auth}*-correct and* INS-OW *secure. Let* AEAD *be an AEAD scheme that is* OW-CPA, IND-CCA *and* INT-CTXT *secure. Let* MAC *be a MAC scheme. Then* CS $:=$ CS[Auth, AEAD, MAC] *is* Conf *secure.*

Proof (Idea). The main path of the proof is the same as in [BDG+24]: we go through the key hierarchy and use that "honest keys" are indistinguishable from random (honest client's raw secrets, their encryption key, their personal master key, and eventually file keys which leads to confidential files). The difference to [BDG+24] is that we allow for adaptive compromises which is why we cannot commit to any keys because the reduction does not know which ciphertexts might be opened later due to a compromise. Hence, we cannot simply set keys to random. Our proof's main idea is to stay in an uncommitted state by only encrypting seeds and patching the random oracle upon compromises. While this is straightforward for a single non-committing encryption, the key hierarchy adds additional challenges because we need to make sure that random oracles are simulated consistently in each step and can be patched without unnecessary reduction losses. □

Theorem 6 (Integrity, Informal). *Let* Auth *be an authentication scheme that is* δ_{Auth}*-correct and* INS-OW *secure. Let* AEAD *be an AEAD scheme that is* OW-CPA *and* INT-CTXT *secure. Let* MAC *be a MAC scheme. Then* CS $:=$ CS[Auth, AEAD, MAC] *is* Int *secure.*

Proof (Idea). The proof follows the same structure as for Theorem 10. After applying the same game changes, we are in a state where the seeds for all the file keys are uniformly random if the files are only owned by honest users. Instead of the AEAD's IND-CCA security, we can reduce to its INT-CTXT security. □

6 Network Adversary Model

The security model from [BDG+24] which we considered so far allows for malicious servers, also denoted as C2C (client-to-client) security. While this is a relevant model, it is also quite strong. For example, if the adversary controls the server, we cannot circumvent offline password guessing attacks since the server has all information except the password. This is reflected in the theorem bounds via the INS-OW term for Auth which is reduced to offline password guessing.

In addition to the malicious server model, [BDG+24] also discusses a weaker model against "external adversaries" where the adversary can still compromise clients but does not control the server anymore; denoted as C2S (client-to-server). While this model is weaker, it allows for better security bounds or advanced security guarantees. An example is the aforementioned problem with offline password guessing which can be avoided if the server cannot be compromised.

The Proposed Model from [BDG+24]. For the weaker model, the authors propose that the server is honest but adversaries can compromise clients, and that there are secure channels between clients and server. Overall, this should give better security in the sense that an adversary might now require online queries for password guessing which is why they use 2HDH in their construction. Since we introduced the Auth primitive to abstract the authentication procedure, we aim to identify the correct security notion that is sufficient for the given setting, and can then think about appropriate instantiations. Considering the notions in Section 3, we can see that INS-OW as needed for the E2EE case is stronger than what we need, which was expected. The additionally defined notion OUT-OW is still too strong because for secure channels we do not need an Init or Fin oracle. This is because compromised users can still only communicate with the server and not interfere or see the communication between honest clients and the server. For the sake of completeness, we present such an Auth notion in Appendix D along with an instantiation much simpler than 2HDH.

A New Model. We target a stronger model than the one just discussed. Instead of secure channels between clients and the server, we assume an adversary controlling the whole network with one exception: adversaries are not allowed to see the communication of client and server during the account registration. This is similar to the setting of asymmetric password-authenticated key exchange protocols [BM93]. If we would allow that, an adversary could simply observe the communication between client and server and in particular obtain the password file to act as the server in the following; this would not be different from the C2C model. We also need the secret key of the server setup to remain secret. Otherwise, the adversary could impersonate the server directly. Since there is only one server, we do not add a dedicated corrupt oracle for long-term keys. This model lies in between that model and the weaker C2S model but allows for relying on online queries for password guesses. In detail, it is sufficient to rely on OUT-OW security of the Auth primitive which can be instantiated by 2HDH based on online queries.

Explicit Authentication. Another property of interest in the network model is *explicit authentication*. In the C2C setting, it does not make sense to formalize such a notion because only clients authenticate to the server and the server can be malicious. So why does it make sense to require explicit authentication in the network setting? The reason is that it is possible to reduce the security to online authentication queries for password guesses. In practice, that can be very useful because online queries can be easily restricted. A query to the auth oracle means that a user tries to log into an account. Failed attempts can only be recognized by an explicit authentication procedure. The same holds for successful attempts which can be useful in practical applications to notify clients via a different channel such that they can take action if the login was malicious.

6.1 Definitions

We give our new security definitions, strengthening external adversaries to a network adversary.

Definition 11 (Net-Conf). *Confidentiality for a cloud storage scheme* CS *is defined via the game in Figure 13. We define the advantage of an adversary* $\mathcal{A}$ *as*

$$\mathsf{Adv}^{Q\text{-Net-Conf}}_{\mathsf{CS},\mathcal{PW}}(\mathcal{A}) := 2\,\mathsf{Pr}\left[Q\text{-Net-Conf}_{\mathsf{CS},\mathcal{PW}}(\mathcal{A}) \Rightarrow 1\right] - 1,$$

for $Q = (Q_{\textsc{Step}}, Q_{\textsc{StepS}_1}, Q_{\textsc{Reg}_1}, Q_{\textsc{Auth}_1}, Q_{\textsc{Put}_1}, Q_{\textsc{Upd}_1}, Q_{\textsc{Get}_1}, Q_{\textsc{Share}_1}, Q_{\textsc{Acpt}_1}, Q_{\textsc{Chall}}, Q_{\textsc{Comp}}, Q_{\textsc{RO}})$.

In addition to the original C2C model from [BDG+24], we need to model that the adversary does not have control of server but can control the network. This means they can control all the communication between honest clients and servers which is modeled by providing the adversary not only oracles executing client's protocol's steps but also server's protocol's steps. In more detail, the adversary gets an additional oracle $\textsc{StepS}_1$ to execute the server's first step of a protocol for some client message. The usual Step can now also be executed for the server which was not necessary before. As mentioned in the model description, we do not want to allow the adversary to see or interfere during the registration procedure because this would not distinguish the model from the C2C model of [BDG+24]. This is modeled by not returning the real protocol messages when executing the registration protocol but only outputting the length which is usually leaked for secure channels. The winning condition for an adversary does not change compared to the original Conf notion (Figure 9).

Definition 12 (Net-Int). *Integrity for a cloud storage scheme* CS *is defined via the game in Figure 14. We define the advantage of an adversary* $\mathcal{A}$ *as*

$$\mathsf{Adv}^{Q\text{-Net-Int}}_{\mathsf{CS},\mathcal{PW}}(\mathcal{A}) := \mathsf{Pr}\left[Q\text{-Net-Int}_{\mathsf{CS},\mathcal{PW}}(\mathcal{A}) \Rightarrow 1\right],$$

for $Q = (Q_{\textsc{Step}}, Q_{\textsc{StepS}_1}, Q_{\textsc{Reg}_1}, Q_{\textsc{Auth}_1}, Q_{\textsc{Put}_1}, Q_{\textsc{Upd}_1}, Q_{\textsc{Get}_1}, Q_{\textsc{Share}_1}, Q_{\textsc{Acpt}_1}, Q_{\textsc{Comp}}, Q_{\textsc{RO}})$.

The adversarial scenario is modeled in the same way as in Net-Conf and the winning condition is analogously taken from Int (Figure 10).

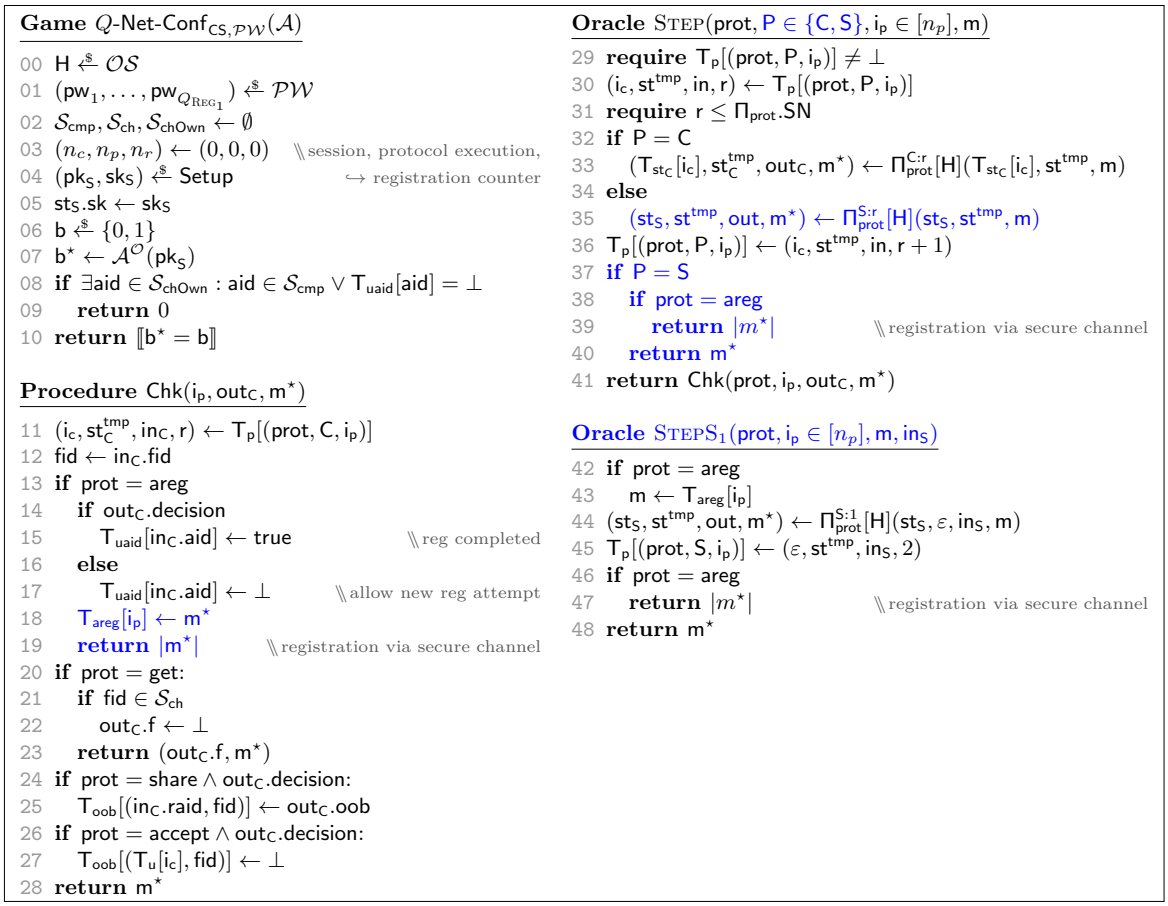

Fig. 13. Network adversary confidentiality game for a cloud storage scheme CS. Adversary $\mathcal{A}$ has access to oracles $\mathcal{O} = \{\textsc{Step}, \textsc{StepS}_1, \textsc{Reg}_1, \textsc{Auth}_1, \textsc{Put}_1, \textsc{Upd}_1, \textsc{Get}_1, \textsc{Share}_1, \textsc{Acpt}_1, \textsc{Chall}, \textsc{Comp}, \mathrm{RO}\}$ making at most Q_{O} queries to oracle O. Oracles not shown here as well as the procedure Step_1 are defined as in Figure 9. Differences to Conf (Figure 9) are highlighted in blue.

Definition 13 (Exp-Auth). *Explicit authentication for a cloud storage scheme* CS *is defined via the game in Figure 15. We define the advantage of an adversary* $\mathcal{A}$ *as*

$$\mathsf{Adv}_{\mathsf{CS}}^{Q\text{-Exp-Auth}}(\mathcal{A}) := \Pr\left[Q\text{-Exp-Auth}_{\mathsf{CS}}(\mathcal{A}) \Rightarrow 1\right],$$

for $Q = (Q_{\textsc{Step}}, Q_{\textsc{StepS}_1}, Q_{\textsc{Reg}_1}, Q_{\textsc{Auth}_1}, Q_{\textsc{Put}_1}, Q_{\textsc{Upd}_1}, Q_{\textsc{Get}_1}, Q_{\textsc{Share}_1}, Q_{\textsc{Acpt}_1}, Q_{\textsc{Comp}}, Q_{\mathrm{RO}})$.

The adversarial model is the same as for the previous notions and the adversary has access to the same oracles. The only possibility for the adversary to win is if an execution of the authentication protocol succeeds for an uncompromised user and the corresponding message transcript for that authentication session was not just forwarded from the honest client and server executions. The last condition is a triviality condition because the adversary has access to the whole network and can therefore decide to forward messages from honest clients and the server. These messages will obviously lead to a successful protocol execution.

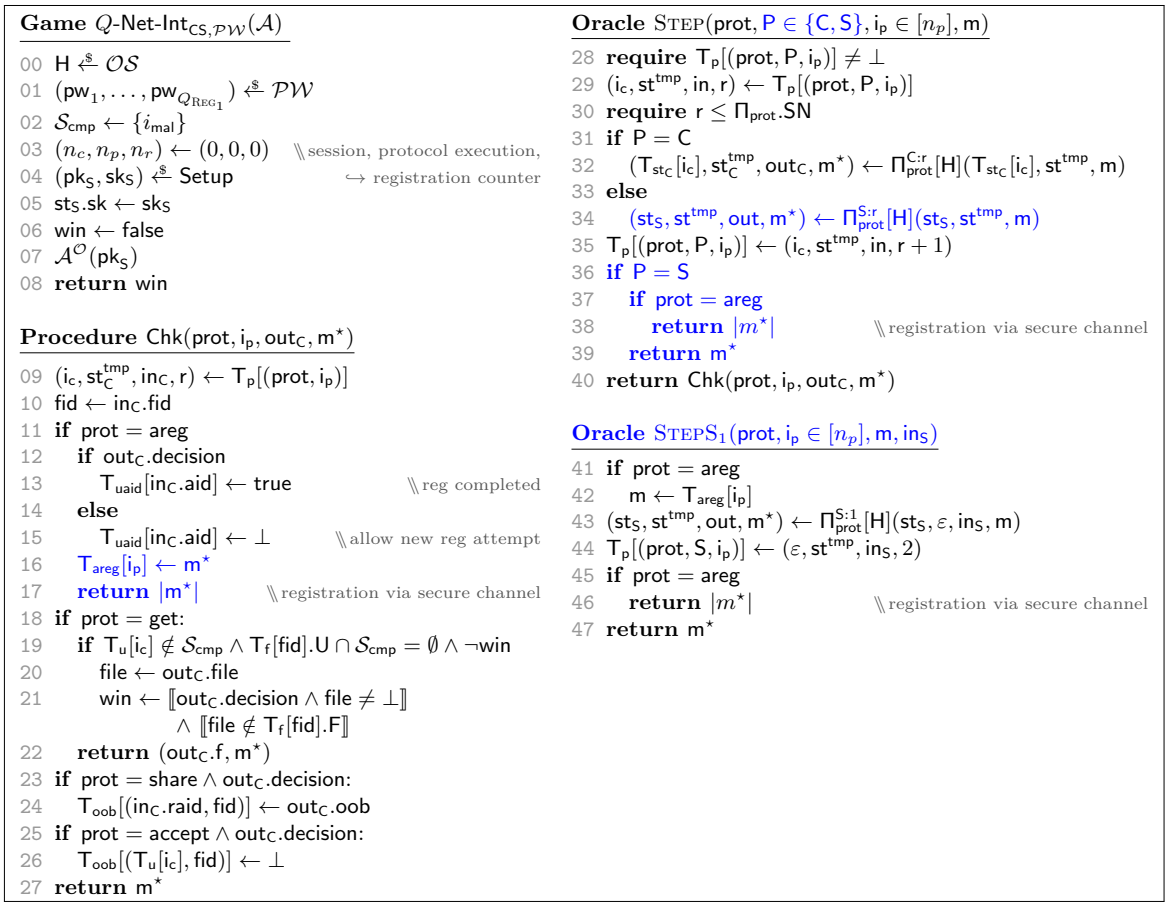

Fig. 14. Network adversary integrity game for a cloud storage scheme CS. Adversary $\mathcal{A}$ has access to oracles $\mathcal{O} = \{\textsc{Step}, \textsc{StepS}_1, \textsc{Reg}_1, \textsc{Auth}_1, \textsc{Put}_1, \textsc{Upd}_1, \textsc{Get}_1, \textsc{Share}_1, \textsc{Acpt}_1, \textsc{Comp}, \mathrm{RO}\}$ making at most Q_{O} queries to oracle O. Oracles not shown here as well as the procedure Step_1 are defined as in Figure 10. Differences to Int (Figure 10) are highlighted in blue.

6.2 Security

Similar to the previous section, we state the informal theorems here. The formal theorems can be found as Theorems 12 to 14 in Appendix E, where we also give the full proofs.

Theorem 7 (Net-Conf, Informal). *Let* Auth *be an authentication scheme that is* δ_{Auth}*-correct and* OUT-OW *secure. Let* AEAD *be an AEAD scheme that is* OW-CPA*,* IND-CCA *and* INT-CTXT *secure. Let* MAC *be a MAC scheme. Then* CS := CS[Auth, AEAD, MAC] *is* Net-Conf *secure.*

Remark 4. As mentioned previously, the crucial aspect here is to rely only on OUT-OW for the Auth primitive which can lead to online guessing bounds for appropriate instantiations. Together with explicit authentication this gives major advantages for practical systems.

Proof (Sketch). The proof follows a similar structure as the proof of Theorem 10 with the difference that the reduction to OUT-OW does not know the server's secret. They can still simulate the game by using their Resp oracle. If the random oracle is queried on the correct raw secret, the reduction wins. After this change, the game is in the same state as in the proof of Theorem 10 and we can apply the same remaining changes. □

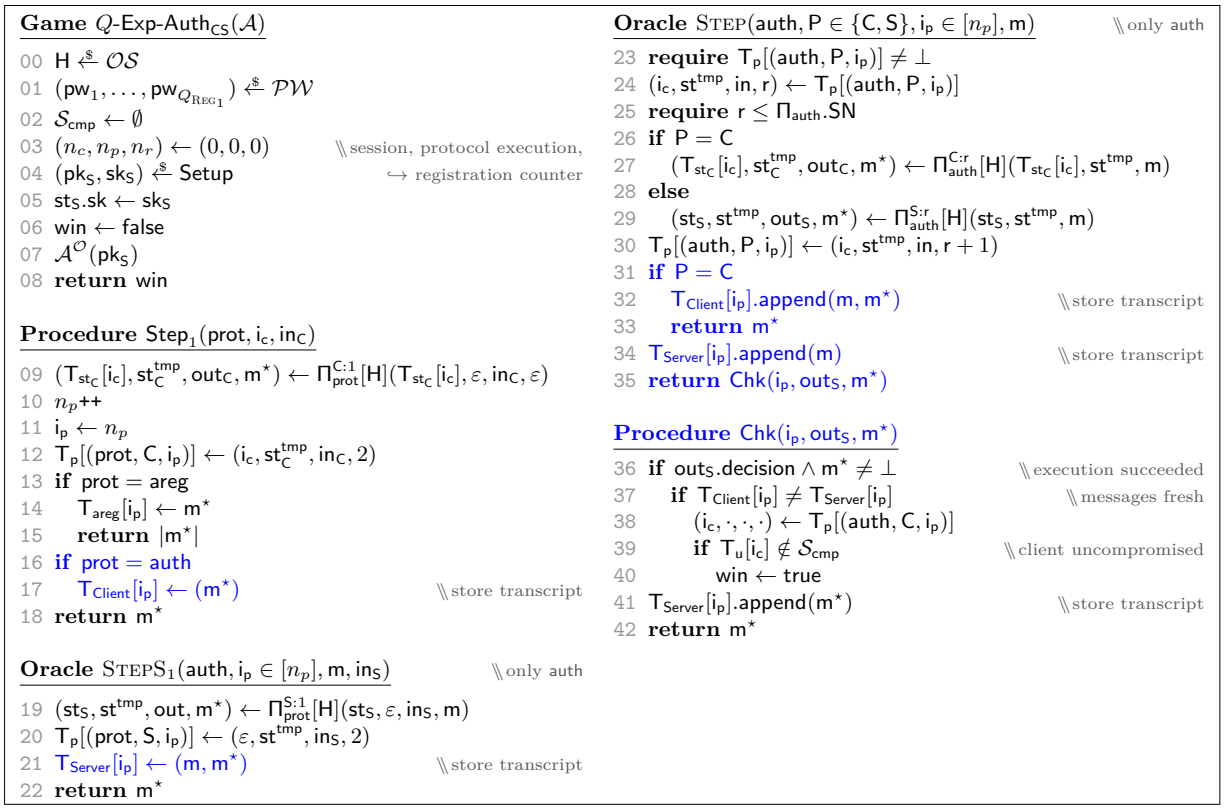

Fig. 15. Explicit authentication game for a cloud storage scheme CS. Adversary $\mathcal{A}$ has access to oracles $\mathcal{O} = \{\textsc{Step}, \textsc{StepS}_1, \textsc{Reg}_1, \textsc{Auth}_1, \textsc{Put}_1, \textsc{Upd}_1, \textsc{Get}_1, \textsc{Share}_1, \textsc{Acpt}_1, \textsc{Comp}, \mathrm{RO}\}$ making at most Q_{O} queries to oracle O. Oracles $\textsc{Reg}_1, \textsc{Auth}_1, \textsc{Put}_1, \textsc{Upd}_1, \textsc{Get}_1, \textsc{Share}_1, \textsc{Acpt}_1, \textsc{Comp}$, and RO are defined as in Figure 10. Differences to Int (Figure 10) are highlighted in blue.

Theorem 8 (Net-Int, Informal). *Let* Auth *be an authentication scheme that is* δ_{Auth}*-correct and* OUT-OW *secure. Let* AEAD *be an AEAD scheme that is* OW-CPA *and* INT-CTXT *secure. Let* MAC *be a MAC scheme. Then* CS $\coloneqq$ CS[Auth, AEAD, MAC] *is* Net-Int *secure.*

Proof. The proof follows from the proofs of Theorem 11 and Theorem 12 combining the first two steps of the proof of Theorem 12, to rely on OUT-OW instead of INS-OW, with the remaining steps of the proof of Theorem 11 to reduce to integrity and not confidentiality. □

Theorem 9 (Explicit Authentication, Informal). *Let* Auth *be an authentication scheme that is* δ_{Auth}*-correct and* OUT-OW *secure. Let* AEAD *be an AEAD scheme. Let* MAC *be a MAC scheme that is* SUF *secure. Then* CS $\coloneqq$ CS[Auth, AEAD, MAC] *is* Exp-Auth *secure.*

Proof (Sketch). Applying OUT-OW in the same way as for the proofs of Theorem 12 and Theorem 13. This allows us to have a uniformly random MAC key for the following reduction to SUF. An additional challenge to reduce the security of the MAC is that during the registration procedure, the client sends the MAC key to the server. We can simulate this step because the security model assumes a secure channel during registration and the length of the MAC key is fixed. If the adversary wins the Exp-Auth game, it must be with a new message or tag for the MAC because just relaying transcripts from the client/server interaction does not yield a valid winning condition. □

7 Conclusion and Discussion

We have presented several security definitions and proofs for the cloud storage scheme. Here, we want to summarize and discuss the results and implications for the scheme when instantiated with 2HDH as the authentication protocol, as originally suggested by [BDG+24], plus the changes we employed to achieve security in the presence of adaptive corruptions.

For the overview in Sections 7.1 and 7.2, we want to minimize the number of variables. In particular, we set the number of users to $N := Q_{\text{REG}_1}$ (the number of initial registration queries) and the number of adaptive corruptions to $C := Q_{\text{COMP}} \leq N$. We unify the remaining queries, those to all the other oracles except for the random oracle(s), as *online queries*, i.e. $Q_{\text{ON}} := \max\{Q_{\text{STEP}}, Q_{\text{AUTH}_1}, Q_{\text{PUT}_1}, Q_{\text{UPD}_1}, Q_{\text{GET}_1}, Q_{\text{SHARE}_1}, Q_{\text{ACPT}_1}, Q_{\text{CHALL}}\}$, and the random oracle queries as *offline queries*, i.e. $Q_{\text{OFF}} := Q_{\text{RO}}$. We give an interpretation of the assumptions in the two attacker models, and we then also provide a more general discussion in Section 7.3.

7.1 Client-to-Client Security

The following corollaries are obtained by combining Theorem 2 and Lemma 4 with Theorems 10 and 11, respectively, where we fold together adversaries against the same AEAD notions and upper bound the number of queries accordingly. We provide an interpretation of these bounds below.

Corollary 1 (Confidentiality). *For any password distribution $\mathcal{PW}$ and adversary $\mathcal{A}$ against the* Conf *security of* $\mathsf{CS} := \mathsf{CS}[\mathsf{2HDH}, \mathsf{AEAD}, \mathsf{MAC}]$*, there exist a* PW-Guess *adversary $\mathcal{B}$ against $\mathcal{PW}$, an* INT-CTXT *adversary $\mathcal{C}$ against* AEAD*, an* IND-PAS *adversary $\mathcal{D}$ against* AEAD*, and an* IND-CCA *adversary $\mathcal{E}$ against* AEAD *with $t_{\mathcal{A}} \approx t_{\mathcal{B}} \approx t_{\mathcal{C}} - C \cdot Q_{\text{ON}} \approx t_{\mathcal{D}} - C \cdot Q_{\text{ON}} \approx t_{\mathcal{E}}$ such that*

$$\mathsf{Adv}^{Q_{\mathcal{A}}\text{-Conf}}_{\mathsf{CS},\mathcal{PW}}(\mathcal{A}) \leq 2 \cdot \mathsf{Adv}^{Q_{\mathcal{B}}\text{-PW-Guess}}_{\mathcal{PW}}(\mathcal{B}) + 4 \cdot \mathsf{Adv}^{Q_{\mathcal{C}}\text{-INT-CTXT}}_{\mathsf{AEAD}}(\mathcal{C}) + 4 \cdot \mathsf{Adv}^{Q_{\mathcal{D}}\text{-IND-PAS}}_{\mathsf{AEAD}}(\mathcal{D}) + \mathsf{Adv}^{Q_{\mathcal{E}}\text{-IND-CCA}}_{\mathsf{AEAD}}(\mathcal{E}) + \frac{Q_{\text{ON}}^2}{2^{\mathsf{nl}-3}} + \frac{Q_{\text{OFF}}}{2^{\mathsf{kl}-4}},$$

with $Q_{\mathcal{A}} = (Q_{\text{ON}}, N, Q_{\text{ON}}, Q_{\text{ON}}, Q_{\text{ON}}, Q_{\text{ON}}, Q_{\text{ON}}, Q_{\text{ON}}, C, Q_{\text{OFF}})$ *and*

$$\begin{aligned} Q_{\mathcal{B}} &= (N, Q_{\text{OFF}}, C), \\ Q_{\mathcal{C}} &= (N, C, 3Q_{\text{ON}}, 4Q_{\text{ON}}), \\ Q_{\mathcal{D}} &= (N, C, 4Q_{\text{ON}}), \\ Q_{\mathcal{E}} &= (2Q_{\text{ON}}, 2Q_{\text{ON}}, 2Q_{\text{ON}}, Q_{\text{ON}}, Q_{\text{ON}}). \end{aligned}$$

Corollary 2 (Integrity). *For any password distribution $\mathcal{PW}$ and adversary $\mathcal{A}$ against the* Int *security of* $\mathsf{CS} := \mathsf{CS}[\mathsf{2HDH}, \mathsf{AEAD}, \mathsf{MAC}]$*, there exist a* PW-Guess *adversary $\mathcal{B}$ against $\mathcal{PW}$, an* INT-CTXT *adversary $\mathcal{C}$ against* AEAD*, and an* IND-PAS *adversary $\mathcal{D}$ against* AEAD *with $t_{\mathcal{A}} \approx t_{\mathcal{B}} \approx t_{\mathcal{C}} - C \cdot Q_{\text{ON}} \approx t_{\mathcal{D}} - C \cdot Q_{\text{ON}}$ such that*

$$\mathsf{Adv}^{Q_{\mathcal{A}}\text{-Int}}_{\mathsf{CS},\mathcal{PW}}(\mathcal{A}) \leq \mathsf{Adv}^{Q_{\mathcal{B}}\text{-PW-Guess}}_{\mathcal{PW}}(\mathcal{B}) + 3 \cdot \mathsf{Adv}^{Q_{\mathcal{C}}\text{-INT-CTXT}}_{\mathsf{AEAD}}(\mathcal{C}) + 2 \cdot \mathsf{Adv}^{Q_{\mathcal{D}}\text{-IND-PAS}}_{\mathsf{AEAD}}(\mathcal{D}) + \frac{Q_{\text{ON}}^2}{2^{\mathsf{nl}-2}} + \frac{Q_{\text{OFF}}}{2^{\mathsf{kl}-3}}$$

with $Q_{\mathcal{A}} = (Q_{\text{ON}}, N, Q_{\text{ON}}, Q_{\text{ON}}, Q_{\text{ON}}, Q_{\text{ON}}, Q_{\text{ON}}, Q_{\text{ON}}, C, Q_{\text{OFF}})$ *and*

$$\begin{aligned} Q_{\mathcal{B}} &= (N, Q_{\text{OFF}}, C), \\ Q_{\mathcal{C}} &= (Q_{\text{ON}}, C, 2Q_{\text{ON}}, 3Q_{\text{ON}}), \\ Q_{\mathcal{D}} &= (N, C, 3Q_{\text{ON}}). \end{aligned}$$

INTERPRETATION. Apart from statistical terms, the above stated theorems use multi-user AEAD notions and the password guessing advantage with adaptive compromises. For the latter, we have highlighted the number of guesses in gray. Observe that this number is linear in the number of offline queries, which can be quite large and therefore imply a significant security loss when considering low-entropy password distributions. This is, however, expected in this adversary model since the adversary controls the server and can therefore brute-force passwords offline.

With regard to the AEAD notions, we stress that they are all standard notions that are lifted to the multi-user setting. It is worth noting that IND-PAS is the potentially less standard multi-bit guessing model, cf. also [JSSOW17], whereas the IND-CCA notion is in the single-bit guessing model. This makes sense for the proof strategy and is the main reason how we can achieve security in the adaptive compromise setting. The IND-PAS security is used for the encrypted master keys (more concretely, the seed), whereas IND-CCA security is used for files. Clearly, it must be disallowed in the confidentiality game to compromise the user and ask for a challenge. However, we need to replace the master key by a random key independently of whether this key will be corrupted later or whether a file key (derived from the master key) will later be used to encrypt a challenge file. Both IND-PAS and IND-CCA can be shown to be implied non-tightly by the respective single-user variant. In terms of tightness, [JSSOW17] shows that a linear loss is inherent. However, since our games are parameterized by compromises, we can apply the result from [BRTZ24] to get a loss linear in the number of compromises for the IND-PAS advantage. Unfortunately, this does not work for the IND-CCA advantage in a straightforward way. Modifying the proof in a non-trivial way, we conjecture however that it is possible here as well.

7.2 Security against Network Adversaries

The following corollaries are obtained by combining Theorem 3 and Lemma 4 with Theorems 12 to 14, respectively, where we fold together adversaries against the same AEAD notions and upper bound the number of queries accordingly. We provide an interpretation of these bounds below.

Corollary 3 (Net-Conf). *For any password distribution $\mathcal{PW}$ and adversary $\mathcal{A}$ against the* Net-Conf *security of* $\mathsf{CS} := \mathsf{CS}[\mathsf{Sig2HDH}, \mathsf{AEAD}, \mathsf{MAC}]$*, there exist a* PW-Guess *adversary $\mathcal{B}_1$ against $\mathcal{PW}$, a* CT-OMCDH *adversary $\mathcal{B}_2$ against $(\mathbb{G}, q, g)$, a* UF *adversary $\mathcal{B}_3$ against* Sig*, an* INT-CTXT *adversary $\mathcal{C}_1$ against* AEAD*, an* OW-CPA *adversary $\mathcal{D}$ against* AEAD*, and an* IND-CCA *adversary $\mathcal{E}$ against* AEAD *with $t_{\mathcal{A}} \approx t_{\mathcal{B}_1} \approx t_{\mathcal{B}_2} \approx t_{\mathcal{B}_3} \approx t_{\mathcal{C}} - C \cdot Q_{\mathrm{ON}} \approx t_{\mathcal{D}} - C \cdot Q_{\mathrm{ON}} \approx t_{\mathcal{E}}$ such that*

$$\begin{aligned}\mathsf{Adv}^{Q_{\mathcal{A}}\text{-Net-Conf}}_{\mathsf{CS},\mathcal{PW}}(\mathcal{A}) \leq\ & 2 \cdot \mathsf{Adv}^{Q_{\mathcal{B}_1}\text{-PW-Guess}}_{\mathcal{PW}}(\mathcal{B}_1) + 2 \cdot \mathsf{Adv}^{Q_{\mathcal{B}_2}\text{-CT-OMCDH}}_{\mathbb{G},q,g}(\mathcal{B}_2) + 2 \cdot \mathsf{Adv}^{Q_{\mathrm{ON}}\text{-UF}}_{\mathsf{Sig}}(\mathcal{B}_3) \\ & + 4 \cdot \mathsf{Adv}^{Q_{\mathcal{C}}\text{-INT-CTXT}}_{\mathsf{AEAD}}(\mathcal{C}) + 4 \cdot \mathsf{Adv}^{Q_{\mathcal{D}}\text{-IND-PAS}}_{\mathsf{AEAD}}(\mathcal{D}) + \mathsf{Adv}^{Q_{\mathcal{E}}\text{-IND-CCA}}_{\mathsf{AEAD}}(\mathcal{E}) \\ & + \frac{Q_{\mathrm{ON}}^2}{2^{\mathsf{nl}-2}} + \frac{Q_{\mathrm{OFF}}}{2^{\mathsf{kl}-4}}\end{aligned}$$

with $Q_{\mathcal{A}} = (Q_{\mathrm{ON}}, N, Q_{\mathrm{ON}}, Q_{\mathrm{ON}}, Q_{\mathrm{ON}}, Q_{\mathrm{ON}}, Q_{\mathrm{ON}}, Q_{\mathrm{ON}}, Q_{\mathrm{ON}}, C, Q_{\mathrm{OFF}})$ *and*

$$\begin{aligned}Q_{\mathcal{B}_1} &= (N, \boxed{Q_{\mathrm{ON}}}, C) \\ Q_{\mathcal{B}_2} &= (N, Q_{\mathrm{OFF}}, C, \boxed{Q_{\mathrm{ON}}}, 3Q_{\mathrm{OFF}}) \\ Q_{\mathcal{C}} &= (N, C, 3Q_{\mathrm{ON}}, 4Q_{\mathrm{ON}}), \\ Q_{\mathcal{D}} &= (N, C, 4Q_{\mathrm{ON}}), \\ Q_{\mathcal{E}} &= (2Q_{\mathrm{ON}}, 2Q_{\mathrm{ON}}, 2Q_{\mathrm{ON}}, Q_{\mathrm{ON}}, Q_{\mathrm{ON}}).\end{aligned}$$

Corollary 4 (Net-Int). *For any password distribution $\mathcal{PW}$ and adversary $\mathcal{A}$ against the* Net-Int *security of* $\mathsf{CS} := \mathsf{CS}[\mathsf{Sig2HDH}, \mathsf{AEAD}, \mathsf{MAC}]$*, there exist a* PW-Guess *adversary $\mathcal{B}_1$ against $\mathcal{PW}$, a* CT-OMCDH *adversary $\mathcal{B}_2$ against $(\mathbb{G}, q, g)$, a* UF *adversary $\mathcal{B}_3$ against* Sig*, an* INT-CTXT *adversary $\mathcal{C}$ against* AEAD*, and an*

IND-PAS *adversary* $\mathcal{D}$ *against* AEAD *with* $t_{\mathcal{A}} \approx t_{\mathcal{B}_1} \approx t_{\mathcal{B}_2} \approx t_{\mathcal{B}_3} \approx t_{\mathcal{C}} - C \cdot Q_{\text{ON}} \approx t_{\mathcal{D}} - C \cdot Q_{\text{ON}}$ *such that*

$$\begin{aligned}\mathsf{Adv}^{Q_{\mathcal{A}}\text{-Net-Int}}_{\mathsf{CS},\mathcal{PW}}(\mathcal{A}) \leq \mathsf{Adv}^{Q_{\mathcal{B}_1}\text{-PW-Guess}}_{\mathcal{PW}}(\mathcal{B}_1) + \mathsf{Adv}^{Q_{\mathcal{B}_2}\text{-CT-OMCDH}}_{\mathbb{G},q,g}(\mathcal{B}_2) + 2 \cdot \mathsf{Adv}^{Q_{\text{ON}}\text{-UF}}_{\mathsf{Sig}}(\mathcal{B}_3) \\ + \mathsf{Adv}^{Q_{\mathcal{C}}\text{-INT-CTXT}}_{\mathsf{AEAD}}(\mathcal{C}) + \mathsf{Adv}^{Q_{\mathcal{D}}\text{-IND-PAS}}_{\mathsf{AEAD}}(\mathcal{D}) + \frac{Q_{\text{ON}}^2}{2^{\mathsf{nl}-2}} + \frac{Q_{\text{OFF}}}{2^{\mathsf{kl}-3}}\end{aligned}$$

with $Q_{\mathcal{A}} = (Q_{\text{ON}}, Q_{\text{ON}}, N, Q_{\text{ON}}, Q_{\text{ON}}, Q_{\text{ON}}, Q_{\text{ON}}, Q_{\text{ON}}, Q_{\text{ON}}, C, Q_{\text{OFF}})$ *and*

$$\begin{aligned}Q_{\mathcal{B}_1} &= (N, \boxed{Q_{\text{ON}}}, C) \\ Q_{\mathcal{B}_2} &= (N, Q_{\text{OFF}}, C, \boxed{Q_{\text{ON}}}, 3Q_{\text{OFF}}) \\ Q_{\mathcal{C}} &= (Q_{\text{ON}}, C, 2Q_{\text{ON}}, 3Q_{\text{ON}}), \\ Q_{\mathcal{D}} &= (N, C, 3Q_{\text{ON}}).\end{aligned}$$

Corollary 5 (Explicit Authentication). *For any password distribution* $\mathcal{PW}$ *and adversary* $\mathcal{A}$ *against the* Exp-Auth *security of* $\mathsf{CS} := \mathsf{CS}[\mathsf{Sig2HDH}, \mathsf{AEAD}, \mathsf{MAC}]$*, there exist a* PW-Guess *adversary* $\mathcal{B}_1$ *against* $\mathcal{PW}$*, a* CT-OMCDH *adversary* $\mathcal{B}_2$ *against* $(\mathbb{G}, q, g)$*, a* UF *adversary* $\mathcal{B}_3$ *against* Sig*, and an* SUF *adversary* $\mathcal{C}$ *against* MAC *with* $t_{\mathcal{A}} \approx t_{\mathcal{B}_1} \approx t_{\mathcal{B}_2} \approx t_{\mathcal{B}_3} \approx t_{\mathcal{C}}$ *such that*

$$\begin{aligned}\mathsf{Adv}^{Q_{\mathcal{A}}\text{-Exp-Auth}}_{\mathsf{CS},\mathcal{PW}}(\mathcal{A}) \leq \mathsf{Adv}^{Q_{\mathcal{B}_1}\text{-PW-Guess}}_{\mathcal{PW}}(\mathcal{B}_1) + \mathsf{Adv}^{Q_{\mathcal{B}_2}\text{-CT-OMCDH}}_{\mathbb{G},q,g}(\mathcal{B}_2) + 2 \cdot \mathsf{Adv}^{Q_{\text{ON}}\text{-UF}}_{\mathsf{Sig}}(\mathcal{B}_3) \\ + \mathsf{Adv}^{Q_{\mathcal{C}}\text{-SUF}}_{\mathsf{MAC}}(\mathcal{C}) + \frac{Q_{\text{OFF}}}{2^{\mathsf{kl}-3}}\end{aligned}$$

with $Q_{\mathcal{A}} = (Q_{\text{ON}}, Q_{\text{ON}}, N, Q_{\text{ON}}, Q_{\text{ON}}, Q_{\text{ON}}, Q_{\text{ON}}, Q_{\text{ON}}, Q_{\text{ON}}, C, Q_{\text{OFF}})$ *and*

$$\begin{aligned}Q_{\mathcal{B}_1} &= (N, \boxed{Q_{\text{ON}}}, C) \\ Q_{\mathcal{B}_2} &= (N, Q_{\text{OFF}}, C, \boxed{Q_{\text{ON}}}, 3Q_{\text{OFF}}) \\ Q_{\mathcal{C}} &= (N, C, Q_{\text{ON}}, Q_{\text{ON}}).\end{aligned}$$

Interpretation. The above statements show that in this weaker adversarial model, we can achieve better bounds for the password guessing advantage. The number of password guesses (highlighted in gray) now depends on online queries and, more specifically, only on the number of online authentication attempts. This is clearly desirable in practice and is state-of-the-art for other password-based primitives like password authenticated key exchange [BPR00]. We note that this property is due to the use of 2HDH in combination with server authentication (cf. also Remark 3). In a sense, we trade offline password guessing for additional computational assumptions. We also highlight the number of Exp queries for the CT-OMCDH advantage since those can affect the concrete security, cf. the hierarchy of OMDH in the AGM [JX25].

7.3 Further Discussion and Future Work

We want to highlight that our new definitions for network adversaries are not meant to replace the original definitions of [BDG+24]. Our definitions themselves are not sufficient because they do not capture the scenario of a malicious server. For real-world systems, we would therefore recommend that they satisfy security in both adversary models.

It might be beneficial to present a combined notion for each property and prove that the two notions imply the combined one. However, given that the area is still early-stage, small changes would require to reprove that result. Therefore, we motivate more research on what are the actual requirements and whether the current security definitions still need slight adaptations. In our network adversary notions, the adversary is never allowed to corrupt the server. This is in contrast to definitions for other contexts, e.g., authenticated key exchange, where long-term key corruptions are allowed in certain cases, to capture properties like forward

secrecy or resistance against key compromise impersonation attacks. We consider it an interesting open problem whether similar properties can be defined and achieved in the cloud storage setting.

Our contributions also provide the first step towards a post-quantum secure construction in the network adversary setting. We hope that our abstraction of the authentication protocol will allow for instantiations using post-quantum secure OPRFs (or variants thereof), which we leave as an interesting open problem. Similarly, there is currently no proof in the standard model. We do not see any reason why a standard-model proof should not work in the setting with selective corruptions. However, in the adaptive corruption setting, this seems much more challenging and a more promising direction could be a proof in the quantum-accessible random oracle model.

Acknowledgments

We thank the anonymous reviewers for their careful reading of this work and for pointing out several points that helped improve the paper, in particular, the argument for explicit authentication and the subtleties in the security model of BDGHP. We also thank Aysan Nishaburi and Emiel Wiedijk for pointing out an issue with the previous proof of 2HDH. Jonas Janneck was supported by the European Union (ERC AdG REWORC - 101054911).

References

ABCP24. Martin R. Albrecht, Matilda Backendal, Daniele Coppola, and Kenneth G. Paterson. Share with care: Breaking E2EE in Nextcloud. In *2024 IEEE European Symposium on Security and Privacy*, pages 828–840. IEEE Computer Society Press, July 2024. doi:10.1109/EuroSP60621.2024.00051. (Cited on p. 4).

ABR01. Michel Abdalla, Mihir Bellare, and Phillip Rogaway. The oracle Diffie-Hellman assumptions and an analysis of DHIES. In David Naccache, editor, *CT-RSA 2001*, volume 2020 of *LNCS*, pages 143–158. Springer, Berlin, Heidelberg, April 2001. doi:10.1007/3-540-45353-9_12. (Cited on p. 11).

AHMP23. Martin R. Albrecht, Miro Haller, Lenka Mareková, and Kenneth G. Paterson. Caveat implementor! Key recovery attacks on MEGA. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 190–218. Springer, Cham, April 2023. doi:10.1007/978-3-031-30589-4_7. (Cited on p. 4).

App. Apple. Advanced data protection for icloud. https://support.apple.com/guide/security/advanced-data-protection-for-icloud-sec973254c5f/web. Accessed: 2025-09-30. (Cited on p. 4).

BCQ+13. Alysson Bessani, Miguel Correia, Bruno Quaresma, Fernando André, and Paulo Sousa. Depsky: Dependable and secure storage in a cloud-of-clouds. *ACM Trans. Storage*, 9(4), November 2013. doi:10.1145/2535929. (Cited on p. 8).

BDFH25. Ward Beullens, Lucas Dodgson, Sebastian H. Faller, and Julia Hesse. The 2Hash OPRF framework and efficient post-quantum instantiations. In Serge Fehr and Pierre-Alain Fouque, editors, *EUROCRYPT 2025, Part VIII*, volume 15608 of *LNCS*, pages 332–362. Springer, Cham, May 2025. doi:10.1007/978-3-031-91101-9_12. (Cited on p. 5).

BDG+24. Matilda Backendal, Hannah Davis, Felix Günther, Miro Haller, and Kenneth G. Paterson. A formal treatment of end-to-end encrypted cloud storage. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part II*, volume 14921 of *LNCS*, pages 40–74. Springer, Cham, August 2024. doi:10.1007/978-3-031-68379-4_2. (Cited on pp. 4, 8, 11, 16, 17, 18, 19, 24, 25, 29, 31, and 66).

BFL20. Balthazar Bauer, Georg Fuchsbauer, and Julian Loss. A classification of computational assumptions in the algebraic group model. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 121–151. Springer, Cham, August 2020. doi:10.1007/978-3-030-56880-1_5. (Cited on p. 11).

BFP21. Balthazar Bauer, Georg Fuchsbauer, and Antoine Plouviez. The one-more discrete logarithm assumption in the generic group model. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part IV*, volume 13093 of *LNCS*, pages 587–617. Springer, Cham, December 2021. doi:10.1007/978-3-030-92068-5_20. (Cited on p. 11).

BHP23. Matilda Backendal, Miro Haller, and Kenneth G. Paterson. MEGA: Malleable encryption goes awry. In *2023 IEEE Symposium on Security and Privacy*, pages 146–163. IEEE Computer Society Press, May 2023. doi:10.1109/SP46215.2023.10179290. (Cited on p. 4).

BLMR13. Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. Key homomorphic PRFs and their applications. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 410–428. Springer, Berlin, Heidelberg, August 2013. doi:10.1007/978-3-642-40041-4_23. (Cited on p. 8).

BM93. Steven M. Bellovin and Michael Merritt. Augmented encrypted key exchange: A password-based protocol secure against dictionary attacks and password file compromise. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93*, pages 244–250. ACM Press, November 1993. doi:10.1145/168588.168618. (Cited on pp. 8 and 25).

Bol03. Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In Yvo Desmedt, editor, *PKC 2003*, volume 2567 of *LNCS*, pages 31–46. Springer, Berlin, Heidelberg, January 2003. doi:10.1007/3-540-36288-6_3. (Cited on pp. 10 and 11).

BPR00. Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 139–155. Springer, Berlin, Heidelberg, May 2000. doi:10.1007/3-540-45539-6_11. (Cited on pp. 5 and 31).

BR93. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93*, pages 62–73. ACM Press, November 1993. doi:10.1145/168588.168596. (Cited on p. 9).

BR94. Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 232–249. Springer, Berlin, Heidelberg, August 1994. doi:10.1007/3-540-48329-2_21. (Cited on p. 5).

BR06. Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 409–426. Springer, Berlin, Heidelberg, May / June 2006. doi:10.1007/11761679_25. (Cited on p. 9).

BRTZ24. Mihir Bellare, Doreen Riepel, Stefano Tessaro, and Yizhao Zhang. Count corruptions, not users: Improved tightness for signatures, encryption and authenticated key exchange. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part II*, volume 15485 of *LNCS*, pages 326–360. Springer, Singapore, December 2024. doi:10.1007/978-981-96-0888-1_11. (Cited on p. 30).

BS23. Mihir Bellare and Laura Shea. Flexible password-based encryption: Securing cloud storage and provably resisting partitioning-oracle attacks. In Mike Rosulek, editor, *CT-RSA 2023*, volume 13871 of *LNCS*, pages 594–621. Springer, Cham, April 2023. doi:10.1007/978-3-031-30872-7_23. (Cited on p. 8).

CHGY22. Weikeng Chen, Thang Hoang, Jorge Guajardo, and Attila A. Yavuz. Titanium: A metadata-hiding file-sharing system with malicious security. In *NDSS 2022*. The Internet Society, April 2022. (Cited on p. 8).

CKM23. Elizabeth C. Crites, Chelsea Komlo, and Mary Maller. Fully adaptive Schnorr threshold signatures. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part I*, volume 14081 of *LNCS*, pages 678–709. Springer, Cham, August 2023. doi:10.1007/978-3-031-38557-5_22. (Cited on p. 5).

CLTY22. Long Chen, Ya-Nan Li, Qiang Tang, and Moti Yung. End-to-same-end encryption: Modularly augmenting an app with an efficient, portable, and blind cloud storage. In Kevin R. B. Butler and Kurt Thomas, editors, *USENIX Security 2022*, pages 2353–2370. USENIX Association, August 2022. URL: https://www.usenix.org/conference/usenixsecurity22/presentation/chen-long. (Cited on p. 8).

CP20. Weikeng Chen and Raluca Ada Popa. Metal: A metadata-hiding file-sharing system. In *NDSS 2020*. The Internet Society, February 2020. doi:10.14722/ndss.2020.24095. (Cited on p. 8).

DGRW18. Yevgeniy Dodis, Paul Grubbs, Thomas Ristenpart, and Joanne Woodage. Fast message franking: From invisible salamanders to encryptment. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 155–186. Springer, Cham, August 2018. doi:10.1007/978-3-319-96884-1_6. (Cited on p. 7).

DHL22. Poulami Das, Julia Hesse, and Anja Lehmann. DPaSE: Distributed password-authenticated symmetric-key encryption, or how to get many keys from one password. In Yuji Suga, Kouichi Sakurai, Xuhua Ding, and Kazue Sako, editors, *ASIACCS 22*, pages 682–696. ACM Press, May / June 2022. doi:10.1145/3488932.3517389. (Cited on p. 8).

FL25. Karla Friedrichs, Anja Lehmann, and Cavit Özbay. Game changer: A modular framework for OPRF security. In *ASIACRYPT 2025*, LNCS. Springer, Singapore, December 2025. (Cited on p. 7).

GJ18. Kristian Gjøsteen and Tibor Jager. Practical and tightly-secure digital signatures and authenticated key exchange. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 95–125. Springer, Cham, August 2018. doi:10.1007/978-3-319-96881-0_4. (Cited on p. 6).

GSMB03. Eu-Jin Goh, Hovav Shacham, Nagendra Modadugu, and Dan Boneh. SiRiUS: Securing remote untrusted storage. In *NDSS 2003*. The Internet Society, February 2003. (Cited on p. 8).

HKL+25. Lena Heimberger, Daniel Kales, Riccardo Lolato, Omid Mir, Sebastian Ramacher, and Christian Rechberger. Leap: A fast, lattice-based OPRF with application to private set intersection. In Serge Fehr and Pierre-Alain Fouque, editors, *EUROCRYPT 2025, Part VII*, volume 15607 of *LNCS*, pages 254–283. Springer, Cham, May 2025. doi:10.1007/978-3-031-91098-2_10. (Cited on p. 5).

HR23. Nadia Heninger and Keegan Ryan. The hidden number problem with small unknown multipliers: Cryptanalyzing MEGA in six queries and other applications. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *PKC 2023, Part I*, volume 13940 of *LNCS*, pages 147–176. Springer, Cham, May 2023. doi:10.1007/978-3-031-31368-4_6. (Cited on p. 4).

HT24. Jonas Hofmann and Kien Tuong Truong. End-to-end encrypted cloud storage in the wild: A broken ecosystem. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *ACM CCS 2024*, pages 3988–4001. ACM Press, October 2024. doi:10.1145/3658644.3690309. (Cited on p. 4).

Jae23. Joseph Jaeger. Let attackers program ideal models: Modularity and composability for adaptive compromise. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part III*, volume 14006 of *LNCS*, pages 101–131. Springer, Cham, April 2023. doi:10.1007/978-3-031-30620-4_4. (Cited on p. 6).

Jae25. Joseph Jaeger. Nonadaptive one-way to hiding implies adaptive quantum reprogramming. In *ASIACRYPT 2025, Part VIII*, LNCS, pages 287–319. Springer, Singapore, December 2025. doi:10.1007/978-981-95-5125-5_10. (Cited on p. 8).

JKK14. Stanislaw Jarecki, Aggelos Kiayias, and Hugo Krawczyk. Round-optimal password-protected secret sharing and T-PAKE in the password-only model. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part II*, volume 8874 of *LNCS*, pages 233–253. Springer, Berlin, Heidelberg, December 2014. doi:10.1007/978-3-662-45608-8_13. (Cited on p. 4).

JKR19. Stanislaw Jarecki, Hugo Krawczyk, and Jason K. Resch. Updatable oblivious key management for storage systems. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 379–393. ACM Press, November 2019. doi:10.1145/3319535.3363196. (Cited on p. 8).

JKRS21. Tibor Jager, Eike Kiltz, Doreen Riepel, and Sven Schäge. Tightly-secure authenticated key exchange, revisited. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 117–146. Springer, Cham, October 2021. doi:10.1007/978-3-030-77870-5_5. (Cited on p. 6).

JR26. Jonas Janneck and Doreen Riepel. Secure cloud storage: Modularization, network adversaries and adaptive corruptions. LNCS, pages 66–96, 2026. doi:10.1007/978-3-032-25317-0_3. (Cited on p. 3).

JSSOW17. Tibor Jager, Martijn Stam, Ryan Stanley-Oakes, and Bogdan Warinschi. Multi-key authenticated encryption with corruptions: Reductions are lossy. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 409–441. Springer, Cham, November 2017. doi:10.1007/978-3-319-70500-2_14. (Cited on pp. 10, 30, and 35).

JT20. Joseph Jaeger and Nirvan Tyagi. Handling adaptive compromise for practical encryption schemes. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 3–32. Springer, Cham, August 2020. doi:10.1007/978-3-030-56784-2_1. (Cited on p. 6).

JX25. Jake Januzelli and Jiayu Xu. A complete characterization of one-more assumptions in the algebraic group model. In *ASIACRYPT 2025, Part IV*, LNCS, pages 206–238. Springer, Singapore, December 2025. doi:10.1007/978-981-95-5113-2_7. (Cited on pp. 11 and 31).

LER+18. Russell W. F. Lai, Christoph Egger, Manuel Reinert, Sherman S. M. Chow, Matteo Maffei, and Dominique Schröder. Simple password-hardened encryption services. In William Enck and Adrienne Porter Felt, editors, *USENIX Security 2018*, pages 1405–1421. USENIX Association, August 2018. URL: https://www.usenix.org/conference/usenixsecurity18/presentation/lai. (Cited on p. 8).

LESC17. Russell W. F. Lai, Christoph Egger, Dominique Schröder, and Sherman S. M. Chow. Phoenix: Rebirth of a cryptographic password-hardening service. In Engin Kirda and Thomas Ristenpart, editors, *USENIX Security 2017*, pages 899–916. USENIX Association, August 2017. URL: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/lai. (Cited on p. 8).

LGR21. Julia Len, Paul Grubbs, and Thomas Ristenpart. Partitioning oracle attacks. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021*, pages 195–212. USENIX Association, August 2021. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/len. (Cited on p. 7).

LSTY25. Ya-Nan Li, Yaqing Song, Qiang Tang, and Moti Yung. End-to-end encrypted git services. In *ACM CCS 2025*, pages 468–482. ACM Press, November 2025. doi:10.1145/3719027.3744815. (Cited on p. 8).
MEG. MEGA. https://mega.io. Accessed: 2025-09-30. (Cited on p. 4).
Nex. Nextcloud. https://nextcloud.com. Accessed: 2025-09-30. (Cited on p. 4).
Nie02. 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. Springer, Berlin, Heidelberg, August 2002. doi:10.1007/3-540-45708-9_8. (Cited on p. 6).
PSV+14. Raluca Ada Popa, Emily Stark, Steven Valdez, Jonas Helfer, Nickolai Zeldovich, and Hari Balakrishnan. Building web applications on top of encrypted data using mylar. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 157–172, Seattle, WA, April 2014. USENIX Association. URL: https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/popa. (Cited on p. 8).
PWZ23. Jiaxin Pan, Benedikt Wagner, and Runzhi Zeng. Tighter security for generic authenticated key exchange in the QROM. In Jian Guo and Ron Steinfeld, editors, *ASIACRYPT 2023, Part IV*, volume 14441 of *LNCS*, pages 401–433. Springer, Singapore, December 2023. doi:10.1007/978-981-99-8730-6_13. (Cited on p. 8).
TMRM18. Nirvan Tyagi, Muhammad Haris Mughees, Thomas Ristenpart, and Ian Miers. BurnBox: Self-revocable encryption in a world of compelled access. In William Enck and Adrienne Porter Felt, editors, *USENIX Security 2018*, pages 445–461. USENIX Association, August 2018. URL: https://www.usenix.org/conference/usenixsecurity18/presentation/tyagi. (Cited on p. 8).

A Additional Preliminaries

A.1 AEAD Security Notions

The following multi-user multi-bit security notion is taken from [JSSOW17] where it is called $\mathsf{LRIND\text{-}PAS}^{\mathsf{NR},n}$. This stands for left-or-right indistinguishability under passive attacks for adversaries without nonce-reuse. They are defined in a multi-bit setting.

Definition 14 (IND-PAS). *Indistinguishability under chosen passive attacks for an* AEAD *is defined via the game in Figure 16. We define the advantage of an adversary $\mathcal{A}$ as*

$$\mathsf{Adv}^{Q\text{-}\mathsf{IND\text{-}PAS}}_{\mathsf{AEAD}}(\mathcal{A}) := 2\,\mathsf{Pr}\left[Q\text{-}\mathsf{IND\text{-}PAS}_{\mathsf{AEAD}}(\mathcal{A}) \Rightarrow 1\right] - 1 \ ,$$

where $Q = (n, Q_{\textsc{Comp}}, Q_{\textsc{Chl}})$.

Game Q-IND-PAS$_{\mathsf{AEAD}}(\mathcal{A})$
00 $\mathcal{S}_{\mathsf{cmp}}, \mathcal{S}_{\mathsf{n}} \leftarrow \emptyset$
01 $\mathsf{b}_1, \ldots, \mathsf{b}_n \xleftarrow{\$} \{0,1\}$
02 **for** $i \in [n]$
03 $\quad \mathsf{k}_i \xleftarrow{\$} \{0,1\}^{\mathsf{kl}}$
04 $(i, \mathsf{b}') \xleftarrow{\$} \mathcal{A}^{\textsc{Chl},\textsc{Comp}}$
05 **return** $[\![\mathsf{b}_i = \mathsf{b}' \wedge i \notin \mathcal{S}_{\mathsf{cmp}}]\!]$

Oracle $\textsc{Comp}(i \in [n])$
06 $\mathcal{S}_{\mathsf{cmp}} \leftarrow \mathcal{S}_{\mathsf{cmp}} \cup \{i\}$
07 **return** k_i

Oracle $\textsc{Chl}(i \in [n], \mathsf{n}, \mathsf{m}_0, \mathsf{m}_1, \mathsf{ad})$
08 **if** $(i, \mathsf{n}) \in \mathcal{S}_{\mathsf{n}}$
09 $\quad$ **return** $\perp$
10 $\mathcal{S}_{\mathsf{n}} \leftarrow \mathcal{S}_{\mathsf{n}} \cup \{(i, \mathsf{n})\}$
11 $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{k}_i, \mathsf{n}, \mathsf{m}_{\mathsf{b}_i}, \mathsf{ad})$
12 **return** ct

Fig. 16. Game defining IND-PAS security for an AEAD scheme $\mathsf{AEAD} := (\mathsf{Enc}, \mathsf{Dec})$ with adversary $\mathcal{A}$ making at most $Q_{\textsc{Comp}}$ queries to Comp and at most $Q_{\textsc{Chl}}$ queries to Chl.

For our proofs we actually need two other variants. First, a standard IND variant in the single-bit setting with a separate challenge oracle. Second, a one-wayness notion, as captured in Definition 16. The relations can be found in Appendix A.2.

Definition 15 (IND-CCA). *Indistinguishability under chosen ciphertext attacks for an* AEAD *is defined via the game in Figure 17. We define the advantage of an adversary* $\mathcal{A}$ *as*

$$\mathsf{Adv}^{Q\text{-IND-CCA}}_{\mathsf{AEAD}}(\mathcal{A}) \coloneqq 2\,\Pr\left[Q\text{-IND-CCA}_{\mathsf{AEAD}}(\mathcal{A}) \Rightarrow 1\right] - 1\ ,$$

where $Q = (n, Q_{\textsc{Comp}}, Q_{\textsc{Enc}}, Q_{\textsc{Dec}}, Q_{\textsc{Chl}})$.

Game Q-IND-CCA$_{\mathsf{AEAD}}(\mathcal{A})$

$\mathcal{S}_{\mathsf{cmp}}, \mathcal{S}_{\mathsf{n}}, \mathcal{S}_{\mathsf{chl}} \leftarrow \emptyset$
$\mathsf{b} \xleftarrow{\$} \{0,1\}$
for $i \in [n]$
  $\mathsf{k}_i \xleftarrow{\$} \{0,1\}^{\mathsf{kl}}$
$\mathsf{b}' \xleftarrow{\$} \mathcal{A}^{\textsc{Enc},\textsc{Dec},\textsc{Chl},\textsc{Comp}}$
return $[\![\mathsf{b} = \mathsf{b}' \wedge \mathcal{S}_{\mathsf{chl}} \cap \mathcal{S}_{\mathsf{cmp}} = \emptyset]\!]$

Oracle $\textsc{Chl}(i \in [n], \mathsf{n}, \mathsf{m}_0, \mathsf{m}_1, \mathsf{ad})$

if $(i, \mathsf{n}) \in \mathcal{S}_{\mathsf{n}}$
  return $\perp$
$\mathcal{S}_{\mathsf{n}} \leftarrow \mathcal{S}_{\mathsf{n}} \cup \{(i, \mathsf{n})\}$
$\mathcal{S}_{\mathsf{chl}} \leftarrow \mathcal{S}_{\mathsf{chl}} \cup \{i\}$
$\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{k}_i, \mathsf{n}, \mathsf{m}_{\mathsf{b}}, \mathsf{ad})$
$\mathsf{T}_{\mathsf{Enc}}[i, \mathsf{n}, \mathsf{ct}, \mathsf{ad}] \leftarrow \mathsf{m}$
return ct

Oracle $\textsc{Enc}(i \in [n], \mathsf{n}, \mathsf{m}, \mathsf{ad})$

if $(i, \mathsf{n}) \in \mathcal{S}_{\mathsf{n}}$
  return $\perp$
$\mathcal{S}_{\mathsf{n}} \leftarrow \mathcal{S}_{\mathsf{n}} \cup \{(i, \mathsf{n})\}$
$\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{k}_i, \mathsf{n}, \mathsf{m}, \mathsf{ad})$
$\mathsf{T}_{\mathsf{Enc}}[i, \mathsf{n}, \mathsf{ct}, \mathsf{ad}] \leftarrow \mathsf{m}$
return ct

Oracle $\textsc{Dec}(i \in [n], \mathsf{n}, \mathsf{ct}, \mathsf{ad})$

if $\mathsf{T}_{\mathsf{Enc}}[i, \mathsf{n}, \mathsf{ct}, \mathsf{ad}] \neq \perp$
  return $\perp$
$\mathsf{m} \leftarrow \mathsf{Dec}(\mathsf{k}_i, \mathsf{n}, \mathsf{ct}, \mathsf{ad})$
return m

Oracle $\textsc{Comp}(i \in [n])$

$\mathcal{S}_{\mathsf{cmp}} \leftarrow \mathcal{S}_{\mathsf{cmp}} \cup \{i\}$
return k_i

Fig. 17. Game defining IND-CCA security for an AEAD scheme $\mathsf{AEAD} \coloneqq (\mathsf{Enc}, \mathsf{Dec})$ with $Q = (n, Q_{\textsc{Comp}}, Q_{\textsc{Enc}}, Q_{\textsc{Dec}}, Q_{\textsc{Chl}})$ and adversary $\mathcal{A}$ making at most Q_{O} queries to oracle O.

Definition 16 (OW-CPA). *One-wayness under chosen plaintext attacks for an* AEAD *and a message length* ml *is defined via the game in Figure 18. We define the advantage of an adversary* $\mathcal{A}$ *as*

$$\mathsf{Adv}^{Q\text{-OW-CPA}}_{\mathsf{AEAD},\mathsf{ml}}(\mathcal{A}) \coloneqq \Pr\left[Q\text{-OW-CPA}_{\mathsf{AEAD},\mathsf{ml}}(\mathcal{A}) \Rightarrow 1\right],$$

where $Q = (n, Q_{\textsc{Comp}}, Q_{\textsc{Enc}}, Q_{\textsc{Chl}}, Q_{\textsc{Check}})$.

Definition 17 (INT-CTXT). *Ciphertext integrity is defined via the game in Figure 19. We define the advantage of an adversary* $\mathcal{A}$ *as*

$$\mathsf{Adv}^{Q\text{-INT-CTXT}}_{\mathsf{AEAD}}(\mathcal{A}) \coloneqq \Pr\left[Q\text{-INT-CTXT}_{\mathsf{AEAD}}(\mathcal{A}) \Rightarrow 1\right],$$

for $Q = (n, Q_{\textsc{Comp}}, Q_{\textsc{Enc}}, Q_{\textsc{Check}})$.

A.2 Relation for OW-CPA

Lemma 4 (IND-PAS $\Rightarrow$ OW-CPA). *If* AEAD *is correct, then for any adversary* $\mathcal{A}$ *against* OW-CPA *there exists an adversary* $\mathcal{B}$ *against* IND-PAS *with* $t_{\mathcal{A}} \approx t_{\mathcal{B}}$ *such that*

$$\mathsf{Adv}^{Q\text{-OW-CPA}}_{\mathsf{AEAD},\mathsf{ml}}(\mathcal{A}) \leq \mathsf{Adv}^{(n, Q_{\textsc{Comp}}, Q_{\textsc{Enc}}+Q_{\textsc{Chl}})\text{-IND-PAS}}_{\mathsf{AEAD}}(\mathcal{B}) + \frac{Q_{\textsc{Chl}} + Q_{\textsc{Check}}}{2^{\mathsf{ml}}},$$

with $Q = (n, Q_{\textsc{Comp}}, Q_{\textsc{Enc}}, Q_{\textsc{Chl}}, Q_{\textsc{Check}})$.

Game Q-OW-CPA$_{\mathsf{AEAD},\mathsf{ml}}(\mathcal{A})$

$\mathcal{S}_{\mathsf{cmp}}, \mathcal{S}_{\mathsf{n}} \leftarrow \emptyset$
for $i \in [n]$
  $\mathsf{k}_i \xleftarrow{\$} \{0,1\}^{\mathsf{kl}}$
$(i^\star, \mathsf{n}^\star, \mathsf{ct}^\star, \mathsf{ad}^\star, \mathsf{m}^\star) \xleftarrow{\$} \mathcal{A}^{\textsc{Enc},\textsc{Chl},\textsc{Comp},\textsc{Check}}$
return $[\![\mathsf{m}^\star \neq \bot \wedge \mathsf{m}^\star = \mathsf{T}_{\mathsf{Chl}}[i^\star, \mathsf{n}^\star, \mathsf{ct}^\star, \mathsf{ad}^\star]]\!]$
  $\wedge\ [\![i^\star \notin \mathcal{S}_{\mathsf{cmp}}]\!]$

Oracle $\textsc{Chl}(i \in [n], \mathsf{n}, j, \mathsf{ad})$

if $(i, \mathsf{n}) \in \mathcal{S}_{\mathsf{n}}$
  return $\bot$
$\mathcal{S}_{\mathsf{n}} \leftarrow \mathcal{S}_{\mathsf{n}} \cup \{(i, \mathsf{n})\}$
if $\mathsf{T}_{\mathsf{m}}[j] = \bot$
  $\mathsf{T}_{\mathsf{m}}[j] \xleftarrow{\$} \{0,1\}^{\mathsf{ml}}$
$\mathsf{m} \leftarrow \mathsf{T}_{\mathsf{m}}[j]$
$\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{k}_i, \mathsf{n}, \mathsf{m}, \mathsf{ad})$
$\mathsf{T}_{\mathsf{Chl}}[i, \mathsf{n}, \mathsf{ct}, \mathsf{ad}] \leftarrow \mathsf{m}$
return ct

Oracle $\textsc{Enc}(i \in [n], \mathsf{n}, \mathsf{m}, \mathsf{ad})$

if $(i, \mathsf{n}) \in \mathcal{S}_{\mathsf{n}}$
  return $\bot$
$\mathcal{S}_{\mathsf{n}} \leftarrow \mathcal{S}_{\mathsf{n}} \cup \{(i, \mathsf{n})\}$
$\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{k}_i, \mathsf{n}, \mathsf{m}, \mathsf{ad})$
return ct

Oracle $\textsc{Comp}(i \in [n])$

$\mathcal{S}_{\mathsf{cmp}} \leftarrow \mathcal{S}_{\mathsf{cmp}} \cup \{i\}$
return k_i

Oracle $\textsc{Check}(i \in [n], \mathsf{n}, \mathsf{ct}, \mathsf{ad}, \mathsf{m})$

if $\mathsf{T}_{\mathsf{Chl}}[i, \mathsf{n}, \mathsf{ct}, \mathsf{ad}] = \bot$
  return $\bot$
return $[\![\mathsf{m} = \mathsf{Dec}(\mathsf{k}_i, \mathsf{n}, \mathsf{ct}, \mathsf{ad})]\!]$

Fig. 18. Game defining OW-CPA security for an AEAD scheme $\mathsf{AEAD} \coloneqq (\mathsf{Enc}, \mathsf{Dec})$ and message length ml with $Q = (n, Q_{\textsc{Comp}}, Q_{\textsc{Enc}}, Q_{\textsc{Chl}}, Q_{\textsc{Check}})$ and adversary $\mathcal{A}$ making at most Q_{O} queries to oracle O.

Game Q-INT-CTXT$_{\mathsf{AEAD}}(\mathcal{A})$

$\mathcal{S}_{\mathsf{cmp}}, \mathcal{S}_{\mathsf{n}} \leftarrow \emptyset$
for $i \in [n]$
  $\mathsf{k}_i \xleftarrow{\$} \{0,1\}^{\mathsf{kl}}$
$(i^\star, \mathsf{n}^\star, \mathsf{ct}^\star, \mathsf{ad}^\star) \xleftarrow{\$} \mathcal{A}^{\textsc{Enc},\textsc{Comp},\textsc{Check}}$
return $[\![\mathsf{Dec}(\mathsf{k}_{i^\star}, \mathsf{n}^\star, \mathsf{ct}^\star, \mathsf{ad}^\star) \neq \bot]\!]$
  $\wedge\ [\![\mathsf{T}_{\mathsf{Enc}}[i^\star, \mathsf{n}^\star, \mathsf{ct}^\star, \mathsf{ad}^\star] = \bot \wedge i^\star \notin \mathcal{S}_{\mathsf{cmp}}]\!]$

Oracle $\textsc{Comp}(i \in [n])$

$\mathcal{S}_{\mathsf{cmp}} \leftarrow \mathcal{S}_{\mathsf{cmp}} \cup \{i\}$
return k_i

Oracle $\textsc{Enc}(i \in [n], \mathsf{n}, \mathsf{m}, \mathsf{ad})$

if $(i, \mathsf{n}) \in \mathcal{S}_{\mathsf{n}}$
  return $\bot$
$\mathcal{S}_{\mathsf{n}} \leftarrow \mathcal{S}_{\mathsf{n}} \cup \{(i, \mathsf{n})\}$
$\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{k}_i, \mathsf{n}, \mathsf{m}, \mathsf{ad})$
$\mathsf{T}_{\mathsf{Enc}}[i, \mathsf{n}, \mathsf{ct}, \mathsf{ad}] \leftarrow 1$
return ct

Oracle $\textsc{Check}(i \in [n], \mathsf{n}, \mathsf{ct}, \mathsf{ad})$

return $[\![\mathsf{Dec}(\mathsf{k}_i, \mathsf{n}, \mathsf{ct}, \mathsf{ad}) \neq \bot]\!]$

Fig. 19. Game defining INT-CTXT security for an AEAD scheme $\mathsf{AEAD} \coloneqq (\mathsf{Enc}, \mathsf{Dec})$ with $Q = (n, Q_{\textsc{Comp}}, Q_{\textsc{Enc}}, Q_{\textsc{Check}})$ and adversary $\mathcal{A}$ making at most Q_{O} queries to oracle O.

Proof. To prove the statement, we depict a sequence of games in Figure 20.

Game G_0**.** This is the Q-OW-CPA game for $Q = (n, Q_{\textsc{Comp}}, Q_{\textsc{Enc}}, Q_{\textsc{Chl}}, Q_{\textsc{Check}})$:

$$\Pr[\mathsf{G}_0(\mathcal{A}) \Rightarrow 1] = \mathsf{Adv}^{Q\text{-OW-CPA}}_{\mathsf{AEAD},\mathsf{ml}}(\mathcal{A}).$$

Game G_1**.** This game is the same as the previous one except that the challenger is choosing an additional message m_1 in the challenge oracle. This message is drawn uniformly at random from the message space. Further, the new message is stored together with the old one in table $\mathsf{T}_{\mathsf{Chl}}$. Since this conceptual change does not influence the winning probability of the, we obtain

$$\Pr[\mathsf{G}_0(\mathcal{A}) \Rightarrow 1] = \Pr[\mathsf{G}_1(\mathcal{A}) \Rightarrow 1].$$

Game G_2**.** This is the same game as before except that in the check oracle the challenger does not compute the real decryption of the input ciphertext but checks if the input message m equals the message stored in table $\mathsf{T}_{\mathsf{Chl}}$, i.e. the message that was encrypted in the challenge oracle. Note that the check oracle can only be queried on elements corresponding to a previous challenge query. Due to the assumption of (perfect) correctness of AEAD, this change is only conceptual:

$$\Pr[\mathsf{G}_1(\mathcal{A}) \Rightarrow 1] = \Pr[\mathsf{G}_2(\mathcal{A}) \Rightarrow 1].$$

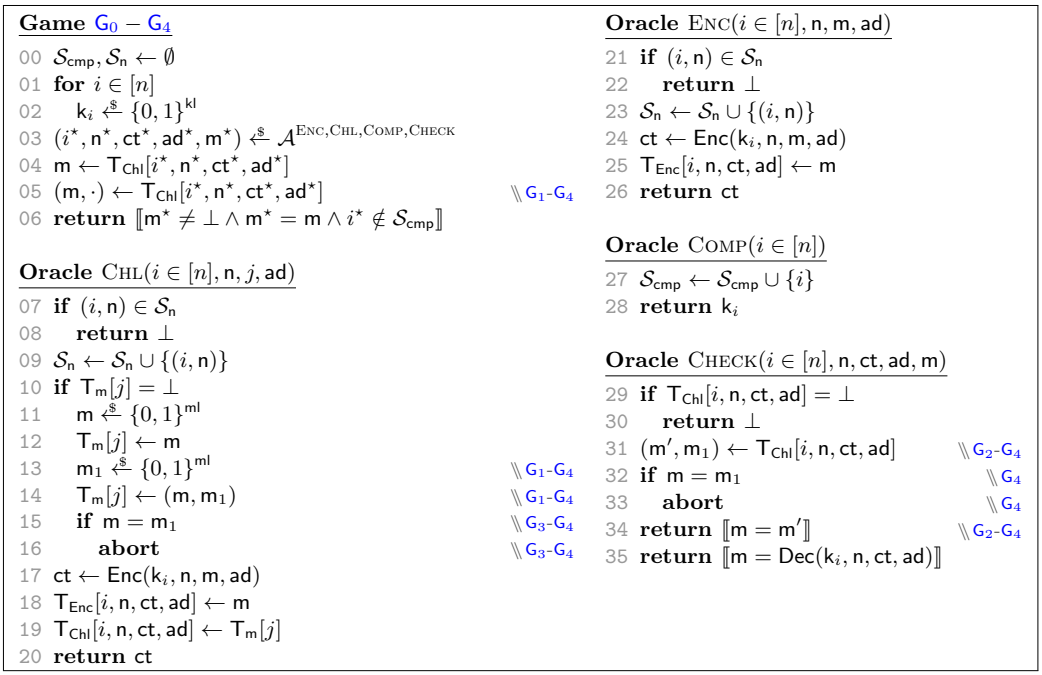

Fig. 20. Games $\mathsf{G}_0 - \mathsf{G}_4$ for the proof of Lemma 4.

Game G_3. This game is the same as the previous one except that it aborts if the two messages chosen in the challenge oracle are the same. The collision probability for one query is $1/2^{\mathsf{ml}}$ resulting in an overall bound of

$$\Pr[\mathsf{G}_2(\mathcal{A}) \Rightarrow 1] - \Pr[\mathsf{G}_3(\mathcal{A}) \Rightarrow 1] \leq \frac{Q_{\textsc{Chl}}}{2^{\mathsf{ml}}}.$$

Game G_4. This game is the same as the previous one except that it aborts if the check oracle is queried on the second message stored in $\mathsf{T}_{\mathsf{Chl}}$, i.e. m_1.

Claim. It holds that

$$\Pr[\mathsf{G}_2(\mathcal{A}) \Rightarrow 1] - \Pr[\mathsf{G}_4(\mathcal{A}) \Rightarrow 1] \leq \frac{Q_{\textsc{Check}}}{2^{\mathsf{ml}}}.$$

Proof (Claim). We need to upper bound the probability of an abort. Note that the second message, i.e. m_1, is never encrypted nor used anywhere in the game except from being stored in $\mathsf{T}_{\mathsf{Chl}}$. This means that adversary $\mathcal{A}$ cannot have any knowledge about it and the best chances are to guess the correct message which yields a probability of at most $1/2^{\mathsf{ml}}$. Considering $Q_{\textsc{Check}}$ queries to $\textsc{Check}$ yields the claimed bound. □

Reduction to IND-PAS.

Claim. There exists an IND-PAS adversary $\mathcal{B}$ such that

$$\Pr[\mathsf{G}_4(\mathcal{A}) \Rightarrow 1] \leq \mathsf{Adv}^{(n, Q_{\textsc{Comp}}, Q_{\textsc{Enc}}+Q_{\textsc{Chl}})\text{-IND-PAS}}_{\mathsf{AEAD}}(\mathcal{B}).$$

Proof (Claim). The reduction $\mathcal{B}$ is presented in Figure 21. Simulating G_4 for adversary $\mathcal{A}$ is straightforward because the reduction can use their own encryption oracle. Note that the "real" encryption oracle can be

Adversary $\mathcal{B}^{\text{ENC}_\mathcal{B},\text{COMP}_\mathcal{B}}$

00 $(i^\star, \mathsf{n}^\star, \mathsf{ct}^\star, \mathsf{ad}^\star, \mathsf{m}^\star) \xleftarrow{\$} \mathcal{A}^{\text{ENC},\text{DEC},\text{CHL},\text{COMP},\text{CHECK}}$
01 $(\mathsf{m}_0, \mathsf{m}_1) \leftarrow \mathsf{T}_{\mathsf{Chl}}[i^\star, \mathsf{n}^\star, \mathsf{ct}^\star, \mathsf{ad}^\star]$
02 **output** $(i^\star, [\![\mathsf{m}^\star = \mathsf{m}_1]\!])$

Oracle $\text{CHL}(i \in [n], \mathsf{n}, j, \mathsf{ad})$

03 **if** $\mathsf{T}_\mathsf{m}[j] = \bot$
04 $\quad(\mathsf{m}_0, \mathsf{m}_1) \xleftarrow{\$} \{0,1\}^{\mathsf{ml}} \times \{0,1\}^{\mathsf{ml}}$
05 $\quad\mathsf{T}_\mathsf{m}[j] \leftarrow (\mathsf{m}_0, \mathsf{m}_1)$
06 $\quad$**if** $\mathsf{m}_0 = \mathsf{m}_1$
07 $\quad\quad$**abort**
08 $\mathsf{ct} \leftarrow \text{ENC}_\mathcal{B}(i, \mathsf{n}, \mathsf{m}_0, \mathsf{m}_1, \mathsf{ad})$
09 $\mathsf{T}_{\mathsf{Chl}}[i, \mathsf{n}, \mathsf{ct}, \mathsf{ad}] \leftarrow \mathsf{T}_\mathsf{m}[j]$
10 **return** ct

Oracle $\text{ENC}(i \in [n], \mathsf{n}, \mathsf{m}, \mathsf{ad})$

11 $\mathsf{ct} \leftarrow \text{ENC}_\mathcal{B}(i, \mathsf{n}, \mathsf{m}, \mathsf{m}, \mathsf{ad})$
12 **return** ct

Oracle $\text{COMP}(i \in [n])$

13 **return** $\text{COMP}_\mathcal{B}(i)$

Oracle $\text{CHECK}(i \in [n], \mathsf{n}, \mathsf{ct}, \mathsf{ad}, \mathsf{m})$

14 **if** $\mathsf{T}_{\mathsf{Chl}}[i, \mathsf{n}, \mathsf{ct}, \mathsf{ad}] = \bot$
15 $\quad$**return** $\bot$
16 $(\mathsf{m}_0, \mathsf{m}_1) \leftarrow \mathsf{T}_{\mathsf{Chl}}[i, \mathsf{n}, \mathsf{ct}, \mathsf{ad}]$
17 **if** $\exists \mathsf{b} : \mathsf{m} = \mathsf{m}_\mathsf{b}$
18 $\quad$**output** (i, b)
19 **return** 0

Fig. 21. Adversary $\mathcal{B}$ against IND-PAS having access to oracles $\text{ENC}_\mathcal{B}$ and $\text{COMP}_\mathcal{B}$ simulating G_4 for adversary $\mathcal{A}$.

simulated by calling $\mathcal{B}$'s (left-or-right) encryption oracle on $\mathsf{m}_0 = \mathsf{m}_1$. Further, $\mathcal{B}$ terminates and outputs either in the check oracle in case $\mathcal{A}$ is querying the oracle on a message stored in $\mathsf{T}_{\mathsf{Chl}}$ for the corresponding query or after $\mathcal{A}$ returned their solution (Line 02).

Case 1: $\mathcal{B}$ outputs in CHECK: If $\mathcal{A}$'s query to the check oracle does not include the challenge message but the second message stored in $\mathsf{T}_{\mathsf{Chl}}$, $\mathcal{A}$'s game would abort due to the changes introduced in G_4. Hence, the only possibility for $\mathcal{A}$ to win is to query the check oracle on message m where m was actually encrypted during the challenge. This leads to $\mathcal{B}$ winning their indistinguishability game since they choose the bit corresponding to the message queried by $\mathcal{A}$.

Case 2: $\mathcal{B}$ outputs after $\mathcal{A}$ terminated: If $\mathcal{A}$ wins their OW-CPA game, the output message $\mathsf{m}^\star$ was the message used in the challenge oracle, i.e. $\mathsf{ct}^\star$ is an encryption of $\mathsf{m}^\star$. Iff $\mathsf{m}^\star = \mathsf{m}_1$, $\mathcal{B}$ is playing their IND-PAS game with $b_j = 1$. Hence, they also win their game.

In both cases it holds $\mathsf{Pr}[\mathcal{A} \text{ wins}] \leq \mathsf{Pr}[\mathcal{B} \text{ wins}]$ which concludes the proof by counting the oracle queries for $\mathcal{B}$. □

Collecting the probabilities completes the proof of Lemma 4. □

A.3 Security Notion for MAC and Signature Schemes

Definition 18 (SUF for MAC). *Unforgeability for a* MAC *is defined via the game in Figure 22. We define the advantage of an adversary $\mathcal{A}$ as*

$$\mathsf{Adv}^{Q\text{-SUF}}_{\mathsf{MAC}}(\mathcal{A}) := \mathsf{Pr}\left[Q\text{-}\mathsf{SUF}_{\mathsf{MAC}}(\mathcal{A}) \Rightarrow 1\right] ,$$

where $Q = (n, Q_{\text{COMP}}, Q_{\text{TAG}}, Q_{\text{VER}})$.

Definition 19 (UF for Sig). *Unforgeability for a signature scheme* Sig *is defined via the game in Figure 23. We define the advantage of an adversary $\mathcal{A}$ as*

$$\mathsf{Adv}^{Q_{\text{SIGN}}\text{-UF}}_{\mathsf{Sig}}(\mathcal{A}) := \mathsf{Pr}\left[Q_{\text{SIGN}}\text{-}\mathsf{UF}_{\mathsf{Sig}}(\mathcal{A}) \Rightarrow 1\right] .$$

Game Q-$\mathsf{SUF}_{\mathsf{MAC}}(\mathcal{A})$

00 $\mathcal{S}_{\mathsf{cmp}}, \mathcal{S}_{\mathsf{tag}} \leftarrow \emptyset$
01 **for** $i \in [n]$
02 $\quad \mathsf{k}_i \xleftarrow{\$} \{0,1\}^{\mathsf{kl}}$
03 $\mathsf{win} \leftarrow \mathsf{false}$
04 $\mathcal{A}^{\textsc{Tag},\textsc{Ver},\textsc{Comp}}$
05 **return** win

Oracle $\textsc{Tag}(i \in [n], \mathsf{m})$

06 $\mathsf{tag} \leftarrow \mathsf{Tag}(\mathsf{k}_i, \mathsf{m})$
07 $\mathcal{S}_{\mathsf{tag}} \leftarrow \mathcal{S}_{\mathsf{tag}} \cup \{(i, \mathsf{m}, \mathsf{tag})\}$
08 **return** tag

Oracle $\textsc{Ver}(i \in [n], \mathsf{m}, \mathsf{tag})$

09 $b \leftarrow \mathsf{Ver}(\mathsf{k}_i, \mathsf{m}, \mathsf{tag})$
10 **if** $b \wedge i \notin \mathcal{S}_{\mathsf{cmp}} \wedge (i, \mathsf{m}, \mathsf{tag}) \notin \mathcal{S}_{\mathsf{tag}}$
11 $\quad \mathsf{win} \leftarrow \mathsf{true}$
12 **return** b

Oracle $\textsc{Comp}(i \in [n])$

13 $\mathcal{S}_{\mathsf{cmp}} \leftarrow \mathcal{S}_{\mathsf{cmp}} \cup \{i\}$
14 **return** k_i

Fig. 22. Game defining SUF security for a MAC scheme $\mathsf{MAC} := (\mathsf{Tag}, \mathsf{Ver})$ with $Q = (n, Q_{\textsc{Comp}}, Q_{\textsc{Tag}}, Q_{\textsc{Ver}})$ and adversary $\mathcal{A}$ making at most Q_{O} queries to oracle O.

Game $Q_{\textsc{Sign}}$-$\mathsf{UF}_{\mathsf{Sig}}(\mathcal{A})$

00 $\mathcal{S}_{\mathsf{sig}} \leftarrow \emptyset$
01 $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{Gen}$
02 $(\mathsf{m}^\star, \sigma^\star) \xleftarrow{\$} \mathcal{A}^{\textsc{Sign}}(\mathsf{pk})$
03 **return** $[\![\mathsf{Ver}(\mathsf{pk}, \mathsf{m}^\star, \sigma^\star) \wedge \mathsf{m}^\star \notin Q]\!]$

Oracle $\textsc{Sign}(i \in [n], \mathsf{m})$

04 $\sigma \xleftarrow{\$} \mathsf{Sign}(\mathsf{sk}, \mathsf{m})$
05 $\mathcal{S}_{\mathsf{sig}} \leftarrow \mathcal{S}_{\mathsf{sig}} \cup \{\mathsf{m}\}$
06 **return** σ

Fig. 23. Game defining UF security for a signature scheme $\mathsf{Sig} := (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ with adversary $\mathcal{A}$ making at most $Q_{\textsc{Sign}}$ queries to oracle Sign.

B Omitted Proofs from Section 3

B.1 Proof of Theorem 1

Theorem 1 (INS-OW of SHash). *For any distribution $\mathcal{D}_{\mathsf{C}}$ and adversary $\mathcal{A}$ against* INS-OW *of* $\mathsf{SHash} := \mathsf{SHash}[\mathcal{D}_{\mathsf{S}}, \mathsf{kl}]$*, there exists a* PW-Guess *adversary $\mathcal{B}$ against $\mathcal{D}_{\mathsf{C}}$ with $t_{\mathcal{A}} \approx t_{\mathcal{B}}$ such that*

$$\mathsf{Adv}^{Q\text{-}\mathsf{INS}\text{-}\mathsf{OW}}_{\mathsf{SHash},\mathcal{D}_{\mathsf{C}}}(\mathcal{A}) \leq \mathsf{Adv}^{(Q_{\textsc{Reg}},Q_{\mathrm{RO}},Q_{\textsc{Comp}})\text{-}\mathsf{PW}\text{-}\mathsf{Guess}}_{\mathcal{D}_{\mathsf{C}}}(\mathcal{B}) + \frac{Q_{\textsc{Check}} + 1}{2^{\mathsf{kl}} - Q_{\textsc{Check}}}$$

with $Q = (Q_{\textsc{Reg}}, Q_{\textsc{Comp}}, Q_{\textsc{Init}}, Q_{\textsc{Fin}}, Q_{\textsc{Check}}, Q_{\mathrm{RO}})$.

Proof. We use the sequence of games described in Figure 24.

Game G_0. This is the INS-OW game for SHash and $\mathcal{D}_{\mathsf{C}}$.

Game G_1. This is the same game as the previous one except that it introduces several conceptual changes. We introduce a new random oracle, RO', which behaves exactly the same as RO with the only difference that it maintains an additional table T' which indicates that the query was made via the new oracle. The adversary still only has access to the original random oracle but the new random oracle is called from all other oracle queries in the game. Hence, we mark if a query was made directly by the attacker or internally. Since the oracle still maintains the same list to answer queries, the change is only conceptual and it holds that

$$\Pr[\mathsf{G}_0(\mathcal{A}) \Rightarrow 1] = \Pr[\mathsf{G}_1(\mathcal{A}) \Rightarrow 1].$$

Game G_2. This is the same game as the previous one except that it aborts if random oracle RO is queried on a registered and uncompromised auxiliary input aux and the input x is the correct client secret for that aux. The game further aborts in the registration oracle if there already exists a previous random oracle query on the correct client secret and the registered aux.

Game $\mathsf{G}_0 - \mathsf{G}_2$

00 $\mathcal{S}_{\mathsf{cmp}} \leftarrow \emptyset$
01 $n_r \leftarrow 0$
02 $(x_{\mathsf{C},1}, \ldots, x_{\mathsf{C},Q_{\mathrm{REG}}}) \xleftarrow{\$} \mathcal{D}_{\mathsf{C}}$
03 **for** $i \in [Q_{\mathrm{REG}}]$
04 $\quad x_{\mathsf{S},i} \xleftarrow{\$} \mathcal{D}_{\mathsf{S}}$
05 $(y^\star, \mathsf{aux}^\star) \xleftarrow{\$} \mathcal{A}^{\mathrm{REG},\mathrm{INIT},\mathrm{FIN},\mathrm{COMP},\mathrm{CHECK},\mathrm{RO}}$
06 **if** $\mathsf{T}_{\mathsf{aux}}[\mathsf{aux}] = \bot$ **return** 0
07 $i^\star \leftarrow \mathsf{T}_{\mathsf{aux}}[\mathsf{aux}]$
08 **if** $i^\star \in \mathcal{S}_{\mathsf{cmp}}$ **return** 0
09 **return** $[\![y^\star = \mathrm{RO}(x_{\mathsf{C},i^\star}, \mathsf{aux}^\star, x_{\mathsf{S},i^\star})]\!]$ \\ G_0
10 **return** $[\![y^\star = \mathrm{RO}'(x_{\mathsf{C},i^\star}, \mathsf{aux}^\star, x_{\mathsf{S},i^\star})]\!]$ \\ G_1-G_2

Oracle $\mathrm{REG}(\mathsf{aux})$

11 **require** $\mathsf{T}_{\mathsf{aux}}[\mathsf{aux}] = \bot$
12 n_r++
13 $\mathsf{T}_{\mathsf{aux}}[\mathsf{aux}] \leftarrow n_r$
14 **if** $(x_{\mathsf{C},n_r}, \mathsf{aux}, \cdot) \in \mathsf{T}$ \\ G_2
15 $\quad$ **abort** \\ G_2
16 **return** x_{S,n_r}

Oracle $\mathrm{INIT}(\mathsf{aux}, j)$

17 **require** $\mathsf{T}_{\mathsf{aux}}[\mathsf{aux}] \neq \bot$
18 $\mathsf{auth}_1 \leftarrow \mathsf{aux}$
19 $\mathsf{T}[i,j] \leftarrow (\varepsilon, \mathsf{aux})$
20 **return** auth_1

Oracle $\mathrm{CHECK}(y, \mathsf{aux})$

21 **if** $\mathsf{T}_{\mathsf{aux}}[\mathsf{aux}] = \bot$ **return** 0
22 $i \leftarrow \mathsf{T}_{\mathsf{aux}}[\mathsf{aux}]$
23 **return** $[\![y = \mathrm{RO}(x_{\mathsf{C},i}, \mathsf{aux}, x_{\mathsf{S},i})]\!]$ \\ G_0
24 **return** $[\![y = \mathrm{RO}'(x_{\mathsf{C},i}, \mathsf{aux}, x_{\mathsf{S},i})]\!]$ \\ G_1-G_2

Oracle $\mathrm{FIN}(\mathsf{auth}_2, i \in [n_r], j)$

25 **if** $\mathsf{T}[i,j] = \bot$
26 $\quad$ **return** $\bot$
27 $(\varepsilon, \mathsf{aux}) \leftarrow \mathsf{T}[i,j]$
28 $\mathsf{T}[i,j] \leftarrow \bot$
29 $y \leftarrow \mathrm{RO}(x_{\mathsf{C},i}, \mathsf{aux}, \mathsf{auth}_2)$ \\ G_0
30 $y \leftarrow \mathrm{RO}'(x_{\mathsf{C},i}, \mathsf{aux}, \mathsf{auth}_2)$ \\ G_1-G_2
31 **if** $y = \mathrm{RO}(x_{\mathsf{C},i}, \mathsf{aux}, x_{\mathsf{S},i})$ **return** $\top$ \\ G_0
32 **if** $y = \mathrm{RO}'(x_{\mathsf{C},i}, \mathsf{aux}, x_{\mathsf{S},i})$ **return** $\top$ \\ G_1-G_2
33 **return** y

Oracle $\mathrm{COMP}(i \in [n_r])$

34 $\mathcal{S}_{\mathsf{cmp}} \leftarrow \mathcal{S}_{\mathsf{cmp}} \cup \{i\}$
35 **return** $x_{\mathsf{C},i}$

Oracle $\mathrm{RO}(x_{\mathsf{C}}, \mathsf{aux}, x_{\mathsf{S}})$

36 **if** $\mathsf{T}_{\mathsf{aux}}[\mathsf{aux}] \neq \bot \wedge \mathsf{T}_{\mathsf{aux}}[\mathsf{aux}] \notin \mathcal{S}_{\mathsf{cmp}} \wedge x_{\mathsf{C}} = x_{\mathsf{C},i}$ \\ G_2
37 $\quad$ **abort** \\ G_2
38 **if** $(x_{\mathsf{C}}, \mathsf{aux}, x_{\mathsf{S}}) \notin \mathsf{T}$
39 $\quad \mathsf{T}[x_{\mathsf{C}}, \mathsf{aux}, x_{\mathsf{S}}] \xleftarrow{\$} \{0,1\}^{\mathsf{kl}}$
40 **return** $\mathsf{T}[x_{\mathsf{C}}, \mathsf{aux}, x_{\mathsf{S}}]$

Oracle $\mathrm{RO}'(x_{\mathsf{C}}, \mathsf{aux}, x_{\mathsf{S}})$ \\ G_1-G_2

41 **if** $(x_{\mathsf{C}}, \mathsf{aux}, x_{\mathsf{S}}) \notin \mathsf{T}$
42 $\quad \mathsf{T}[x_{\mathsf{C}}, \mathsf{aux}, x_{\mathsf{S}}] \xleftarrow{\$} \{0,1\}^{\mathsf{kl}}$
43 $\quad \mathsf{T}'[x_{\mathsf{C}}, \mathsf{aux}, x_{\mathsf{S}}] \xleftarrow{\$} 1$
44 **return** $\mathsf{T}[x_{\mathsf{C}}, \mathsf{aux}, x_{\mathsf{S}}]$

Fig. 24. Games $\mathsf{G}_0 - \mathsf{G}_2$ for the proof of Theorem 1.

Claim. There exists an adversary $\mathcal{B}$ against PW-Guess such that

$$\Pr[\mathsf{G}_1(\mathcal{A}) \Rightarrow 1] - \Pr[\mathsf{G}_2(\mathcal{A}) \Rightarrow 1] \leq \mathsf{Adv}^{(Q_{\mathrm{REG}}, Q_{\mathrm{RO}}, Q_{\mathrm{COMP}})\text{-PW-Guess}}_{\mathcal{D}_{\mathsf{C}}}(\mathcal{B}).$$

Proof (Claim). The reduction is formally depicted in Figure 25. The oracles are simulated by querying the internal random oracles on some unique tag "chl" which acts as a placeholder and is considered to be a value outside of the valid random oracle input space. This allows the reduction $\mathcal{B}$ to consistently answer oracle queries. In case of a compromise, the random oracles must be patched on the respective values. This is possible because the client secret is known to the reduction upon compromise and done in Line 32. Note that this simulation is sound because queries on the correct client secret *before* compromise are captured by the newly introduced abort condition. To capture this condition, reduction $\mathcal{B}$ uses their TEST oracle and therefore knows whenever the abort condition is met. Moreover, in such a case $\mathcal{B}$ correctly guessed an uncompromised password and thus wins their game. Note that $\mathcal{B}$ issues at most Q_{RO} queries to TEST because queries from REG and RO are mutually exclusive. □

Final game.

Claim. It holds that

$$\Pr[\mathsf{G}_2(\mathcal{A}) \Rightarrow 1] \leq \frac{Q_{\mathrm{CHECK}} + 1}{2^{\mathsf{kl}} - Q_{\mathrm{CHECK}}}.$$

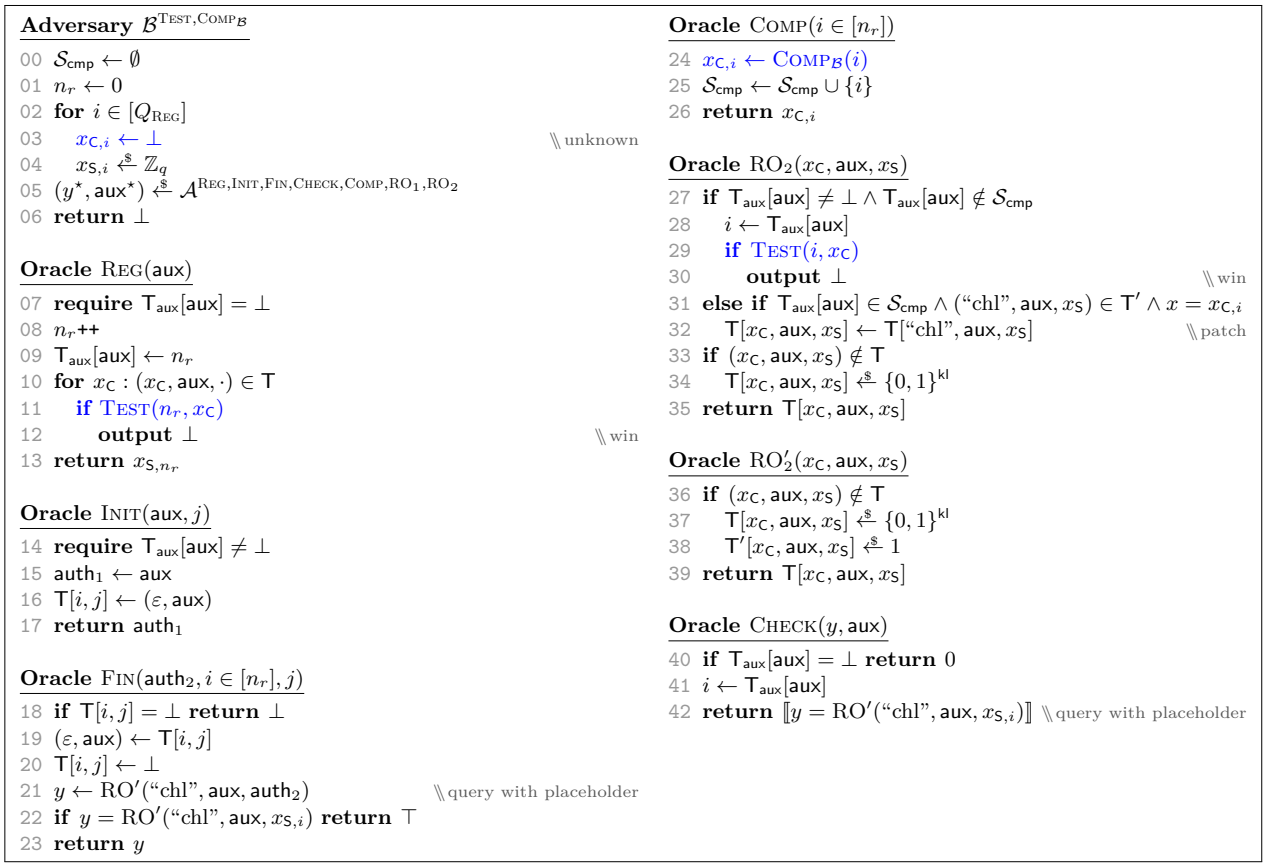

Fig. 25. Adversary $\mathcal{B}$ against PW-Guess having access to oracles TEST and $\mathsf{Comp}_\mathcal{B}$ simulating $\mathsf{G}_1/\mathsf{G}_2$

Proof (Claim). The random oracle was never queried directly on the correct input due to the abort introduced in the previous game. Further, the correct output $y^\star$ was also never output by FIN because FIN outputs $\top$ instead of the actual value in such a case. Hence, adversary $\mathcal{A}$'s view is independent of the output of RO_2. The only possibility for $\mathcal{A}$ to win is to guess the correct output in the space 2^{kl}. This possible once per query to CHECK and for the final output resulting in $Q_{\text{CHECK}} + 1$ many guesses. Since one guess rules out one possible value, we need to sum up $\sum_{i=0}^{Q_{\text{CHECK}}} 1/(2^{\mathsf{kl}} - i)$ which can be upper-bounded to obtain the claim. □

Collecting the probabilities completes the proof of Theorem 1. □

B.2 Proof of Theorem 2

Theorem 2 (INS-OW of 2HDH). *For any distribution $\mathcal{D}_\mathsf{C}$ and adversary $\mathcal{A}$ against* INS-OW *of* $\mathsf{2HDH} := \mathsf{2HDH}[\mathbb{G}, q, g, \mathsf{kl}]$ *there exists a* PW-Guess *adversary $\mathcal{B}$ against $\mathcal{D}_\mathsf{C}$ with $t_\mathcal{A} \approx t_\mathcal{B}$ such that*

$$\mathsf{Adv}^{Q\text{-INS-OW}}_{\mathsf{2HDH},\mathcal{D}_\mathsf{C}}(\mathcal{A}) \leq \mathsf{Adv}^{(Q_{\text{REG}},Q_{\text{RO}},Q_{\text{COMP}})\text{-PW-Guess}}_{\mathcal{D}_\mathsf{C}}(\mathcal{B}) + \frac{Q_{\text{CHECK}} + 1}{2^{\mathsf{kl}} - Q_{\text{CHECK}}}$$

with $Q = (Q_{\text{REG}}, Q_{\text{COMP}}, Q_{\text{INIT}}, Q_{\text{FIN}}, Q_{\text{CHECK}}, Q_{\text{RO}})$.

Proof. We use the sequence of games described in Figure 26.

Game G_0. This is the INS-OW game for 2HDH and distribution $\mathcal{D}_{\mathsf{C}}$.

Game $\mathsf{G}_0 - \mathsf{G}_2$

00 $\mathcal{S}_{\mathsf{cmp}} \leftarrow \emptyset$
01 $n_r \leftarrow 0$
02 $(x_{\mathsf{C},1}, \ldots, x_{\mathsf{C},Q_{\mathrm{REG}}}) \xleftarrow{\$} \mathcal{D}_{\mathsf{C}}$
03 **for** $i \in [Q_{\mathrm{REG}}]$
04 $\quad x_{\mathsf{S},i} \xleftarrow{\$} \mathbb{Z}_q$
05 $(y^\star, \mathsf{aux}^\star) \xleftarrow{\$} \mathcal{A}^{\mathrm{REG},\mathrm{INIT},\mathrm{FIN},\mathrm{CHL},\mathrm{RO}_1,\mathrm{RO}_2}$
06 **if** $\mathsf{T}_{\mathsf{aux}}[\mathsf{aux}^\star] = \bot$ **return** 0
07 $i^\star \leftarrow \mathsf{T}_{\mathsf{aux}}[\mathsf{aux}^\star]$
08 **if** $i^\star \in \mathcal{S}_{\mathsf{cmp}}$ **return** 0
09 **return** $[\![y^\star = \mathrm{RO}_2(x_{\mathsf{C},i^\star}, \mathsf{aux}^\star, \mathrm{RO}_1(x_{\mathsf{C},i^\star}, \mathsf{aux}^\star)^{x_{\mathsf{S},i^\star}})]\!]$ \\ G_0
10 **return** $[\![y^\star = \mathrm{RO}_2(x_{\mathsf{C},i^\star}, \mathsf{aux}^\star, \mathrm{RO}'_1(x_{\mathsf{C},i^\star}, \mathsf{aux}^\star)^{x_{\mathsf{S},i^\star}})]\!]$ \\ G_1-G_2

Oracle $\mathrm{REG}(\mathsf{aux})$

11 **require** $\mathsf{T}_{\mathsf{aux}}[\mathsf{aux}] = \bot$
12 n_r++
13 $\mathsf{T}_{\mathsf{aux}}[\mathsf{aux}] \leftarrow n_r$
14 **if** $(x_{\mathsf{C},n_r}, \mathsf{aux}) \in \mathsf{T}_1 \vee (x_{\mathsf{C},n_r}, \mathsf{aux}, \cdot) \in \mathsf{T}_2$ \\ G_2
15 $\quad$ **abort** \\ G_2
16 **return** x_{S,n_r}

Oracle $\mathrm{INIT}(\mathsf{aux}, j)$

17 **require** $\mathsf{T}_{\mathsf{aux}}[\mathsf{aux}] \neq \bot$
18 $i \leftarrow \mathsf{T}_{\mathsf{aux}}[\mathsf{aux}]$
19 $r \xleftarrow{\$} \mathbb{Z}_q$
20 $\alpha \leftarrow \mathrm{RO}_1(x_{\mathsf{C},i}, \mathsf{aux})^r$ \\ G_0
21 $\alpha \leftarrow \mathrm{RO}'_1(x_{\mathsf{C},i}, \mathsf{aux})^r$ \\ G_1-G_2
22 $\mathsf{T}[i,j] \leftarrow (r, \mathsf{aux})$
23 **return** α

Oracle $\mathrm{FIN}(\beta, i \in [n_r], j)$

24 **if** $\mathsf{T}[i,j] = \bot$ **return** $\bot$
25 $(r, \mathsf{aux}) \leftarrow \mathsf{T}[i,j]$
26 $\mathsf{T}[i,j] \leftarrow \bot$
27 **if** $\beta \notin \mathbb{G}$ **return** $\bot$
28 $\gamma \leftarrow \beta^{1/r}$
29 $y \leftarrow \mathrm{RO}_2(x_{\mathsf{C},i}, \mathsf{aux}, \gamma)$ \\ G_0
30 $y \leftarrow \mathrm{RO}'_2(x_{\mathsf{C},i}, \mathsf{aux}, \gamma)$ \\ G_1-G_2
31 **if** $y = \mathrm{RO}_2(x_{\mathsf{C},i}, \mathsf{aux}, \mathrm{RO}_1(x_{\mathsf{C},i}, \mathsf{aux})^{x_{\mathsf{S},i}})$ **return** $\top$ \\ G_0
32 **if** $y = \mathrm{RO}'_2(x_{\mathsf{C},i}, \mathsf{aux}, \mathrm{RO}'_1(x_{\mathsf{C},i}, \mathsf{aux})^{x_{\mathsf{S},i}})$ **return** $\top$ \\ G_1-G_2
33 **return** y

Oracle $\mathrm{COMP}(i \in [n_r])$

34 $\mathcal{S}_{\mathsf{cmp}} \leftarrow \mathcal{S}_{\mathsf{cmp}} \cup \{i\}$
35 **return** $x_{\mathsf{C},i}$

Oracle $\mathrm{RO}_1(x, \mathsf{aux})$

36 **if** $\mathsf{T}_{\mathsf{aux}}[\mathsf{aux}] \neq \bot \wedge \mathsf{T}_{\mathsf{aux}}[\mathsf{aux}] \notin \mathcal{S}_{\mathsf{cmp}} \wedge x = x_{\mathsf{C},i}$ \\ G_2
37 $\quad$ **abort** \\ G_2
38 **if** $(x, \mathsf{aux}) \notin \mathsf{T}_1$
39 $\quad \mathsf{T}_1[x, \mathsf{aux}] \xleftarrow{\$} \mathbb{G}$
40 **return** $\mathsf{T}_1[x, \mathsf{aux}]$

Oracle $\mathrm{RO}'_1(x, \mathsf{aux})$ \\ G_1-G_2

41 **if** $(x, \mathsf{aux}) \notin \mathsf{T}_1$
42 $\quad \mathsf{T}_1[x, \mathsf{aux}] \xleftarrow{\$} \mathbb{G}$
43 $\quad \mathsf{T}'_1[x, \mathsf{aux}] \leftarrow 1$
44 **return** $\mathsf{T}_1[x, \mathsf{aux}]$

Oracle $\mathrm{RO}_2(x, \mathsf{aux}, \gamma)$

45 **if** $\mathsf{T}_{\mathsf{aux}}[\mathsf{aux}] \neq \bot \wedge \mathsf{T}_{\mathsf{aux}}[\mathsf{aux}] \notin \mathcal{S}_{\mathsf{cmp}} \wedge x = x_{\mathsf{C},i}$ \\ G_2
46 $\quad$ **abort** \\ G_2
47 **if** $(x, \mathsf{aux}, \gamma) \notin \mathsf{T}_2$
48 $\quad \mathsf{T}_2[x, \mathsf{aux}, \gamma] \xleftarrow{\$} \{0,1\}^{\mathsf{kl}}$
49 **return** $\mathsf{T}_2[x, \mathsf{aux}, \gamma]$

Oracle $\mathrm{RO}'_2(x, \mathsf{aux}, \gamma)$ \\ G_1-G_2

50 **if** $(x, \mathsf{aux}, \gamma) \notin \mathsf{T}_2$
51 $\quad \mathsf{T}_2[x, \mathsf{aux}, \gamma] \xleftarrow{\$} \{0,1\}^{\mathsf{kl}}$
52 $\quad \mathsf{T}'_2[x, \mathsf{aux}, \gamma] \xleftarrow{\$} 1$
53 **return** $\mathsf{T}_2[x, \mathsf{aux}, \gamma]$

Oracle $\mathrm{CHECK}(y, \mathsf{aux})$

54 **if** $\mathsf{T}_{\mathsf{aux}}[\mathsf{aux}] = \bot$ **return** 0
55 $i \leftarrow \mathsf{T}_{\mathsf{aux}}[\mathsf{aux}]$
56 **return** $[\![y = \mathrm{RO}_2(x_{\mathsf{C},i}, \mathsf{aux}, \mathrm{RO}_1(x_{\mathsf{C},i}, \mathsf{aux})^{x_{\mathsf{S},i}})]\!]$ \\ G_0
57 **return** $[\![y = \mathrm{RO}'_2(x_{\mathsf{C},i}, \mathsf{aux}, \mathrm{RO}'_1(x_{\mathsf{C},i}, \mathsf{aux})^{x_{\mathsf{S},i}})]\!]$ \\ G_1-G_2

Fig. 26. Games $\mathsf{G}_0 - \mathsf{G}_2$ for the proof of Theorem 2.

Game G_1. This is the same game as the previous one except that it introduces several conceptual changes. We introduce two new random oracles, RO'_1 and RO'_2 which behave exactly the same as RO_1 and RO_2 with the only difference that they maintain additional tables T'_1 and T'_2 which indicate that the query was made via the new oracle. The adversary still has access via the original random oracles but the new random oracles are called from all other oracle queries in the game. Hence, we mark if a query was made directly by the attacker or internally. Since the oracles still maintain the same list to answer queries, the change is only conceptual and it holds that

$$\Pr[\mathsf{G}_0(\mathcal{A}) \Rightarrow 1] = \Pr[\mathsf{G}_1(\mathcal{A}) \Rightarrow 1].$$

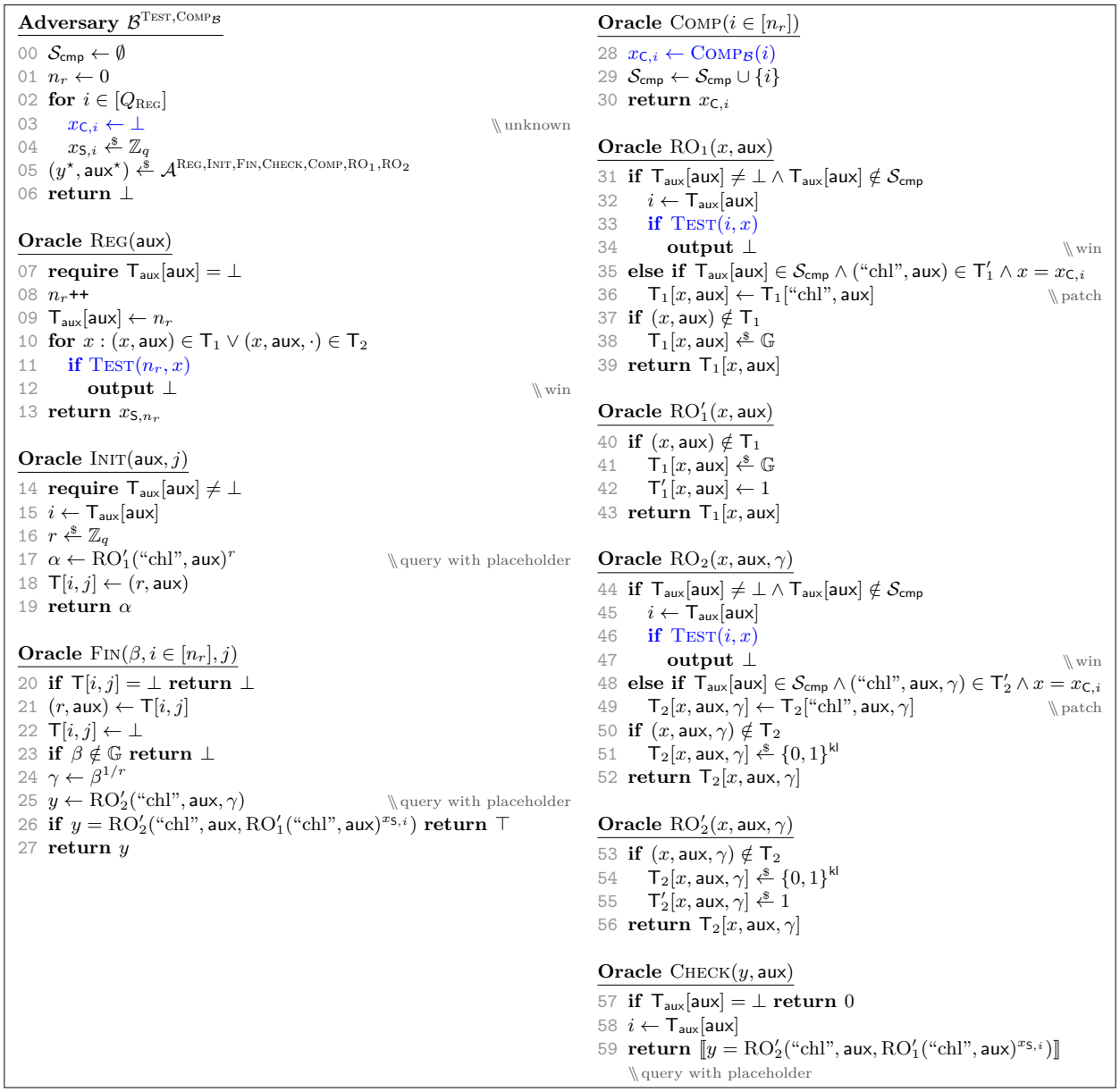

Fig. 27. Adversary $\mathcal{B}$ against PW-Guess having access to oracles $\text{TEST}, \text{COMP}_\mathcal{B}$ simulating $\mathsf{G}_1/\mathsf{G}_2$ for adversary $\mathcal{A}$.

Game G_2. This is the same game as the previous one except that it aborts if random oracles RO_1 or RO_2 are queried on a registered and uncompromised auxiliary input aux and the input x is the correct client secret for that aux. The game further aborts in the registration oracle if there already exists a previous random oracle query on the correct client secret and the registered aux.

Claim. It holds that

$$\Pr[\mathsf{G}_1(\mathcal{A}) \Rightarrow 1] - \Pr[\mathsf{G}_2(\mathcal{A}) \Rightarrow 1] \leq \mathsf{Adv}^{(Q_{\text{REG}}, Q_{\text{RO}}, Q_{\text{COMP}})\text{-PW-Guess}}_{\mathcal{D}_\mathsf{C}}(\mathcal{B}).$$

Proof (Claim). The reduction is formally depicted in Figure 27. The oracles are simulated by querying the internal random oracles on some unique tag "chl" which acts as a placeholder and is considered to be a value outside of the valid random oracle input space. This allows the reduction $\mathcal{B}$ to consistently answer oracle queries. In case of a compromise, the random oracles must be patched on the respective values. This is possible because the client secret is known to the reduction upon compromise and done in Line 36 and Line 49. Note that this simulation is sound because queries on the correct client secret *before* compromise are captured by the newly introduced abort condition. To capture this condition, reduction $\mathcal{B}$ uses their TEST oracle and therefore knows whenever the abort condition is met. Moreover, in such a case $\mathcal{B}$ correctly guessed an uncompromised password and thus wins their game. Note that $\mathcal{B}$ issues at most Q_{RO} queries to TEST because queries from REG and $\mathrm{RO}_1/\mathrm{RO}_2$ are mutually exclusive. □

Final game.

Claim. It holds that

$$\mathsf{Pr}[\mathsf{G}_2(\mathcal{A}) \Rightarrow 1] \leq \frac{Q_{\textsc{Check}} + 1}{2^{\mathsf{kl}} - Q_{\textsc{Check}}}.$$

Proof (Claim). The random oracle was never queried directly on the correct input due to the abort introduced in the previous game. Further, the correct output $y^\star$ was also never output by FIN because FIN outputs $\top$ instead of the actual value in such a case. Hence, adversary $\mathcal{A}$'s view is independent of the output of RO_2 with the correct client secret on input. The only possibility for $\mathcal{A}$ to win is to guess the correct output in the space 2^{kl}. This is possible once per query to CHECK and for the final output resulting in $Q_{\textsc{Check}} + 1$ many guesses. Since one guess rules out one possible value, we need to sum up $\sum_{i=0}^{Q_{\textsc{Check}}} 1/(2^{\mathsf{kl}} - i)$ which can be upper-bounded to obtain the claim. □

Collecting the probabilities completes the proof of Theorem 2. □

B.3 Proof of Theorem 3

Theorem 3 (OUT-OW of Sig2HDH). *For any distribution $\mathcal{D}_{\mathsf{C}}$ and adversary $\mathcal{A}$ against* OUT-OW *of* $\mathsf{Sig2HDH} := \mathsf{Sig2HDH}[\mathsf{Sig}, \mathbb{G}, q, g, \mathsf{kl}]$*, there exist a* UF *adversary $\mathcal{B}$ against* Sig*, a* CT-OMCDH *adversary $\mathcal{C}$ against $(\mathbb{G}, q, g)$, and a* PW-Guess *adversary $\mathcal{D}$ against $\mathcal{D}_{\mathsf{C}}$ with $t_{\mathcal{A}} \approx t_{\mathcal{B}} \approx t_{\mathcal{C}} \approx t_{\mathcal{D}}$ such that*

$$\begin{aligned}\mathsf{Adv}^{Q\text{-}\mathsf{OUT}\text{-}\mathsf{OW}}_{\mathsf{Sig2HDH},\mathcal{D}_{\mathsf{C}}}(\mathcal{A}) \leq\ & \mathsf{Adv}^{Q_{\textsc{Resp}}\text{-}\mathsf{UF}}_{\mathsf{Sig}}(\mathcal{B}) + \mathsf{Adv}^{(Q_{\textsc{Reg}}, Q_{\mathrm{RO}}, Q_{\textsc{Comp}}, Q_{\textsc{Resp}}, Q_{\mathrm{RO}} + Q_{\textsc{Check}})\text{-}\mathsf{CT}\text{-}\mathsf{OMCDH}}_{\mathbb{G},q,g}(\mathcal{C}) \\ & + \mathsf{Adv}^{(Q_{\textsc{Reg}}, Q_{\textsc{Resp}}, Q_{\textsc{Comp}})\text{-}\mathsf{PW}\text{-}\mathsf{Guess}}_{\mathcal{D}_{\mathsf{C}}}(\mathcal{D}) + \frac{Q_{\textsc{Check}} + 1}{2^{\mathsf{kl}} - Q_{\textsc{Check}}} + \frac{Q_{\mathrm{RO}}^2}{2^{\mathsf{kl}}}\end{aligned}$$

with $Q = (Q_{\textsc{Reg}}, Q_{\textsc{Comp}}, Q_{\textsc{Init}}, Q_{\textsc{Resp}}, Q_{\textsc{Fin}}, Q_{\textsc{Check}}, Q_{\mathrm{RO}})$.

Proof. Sequence of games described in Figure 28.

Game G_0. This is the OUT-OW game for Sig2HDH and distribution $\mathcal{D}_{\mathsf{C}}$.

Game G_1. Here, we introduce an additional book-keeping list $\mathcal{S}_{\mathsf{resp}}$ to keep track of matching transcripts, i.e., the oracle FIN is queried on β which was forwarded from a RESP query with the corresponding first message α. In this case, the FIN oracle outputs $\top$ immediately.

By perfect correctness, these are only conceptual changes and we have

$$\mathsf{Pr}[\mathsf{G}_1(\mathcal{A}) \Rightarrow 1] = \mathsf{Pr}[\mathsf{G}_0(\mathcal{A}) \Rightarrow 1].$$

Game G_2. In this game, we abort in the FIN oracle if it has not returned $\perp$ (due to invalid β or σ) or $\top$ (due to the previous change).

Game $\mathsf{G}_0 - \mathsf{G}_5$

00 $\mathcal{S}_{\mathsf{cmp}}, \mathcal{S}_{\mathsf{resp}} \leftarrow \emptyset$
01 $n_r \leftarrow 0$
02 $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{Gen}$
03 $(x_{\mathsf{C},1}, \ldots, x_{\mathsf{C},Q_{\mathrm{REG}}}) \xleftarrow{\$} \mathcal{D}_{\mathsf{C}}$
04 **for** $i \in [Q_{\mathrm{REG}}]$
05 $\quad x_{\mathsf{S},i} \xleftarrow{\$} \mathbb{Z}_q$
06 $\quad \mathcal{S}_{\mathsf{guess},i} \leftarrow \emptyset$ \\ G_3-G_5
07 $(y^\star, \mathsf{aux}^\star) \xleftarrow{\$} \mathcal{A}^{\textsc{Init},\textsc{Resp},\textsc{Fin},\textsc{Comp},\textsc{Check},\mathrm{RO}_1,\mathrm{RO}_2}(\mathsf{pk})$
08 **if** $\mathsf{T}_{\mathsf{aux}}[\mathsf{aux}^\star] = \bot$ **return** 0
09 $i^\star \leftarrow \mathsf{T}_{\mathsf{aux}}[\mathsf{aux}^\star]$
10 **if** $i^\star \in \mathcal{S}_{\mathsf{cmp}}$ **return** 0
11 **return** $[\![y^\star = \mathrm{RO}_2(\mathsf{pk}, x_{\mathsf{C},i^\star}, \mathsf{aux}^\star, \mathrm{RO}_1(\mathsf{pk}, x_{\mathsf{C},i^\star}, \mathsf{aux}^\star)^{x_{\mathsf{S},i^\star}})]\!]$

Oracle $\textsc{Reg}(\mathsf{aux})$

12 **require** $\mathsf{T}_{\mathsf{aux}}[\mathsf{aux}] = \bot$
13 n_r++
14 $\mathsf{T}_{\mathsf{aux}}[\mathsf{aux}] \leftarrow n_r$
15 $\mathsf{ctr}_{n_r} \leftarrow 0$ \\ G_3-G_5
16 **if** $(x_{\mathsf{C},i}, \mathsf{aux}, \cdot) \in \mathsf{T}_2$ \\ G_5
17 $\quad$ **abort** \\ G_5

Oracle $\textsc{Init}(\mathsf{aux}, j)$ \\ G_0-G_2

18 **require** $\mathsf{T}_{\mathsf{aux}}[\mathsf{aux}] \neq \bot$
19 $i \leftarrow \mathsf{T}_{\mathsf{aux}}[\mathsf{aux}]$
20 $r \xleftarrow{\$} \mathbb{Z}_q$
21 $h \leftarrow \mathrm{RO}_1(x_{\mathsf{C},i}, \mathsf{aux})$
22 $\alpha \leftarrow h^r$
23 $\mathsf{T}[i,j] \leftarrow (r, \mathsf{aux}, \alpha)$
24 **return** α

Oracle $\textsc{Init}(\mathsf{aux}, j)$ \\ G_3-G_5

25 **require** $\mathsf{T}_{\mathsf{aux}}[\mathsf{aux}] \neq \bot$
26 $i \leftarrow \mathsf{T}_{\mathsf{aux}}[\mathsf{aux}]$
27 $\hat{r} \xleftarrow{\$} \mathbb{Z}_q$
28 $\alpha \leftarrow g^{\hat{r}}$
29 $\mathsf{T}[i,j] \leftarrow (\hat{r}, \mathsf{aux}, \alpha)$
30 **return** α

Oracle $\textsc{Fin}((\beta, \sigma), i \in [n], j)$

31 **if** $\mathsf{T}[i,j] = \bot$ **return** $\bot$
32 $(r, \mathsf{aux}, \alpha) \leftarrow \mathsf{T}[i,j]$ \\ G_0-G_2
33 $(\hat{r}, \mathsf{aux}, \alpha) \leftarrow \mathsf{T}[i,j]$ \\ G_3-G_5
34 $\mathsf{T}[i,j] \leftarrow \bot$
35 **if** $\beta \notin \mathbb{G}$ **return** $\bot$
36 **if** $\mathsf{Ver}(\mathsf{pk}, (\alpha, \beta), \sigma) \neq 1$ **return** $\bot$
37 **if** $(\alpha, i, \beta) \in \mathcal{S}_{\mathsf{resp}}$ **return** $\top$ \\ G_1-G_5
38 **abort** \\ G_2-G_5
39 $y \leftarrow \mathrm{RO}_2(x_{\mathsf{C},i}, \mathsf{aux}, \beta^{1/r})$ \\ G_0-G_2
40 $y' \leftarrow \mathrm{RO}_2(x_{\mathsf{C},i}, \mathsf{aux}, \mathrm{RO}_1(x_{\mathsf{C},i}, \mathsf{aux})^{x_{\mathsf{S},i}})$ \\ G_0-G_2
41 **if** $y = y'$: **return** $\top$ \\ G_0-G_2
42 **return** y \\ G_0-G_2

Oracle $\textsc{Resp}(\alpha, i \in [n_r])$

43 **if** $\alpha \notin \mathbb{G}$ **return** $\bot$
44 ctr_i++ \\ G_3-G_5
45 $\beta \leftarrow \alpha^{x_{\mathsf{S},i}}$
46 $\sigma \xleftarrow{\$} \mathsf{Sign}(\mathsf{sk}, (\alpha, \beta))$
47 $\mathcal{S}_{\mathsf{resp}} \leftarrow \mathcal{S}_{\mathsf{resp}} \cup \{(\alpha, i, \beta)\}$ \\ G_1-G_5
48 **return** (β, σ)

Oracle $\mathrm{RO}_1(x, \mathsf{aux})$

49 **if** $(x, \mathsf{aux}) \notin \mathsf{T}_1$
50 $\quad \mathsf{T}_1[x, \mathsf{aux}] \xleftarrow{\$} \mathbb{G}$
51 **return** $\mathsf{T}_1[x, \mathsf{aux}]$

Oracle $\mathrm{RO}_2(x, \mathsf{aux}, \gamma)$

52 **if** $(x, \mathsf{aux}) \in \mathsf{T}_1 \wedge \mathsf{T}_{\mathsf{aux}}[\mathsf{aux}] \neq \bot$ \\ G_3-G_5
53 $\quad h \leftarrow \mathsf{T}_1[x, \mathsf{aux}]$ \\ G_3-G_5
54 $\quad i \leftarrow \mathsf{T}_{\mathsf{aux}}[\mathsf{aux}]$ \\ G_3-G_5
55 $\quad$ **if** $\gamma = h^{x_{\mathsf{S},i}} \wedge i \notin \mathcal{S}_{\mathsf{cmp}}$ \\ G_3-G_5
56 $\quad\quad \mathcal{S}_{\mathsf{guess},i} \leftarrow \mathcal{S}_{\mathsf{guess},i} \cup \{(x, \mathsf{aux}, h, \gamma)\}$ \\ G_3-G_5
57 $\quad\quad$ **if** $|\mathcal{S}_{\mathsf{guess},i}| > \mathsf{ctr}_i$: **abort** \\ G_4-G_5
58 $\quad\quad$ **if** $x = x_{\mathsf{C},i}$: **abort** \\ G_5
59 **if** $(x, \mathsf{aux}, \gamma) \notin \mathsf{T}_2$
60 $\quad y \xleftarrow{\$} \{0,1\}^{\mathsf{kl}}$
61 $\quad \mathsf{T}_2[x, \mathsf{aux}, \gamma] \leftarrow y$
62 **return** $\mathsf{T}_2[x, \mathsf{aux}, \gamma]$

Oracle $\textsc{Comp}(i \in [n_r])$

63 $\mathcal{S}_{\mathsf{cmp}} \leftarrow \mathcal{S}_{\mathsf{cmp}} \cup \{i\}$
64 **return** $(x_{\mathsf{C},i}, x_{\mathsf{S},i})$

Oracle $\textsc{Check}(y, \mathsf{aux})$

65 **if** $\mathsf{T}_{\mathsf{aux}}[\mathsf{aux}] = \bot$ **return** 0
66 $i \leftarrow \mathsf{T}_{\mathsf{aux}}[\mathsf{aux}]$
67 **return** $[\![y = \mathrm{RO}_2(x_{\mathsf{C},i}, \mathsf{aux}, \mathrm{RO}_1(x_{\mathsf{C},i}, \mathsf{aux})^{x_{\mathsf{S},i}})]\!]$

Fig. 28. Games $\mathsf{G}_0 - \mathsf{G}_5$ for the proof of Theorem 3.

Claim. There exists an adversary $\mathcal{B}$ against UF such that

$$\Pr[\mathsf{G}_1(\mathcal{A}) \Rightarrow 1] - \Pr[\mathsf{G}_2(\mathcal{A}) \Rightarrow 1] \leq \mathsf{Adv}_{\mathsf{Sig}}^{Q_{\mathrm{RESP}}\text{-}\mathsf{UF}}(\mathcal{B}).$$

Proof (Claim). The reduction $\mathcal{B}$ is presented in Figure 29. $\mathcal{B}$ gets as input the public verification key and has access to a signing oracle. It simulates all oracles except for $\textsc{Resp}$ and $\textsc{Fin}$ as in the game description. To generate the server's signature, it queries $\textsc{Sign}$ on (α, β). Whenever $\mathcal{A}$ queries $\textsc{Fin}$ on (β, σ) such that β

was not an output of REG (with α as input) and σ is valid, $\mathcal{B}$ can win the unforgeability game. Note that the signature is on both α and β, hence, $\mathcal{B}$ has not queries SIGN on the same message (α, β) yet. Note also that we do not need strong unforgeability since the output y would be the same if σ would be a valid signature different from the one output by RESP. Therefore, the previous game hop already implicitly takes care of such an event. □

Game G_3. The next game is to clean up the code and introduce additional variables for the next game hops. First, we remove the now unused code after the abort in FIN. Second, we do not use RO_1 in INIT anymore, but simply sample an exponent $\hat{r}$ and compute $\alpha = g^{\hat{r}}$. Note that this does not change the distribution and also that r is not used anymore in FIN. We then introduce a counter ctr_i in REG that counts queries to RESP for each user i. Finally, we introduce a set $\mathcal{S}_{\mathsf{guess},i}$ during initialization. The set is filled during queries to random oracle RO_2, i.e., via a query by the adversary, whenever (x, aux) was previously also queried to RO_1.[11] We will use this set to count guessing attempts for the target user. Hence, it is only updated as long as i is not compromised.

All these changes are only conceptual, therefore

$$\mathsf{Pr}[\mathsf{G}_3(\mathcal{A}) \Rightarrow 1] = \mathsf{Pr}[\mathsf{G}_2(\mathcal{A}) \Rightarrow 1].$$

Game G_4. Now we introduce the next abort event. Namely, this game aborts as soon as the set $\mathcal{S}_{\mathsf{guess},i}$ is larger than the counter ctr_i for any i.

Claim. There exists an adversary $\mathcal{C}$ against CT-OMCDH such that

$$\mathsf{Pr}\left[\mathsf{G}_3(\mathcal{A}) \Rightarrow 1\right] - \mathsf{Pr}\left[\mathsf{G}_4(\mathcal{A}) \Rightarrow 1\right] \leq \mathsf{Adv}^{(Q_{\mathrm{REG}}, Q_{\mathrm{RO}}, Q_{\mathrm{COMP}}, Q_{\mathrm{RESP}}, Q_{\mathrm{RO}}+Q_{\mathrm{CHECK}})\text{-CT-OMCDH}}_{\mathbb{G},p,g}(\mathcal{C}).$$

Note in particular that we only call the EXP oracle Q_{RESP} times. This later translates into the number of password guesses.

Proof (Claim). The reduction $\mathcal{C}$ is presented in Figure 30. It gets as input $(V_1, \ldots, V_{Q_{\mathrm{REG}}}, U_1, \ldots, U_{Q_{\mathrm{RO}}})$, where we embed the exponent of V_i as the server secret for registered user i and U_j is embedded as the j-th output of RO_1. $\mathcal{C}$ simulates the RESP oracle by calling its EXP oracle. The CHECK oracle can be simulated via $\mathcal{C}$'s $\mathrm{CHECK}_{\mathcal{C}}$ oracle in case RO_2 was already queried on the checked value and the output y is correct. In case, the random oracle was not queried yet, a randomly chosen value is used and the random oracle is patched upon the correct random oracle query (Line 47). To simulate the random oracle RO_2 and to decide whether an element should be added to set $\mathcal{S}_{\mathsf{guess},i}$, $\mathcal{C}$ queries its $\mathrm{CHECK}_{\mathcal{C}}$ oracle. If the oracle returns true, then the adversary has found a valid CDH solution for values V_i and U_j. In case the list of valid solutions is larger than the queries to RESP for an index i, denoted by ctr_i, the adversary outputs $(i, \mathcal{S}_{\mathsf{guess},i})$ and wins their game. Hence, whenever the newly introduced abort condition triggers, $\mathcal{C}$ wins. □

Game G_5. In this game, we abort when the ith client secret is contained in the set $\mathcal{S}_{\mathsf{guess},i}$ (Line 58) and if the correct client secret was input to RO_2 before registration (Line 17).

Claim. There exists an adversary $\mathcal{D}$ against PW-Guess such that

$$\mathsf{Pr}\left[\mathsf{G}_4(\mathcal{A}) \Rightarrow 1\right] - \mathsf{Pr}\left[\mathsf{G}_5(\mathcal{A}) \Rightarrow 1\right] \leq \mathsf{Adv}^{(Q_{\mathrm{REG}}, Q_{\mathrm{RESP}}, Q_{\mathrm{COMP}})\text{-PW-Guess}}_{\mathcal{D}_{\mathsf{C}}}(\mathcal{D}).$$

Proof (Claim). The reduction is formally constructed in Figure 31. Due to the changes in the previous games, the oracles are independent of the client secrets except for CHECK and RO_2. The abort condition can be checked by $\mathcal{D}$'s check oracle $\mathrm{CHECK}_{\mathcal{D}}$ and if it triggers, $\mathcal{D}$ wins their game. To consistently answer random oracle queries with respect to the CHECK oracle, the reduction has to potentially patch the random

[11] We could already use the server secret here to only identify relevant queries, but this would require to iterate over all indices $i \in [n]$, which unnecessarily increases the runtime (of the game, but also of the reduction later).

Adversary $\mathcal{B}^{\text{SIGN}}(\mathsf{pk})$

00 $\mathcal{S}_{\mathsf{cmp}}, \mathcal{S}_{\mathsf{resp}} \leftarrow \emptyset$
01 $n_r \leftarrow 0$
02 $(x_{\mathsf{C},1}, \ldots, x_{\mathsf{C},Q_{\text{REG}}}) \xleftarrow{\$} \mathcal{D}_{\mathsf{C}}$
03 **for** $i \in [Q_{\text{REG}}]$
04 $\quad x_{\mathsf{S},i} \xleftarrow{\$} \mathbb{Z}_q$
05 $(y^\star, \mathsf{aux}^\star) \xleftarrow{\$} \mathcal{A}^{\text{INIT},\text{RESP},\text{FIN},\text{COMP},\text{CHECK},\text{RO}_1,\text{RO}_2}(\mathsf{pk})$
06 **return** $\bot$

Oracle $\text{REG}(\mathsf{aux})$

07 **require** $\mathsf{T}_{\mathsf{aux}}[\mathsf{aux}] = \bot$ \\enforce unique aux
08 n_r++
09 $\mathsf{T}_{\mathsf{aux}}[\mathsf{aux}] \leftarrow n_r$

Oracle $\text{INIT}(\mathsf{aux}, j)$

10 **require** $\mathsf{T}_{\mathsf{aux}}[\mathsf{aux}] \neq \bot$
11 $i \leftarrow \mathsf{T}_{\mathsf{aux}}[\mathsf{aux}]$
12 $r \xleftarrow{\$} \mathbb{Z}_q$
13 $h \leftarrow \text{RO}_1(x_{\mathsf{C},i}, \mathsf{aux})$
14 $\alpha \leftarrow h^r$
15 $\mathsf{T}[i,j] \leftarrow (r, \mathsf{aux}, \alpha)$
16 **return** α

Oracle $\text{FIN}((\beta,\sigma), i \in [n], j)$

17 **if** $\mathsf{T}[i,j] = \bot$ **return** $\bot$
18 $(r, \mathsf{aux}, \alpha) \leftarrow \mathsf{T}[i,j]$
19 $\mathsf{T}[i,j] \leftarrow \bot$
20 **if** $\beta \notin \mathbb{G}$ **return** $\bot$
21 **if** $\mathsf{Ver}(\mathsf{pk}, (\alpha,\beta), \sigma) \neq 1$ **return** $\bot$
22 **if** $(\alpha, i, \beta) \in \mathcal{S}_{\mathsf{resp}}$ **return** $\top$
23 **output** $((\alpha,\beta),\sigma)$

Oracle $\text{RESP}(\alpha, i \in [n_r])$

24 **if** $\alpha \notin \mathbb{G}$ **return** $\bot$
25 $\beta \leftarrow \alpha^{x_{\mathsf{S},i}}$
26 $\sigma \leftarrow \text{SIGN}((\alpha,\beta))$
27 $\mathcal{S}_{\mathsf{resp}} \leftarrow \mathcal{S}_{\mathsf{resp}} \cup \{(\alpha, i, \beta)\}$
28 **return** (β,σ)

Oracle $\text{RO}_1(x, \mathsf{aux})$

29 **if** $(x, \mathsf{aux}) \notin \mathsf{T}_1$
30 $\quad \mathsf{T}_1[x, \mathsf{aux}] \xleftarrow{\$} \mathbb{G}$
31 **return** $\mathsf{T}_1[x, \mathsf{aux}]$

Oracle $\text{RO}_2(x, \mathsf{aux}, \gamma)$

32 **if** $(x, \mathsf{aux}, \gamma) \notin \mathsf{T}_2$
33 $\quad y \xleftarrow{\$} \{0,1\}^{\mathsf{kl}}$
34 $\quad \mathsf{T}_2[x, \mathsf{aux}, \gamma] \leftarrow y$
35 **return** $\mathsf{T}_2[x, \mathsf{aux}, \gamma]$

Oracle $\text{COMP}(i \in [n_r])$

36 $\mathcal{S}_{\mathsf{cmp}} \leftarrow \mathcal{S}_{\mathsf{cmp}} \cup \{i\}$
37 **return** $(x_{\mathsf{C},i}, x_{\mathsf{S},i})$

Oracle $\text{CHECK}(y, \mathsf{aux})$

38 **if** $\mathsf{T}_{\mathsf{aux}}[\mathsf{aux}] = \bot$ **return** 0
39 $i \leftarrow \mathsf{T}_{\mathsf{aux}}[\mathsf{aux}]$
40 **return** $[\![y = \text{RO}_2(x_{\mathsf{C},i}, \mathsf{aux}, \text{RO}_1(x_{\mathsf{C},i}, \mathsf{aux})^{x_{\mathsf{S},i}})]\!]$

Fig. 29. Adversary $\mathcal{B}$ against UF having access to oracle SIGN simulating $\mathsf{G}_1/\mathsf{G}_2$ for adversary $\mathcal{A}$.

oracle RO_2 when queried on a compromised user (Line 50). Due to the previous condition, the set to be tested is of size at most ctr_i for each user i which can again be upper bounded by Q_{RESP}. Note that $\mathcal{D}$ issues at most Q_{RO} queries to TEST because queries from REG and RO_1/RO_2 are mutually exclusive. □

Final game.

Claim. It holds that

$$\Pr[\mathsf{G}_5(\mathcal{A}) \Rightarrow 1] \leq \frac{Q_{\text{CHECK}} + 1}{2^{\mathsf{kl}} - Q_{\text{CHECK}}}.$$

Proof (Claim). If the ith client secret is not in the set $\mathcal{S}_{\mathsf{guess},i}$, then the random oracle has never been queried on the right input. Further, the correct output $y^\star$ was also never output by FIN because FIN outputs $\top$ instead of the actual value in such a case. Hence, adversary $\mathcal{A}$'s view is independent of the output of RO_2 with the correct client secret on input. The only possibility for $\mathcal{A}$ to win the game is to guess the correct output in the space 2^{kl}. This is possible once per query to CHECK and for the final output resulting in $Q_{\text{CHECK}} + 1$ many guesses. Since one guess rules out one possible value, we need to sum up $\sum_{i=0}^{Q_{\text{CHECK}}} 1/(2^{\mathsf{kl}} - i)$ which can be upper-bounded to obtain the claim. □

Collecting the probabilities completes the proof of Theorem 3. □

Fig. 30. Adversary $\mathcal{C}$ against CT-OMCDH having access to oracles Exp, $\text{Check}_\mathcal{B}$, and $\text{Comp}_\mathcal{B}$ simulating $\mathsf{G}_3/\mathsf{G}_4$ for adversary $\mathcal{A}$.

C Theorems and Proofs from Section 5

C.1 Confidentiality

We restate Theorem 5 using concrete bounds and provide the full proof.

Theorem 10 (Confidentiality). *For any password distribution $\mathcal{PW}$ and adversary $\mathcal{A}$ against the* Conf *security of* $\text{CS} := \text{CS}[\text{Auth}, \text{AEAD}, \text{MAC}]$ *there exist an* INS-OW *adversary $\mathcal{B}$ against* Auth, INT-CTXT *adversaries $\mathcal{C}_1, \mathcal{C}_2$ against* AEAD, OW-CPA *adversaries $\mathcal{D}_1, \mathcal{D}_2$ against* AEAD, *and an* IND-CCA *adversary $\mathcal{E}$ against* AEAD *with* $t_\mathcal{A} \approx t_\mathcal{B} \approx t_{\mathcal{C}_1} \approx t_{\mathcal{D}_1} \approx t_{\mathcal{C}_2} - Q_{\text{Comp}} \cdot Q_{\text{Auth}_1} \approx t_{\mathcal{D}_2} - Q_{\text{Comp}} \cdot Q_{\text{Auth}_1} \approx t_\mathcal{E}$ *such that*

$$\begin{aligned}\text{Adv}^{Q_\mathcal{A}\text{-Conf}}_{\text{CS},\mathcal{PW}}(\mathcal{A}) \leq 2 \cdot \Big(&\text{Adv}^{Q_\mathcal{B}\text{-INS-OW}}_{\text{Auth},\mathcal{PW}}(\mathcal{B}) + \text{Adv}^{Q_{\mathcal{C}_1}\text{-INT-CTXT}}_{\text{AEAD}}(\mathcal{C}_1) + \text{Adv}^{Q_{\mathcal{D}_1}\text{-OW-CPA}}_{\text{AEAD,kl}}(\mathcal{D}_1) \\ &+ \text{Adv}^{Q_{\mathcal{C}_2}\text{-INT-CTXT}}_{\text{AEAD}}(\mathcal{C}_2) + \text{Adv}^{Q_{\mathcal{D}_2}\text{-OW-CPA}}_{\text{AEAD,kl}}(\mathcal{D}_2)\Big) + \text{Adv}^{Q_\mathcal{E}\text{-IND-CCA}}_{\text{AEAD}}(\mathcal{E}) \\ &+ 2Q_{\text{Auth}_1} \cdot \delta_{\text{Auth}} + \frac{(Q_{\text{Put}_1} + Q_{\text{Acpt}_1})^2 + (Q_{\text{Upd}_1} + 1)^2}{2^{\text{nl}}}\end{aligned}$$

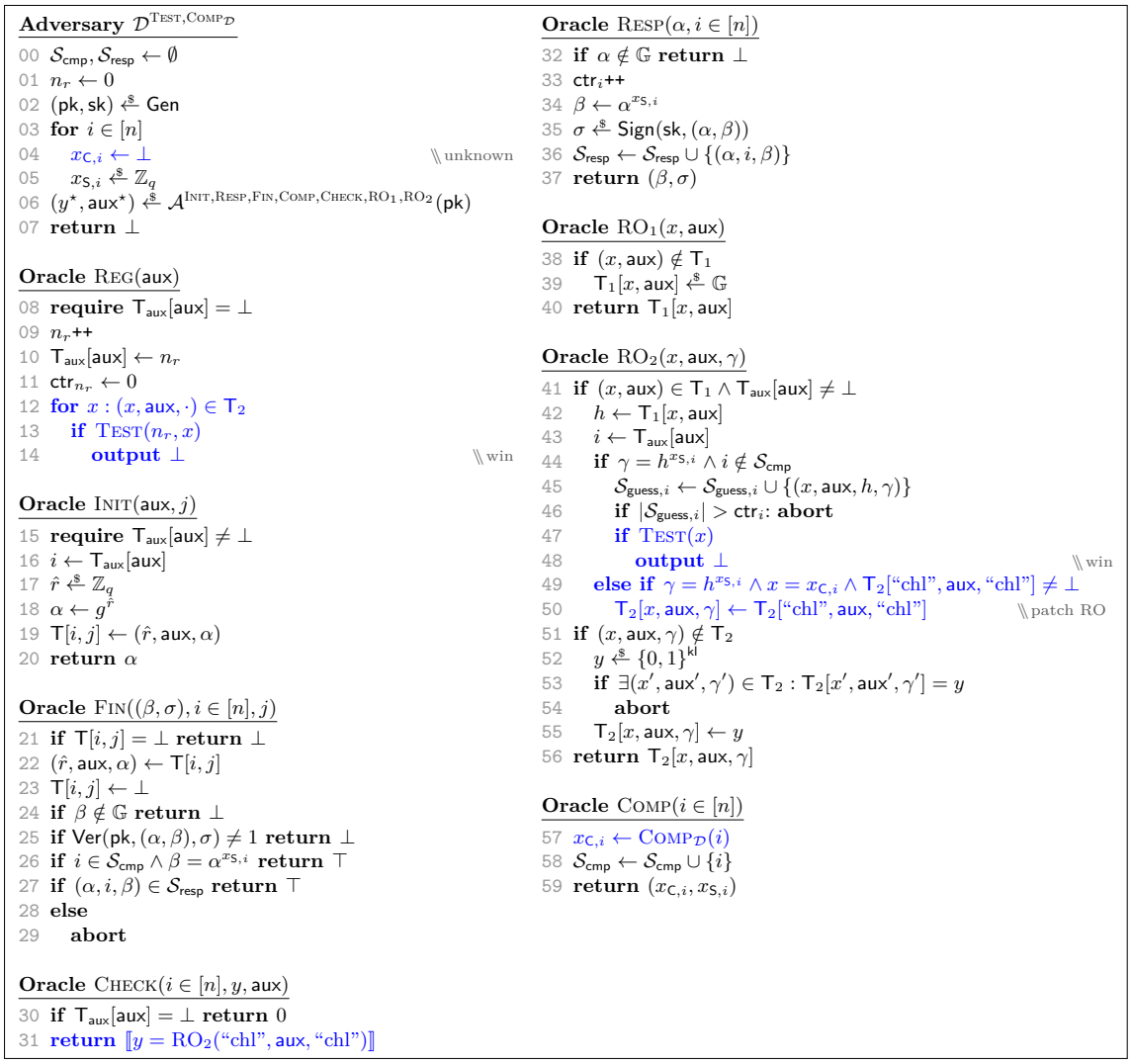

Fig. 31. Adversary $\mathcal{D}$ against PW-Guess having access to oracle TEST and $\mathsf{Comp}_\mathcal{D}$ simulating $\mathsf{G}_4/\mathsf{G}_5$.

with

$$
\begin{aligned}
Q_\mathcal{A} &= (Q_{\text{STEP}}, Q_{\text{REG}_1}, Q_{\text{AUTH}_1}, Q_{\text{PUT}_1}, Q_{\text{UPD}_1}, Q_{\text{GET}_1}, Q_{\text{SHARE}_1}, Q_{\text{ACPT}_1}, Q_{\text{CHALL}}, Q_{\text{COMP}}, Q_{\text{RO}}),\\
Q_\mathcal{B} &= (Q_{\text{REG}_1}, Q_{\text{COMP}}, Q_{\text{AUTH}_1}, Q_{\text{AUTH}_1}, Q_{\text{REG}_1} + Q_{\text{RO}}, Q_{\text{RO}}),\\
Q_{\mathcal{C}_1} &= (Q_{\text{REG}_1}, Q_{\text{COMP}}, Q_{\text{REG}_1}, Q_{\text{AUTH}_1}),\\
Q_{\mathcal{D}_1} &= (Q_{\text{REG}_1}, Q_{\text{COMP}}, 0, Q_{\text{REG}_1}, Q_{\text{REG}_1} + Q_{\text{RO}}),\\
Q_{\mathcal{C}_2} &= (Q_{\text{REG}_1}, Q_{\text{COMP}}, Q_{\text{PUT}_1} + Q_{\text{ACPT}_1} + Q_{\text{CHL}}, Q_{\text{UPD}_1} + Q_{\text{GET}_1} + Q_{\text{SHARE}_1} + Q_{\text{CHL}}),\\
Q_{\mathcal{D}_2} &= (Q_{\text{REG}_1}, Q_{\text{COMP}}, Q_{\text{ACPT}_1}, Q_{\text{PUT}_1} + Q_{\text{ACPT}_1} + Q_{\text{CHL}}, Q_{\text{PUT}_1} + Q_{\text{UPD}_1} + Q_{\text{GET}_1} + Q_{\text{SHARE}_1} + Q_{\text{CHL}} + Q_{\text{RO}}),\\
Q_\mathcal{E} &= (Q_{\text{PUT}_1} + Q_{\text{CHL}}, Q_{\text{PUT}_1} + Q_{\text{CHL}}, Q_{\text{PUT}_1} + Q_{\text{UPD}_1}, Q_{\text{GET}_1}, Q_{\text{CHL}}).
\end{aligned}
$$

Proof. We prove the theorem by a sequence of games and depict only those oracles in which a change occurs, starting with Figure 32.

Game G_0. This is the confidentiality game in which we plug in construction CS. To ease the depiction of later changes, we distinguish two random oracles. One can be called by the adversary, which is denoted by

RO as usual, and one is an internal random oracle, denoted by H, which is only called from other oracles. The change is conceptual since both oracles represent the same function and maintain the same table for lazy sampling. Additionally, we introduce a table $\mathsf{T}_{\mathsf{H\text{-}int}}$ which indicates if a value was queried to the internal RO; this will be used in later changes. It holds that

$$\Pr[\mathsf{G}_0(\mathcal{A}) \Rightarrow 1] = \mathsf{Adv}^{Q_{\mathcal{A}}\text{-}\mathsf{Conf}}_{\mathsf{CS},\mathcal{PW}}(\mathcal{A}),$$

where $Q_{\mathcal{A}}$ is the same as in the theorem statement.

Game G_1. This is the same game as the previous one except that in the second step of the authentication protocol it overrides the value rw with the output of a Fin call if the received value auth_2 is correct, i.e. is output by the response algorithm of Auth with the correct inputs. The changes are depicted in Figure 32.

Claim. It holds that

$$\Pr[\mathsf{G}_0(\mathcal{A}) \Rightarrow 1] - \Pr[\mathsf{G}_1(\mathcal{A}) \Rightarrow 1] \leq Q_{\text{AUTH}_1} \cdot \delta_{\mathsf{Auth}}.$$

Proof (Claim). Let us consider one query to the second step of the authentication protocol. The abort condition only occurs if auth_2 is in the support of $\mathsf{Auth.Resp}(\mathsf{auth}_1, \mathsf{k})$. Since auth_1 was computed during the first step of the authentication protocol by the game itself, it is the output of the initialization procedure of Auth. The games can only be distinguished if rw computed by Fin and Full (as done in REG_1) are different. This is exactly the correctness error of Auth. Since the second step of the authentication protocol can be called only if the first step was executed, the change appears at most Q_{AUTH_1} times resulting in the claimed bound. □

Game G_2. This is the same game as the previous one except that it aborts if there exists a random oracle query on a registered and uncompromised user such that the same input was previously queried to the internal RO H and the label is either "ek" or "mac". The game also aborts in the registration oracle if there already exists a random oracle query on the same rw and aid. The changes are depicted in Figure 32.

Claim. There exists an adversary $\mathcal{B}$ against $\mathsf{INS\text{-}OW}$ such that

$$\Pr[\mathsf{G}_1(\mathcal{A}) \Rightarrow 1] - \Pr[\mathsf{G}_2(\mathcal{A}) \Rightarrow 1] \leq \mathsf{Adv}^{Q\text{-}\mathsf{INS\text{-}OW}}_{\mathsf{Auth},\mathcal{PW}}(\mathcal{B})$$

with $Q = (Q_{\text{REG}_1}, Q_{\text{COMP}}, Q_{\text{AUTH}_1}, Q_{\text{AUTH}_1}, Q_{\text{REG}_1} + Q_{\text{RO}}, Q_{\text{RO}})$.

Proof (Claim). We formally construct adversary $\mathcal{B}$ in Figure 33 simulating the game for $\mathcal{A}$. First, we show that $\mathcal{B}$'s simulation is sound. The registration oracle is simulated with the reduction's registration oracle and by choosing a unique challenge tag "chl" (which is not in the valid input space of the random oracle) in case a user is not compromised (yet) and the internal random oracle is queried on that tag instead of the actual rw (Line 15). This is sound because later queries to the random oracle are answered consistently:

1. In the last step of the authentication procedure, i.e. in oracle STEP in Figure 33, the finalize algorithm is simulated by using $\mathcal{B}$'s oracle FIN. Note that FIN outputs $\top$ in case it is queried on the correct value, i.e. the value of the full execution of Auth. In this way, $\mathcal{B}$ recognizes when to use the previously programmed RO and answers the query with the same value. This works because of the changes in G_1.
2. Whenever adversary $\mathcal{A}$ queries the random oracle RO on an input that corresponds to a previous registration query, $\mathcal{B}$ can just abort the game winning their $\mathsf{INS\text{-}OW}$ game. $\mathcal{B}$ can recognize that by using their oracle CHECK. The reduction must also simulate an abort in the registration oracle which can also be done by using the check oracle analogously (Line 13). The other oracles, i.e. AUTH_1 and STEP, can be simulated by using $\mathcal{B}$'s oracles INIT and FIN without the need of any client secrets.

The next challenge is to handle compromises in a sound way. More precisely, after a user is compromised, random oracle queries must still be consistent. This is done by patching the random oracle on values that were previously unknown and marked with "chl". Since the reduction is querying their own compromise

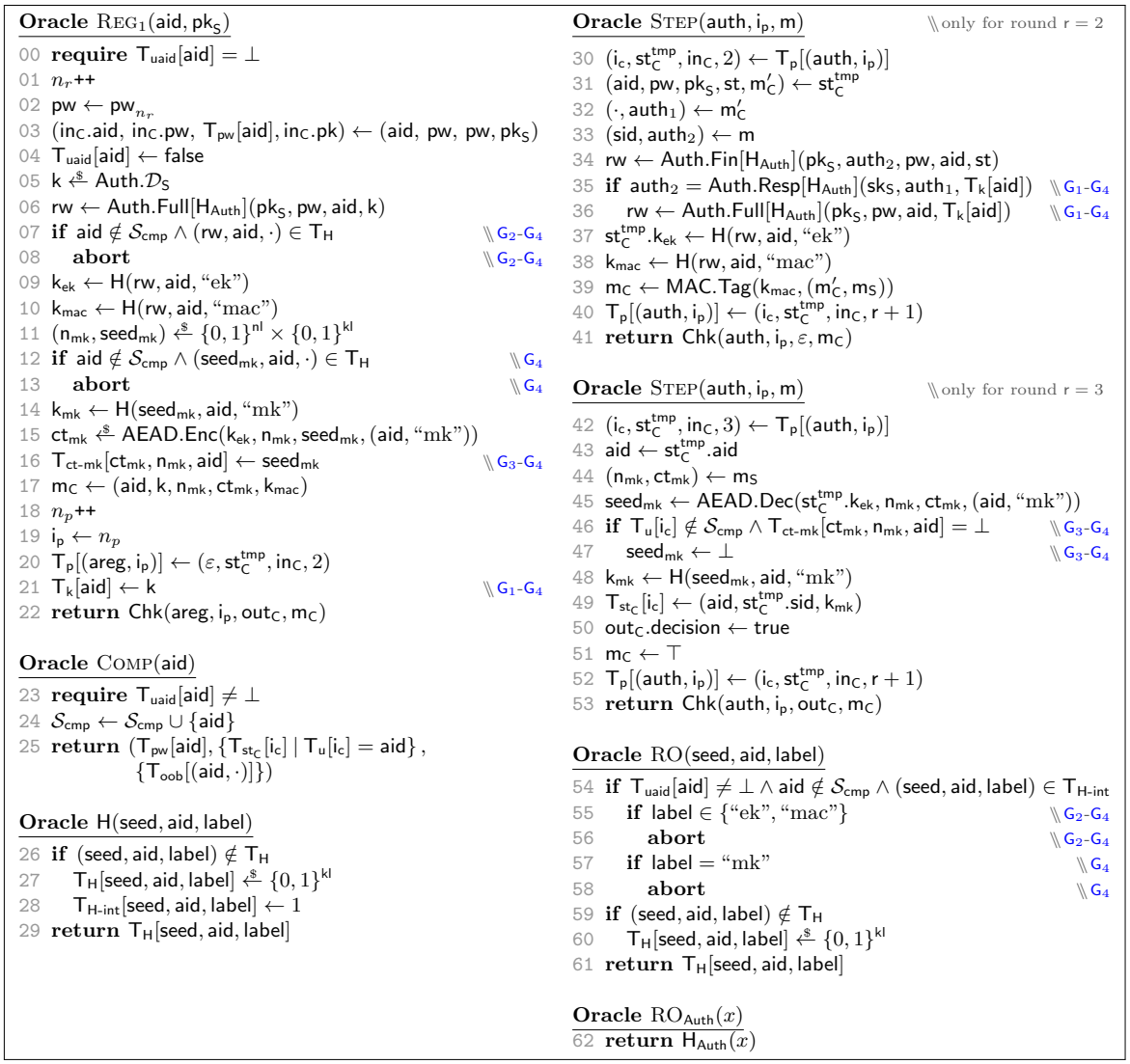

Fig. 32. Games $\mathsf{G}_0 - \mathsf{G}_4$ for the proof of Theorem 10. H is an internal random oracle and cannot be queried by $\mathcal{A}$.

oracle $\textsc{Comp}_{\mathcal{B}}$, they learn the client secret for that user and thus can patch the oracle later (Line 66). Note that this works because the random oracle was not queried on that input before because the game would have been aborted in such a case.

Finally, note that as soon as a newly introduced abort occurs, $\mathcal{B}$ wins their INS-OW game. □

Game G_3. This is the same game as the previous one except that the decryption of every ciphertext in the third round of the authentication procedure on an uncompromised user is set to $\perp$ if the ciphertext was not created by the game. More precisely, if the tuple $(\mathsf{ct_{mk}}, \mathsf{n_{mk}}, \mathsf{aid})$ is not in table $\mathsf{T_{ct\text{-}mk}}$. The changes are depicted in Figure 32.

Claim. There exists an adversary $\mathcal{C}_1$ against INT-CTXT such that

$$\Pr\left[\mathsf{G}_2(\mathcal{A}) \Rightarrow 1\right] - \Pr\left[\mathsf{G}_3(\mathcal{A}) \Rightarrow 1\right] \leq \mathsf{Adv}^{Q\text{-INT-CTXT}}_{\mathsf{AEAD}}(\mathcal{C}_1)$$

with $Q = (Q_{\textsc{Reg}_1}, Q_{\textsc{Comp}}, Q_{\textsc{Reg}_1}, Q_{\textsc{Auth}_1})$.

Fig. 33. Adversary $\mathcal{B}$ against INS-OW security of Auth having access to oracles Reg, Init, Fin, $\text{Comp}_{\mathcal{B}}$, and Check simulating $\mathsf{G}_1/\mathsf{G}_2$ for adversary $\mathcal{A}$.

Proof (Claim). The reduction $\mathcal{C}_1$ is formally constructed in Figure 34 showing all oracles that are not trivially simulatable, i.e. where the reduction cannot just execute the oracle as is. The registration oracle can be simulated by using the reduction's encryption oracle on a fresh instance and the ciphertext is stored to answer decryption queries in Line 60. In the last step of the authentication procedure (oracle Step on the authentication sub protocol in the third step, i.e. $\mathsf{r} = 3$), the reduction can either use the value that was encrypted during the registration process as described before or if the ciphertext is new, it can check if it is a valid ciphertext using oracle Check. This is done only for uncompromised users. If it is a valid ciphertext, i.e. decrypting to a message $\mathsf{seed_{mk}} \neq \bot$, $\mathcal{C}_1$ can win their INT-CTXT game by outputting the valid tuple. Since the ciphertext is new because it is not in table $\mathsf{T_{ct\text{-}mk}}$, the winning condition of $\mathcal{C}_1$ holds.

Fig. 34. Adversary $\mathcal{C}_1$ against INT-CTXT security of AEAD having access to oracles Enc, $\text{Comp}_\mathcal{C}$, and Check simulating $\mathsf{G}_2/\mathsf{G}_3$ for adversary $\mathcal{A}$.

It remains to show that the simulation of $\mathcal{C}_1$ is sound with respective to consistent random oracle outputs. The keys for the AEAD is not known to $\mathcal{C}_1$ until a party is compromised. This also means that the output of the random oracle with input $(\mathsf{rw}, \mathsf{aid}, \text{“ek”})$, where rw is the correct Auth output computed during registration, is not known to the reduction. However, due to the abort introduced in the previous game (Line 73), the game aborts whenever the adversary makes such a query on an uncompromised user. That allows $\mathcal{C}_1$ to patch the random oracle upon a corruption query because they learn the secret $\mathsf{k}_{Q'}$.

An additional challenge in this situation is that the reduction does not only have to make the random oracle consistent but also potential states or temporary states. The only state that contains the value k_{ek} is the temporary client state $\mathsf{st}_{\mathsf{C}}^{\mathsf{tmp}}$ during the authentication protocol. More specifically, in the second step of the authentication protocol k_{ek} is computed and stored in the temporary state (Line 45) and in the third step it is used for decryption (Line 56). To make the third step consistent, the reduction updates the state before it is used. This can be done by storing the value rw during the second step (Line 44) and query the random oracle again on the same values in the third step (Line 53). This captures all cases since for the real rw, the random oracle was reprogrammed during a compromise. □

Game G_4. This is the same game as the previous one except that it aborts if there exists a random oracle query on a registered and uncompromised user such that the same input was previously queried to the internal RO H and the tag is "mk". The game also aborts in the registration oracle if the queried user is not compromised and there already exists a random oracle query on the same $\mathsf{seed}_{\mathsf{mk}}$ and aid. The changes are depicted in Figure 32.

Claim. There exists an adversary $\mathcal{D}_1$ against $\mathsf{OW\text{-}CPA}$ such that

$$\Pr\left[\mathsf{G}_3(\mathcal{A}) \Rightarrow 1\right] - \Pr\left[\mathsf{G}_4(\mathcal{A}) \Rightarrow 1\right] \leq \mathsf{Adv}^{Q\text{-}\mathsf{OW\text{-}CPA}}_{\mathsf{AEAD},\mathsf{kl}}(\mathcal{D}_1),$$

with $Q = (Q_{\mathrm{REG}_1}, Q_{\mathrm{COMP}}, 0, Q_{\mathrm{REG}_1}, Q_{\mathrm{REG}_1} + Q_{\mathrm{RO}})$.

Proof (Claim). Adversary $\mathcal{D}_1$ is formally constructed in Figure 35. They simulate their registration oracle by calling CHL for uncompromised users. This works since $\mathsf{seed}_{\mathsf{mk}}$ is a uniformly random value from the message set $\{0,1\}^{\mathsf{kl}}$. If the adversary queried the random oracle on a value seed which is by chance the encrypted value $\mathsf{seed}_{\mathsf{mk}}$, the reduction can recognize that by using oracle CHECK and win the $\mathsf{OW\text{-}CPA}$ game. As for the previous reductions, the random oracle is queried on a unique symbol "chl" (not being in the valid input space of the random oracle) and programmed later. The authentication protocol can be simulated as follows. Due to the changes in the previous game, $\mathcal{D}_1$ can answer all queries for uncompromised users: either by returning $\perp$ if the ciphertext is new or by querying the random oracle on "chl" if the ciphertext came from a previous query to the registration oracle. For the random oracle, if adversary $\mathcal{A}$ queries it on a value corresponding to a challenge ciphertext, which can be recognized by a query to CHECK again, the reduction wins their one-wayness game. The simulation of compromises can be handled identically to the reduction in Figure 34 from the proof of G_3. □

Game G_5. This is the same game as the previous one except that it aborts if there is a collision in the nonces for the same user or file during queries to the put oracle, the update oracle, or the accept oracle. In more detail, we use set $\mathcal{S}_{\mathsf{keys},\mathsf{aid}}$ to store the nonces used by user aid for key encryption and $\mathcal{S}_{\mathsf{files},\mathsf{fid}}$ to store the nonces used to encrypt files with id fid. If the game chooses a nonce already present in the set, it aborts. The changes are depicted in Figure 36.

Claim. It holds that

$$\Pr\left[\mathsf{G}_4(\mathcal{A}) \Rightarrow 1\right] - \Pr\left[\mathsf{G}_5(\mathcal{A}) \Rightarrow 1\right] \leq \frac{(Q_{\mathrm{PUT}_1} + Q_{\mathrm{ACPT}_1})^2 + (Q_{\mathrm{UPD}_1} + 1)^2}{2^{\mathsf{nl}+1}}.$$

Proof (Claim). For $\mathcal{S}_{\mathsf{keys},\mathsf{aid}}$, there is one nonce chosen per query to PUT_1 or ACPT_1 for a particular user aid. Since nonces have length nl, the first part of the bound is the probability of a collision for the respective number of queries. For a given file id fid, there is one encryption in the call to PUT_1 and at most Q_{UPD_1} encryptions in update oracle calls resulting in the presented bound. □

Game G_6. This is the same game as the previous one except that the decryption of every file key ciphertext owned by uncompromised user is set to $\perp$ if the ciphertext was not created by the game. More precisely, if

Fig. 35. Adversary $\mathcal{D}_1$ against OW-CPA security of AEAD having access to oracles Enc, Chl, $\text{Comp}_{\mathcal{D}}$, and Check simulating $\mathsf{G}_3/\mathsf{G}_4$ for adversary $\mathcal{A}$.

Oracle $\textsc{Put}_1(\mathsf{i_c} \in [n_c], \mathsf{fid}, \mathsf{file})$

00 **require** $\mathsf{T_f}[\mathsf{fid}] = \bot$
01 $\mathsf{T_f}[\mathsf{fid}] \leftarrow \{\mathsf{T_u}[\mathsf{i_c}]\}$
02 $(\mathsf{in_C.fid}, \mathsf{in_C.file}) \leftarrow (\mathsf{fid}, \mathsf{file})$
03 $\mathsf{aid} \leftarrow \mathsf{T_u}[\mathsf{i_c}]$
04 $\mathsf{st_C} \leftarrow \mathsf{T_{st_C}}[\mathsf{i_c}]$
05 $\mathsf{n_k} \xleftarrow{\$} \{0,1\}^{\mathsf{nl}}$
06 $\mathsf{n_f} \xleftarrow{\$} \{0,1\}^{\mathsf{nl}}$
07 **if** $\mathsf{n_k} \in \mathcal{S}_{\mathsf{keys,aid}} \vee \mathsf{n_f} \in \mathcal{S}_{\mathsf{files,fid}}$ \\ G_5-G_7
08 **abort** \\ G_5-G_7
09 $\mathcal{S}_{\mathsf{keys,aid}} \leftarrow \mathcal{S}_{\mathsf{keys,aid}} \cup \{\mathsf{n_k}\}$ \\ G_5-G_7
10 $\mathcal{S}_{\mathsf{files,fid}} \leftarrow \mathcal{S}_{\mathsf{files,fid}} \cup \{\mathsf{n_f}\}$ \\ G_5-G_7
11 $\mathsf{ad} \leftarrow (\mathsf{st_C.aid}, \mathsf{fid}, \text{“fk”})$
12 $\mathsf{seed_k} \xleftarrow{\$} \{0,1\}^{\mathsf{kl}}$
13 $\mathsf{ct_k} \xleftarrow{\$} \mathsf{AEAD.Enc}(\mathsf{st_C.k_{mk}}, \mathsf{n_k}, \mathsf{seed_k}, \mathsf{ad})$
14 $\mathsf{T_{ct\text{-}k}}[\mathsf{aid}, \mathsf{n_k}, \mathsf{fid}, \mathsf{ct_k}] \leftarrow \mathsf{seed_k}$ \\ G_6-G_7
15 **if** $\mathsf{aid} \notin \mathcal{S}_{\mathsf{cmp}} \wedge (\mathsf{seed_k}, \mathsf{fid}, \cdot) \in \mathsf{T_H}$ \\ G_7
16 **abort** \\ G_7
17 $\mathsf{k_f} \xleftarrow{\$} \mathsf{H}(\mathsf{seed_k}, \mathsf{st_C.aid}, \text{“fk”})$
18 $\mathsf{ct_f} \xleftarrow{\$} \mathsf{AEAD.Enc}(\mathsf{k_f}, \mathsf{n_f}, \mathsf{file}, (\mathsf{fid}, \text{“fk”}))$
19 $\mathsf{m_C} \leftarrow (\mathsf{st_C.sid}, \mathsf{fid}, \mathsf{n_k}, \mathsf{n_f}, \mathsf{ct_k}, \mathsf{ct_f})$
20 $\mathsf{out_C.decision} \leftarrow \mathsf{true}$
21 n_p++
22 $\mathsf{i_p} \leftarrow n_p$
23 $\mathsf{T_p}[(\Pi, \mathsf{i_p})] \leftarrow (\mathsf{i_c}, \varepsilon, \mathsf{in_C}, 2)$
24 **return** $\mathsf{Chk}(\Pi, \mathsf{i_p}, \mathsf{out_C}, \mathsf{m_C})$

Oracle $\textsc{Step}(\mathsf{update}, \mathsf{i_p} \in [n_p], \mathsf{m})$ \\ only for round $\mathsf{r} = 2$

25 $(\mathsf{i_c}, \mathsf{st_C^{tmp}}, \mathsf{in_C}, 2) \leftarrow \mathsf{T_p}[(\Pi, \mathsf{i_p})]$
26 $\mathsf{aid} \leftarrow \mathsf{T_u}[\mathsf{i_c}]$
27 $\mathsf{fid} \leftarrow \mathsf{st_C^{tmp}.fid}$
28 $\mathsf{seed_k} \leftarrow \mathsf{getHeader}^{\mathsf{C:2}}(\mathsf{st_C}, \mathsf{fid}, \mathsf{m_S})$
29 $\mathsf{k_f} \leftarrow \mathsf{H}(\mathsf{seed_k}, \mathsf{fid}, \text{“fk”})$
30 $\mathsf{n} \xleftarrow{\$} \{0,1\}^{\mathsf{nl}}$
31 **if** $\mathsf{n} \in \mathcal{S}_{\mathsf{files,fid}}$ \\ G_5-G_7
32 **abort** \\ G_5-G_7
33 $\mathcal{S}_{\mathsf{files,fid}} \leftarrow \mathcal{S}_{\mathsf{files,fid}} \cup \{\mathsf{n}\}$ \\ G_5-G_7
34 $\mathsf{ct_f} \xleftarrow{\$} \mathsf{AEAD.Enc}(\mathsf{k_f}, \mathsf{n}, \mathsf{st_C^{tmp}.file}, (\mathsf{fid}, \text{“file”}))$
35 $\mathsf{m_C} \leftarrow (\mathsf{st_C.sid}, \mathsf{fid}, \mathsf{n}, \mathsf{ct_k}, \mathsf{ct_f})$
36 $\mathsf{out_C.decision} \leftarrow \mathsf{true}$
37 $\mathsf{T_p}[(\Pi, \mathsf{i_p})] \leftarrow (\mathsf{i_c}, \mathsf{st_C^{tmp}}, \mathsf{in_C}, 3)$
38 **return** $\mathsf{Chk}(\Pi, \mathsf{i_p}, \mathsf{out_C}, \mathsf{m_C})$

Oracle $\mathsf{H}(\mathsf{seed}, \mathsf{aid}, \mathsf{label})$

39 **if** $(\mathsf{seed}, \mathsf{aid}, \mathsf{label}) \notin \mathsf{T_H}$
40 $\mathsf{T_H}[\mathsf{seed}, \mathsf{aid}, \mathsf{label}] \xleftarrow{\$} \{0,1\}^{\mathsf{kl}}$
41 $\mathsf{T_{H\text{-}int}}[\mathsf{seed}, \mathsf{aid}, \mathsf{label}] \leftarrow 1$
42 **return** $\mathsf{T_H}[\mathsf{seed}, \mathsf{aid}, \mathsf{label}]$

Oracle $\textsc{Comp}(\mathsf{aid})$

43 **require** $\mathsf{T_{uaid}}[\mathsf{aid}] \neq \bot$
44 $\mathcal{S}_{\mathsf{cmp}} \leftarrow \mathcal{S}_{\mathsf{cmp}} \cup \{\mathsf{aid}\}$
45 **return** $(\mathsf{T_{pw}}[\mathsf{aid}], \{\mathsf{T_{st_C}}[\mathsf{i_c}] \mid \mathsf{T_u}[\mathsf{i_c}] = \mathsf{aid}\}, \{\mathsf{T_{oob}}[(\mathsf{aid}, \cdot)]\})$

Oracle $\textsc{Acpt}_1(\mathsf{i_c} \in [n_c], \mathsf{fid}, \mathsf{oob})$

46 $\mathsf{aid} \leftarrow \mathsf{T_u}[\mathsf{i_c}]$
47 **require** $\mathsf{oob} \neq \bot \vee \mathsf{T_{oob}}[(\mathsf{aid}, \mathsf{fid})] \neq \bot$
48 $\mathsf{T_f}[\mathsf{fid}] \leftarrow \mathsf{T_f}[\mathsf{fid}] \cup \{\mathsf{aid}\}$
49 **if** $\mathsf{oob} = \bot$
50 $\mathsf{in_C.oob} \leftarrow \mathsf{T_{oob}}[(\mathsf{aid}, \mathsf{fid})]$
51 **else**
52 $\mathsf{T_f}[\mathsf{fid}] \leftarrow \mathsf{T_f}[\mathsf{fid}] \cup \{i_{\mathsf{mal}}\}$
53 $\mathsf{in_C.oob} \leftarrow \mathsf{oob}$
54 **if** $\mathsf{fid} \in \mathcal{S}_{\mathsf{ch}}$
55 $\mathcal{S}_{\mathsf{chOwn}} \leftarrow \mathcal{S}_{\mathsf{chOwn}} \cup \mathsf{T_f}[\mathsf{fid}]$
56 $\mathsf{st_C} \leftarrow \mathsf{T_{st_C}}[\mathsf{i_c}]$
57 $\mathsf{ad} \leftarrow (\mathsf{st_C.aid}, \mathsf{in_C.fid}, \text{“fk”})$
58 $\mathsf{n} \xleftarrow{\$} \{0,1\}^{\mathsf{nl}}$
59 **if** $\mathsf{n} \in \mathcal{S}_{\mathsf{keys,aid}}$ \\ G_5-G_7
60 **abort** \\ G_5-G_7
61 $\mathcal{S}_{\mathsf{keys,aid}} \leftarrow \mathcal{S}_{\mathsf{keys,aid}} \cup \{\mathsf{n}\}$ \\ G_5-G_7
62 $\mathsf{seed_k} \leftarrow \mathsf{in_C.oob}$
63 $\mathsf{ct_k} \xleftarrow{\$} \mathsf{AEAD.Enc}(\mathsf{st_C.k_{mk}}, \mathsf{n}, \mathsf{seed_k}, \mathsf{ad})$
64 $\mathsf{T_{ct\text{-}k}}[\mathsf{aid}, \mathsf{n}, \mathsf{fid}, \mathsf{ct_k}] \leftarrow \mathsf{seed_k}$ \\ G_6-G_7
65 $\mathsf{m_C} \leftarrow (\mathsf{st_C.sid}, \mathsf{in_C.fid}, \mathsf{n}, \mathsf{ct_k})$
66 $\mathsf{out_C.decision} \leftarrow \mathsf{true}$
67 n_p++
68 $\mathsf{i_p} \leftarrow n_p$
69 $\mathsf{T_p}[(\Pi, \mathsf{i_p})] \leftarrow (\mathsf{i_c}, \varepsilon, \mathsf{in_C}, 2)$
70 **return** $\mathsf{Chk}(\Pi, \mathsf{i_p}, \mathsf{out_C}, \mathsf{m_C})$

Oracle $\mathrm{RO}(\mathsf{seed}, \mathsf{id}, \mathsf{label})$

71 **if** $\mathsf{T_{uaid}}[\mathsf{id}] = \mathsf{true}, \mathsf{id} \notin \mathcal{S}_{\mathsf{cmp}} \wedge (\mathsf{seed}, \mathsf{id}, \mathsf{label}) \in \mathsf{T_{H\text{-}int}}$
72 **if** $\mathsf{label} \in \{\text{“ek”}, \text{“mac”}\}$
73 **abort**
74 **if** $\mathsf{label} = \text{“mk”}$
75 **abort**
76 **if** $\mathsf{T_f}[\mathsf{id}] \cap \mathcal{S}_{\mathsf{cmp}} = \emptyset \wedge (\mathsf{seed}, \mathsf{id}, \mathsf{label}) \in \mathsf{T_{H\text{-}int}}$
77 **if** $\mathsf{label} = \text{“fk”}$ \\ G_7
78 **abort** \\ G_7
79 **if** $(\mathsf{seed}, \mathsf{id}, \mathsf{label}) \notin \mathsf{T_H}$
80 $\mathsf{T_H}[\mathsf{seed}, \mathsf{id}, \mathsf{label}] \xleftarrow{\$} \{0,1\}^{\mathsf{kl}}$
81 **return** $\mathsf{T_H}[\mathsf{seed}, \mathsf{id}, \mathsf{label}]$

Oracle $\textsc{Chall}(\mathsf{i_c} \in [n_c], \mathsf{fid}, \mathsf{file_0}, \mathsf{file_1}, \mathsf{prot})$

82 **require** $\mathsf{prot} \in \{\mathsf{put}, \mathsf{update}\}$
83 $\mathcal{S}_{\mathsf{ch}} \leftarrow \mathcal{S}_{\mathsf{ch}} \cup \{\mathsf{fid}\}$
84 $\mathcal{S}_{\mathsf{chOwn}} \leftarrow \mathcal{S}_{\mathsf{chOwn}} \cup \{\mathsf{T_u}[\mathsf{i_c}]\} \cup \mathsf{T_f}[\mathsf{fid}]$
85 **return** $\mathsf{prot}(\mathsf{i_c}, \mathsf{fid}, \mathsf{file_b})$

$\mathsf{getHeader}^{\mathsf{C:2}}(\mathsf{st_C}, \mathsf{fid}, \mathsf{m})$

86 $(\mathsf{n_k}, \mathsf{ct_k}) \leftarrow \mathsf{m}$
87 $\mathsf{aid} \leftarrow \mathsf{st_C.aid}$
88 $\mathsf{ad} \leftarrow (\mathsf{aid}, \mathsf{fid}, \text{“fk”})$
89 $\mathsf{seed_k} \leftarrow \mathsf{AEAD.Dec}(\mathsf{st_C.k_{mk}}, \mathsf{n_k}, \mathsf{ct_k}, \mathsf{ad})$
90 **if** $\mathsf{aid} \notin \mathcal{S}_{\mathsf{cmp}} \wedge \mathsf{T_{ct\text{-}k}}[\mathsf{aid}, \mathsf{n_k}, \mathsf{fid}, \mathsf{ct_k}] = \bot$ \\ G_6-G_7
91 $\mathsf{seed_k} \leftarrow \bot$ \\ G_6-G_7
92 **if** $\mathsf{aid} \notin \mathcal{S}_{\mathsf{cmp}} \wedge (\mathsf{seed_k}, \mathsf{fid}, \cdot) \in \mathsf{T_H}$ \\ G_7
93 **abort** \\ G_7
94 **return** $\mathsf{seed_k}$

Fig. 36. Games G_4 – G_7 for the proof of Theorem 10.

the tuple $(\mathsf{aid}, \mathsf{n_k}, \mathsf{fid}, \mathsf{ct_k})$ is not in table $\mathsf{T_{ct\text{-}k}}$. A decryption of file key ciphertexts occurs in the update, get, and share sub protocol. To simplify the depiction, we use subroutine getHeader which is then queried from the three oracles. The changes are depicted in Figure 36.

Claim. There exists an adversary $\mathcal{C}_2$ against INT-CTXT such that

$$\Pr[\mathsf{G}_5(\mathcal{A}) \Rightarrow 1] - \Pr[\mathsf{G}_6(\mathcal{A}) \Rightarrow 1] \leq \mathsf{Adv}^{Q\text{-INT-CTXT}}_{\mathsf{AEAD}}(\mathcal{C}_2)$$

with $Q = (Q_{\text{REG}_1}, Q_{\text{COMP}}, Q_{\text{PUT}_1} + Q_{\text{ACPT}_1} + Q_{\text{CHL}}, Q_{\text{UPD}_1} + Q_{\text{GET}_1} + Q_{\text{SHARE}_1} + Q_{\text{CHL}})$.

Proof (Claim). The reduction $\mathcal{C}_2$ is formally constructed in Figure 37 showing all oracles that are not trivially simulatable, i.e. by just executing the oracle as is. The put and accept oracle can be simulated by using the reduction's encryption oracle and the ciphertext is stored to answer decryption queries. We start a new game instance if there was no instance started for aid yet. The instances are stored in table $\mathsf{T_{key}}$. Note that even though the counter Q is incremented in several oracles, there can be at most Q_{REG_1} many instances since this is an upper bound on the number of aids registered.

The decryption of a file key ciphertext occurs during the update, get, and share protocol but we use the abstraction of the getHeader sub protocol to simplify the depiction. Whenever the reduction needs to simulate such a decryption, they can either use the value that was encrypted by the game before or if the ciphertext is new, it can check if it is a valid ciphertext using oracle CHECK. This is done only for uncompromised users. If it is a valid ciphertext, i.e. decrypting to a message $\mathsf{seed_k} \neq \bot$, $\mathcal{C}_2$ can win their INT-CTXT game by outputting the valid tuple. Since the ciphertext is new because it is not in table $\mathsf{T_{ct\text{-}mk}}$, the winning condition of $\mathcal{C}_2$ holds.

It remains to show that the simulation of $\mathcal{C}_2$ is sound with respective to consistent random oracle outputs. The keys for the AEAD are not known to $\mathcal{C}_2$ until a party is compromised. This also means that the output of the random oracle with input $(\mathsf{seed_{mk}}, \mathsf{aid}, \text{“mk”})$, where $\mathsf{seed_{mk}}$ is the correct seed chosen during the registration process for that user, is not known to the reduction. However, due to the abort introduced in G_4 (Line 93), the game aborts whenever the adversary makes such a query for an uncompromised user. That allows $\mathcal{C}_2$ to patch the random oracle upon a corruption query because they learn the secret $\mathsf{k}_{Q'}$.

An additional challenge in this situation is that the reduction does not only have to make the random oracle consistent but also potential states. This holds for all states of user aid because the persistent states include the master key. Since the states are directly output by the compromise oracle, they must be patched immediately. Hence, the reduction must go through all protocol states for user aid and add $\mathsf{k}_{Q'}$ (Line 45). □

Game G_7. This is the same game as the previous one except that it aborts if there exists a random oracle query on a file id for which the file is only owned by uncompromised users, the same input was previously queried to the internal RO H, and the tag is “fk” (Line 78). The game also aborts in the getHeader routine if the queried user is not compromised and there already exists a random oracle query on the same $\mathsf{seed_{mk}}$ and aid. The changes are depicted in Figure 36.

Claim. There exists an adversary $\mathcal{D}_2$ against OW-CPA such that

$$\Pr[\mathsf{G}_6(\mathcal{A}) \Rightarrow 1] - \Pr[\mathsf{G}_7(\mathcal{A}) \Rightarrow 1] \leq \mathsf{Adv}^{Q\text{-OW-CPA}}_{\mathsf{AEAD,kl}}(\mathcal{D}_2)$$

with $Q = (Q_{\text{REG}_1}, Q_{\text{COMP}}, Q_{\text{ACPT}_1}, Q_{\text{PUT}_1} + Q_{\text{ACPT}_1} + Q_{\text{CHL}}, Q_{\text{PUT}_1} + Q_{\text{UPD}_1} + Q_{\text{GET}_1} + Q_{\text{SHARE}_1} + Q_{\text{CHL}} + Q_{\text{RO}})$.

Proof (Claim). Adversary $\mathcal{D}_2$ is formally constructed in Figure 38. For ease depiction, we use subroutine $\mathsf{getHeader}^{\mathsf{C:2}}$ and model the aborts occurring in the update, get, and share oracle within that subroutine. The game for adversary $\mathcal{A}$ can be simulated by the reduction using their own oracles because due to the changes in G_5 we have unique nonces which is a requirement for the challenge oracle in the OW-CPA game. In more detail, the put oracle can be simulated for uncompromised users using $\mathcal{D}_2$'s challenge oracle with a fresh message. This represents choosing a new (unknown) seed $\mathsf{seed_k}$ and encrypting it. If there was a previous

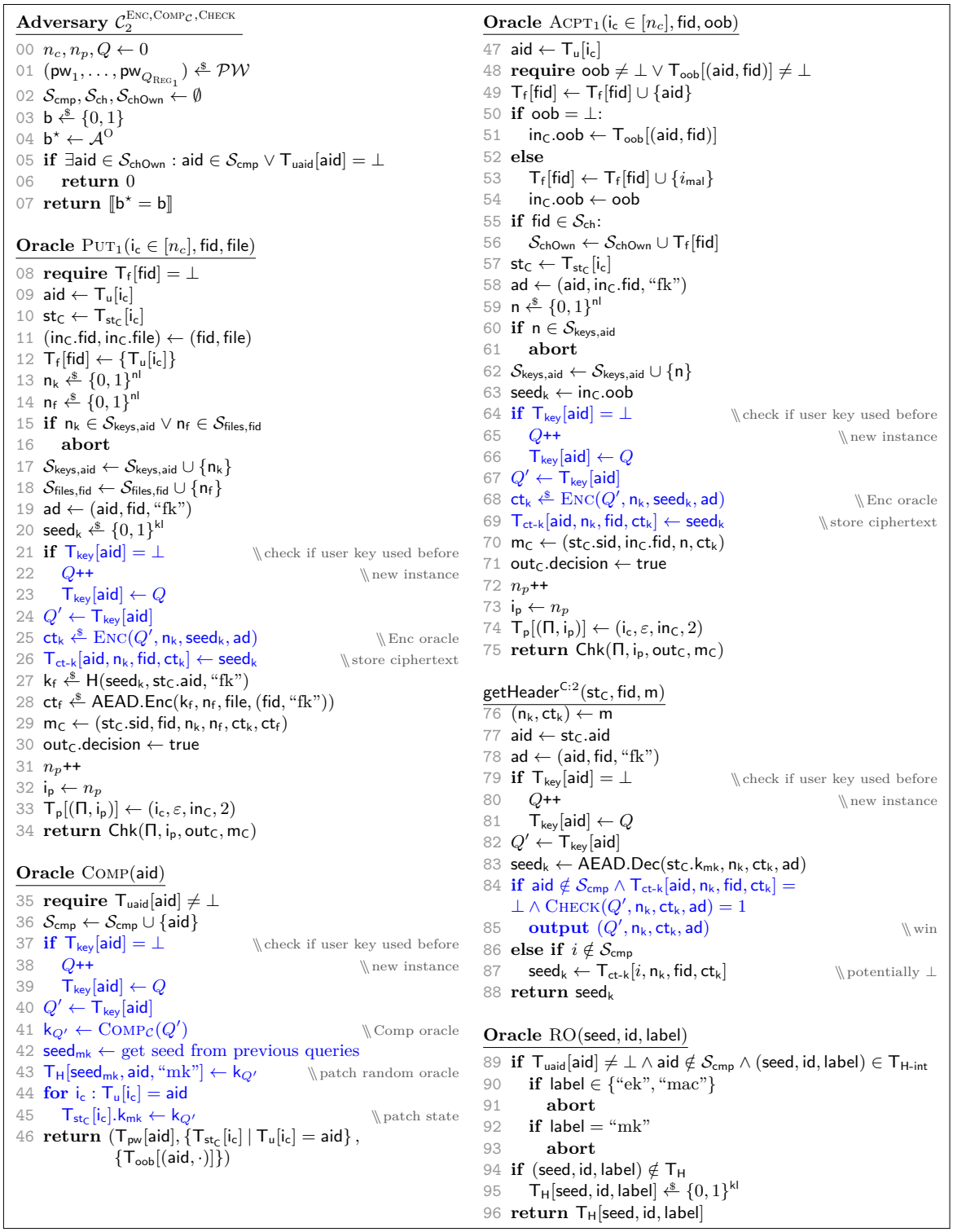

Fig. 37. Adversary $\mathcal{C}_2$ against INT-CTXT security of AEAD having access to oracles Enc, $\text{Comp}_\mathcal{C}$, and Check simulating $\mathsf{G}_5/\mathsf{G}_6$ for adversary $\mathcal{A}$.

random oracle query such that the input is the unknown message of the just created ciphertext, adversary $\mathcal{D}_2$ wins their game. This can be checked using oracle CHECK. If the random oracle was not queried on a correct value, it is queried on challenge tag "chl" to patch it later. The ciphertext together with some relevant information like the message tag Q_m is stored for later use (Line 29). The main challenge for the reduction compared to previous games is that files can be shared by users. More specifically, the seeds for the file keys are shared. Since these seeds are encrypted by each user (who is an owner of some file), the reduction needs to simulate ciphertexts for different users that encrypt the same plaintext. This is why we need the challenge oracle CHL to have the message tag as an input, i.e. adversary $\mathcal{D}_2$ can specify to either obtain the encryption of a new message by querying the challenge on a fresh index Q_m or obtain an encryption of a previous query by querying the challenge on a previously used Q'_m. In the accept oracle, this is how $\mathcal{D}_2$ is simulating a query to an uncompromised user in case the seed for a file with id fid was already encrypted.

The aborts in the update, get, and share oracles are applied in the subroutine $\mathsf{getHeader}$ and follow the same structure as the previous reductions since $\mathcal{D}_2$ can check correct seeds via their check oracle. The same holds for the abort in the random oracle.

For consistent simulation, the reduction needs to patch the random oracle upon corruption. To this end, $\mathcal{D}_2$ recovers the master key seed for user aid and patches the random oracle on the user's master key $\mathsf{k}_{Q'_u}$ which is obtained by $\mathcal{D}_2$'s own compromise oracle. Note that the simulation is sound because the random oracle was not queried before on the same inputs due to the abort in Line 114 introduced in G_4. Patching the states is similar to the reduction for G_6. □

Final reduction. Game G_7 is now in a state where we can reduce it to the IND-CCA security of AEAD.

Claim. There exists an adversary $\mathcal{E}$ against IND-CCA such that

$$2\,\mathsf{Pr}[\mathsf{G}_7(\mathcal{A}) \Rightarrow 1] - 1 \leq \mathsf{Adv}^{Q\text{-IND-CCA}}_{\mathsf{AEAD}}(\mathcal{E})$$

with $Q = (Q_{\text{PUT}_1} + Q_{\text{CHL}}, Q_{\text{PUT}_1} + Q_{\text{CHL}}, Q_{\text{PUT}_1} + Q_{\text{UPD}_1}, Q_{\text{GET}_1}, Q_{\text{CHL}})$.

Proof (Claim). Adversary $\mathcal{E}$ is formally constructed in Figure 39. $\mathcal{E}$ is not sampling file keys themselves but using their own IND-CCA security game against AEAD. For each query to PUT_1, they increase a counter Q and use their own encryption oracle ENC with a new instance to encrypt the file. If there is a challenge query implicitly querying the put oracle but on file file_b, $\mathcal{E}$ queries their challenge oracle CHL instead. This is done using both files, file_0 and file_1, from $\mathcal{A}$'s challenge query. To simplify the depiction in Figure 39, we combine direct put queries and put queries called via the challenge oracle and use the same lines of code. The differences explained before are highlighted with comment PUT_1C. Further, the instance number Q is stored together with the seed seed_k and the file id fid in table T_fid such that the reduction can later query their oracles on the correct instance. This happens in the second step of the get oracle where $\mathcal{E}$ needs to potentially decrypt a ciphertext with respect to a previously used file key (Line 68) which is the only situation in which a file decryption is necessary.

In the second step of the update procedure, $\mathcal{E}$ might require an encryption with respect to a unknown file key again. In case the update oracle was queried directly, i.e. not via a challenge to CHL, the reduction can recognize previously used instances via table T_fid and use oracle ENC. Analogously to the put challenge, challenges for the update routine can be answered via a query to $\text{CHL}_\mathcal{E}$.

Note that this simulation is sound because the adversary cannot learn file keys that belong to any uncompromised user due to the abort introduced in G_7. Further, due to the changes in G_5, we do not have any colliding nonces such that $\mathcal{E}$'s oracles always output a solution.

The remaining issue is to handle compromise queries. A compromise query of $\mathcal{A}$ is with respect to a user while $\mathcal{E}$ is playing a game with several AEAD instances corresponding to files instead of users. Hence, upon a corruption on user aid we could check all files which aid has access to and compromise the corresponding file keys for all these files. To obtain a tighter reduction, we deploy the reverse check for any query to the random oracle (either explicit ones to RO or implicit ones by the game to H. A reverse check means that we check if the file for a query to the random oracle (if there exists a file id id) is owned by a compromised user.

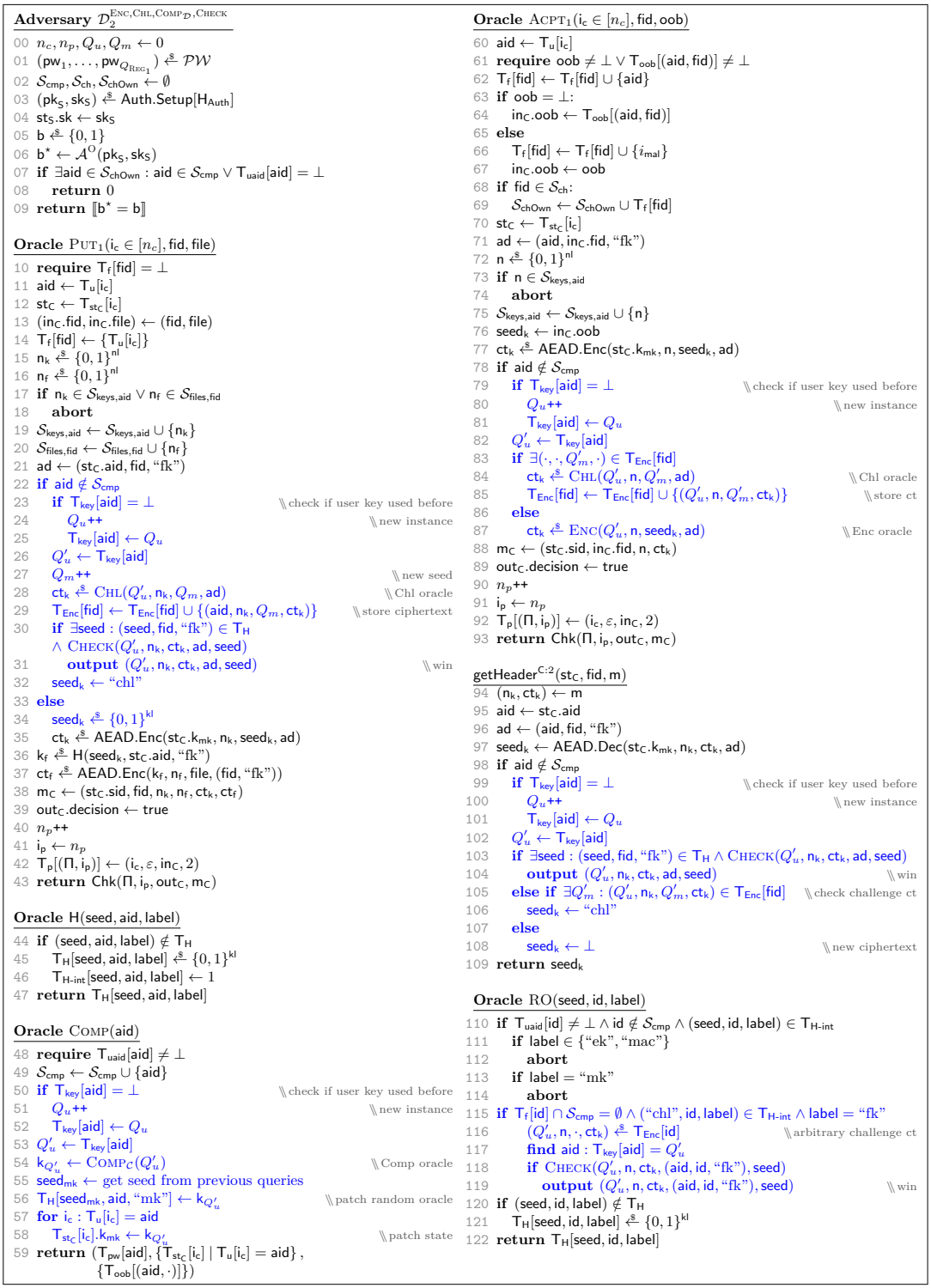

Fig. 38. Adversary $\mathcal{D}_2$ against OW-CPA security of AEAD having access to oracles Enc, $\text{Comp}_\mathcal{D}$, and Check simulating $\mathsf{G}_6/\mathsf{G}_7$ for adversary $\mathcal{A}$.

Adversary $\mathcal{E}^{\text{ENC,DEC,CHL,COMP}_{\mathcal{E}}}$

$k_1, \ldots, k_{Q_{\text{PUT}_1}+Q_{\text{CHL}}} \leftarrow \bot$ \\keys unknown
$n_c, n_p, Q \leftarrow 0$
$(pw_1, \ldots, pw_{Q_{\text{REG}_1}}) \xleftarrow{s} \mathcal{PW}$
$\mathcal{S}_{\text{cmp}}, \mathcal{S}_{\text{ch}}, \mathcal{S}_{\text{chOwn}} \leftarrow \emptyset$
$(pk_S, sk_S) \xleftarrow{s} \text{Auth.Setup}[H_{\text{Auth}}]$
$st_S.sk \leftarrow sk_S$
$b \xleftarrow{s} \{0,1\}$
$b^\star \leftarrow \mathcal{A}^{O}(pk_S, sk_S)$
if $\exists \text{aid} \in \mathcal{S}_{\text{chOwn}} : \text{aid} \in \mathcal{S}_{\text{cmp}} \vee T_{\text{uaid}}[\text{aid}] = \bot$
  return 0
return $[\![b^\star = b]\!]$

Oracle $\text{PUT}_1(i_c \in [n_c], \text{fid}, \text{file})$
Oracle $\text{PUT}_1\text{C}(i_c \in [n_c], \text{fid}, \text{file}_0, \text{file}_1)$ \\queried from CHALL

require $T_{\text{uaid}}[\text{aid}] = \text{true}$
$\text{aid} \leftarrow T_u[i_c]$
$st_C \leftarrow T_{st_C}[i_c]$
$T_f[\text{fid}] \leftarrow \{T_u[i_c]\}$
$n_k \xleftarrow{s} \{0,1\}^{n_l}$
$n_f \xleftarrow{s} \{0,1\}^{n_l}$
if $n_k \in \mathcal{S}_{\text{keys,aid}} \vee n_f \in \mathcal{S}_{\text{files,fid}}$
  abort
$\mathcal{S}_{\text{keys,aid}} \leftarrow \mathcal{S}_{\text{keys,aid}} \cup \{n_k\}$
$\mathcal{S}_{\text{files,fid}} \leftarrow \mathcal{S}_{\text{files,fid}} \cup \{n_f\}$
$\text{ad} \leftarrow (\text{aid}, \text{fid}, \text{“fk”})$
$\text{seed}_k \xleftarrow{s} \{0,1\}^{k_l}$
$ct_k \xleftarrow{s} \text{AEAD.Enc}(st_C.k_{mk}, n_k, \text{seed}_k, \text{ad})$
$T_{\text{ct-k}}[\text{aid}, n_k, \text{fid}, ct_k] \leftarrow \text{seed}_k$
if $\text{aid} \notin \mathcal{S}_{\text{cmp}} \wedge (\text{seed}_k, \text{fid}, \cdot) \in T_H$
  abort
$k_f \xleftarrow{s} H(\text{seed}_k, \text{fid}, \text{“fk”})$
$Q \leftarrow Q + 1$ \\new instance
$T_{\text{fid}}[\text{seed}_k, \text{fid}] \leftarrow Q$
$ct_f \xleftarrow{s} \text{ENC}(Q, n_f, \text{file}, (\text{fid}, \text{“fk”}))$ \\enc oracle
$ct_f \xleftarrow{s} \text{CHL}(Q, n_f, \text{file}_0, \text{file}_1, (\text{fid}, \text{“fk”}))$ \\chl oracle, only PUT_1C
$m_C \leftarrow (st_C.\text{sid}, \text{fid}, n_k, n_f, ct_k, ct_f)$
$\text{out}_C.\text{decision} \leftarrow \text{true}$
n_p++
$i_p \leftarrow n_p$
$T_p[(\Pi, i_p)] \leftarrow (i_c, \varepsilon, \text{in}_C, 2)$
return $\text{Chk}(\Pi, i_p, \text{out}_C, m_C)$

Oracle $H(\text{seed}, \text{id}, \text{label})$

if $T_f[\text{id}] \cap \mathcal{S}_{\text{cmp}} \neq \emptyset \wedge T_{\text{fid}}[\text{seed}, \text{id}] \neq \bot$
  $Q' \leftarrow T_{\text{fid}}[\text{seed}, \text{id}]$
  if $k_{Q'} = \bot$
    $k_{Q'} \xleftarrow{s} \text{COMP}_{\mathcal{E}}(Q')$ \\comp oracle
  $T_H[\text{seed}, \text{id}, \text{label}] \leftarrow k_{Q'}$ \\patch RO
if $(\text{seed}, \text{aid}, \text{label}) \notin T_H$
  $T_H[\text{seed}, \text{aid}, \text{label}] \xleftarrow{s} \{0,1\}^{k_l}$
  $T_{\text{H-int}}[\text{seed}, \text{aid}, \text{label}] \leftarrow 1$
return $T_H[\text{seed}, \text{aid}, \text{label}]$

Oracle $\text{COMP}(\text{aid})$

$\mathcal{S}_{\text{cmp}} \leftarrow \mathcal{S}_{\text{cmp}} \cup \{\text{aid}\}$
return $(T_{\text{pw}}[\text{aid}], \{T_{st_C}[i_c] \mid T_u[i_c] = \text{aid}\},$
$\{T_{\text{oob}}[(\text{aid}, \cdot)]\})$

Oracle $\text{CHALL}(i_c \in [n_c], \text{fid}, \text{file}_0, \text{file}_1, \text{prot})$

require $\text{prot} \in \{\text{PUT}_1, \text{UPD}_1\}$
$\mathcal{S}_{\text{ch}} \leftarrow \mathcal{S}_{\text{ch}} \cup \{\text{fid}\}$
$\mathcal{S}_{\text{chOwn}} \leftarrow \mathcal{S}_{\text{chOwn}} \cup \{T_u[i_c]\} \cup T_f[\text{fid}]$
return $\text{protC}(i_c, \text{fid}, \text{file}_0, \text{file}_1)$

Oracle $\text{STEP}(\text{get}, i_p, m)$ \\only for round $r = 2$

$(i_c, st_C^{\text{tmp}}, \text{in}_C, 2) \leftarrow T_p[(\Pi, i_p)]$
$\text{aid} \leftarrow T_u[i_c]$
$st_C \leftarrow T_{st_C}[i_c]$
$\text{fid} \leftarrow st_C^{\text{tmp}}.\text{fid}$
$((n_k, ct_k), (n_f, ct_f)) \leftarrow m$
$\text{ad} \leftarrow (st_C.\text{aid}, \text{fid}, \text{“fk”})$
$\text{seed}_k \leftarrow \text{AEAD.Dec}(st_C.k_{mk}, n_k, ct_k, \text{ad})$
if $\text{aid} \notin \mathcal{S}_{\text{cmp}} \wedge T_{\text{ct-k}}[\text{aid}, n_k, \text{fid}, ct_k] = \bot$
  $\text{seed}_k \leftarrow \bot$
if $\text{aid} \notin \mathcal{S}_{\text{cmp}} \wedge (\text{seed}_k, \text{fid}, \cdot) \in T_H$
  abort
$k_f \leftarrow H(\text{seed}_k, \text{fid}, \text{“fk”})$
$\text{out}_C.\text{file} \leftarrow \text{AEAD.Dec}(k_f, n_f, ct_f, (\text{fid}, \text{“file”}))$
if $T_{\text{fid}}[\text{seed}_k, \text{fid}] \neq \bot$
  $Q' \leftarrow T_{\text{fid}}[\text{seed}_k, \text{fid}]$
  $\text{out}_C.\text{file} \leftarrow \text{DEC}(Q', n_f, ct_f, (\text{fid}, \text{“file”}))$ \\dec oracle
$\text{out}_C.\text{decision} \leftarrow \text{true}$
$T_p[(\Pi, i_p)] \leftarrow (i_c, st_C^{\text{tmp}}, \text{in}_C, 3)$
return $\text{Chk}(\Pi, i_p, \text{out}_C, m_C)$

Oracle $\text{UPD}_1\text{C}(i_c, \text{fid}, \text{file}_0, \text{file}_1)$ \\store both files

$st_C \leftarrow T_{st_C}[i_c]$
$m_C \leftarrow \text{getHeader}^{C:1}(st_C, \text{in}_C.\text{fid})$
$(st_C^{\text{tmp}}.\text{fid}, st_C^{\text{tmp}}.\text{file}_0, st_C^{\text{tmp}}.\text{file}_1) \leftarrow (\text{fid}, \text{file}_0, \text{file}_1)$
n_p++
$i_p \leftarrow n_p$
$T_p[(\text{update}, i_p)] \leftarrow (i_c, st_C^{\text{tmp}}, \text{in}_C, 2)$
return $\text{Chk}(\text{update}, i_p, \text{out}_C, m_C)$

Oracle $\text{STEP}(\text{update}, i_p, m)$ \\only for round $r = 2$

$(i_c, st_C^{\text{tmp}}, \text{in}_C, 2) \leftarrow T_p[(\Pi, i_p)]$
$\text{aid} \leftarrow T_u[i_c]$
$\text{fid} \leftarrow st_C^{\text{tmp}}.\text{fid}$
$\text{seed}_k \leftarrow \text{getHeader}^{C:2}(st_C, \text{fid}, m)$
$k_f \leftarrow H(\text{seed}_k, \text{fid}, \text{“fk”})$
$n \xleftarrow{s} \{0,1\}^{n_l}$
if $n \in \mathcal{S}_{\text{files,fid}}$
  abort
$\mathcal{S}_{\text{files,fid}} \leftarrow \mathcal{S}_{\text{files,fid}} \cup \{n\}$
$ct_f \xleftarrow{s} \text{AEAD.Enc}(k_f, n, st_C^{\text{tmp}}.\text{file}, (\text{fid}, \text{“file”}))$
if $T_{\text{fid}}[\text{seed}_k, \text{fid}] \neq \bot$
  $Q' \leftarrow T_{\text{fid}}[\text{seed}_k, \text{fid}]$
  if $st_C^{\text{tmp}}.\text{file}_0 \neq \bot$ \\check challenge case
    $ct_f \xleftarrow{s} \text{CHL}_{\mathcal{E}}(Q', n, st_C^{\text{tmp}}.\text{file}_0, st_C^{\text{tmp}}.\text{file}_1, (\text{fid}, \text{“file”}))$ \\chl oracle
  else
    $ct_f \xleftarrow{s} \text{ENC}(Q, n, st_C^{\text{tmp}}.\text{file}, (\text{fid}, \text{“file”}))$ \\enc oracle
$m_C \leftarrow (st_C.\text{sid}, \text{fid}, n, ct_k, ct_f)$
$\text{out}_C.\text{decision} \leftarrow \text{true}$
$T_p[(\Pi, i_p)] \leftarrow (i_c, st_C^{\text{tmp}}, \text{in}_C, 3)$
return $\text{Chk}(\Pi, i_p, \text{out}_C, m_C)$

Oracle $\text{RO}(\text{seed}, \text{id}, \text{label})$

if $T_{\text{uaid}}[\text{id}] \neq \bot \wedge \text{id} \notin \mathcal{S}_{\text{cmp}} \wedge (\text{seed}, \text{id}, \text{label}) \in T_{\text{H-int}}$
  if $\text{label} \in \{\text{“ek”}, \text{“mac”}\}$ **abort**
  if $\text{label} = \text{“mk”}$ **abort**
if $T_f[\text{id}] \cap \mathcal{S}_{\text{cmp}} = \emptyset \wedge (\text{seed}, \text{id}, \text{label}) \in T_{\text{H-int}}$
  if $\text{label} = \text{“fk”}$ **abort**
else if $T_f[\text{id}] \cap \mathcal{S}_{\text{cmp}} \neq \emptyset \wedge T_{\text{fid}}[\text{seed}, \text{id}] \neq \bot$
  $Q' \leftarrow T_{\text{fid}}[\text{seed}, \text{id}]$
  if $k_{Q'} = \bot$
    $k_{Q'} \xleftarrow{s} \text{COMP}_{\mathcal{E}}(Q')$ \\comp oracle
  $T_H[\text{seed}, \text{id}, \text{label}] \leftarrow k_{Q'}$ \\patch RO
if $(\text{seed}, \text{id}, \text{label}) \notin T_H$
  $T_H[\text{seed}, \text{id}, \text{label}] \xleftarrow{s} \{0,1\}^{k_l}$
return $T_H[\text{seed}, \text{id}, \text{label}]$

Fig. 39. Adversary $\mathcal{E}$ against IND-CCA security of AEAD having access to oracles ENC, DEC, CHL, and CHECK simulating G_7 for adversary $\mathcal{A}$.

If this is the case, we look for the correct instance which is stored in $\mathsf{T}_{\mathsf{fid}}$ and compromise the key of said instance if this was not done already. The random oracle can then be patched on this value. Note that the number of compromises is still not the number of random oracle queries since there are at most $Q_{\textsc{Put}_1} + Q_{\textsc{Chl}}$ many file keys that can be corrupted. □

Collecting the probabilities completes the proof of Theorem 10. □

C.2 Integrity

We restate Theorem 6 using concrete bounds and provide the full proof.

Theorem 11 (Integrity). *For any password distribution $\mathcal{PW}$ and adversary $\mathcal{A}$ against the* Int *security of* $\mathsf{CS} := \mathsf{CS}[\mathsf{Auth}, \mathsf{AEAD}, \mathsf{MAC}]$ *there exist an* INS-OW *adversary $\mathcal{B}$ against* Auth, INT-CTXT *adversaries $\mathcal{C}_1, \mathcal{C}_2$ against* AEAD, OW-CPA *adversaries $\mathcal{D}_1, \mathcal{D}_2$ against* AEAD, *and an* INT-CTXT *adversary $\mathcal{E}$ against* AEAD *with $t_{\mathcal{A}} \approx t_{\mathcal{B}} \approx t_{\mathcal{C}_1} \approx t_{\mathcal{D}_1} \approx t_{\mathcal{C}_2} - Q_{\textsc{Comp}} \cdot Q_{\textsc{Auth}_1} \approx t_{\mathcal{D}_2} - Q_{\textsc{Comp}} \cdot Q_{\textsc{Auth}_1} \approx t_{\mathcal{E}}$ such that*

$$\begin{aligned}\mathsf{Adv}^{Q_{\mathcal{A}}\text{-}\mathsf{Int}}_{\mathsf{CS},\mathcal{PW}}(\mathcal{A}) \leq\ & \mathsf{Adv}^{Q_{\mathcal{B}}\text{-}\mathsf{INS\text{-}OW}}_{\mathsf{Auth},\mathcal{PW}}(\mathcal{B}) + \mathsf{Adv}^{Q_{\mathcal{C}_1}\text{-}\mathsf{INT\text{-}CTXT}}_{\mathsf{AEAD}}(\mathcal{C}_1) + \mathsf{Adv}^{Q_{\mathcal{D}_1}\text{-}\mathsf{OW\text{-}CPA}}_{\mathsf{AEAD},\mathsf{kl}}(\mathcal{D}_1) \\ & + \mathsf{Adv}^{Q_{\mathcal{C}_2}\text{-}\mathsf{INT\text{-}CTXT}}_{\mathsf{AEAD}}(\mathcal{C}_2) + \mathsf{Adv}^{Q_{\mathcal{D}_2}\text{-}\mathsf{OW\text{-}CPA}}_{\mathsf{AEAD},\mathsf{kl}}(\mathcal{D}_2) + \mathsf{Adv}^{Q_{\mathcal{E}}\text{-}\mathsf{INT\text{-}CTXT}}_{\mathsf{AEAD}}(\mathcal{E}) \\ & + Q_{\textsc{Auth}_1} \cdot \delta_{\mathsf{Auth}} + \frac{(Q_{\textsc{Put}_1} + Q_{\textsc{Acpt}_1})^2 + (Q_{\textsc{Upd}_1} + 1)^2}{2^{\mathsf{nl}+1}}\end{aligned}$$

with

$$\begin{aligned}Q_{\mathcal{A}} &= (Q_{\textsc{Step}}, Q_{\textsc{Reg}_1}, Q_{\textsc{Auth}_1}, Q_{\textsc{Put}_1}, Q_{\textsc{Upd}_1}, Q_{\textsc{Get}_1}, Q_{\textsc{Share}_1}, Q_{\textsc{Acpt}_1}, Q_{\textsc{Comp}}, Q_{\textsc{RO}}),\\ Q_{\mathcal{B}} &= (Q_{\textsc{Reg}_1}, Q_{\textsc{Comp}}, Q_{\textsc{Auth}_1}, Q_{\textsc{Auth}_1}, Q_{\textsc{Reg}_1} + Q_{\textsc{RO}}, Q_{\textsc{RO}}),\\ Q_{\mathcal{C}_1} &= (Q_{\textsc{Reg}_1}, Q_{\textsc{Comp}}, Q_{\textsc{Reg}_1}, Q_{\textsc{Auth}_1}),\\ Q_{\mathcal{D}_1} &= (Q_{\textsc{Reg}_1}, Q_{\textsc{Comp}}, 0, Q_{\textsc{Reg}_1}, Q_{\textsc{Reg}_1} + Q_{\textsc{RO}}),\\ Q_{\mathcal{C}_2} &= (Q_{\textsc{Reg}_1}, Q_{\textsc{Comp}}, Q_{\textsc{Put}_1} + Q_{\textsc{Acpt}_1}, Q_{\textsc{Upd}_1} + Q_{\textsc{Get}_1} + Q_{\textsc{Share}_1}),\\ Q_{\mathcal{D}_2} &= (Q_{\textsc{Reg}_1}, Q_{\textsc{Comp}}, Q_{\textsc{Acpt}_1}, Q_{\textsc{Put}_1} + Q_{\textsc{Acpt}_1}, Q_{\textsc{Put}_1} + Q_{\textsc{Upd}_1} + Q_{\textsc{Get}_1} + Q_{\textsc{Share}_1} + Q_{\textsc{RO}}),\\ Q_{\mathcal{E}} &= (Q_{\textsc{Put}_1}, Q_{\textsc{Comp}}, Q_{\textsc{Put}_1} + Q_{\textsc{Upd}_1}, Q_{\textsc{Get}_1}).\end{aligned}$$

Proof. We prove the theorem by a sequence of games.

Game G_0. This is the integrity game in which we plug in construction CS. As in the proof of Theorem 10, we distinguish two random oracles. One can be called by the adversary, which is denoted by RO as usual, and one is an internal random oracle, denoted by H, which is only called from other oracles. The change is conceptual since both oracles represent the same function and maintain the same table for lazy sampling. Additionally, we introduce a table $\mathsf{T}_{\mathsf{H\text{-}int}}$ which indicates if a value was queried to the internal RO; this will be used in later changes. It holds that

$$\Pr[\mathsf{G}_0(\mathcal{A}) \Rightarrow 1] = \mathsf{Adv}^{Q_{\mathcal{A}}\text{-}\mathsf{Int}}_{\mathsf{CS},\mathcal{PW}}(\mathcal{A}),$$

where $Q_{\mathcal{A}}$ is the same as in the theorem statement.

Game G_1. This is game G_0 with the same changes as in Game G_7 from the proof of Theorem 10 as depicted in Figure 36. In particular, the game aborts if there are any direct random oracle queries corresponding to file keys which are only owned by uncompromised users and include the correct seed, i.e. the seed to extract the file key. This means that at this point all master keys of uncompromised users are uniformly random. The same holds for file keys which are exclusively owned by uncompromised users. Since the changes are

exactly the same as in the proof of Theorem 10, we obtain the same bounds:

$$\begin{aligned}\Pr[\mathsf{G}_0(\mathcal{A}) \Rightarrow 1] - \Pr[\mathsf{G}_1(\mathcal{A}) \Rightarrow 1] \leq \Big(&\mathsf{Adv}^{Q_{\mathcal{B}}\text{-INS-OW}}_{\mathsf{Auth},\mathcal{PW}}(\mathcal{B}) + \mathsf{Adv}^{Q_{\mathcal{C}_1}\text{-INT-CTXT}}_{\mathsf{AEAD}}(\mathcal{C}_1) \\ &+ \mathsf{Adv}^{Q_{\mathcal{D}_1}\text{-OW-CPA}}_{\mathsf{AEAD},\mathsf{kl}}(\mathcal{D}_1) + \mathsf{Adv}^{Q_{\mathcal{C}_2}\text{-INT-CTXT}}_{\mathsf{AEAD}}(\mathcal{C}_2) + \mathsf{Adv}^{Q_{\mathcal{D}_2}\text{-OW-CPA}}_{\mathsf{AEAD},\mathsf{kl}}(\mathcal{D}_2)\Big) \\ &+ \; Q_{\textsc{Auth}_1} \cdot \delta_{\mathsf{Auth}} + \frac{(Q_{\textsc{Put}_1} + Q_{\textsc{Acpt}_1})^2 + (Q_{\textsc{Upd}_1} + 1)^2}{2^{\mathsf{nl}+1}}\end{aligned}$$

with number of queries denoted in the theorem bound.

Reduction to INT-CTXT of AEAD.

Claim. There exists an adversary $\mathcal{E}$ against INT-CTXT such that

$$\Pr[\mathsf{G}_1(\mathcal{A}) \Rightarrow 1] \leq \mathsf{Adv}^{Q\text{-INT-CTXT}}_{\mathsf{AEAD}}(\mathcal{E})$$

with $Q = (Q_{\textsc{Put}_1}, Q_{\textsc{Comp}}, Q_{\textsc{Put}_1} + Q_{\textsc{Upd}_1}, Q_{\textsc{Check}})$.

Proof (Claim). The reduction is formalized in Figure 40. Note that due to the changes in the previous game, all file keys corresponding to files solely owned by uncompromised users are uniformly random. Hence, the reduction can use their own oracles to simulate encryption queries with respect to these keys. These occur in oracle $\textsc{Put}_1$ and $\textsc{Upd}_1$. While $\mathcal{E}$ chooses a new instance for a $\textsc{Put}_1$ query since the file key is generated in that query, an old instance is used in oracle $\textsc{Upd}_1$. All instances are stored together with their seed seed_k and their id fid in table $\mathsf{T}_{\mathsf{fid}}$ to be recognized later. Further, every file encryption is stored in table $\mathsf{T}_{\mathsf{ct\text{-}f}}$, to be able to completely answer get queries; more specifically the AEAD decryption in the second step of the get procedure (Line 71). If this is not the case, i.e. the file is new, the reduction outputs a solution if the queried user is uncompromised, the file is only owned by uncompromised users and the file ciphertext is valid, i.e. does not decrypt to $\perp$. Note that these conditions imply the winning conditions of adversary $\mathcal{A}$: the only condition that is not syntactically the same is $\mathsf{file} \notin \mathsf{T}_\mathsf{f}[\mathsf{fid}].\mathsf{F}$. Analyzing this condition, we can observe two things. First, in case $\mathsf{T}_\mathsf{f}[\mathsf{fid}].\mathsf{F}$ is empty, the file id is not tracked by the game and hence the corresponding file key was never used. This means that there cannot be a matching entry in table $\mathsf{T}_{\mathsf{ct\text{-}k}}$ which means that the seed seed_k was set to $\perp$ only yielding a $\perp$ output for the oracle which does not fulfill $\mathcal{A}$'s winning condition (or the $\textsc{Check}$ of $\mathcal{E}$). Second, if the challenge file is in $\mathsf{T}_\mathsf{f}[\mathsf{fid}].\mathsf{F}$, then there must be a matching element in $\mathsf{T}_{\mathsf{ct\text{-}f}}$ and hence $\mathcal{E}$ does not reach this line but can just answer the oracle query instead.

To preserve consistent outputs of the random oracle it must be patched upon queries to file keys which correspond to files owned by at least one compromised user. To this end, the instance is restored from table $\mathsf{T}_{\mathsf{fid}}$ and $\mathcal{E}$ queries their own compromise oracle to obtain the key of that instance. The entry of the random oracle table is then patched accordingly (Line 88).

It remains to show that the output tuple of $\mathcal{E}$ is actually a valid solution for their INT-CTXT game. The decryption of the challenge is not $\perp$ due to the check via $\textsc{Check}$. The instance of the AEAD also must not be compromised which is ensured by the queried user not being compromised and the file not being owned by any compromised user because the compromise oracle of $\mathcal{E}$ is only queried if at least one compromised user owns the file. □

Collecting the probabilities completes the proof of Theorem 11. □

D Additional Auth Notion and Construction for External Adversaries

For completeness, we give a security notion which can be used in the external adversary model sketched by BDGHP and described above. We also provide a construction that is much simpler than 2HDH.

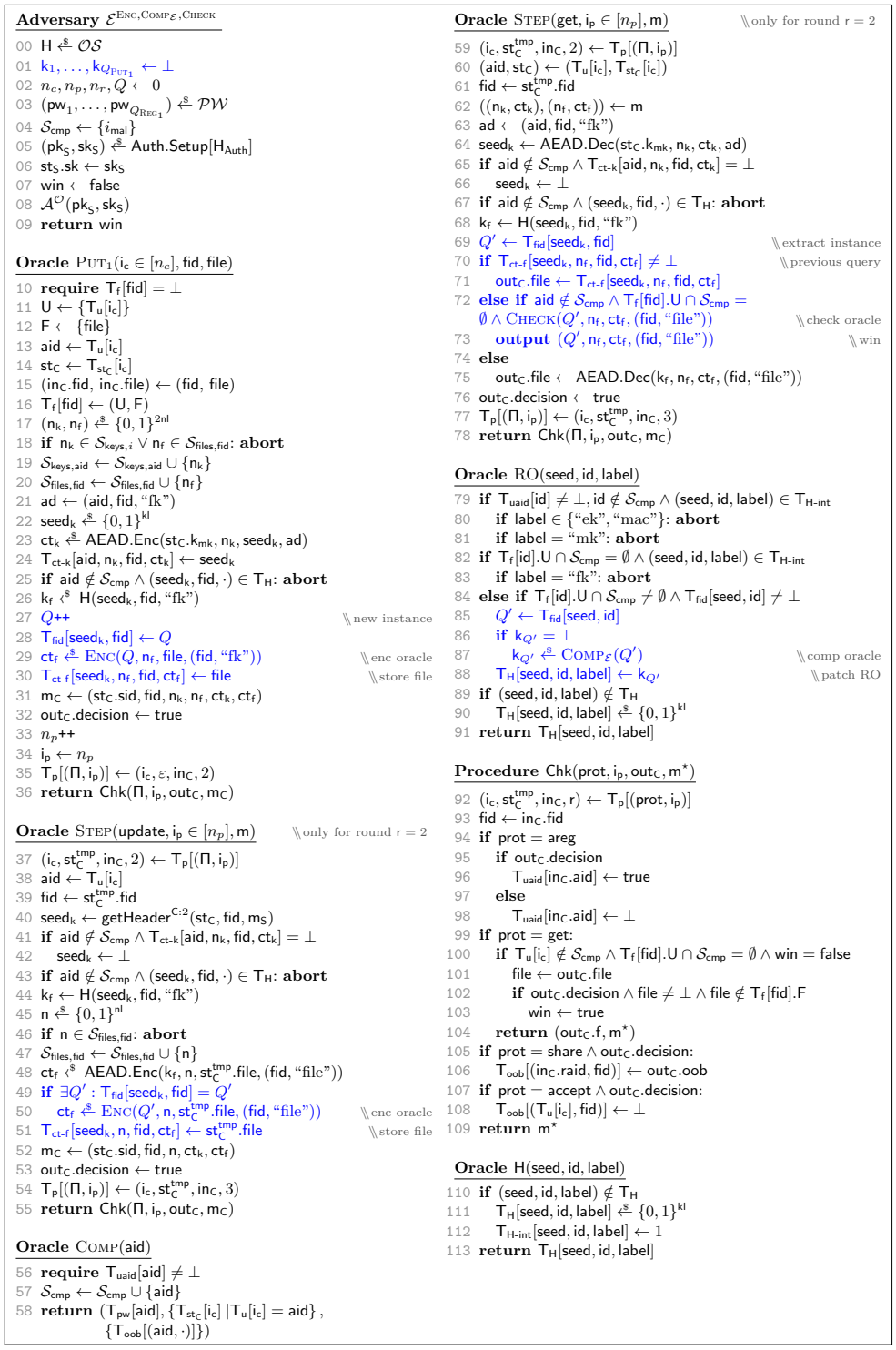

Fig. 40. Adversary $\mathcal{E}$ against INT-CTXT security of AEAD having access to oracles ENC, $\text{COMP}_\mathcal{E}$, and CHECK simulating G_1 for adversary $\mathcal{A}$.

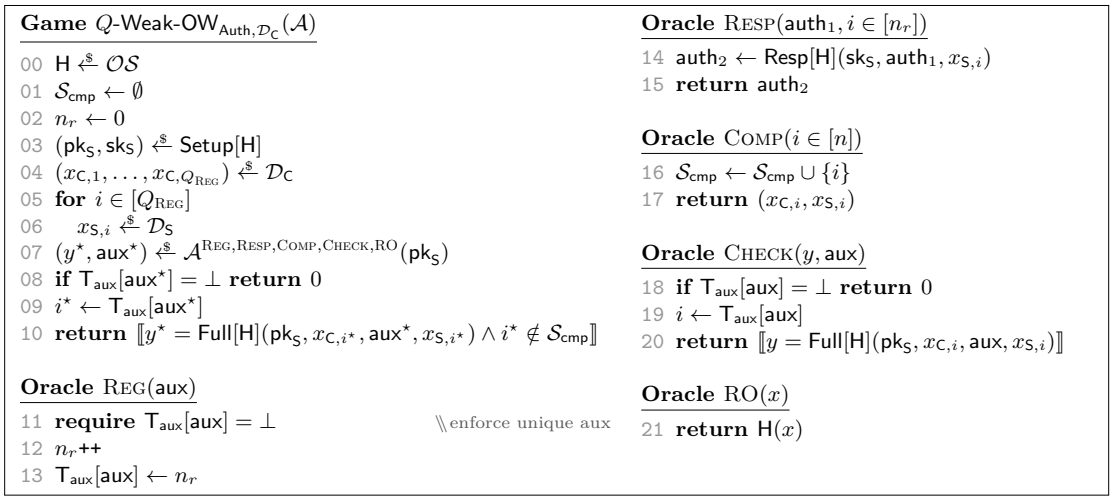

Fig. 41. Game defining Weak-OW security for a two-message authentication scheme $\mathsf{Auth} := (\mathcal{OS}, \mathcal{D}_\mathsf{S}, Y, \mathsf{Setup}, \mathsf{Init}, \mathsf{Resp}, \mathsf{Fin}, \mathsf{Full})$ for distribution $\mathcal{D}_\mathsf{C}$, adversary $\mathcal{A}$, and $Q = (Q_{\mathrm{REG}}, Q_{\mathrm{COMP}}, Q_{\mathrm{RESP}}, Q_{\mathrm{CHECK}}, Q_{\mathrm{RO}})$ making at most Q_O queries to oracle O.

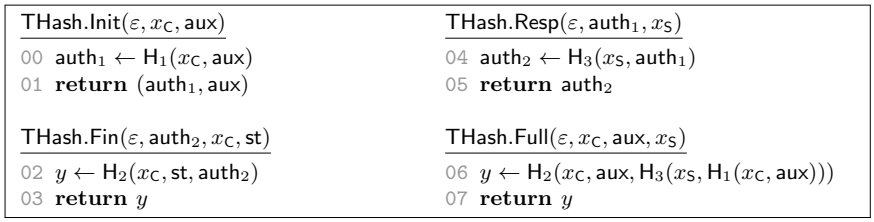

Fig. 42. The authentication protocol $\mathsf{THash} := (\mathcal{OS}, \mathcal{D}_\mathsf{S}, \{0,1\}^{\mathsf{kl}}, \varepsilon, \mathsf{Init}, \mathsf{Resp}, \mathsf{Fin}, \mathsf{Full})$ using three hash functions, where $\mathcal{D}_\mathsf{S}$ is an arbitrary distribution and there is no setup.

Definition 20 (Weak One-Wayness). *Output one-wayness for weak adversaries is defined via the game in Figure 41. We define the advantage of an adversary $\mathcal{A}$ and a client's secrets distribution $\mathcal{D}_\mathsf{C}$ as*

$$\mathsf{Adv}^{Q\text{-Weak-OW}}_{\mathsf{Auth},\mathcal{D}_\mathsf{C}}(\mathcal{A}) := \Pr\left[Q\text{-Weak-OW}_{\mathsf{Auth},\mathcal{D}_\mathsf{C}}(\mathcal{A}) \Rightarrow 1\right]$$

with $Q = (Q_{\mathrm{REG}}, Q_{\mathrm{COMP}}, Q_{\mathrm{INIT}}, Q_{\mathrm{FIN}}, Q_{\mathrm{CHECK}}, Q_{\mathrm{RO}})$.

TRIPLE HASH. In addition to the two authentication schemes from Section 3, we give a third instantiation, called THash in Figure 42. It achieves the same security bound for INS-OW as the other two instantiations and cannot improve its security bound for OUT-OW as in the case of 2HDH. We still state the construction because it achieves a better bound for Weak-OW (compared to SHash, see Table 2) which is needed for the external adversary model from [BDG+24] as discussed in Section 6.

E Theorems and Proofs from Section 6

E.1 Confidentiality

We restate Theorem 7 using concrete bounds and provide the full proof.

Theorem 12 (Net-Conf). *For any password distribution $\mathcal{PW}$ and adversary $\mathcal{A}$ against the* Net-Conf *security of* $\mathsf{CS} := \mathsf{CS}[\mathsf{Auth}, \mathsf{AEAD}, \mathsf{MAC}]$ *there exist an* OUT-OW *adversary $\mathcal{B}$ against* Auth, INT-CTXT *adversaries*

$\mathcal{C}_1, \mathcal{C}_2$ *against* AEAD, OW-CPA *adversaries* $\mathcal{D}_1, \mathcal{D}_2$ *against* AEAD, *and an* IND-CCA *adversary* $\mathcal{E}$ *against* AEAD *with* $t_{\mathcal{A}} \approx t_{\mathcal{B}} \approx t_{\mathcal{C}_1} \approx t_{\mathcal{D}_1} \approx t_{\mathcal{C}_2} - Q_{\text{COMP}} \cdot Q_{\text{AUTH}_1} \approx t_{\mathcal{D}_2} - Q_{\text{COMP}} \cdot Q_{\text{AUTH}_1} \approx t_{\mathcal{E}}$ *such that*

$$\begin{aligned}\mathsf{Adv}^{Q_{\mathcal{A}}\text{-Net-Conf}}_{\mathsf{CS},\mathcal{PW}}(\mathcal{A}) \leq 2 \cdot \Big(&\mathsf{Adv}^{Q_{\mathcal{B}}\text{-OUT-OW}}_{\mathsf{Auth},\mathcal{PW}}(\mathcal{B}) + \mathsf{Adv}^{Q_{\mathcal{C}_1}\text{-INT-CTXT}}_{\mathsf{AEAD}}(\mathcal{C}_1) + \mathsf{Adv}^{Q_{\mathcal{D}_1}\text{-OW-CPA}}_{\mathsf{AEAD},\mathsf{kl}}(\mathcal{D}_1) \\ &+ \mathsf{Adv}^{Q_{\mathcal{C}_2}\text{-INT-CTXT}}_{\mathsf{AEAD}}(\mathcal{C}_2) + \mathsf{Adv}^{Q_{\mathcal{D}_2}\text{-OW-CPA}}_{\mathsf{AEAD},\mathsf{kl}}(\mathcal{D}_2)\Big) + \mathsf{Adv}^{Q_{\mathcal{E}}\text{-IND-CCA}}_{\mathsf{AEAD}}(\mathcal{E}) \\ &+ Q_{\text{AUTH}_1} \cdot \delta_{\mathsf{Auth}} + \frac{(Q_{\text{PUT}_1} + Q_{\text{ACPT}_1})^2 + (Q_{\text{UPD}_1} + 1)^2}{2^{\mathsf{nl}+1}}\end{aligned}$$

with

$$\begin{aligned}
Q_{\mathcal{A}} &= (Q_{\text{STEP}}, Q_{\text{REG}_1}, Q_{\text{AUTH}_1}, Q_{\text{PUT}_1}, Q_{\text{UPD}_1}, Q_{\text{GET}_1}, Q_{\text{SHARE}_1}, Q_{\text{ACPT}_1}, Q_{\text{CHALL}}, Q_{\text{COMP}}, Q_{\text{RO}}),\\
Q_{\mathcal{B}} &= (Q_{\text{REG}_1}, Q_{\text{COMP}}, Q_{\text{AUTH}_1}, Q_{\text{AUTH}_1}, Q_{\text{AUTH}_1}, Q_{\text{REG}_1} + Q_{\text{RO}}),\\
Q_{\mathcal{C}_1} &= (Q_{\text{REG}_1}, Q_{\text{COMP}}, Q_{\text{REG}_1}, Q_{\text{AUTH}_1}),\\
Q_{\mathcal{D}_1} &= (Q_{\text{REG}_1}, Q_{\text{COMP}}, 0, Q_{\text{REG}_1}, Q_{\text{REG}_1} + Q_{\text{RO}}),\\
Q_{\mathcal{C}_2} &= (Q_{\text{REG}_1}, Q_{\text{COMP}}, Q_{\text{PUT}_1} + Q_{\text{ACPT}_1} + Q_{\text{CHL}}, Q_{\text{UPD}_1} + Q_{\text{GET}_1} + Q_{\text{SHARE}_1} + Q_{\text{CHL}}),\\
Q_{\mathcal{D}_2} &= (Q_{\text{REG}_1}, Q_{\text{COMP}}, Q_{\text{ACPT}_1}, Q_{\text{PUT}_1} + Q_{\text{ACPT}_1} + Q_{\text{CHL}}, Q_{\text{PUT}_1} + Q_{\text{UPD}_1} + Q_{\text{GET}_1} + Q_{\text{SHARE}_1} + Q_{\text{CHL}} + Q_{\text{RO}}),\\
Q_{\mathcal{E}} &= (Q_{\text{PUT}_1} + Q_{\text{CHL}}, Q_{\text{PUT}_1} + Q_{\text{CHL}}, Q_{\text{PUT}_1} + Q_{\text{UPD}_1}, Q_{\text{GET}_1}, Q_{\text{CHL}}).
\end{aligned}$$

Proof. We prove the theorem by a sequence of games and depict only those oracles in which a change occurs, starting with Figure 43.

Game G_0. This is the Net-Conf game in which we plug in construction CS. As for the proof of Theorem 10, we distinguish two random oracles. One can be called by the adversary, which is denoted by RO as usual, and one is an internal random oracle, denoted by H, which is only called from other oracles. The change is conceptual since both oracles represent the same function and maintain the same table for lazy sampling. Additionally, we introduce a table $\mathsf{T}_{\mathsf{H\text{-}int}}$ which indicates if a value was queried to the internal RO; this will be used in later changes. It holds that

$$\Pr[\mathsf{G}_0(\mathcal{A}) \Rightarrow 1] = \mathsf{Adv}^{Q_{\mathcal{A}}\text{-Net-Conf}}_{\mathsf{CS},\mathcal{PW}}(\mathcal{A}),$$

where $Q_{\mathcal{A}}$ is the same as in the theorem statement.

Game G_1. This is the same game as the previous one except that in the second step of the authentication protocol it overrides the value rw with the output of a Fin call if the received value auth_2 is correct, i.e. is output by the response algorithm of Auth with the correct inputs. The changes are depicted in Figure 43.

Claim. It holds that

$$\Pr[\mathsf{G}_0(\mathcal{A}) \Rightarrow 1] - \Pr[\mathsf{G}_1(\mathcal{A}) \Rightarrow 1] \leq Q_{\text{AUTH}_1} \cdot \delta_{\mathsf{Auth}}.$$

Proof (Claim). The step is analogous to the proof of Theorem 10. □

Game G_2. This is the same game as the previous one except that it aborts if there exists a random oracle query on a registered and uncompromised user such that the same input was previously queried to the internal RO H and the tag is either "ek" or "mac". The game also aborts in the registration oracle if the queried user is not compromised and there already exists a random oracle query on the same rw and aid. The changes are depicted in Figure 43.

Claim. There exists an adversary $\mathcal{B}$ against OUT-OW such that

$$\Pr[\mathsf{G}_1(\mathcal{A}) \Rightarrow 1] - \Pr[\mathsf{G}_2(\mathcal{A}) \Rightarrow 1] \leq \mathsf{Adv}^{Q\text{-OUT-OW}}_{\mathsf{Auth},\mathcal{PW}}(\mathcal{B})$$

with $Q = (Q_{\text{REG}_1}, Q_{\text{COMP}}, Q_{\text{AUTH}_1}, Q_{\text{AUTH}_1}, Q_{\text{AUTH}_1}, Q_{\text{REG}_1} + Q_{\text{RO}})$.

Oracle $\text{REG}_1(\mathsf{aid}, \mathsf{pk_S})$

00 **require** $\mathsf{T_{uaid}}[\mathsf{aid}] = \bot$
01 n_r++
02 $\mathsf{in_C.pw} \leftarrow \mathsf{pw}_{n_r}$
03 $\mathsf{T_{pw}}[\mathsf{aid}] \leftarrow \mathsf{in_C.pw}$
04 $\mathsf{in_C.aid} \leftarrow \mathsf{aid}$
05 $\mathsf{T_{uaid}}[\mathsf{aid}] \leftarrow \mathsf{false}$
06 $\mathsf{in_C.pk} \leftarrow \mathsf{pk_S}$
07 $\mathsf{k} \xleftarrow{\$} \mathsf{Auth}.\mathcal{D}_\mathsf{S}$
08 $\mathsf{rw} \leftarrow \mathsf{Auth.Full}[\mathsf{H_{Auth}}](\mathsf{pk_S}, \mathsf{pw}, \mathsf{aid}, \mathsf{k})$
09 **if** $\mathsf{aid} \notin \mathcal{S}_{\mathsf{cmp}} \wedge (\mathsf{rw}, \mathsf{aid}, \cdot) \in \mathsf{T_H}$ \\ G_2
10 **abort** \\ G_2
11 $\mathsf{k_{ek}} \leftarrow \mathsf{H}(\mathsf{rw}, \mathsf{aid}, \text{“ek”})$
12 $\mathsf{k_{mac}} \leftarrow \mathsf{H}(\mathsf{rw}, \mathsf{aid}, \text{“mac”})$
13 $\mathsf{seed_{mk}} \xleftarrow{\$} \{0,1\}^{\mathsf{kl}}$
14 $\mathsf{k_{mk}} \leftarrow \mathsf{H}(\mathsf{seed_{mk}}, \mathsf{aid}, \text{“mk”})$
15 $\mathsf{n_{mk}} \xleftarrow{\$} \{0,1\}^{\mathsf{nl}}$
16 $\mathsf{ct_{mk}} \xleftarrow{\$} \mathsf{AEAD.Enc}(\mathsf{k_{ek}}, \mathsf{n_{mk}}, \mathsf{seed_{mk}}, (\mathsf{aid}, \text{“mk”}))$
17 $\mathsf{m_C} \leftarrow (\mathsf{aid}, \mathsf{k}, \mathsf{n_{mk}}, \mathsf{ct_{mk}}, \mathsf{k_{mac}})$
18 n_p++
19 $\mathsf{i_p} \leftarrow n_p$
20 $\mathsf{T_p}[(\Pi, \mathsf{C}, \mathsf{i_p})] \leftarrow (i, \varepsilon, \mathsf{in_C}, 2)$
21 $\mathsf{T_k}[\mathsf{aid}] \leftarrow \mathsf{k}$ \\ G_1-G_2
22 **return** $\mathsf{Chk}(\Pi, \mathsf{i_p}, \mathsf{out_C}, \mathsf{m_C})$

Oracle $\text{COMP}(\mathsf{aid})$

23 **require** $\mathsf{T_{uaid}}[\mathsf{aid}] \neq \bot$
24 $\mathcal{S}_{\mathsf{cmp}} \leftarrow \mathcal{S}_{\mathsf{cmp}} \cup \{\mathsf{aid}\}$
25 **return** $(\mathsf{T_{pw}}[\mathsf{aid}], \{\mathsf{T_{st_C}}[\mathsf{i_c}] \mid \mathsf{T_u}[\mathsf{i_c}] = \mathsf{aid}\},$
$\{\mathsf{T_{oob}}[(\mathsf{aid}, \cdot)]\})$

Oracle $\text{STEP}(\mathsf{auth}, \mathsf{S}, \mathsf{i_p}, \mathsf{m})$ \\ only for server round $r = 1$

26 $(\mathsf{i_c}, \mathsf{st}^{\mathsf{tmp}}, \mathsf{in}, 1) \leftarrow \mathsf{T_p}[(\mathsf{prot}, \mathsf{S}, \mathsf{i_p})]$
27 $(\mathsf{aid}, \mathsf{auth}_1) \leftarrow \mathsf{m}$
28 **if** $\mathsf{aid} \notin \mathsf{st_S.acc}$
29 $\mathsf{fail_S}$
30 $\mathsf{sid} \xleftarrow{\$} \mathsf{NewID}$
31 $(\mathsf{k}, \ldots, \mathsf{k_{mac}}) \leftarrow \mathsf{st_S.acc}[\mathsf{aid}]$
32 $\mathsf{sk_S} \leftarrow \mathsf{st_C.sk}$
33 $\mathsf{auth}_2 \xleftarrow{\$} \mathsf{Auth.Resp}[\mathsf{H_{Auth}}](\mathsf{sk_S}, \mathsf{auth}_1, \mathsf{k})$
34 $\mathsf{m_S} \leftarrow (\mathsf{sid}, \mathsf{auth}_2)$
35 $\mathsf{st}^{\mathsf{tmp}}_{\mathsf{S}} \leftarrow (\mathsf{aid}, \mathsf{k_{mac}}, \mathsf{m}, \mathsf{m_S})$
36 $\mathsf{T_p}[(\mathsf{auth}, \mathsf{S}, \mathsf{i_p})] \leftarrow (\mathsf{i_c}, \mathsf{st}^{\mathsf{tmp}}, \mathsf{in}, 2)$
37 **return** $\mathsf{m_S}$

Oracle $\text{AUTH}_1(\mathsf{aid}, \mathsf{pk_S})$

38 **require** $\mathsf{T_{uaid}}[\mathsf{aid}] = \mathsf{true}$
39 $\mathsf{in_C.aid} \leftarrow \mathsf{aid}$
40 $\mathsf{in_C.pw} \leftarrow \mathsf{T_{pw}}[\mathsf{aid}]$
41 $\mathsf{in_C.pk} \leftarrow \mathsf{pk_S}$
42 n_c++
43 $\mathsf{i_c} \leftarrow n_c$
44 $\mathsf{T_u}[\mathsf{i_c}] \leftarrow \mathsf{aid}$
45 $\mathsf{T_{st_C}}[\mathsf{i_c}] \leftarrow \varepsilon$
46 $(\mathsf{auth}_1, \mathsf{st}) \xleftarrow{\$} \mathsf{Auth.Init}[\mathsf{H_{Auth}}](\mathsf{pk_S}, \mathsf{in_C.pw}, \mathsf{aid})$
47 $\mathsf{m_C} \leftarrow (\mathsf{aid}, \mathsf{auth}_1)$
48 $\mathsf{st}^{\mathsf{tmp}}_{\mathsf{C}} \leftarrow (\mathsf{aid}, \mathsf{in_C.pw}, \mathsf{pk_S}, \mathsf{st}, \mathsf{m_C})$
49 n_p++
50 $\mathsf{i_p} \leftarrow n_p$
51 $\mathsf{T_p}[(\mathsf{auth}, \mathsf{C}, \mathsf{i_p})] \leftarrow (\mathsf{i_c}, \mathsf{st}^{\mathsf{tmp}}_{\mathsf{C}}, \mathsf{in_C}, 2)$
52 **return** $\mathsf{Chk}(\mathsf{prot}, \mathsf{i_p}, \varepsilon, \mathsf{m_C})$

Oracle $\text{STEP}(\mathsf{auth}, \mathsf{C}, \mathsf{i_p}, \mathsf{m})$ \\ only for client round $r = 2$

53 $(\mathsf{i_c}, \mathsf{st}^{\mathsf{tmp}}_{\mathsf{C}}, \mathsf{in_C}, 2) \leftarrow \mathsf{T_p}[(\Pi, \mathsf{C}, \mathsf{i_p})]$
54 $(\mathsf{aid}, \mathsf{pw}, \mathsf{pk_S}, \mathsf{st}, \mathsf{m}'_\mathsf{C}) \leftarrow \mathsf{st}^{\mathsf{tmp}}_{\mathsf{C}}$
55 $(\cdot, \mathsf{auth}_1) \leftarrow \mathsf{m}'_\mathsf{C}$
56 $(\mathsf{sid}, \mathsf{auth}_2) \leftarrow \mathsf{m}$
57 $\mathsf{rw} \leftarrow \mathsf{Auth.Fin}[\mathsf{H_{Auth}}](\mathsf{pk_S}, \mathsf{auth}_2, \mathsf{pw}, \mathsf{aid}, \mathsf{st})$
58 **if** $\mathsf{auth}_2 = \mathsf{Auth.Resp}[\mathsf{H_{Auth}}](\mathsf{sk_S}, \mathsf{auth}_1, \mathsf{T_k}[\mathsf{aid}])$
59 $\mathsf{rw} \leftarrow \mathsf{Auth.Full}[\mathsf{H_{Auth}}](\mathsf{pk_S}, \mathsf{pw}, \mathsf{aid}, \mathsf{T_k}[\mathsf{aid}])$ \\ G_1-G_2
60 $\mathsf{st}^{\mathsf{tmp}}_{\mathsf{C}}.\mathsf{k_{ek}} \leftarrow \mathsf{H}(\mathsf{rw}, \mathsf{aid}, \text{“ek”})$
61 $\mathsf{k_{mac}} \leftarrow \mathsf{H}(\mathsf{rw}, \mathsf{aid}, \text{“mac”})$
62 $\mathsf{m_C} \leftarrow \mathsf{MAC.Tag}(\mathsf{k_{mac}}, (\mathsf{m}'_\mathsf{C}, \mathsf{m_S}))$
63 $\mathsf{T_p}[(\Pi, \mathsf{C}, \mathsf{i_p})] \leftarrow (\mathsf{i_c}, \mathsf{st}^{\mathsf{tmp}}_{\mathsf{C}}, \mathsf{in_C}, 3)$
64 **return** $\mathsf{Chk}(\Pi, \mathsf{i_p}, \varepsilon, \mathsf{m_C})$

Oracle $\mathsf{H}(\mathsf{seed}, \mathsf{aid}, \mathsf{label})$

65 **if** $(\mathsf{seed}, \mathsf{aid}, \mathsf{label}) \notin \mathsf{T_H}$
66 $\mathsf{T_H}[\mathsf{seed}, \mathsf{aid}, \mathsf{label}] \xleftarrow{\$} \{0,1\}^{\mathsf{kl}}$
67 $\mathsf{T_{H\text{-}int}}[\mathsf{seed}, \mathsf{aid}, \mathsf{label}] \leftarrow 1$
68 **return** $\mathsf{T_H}[\mathsf{seed}, \mathsf{aid}, \mathsf{label}]$

Oracle $\text{RO}(\mathsf{seed}, \mathsf{id}, \mathsf{label})$

69 **if** $\mathsf{T_{uaid}}[\mathsf{id}] \neq \bot \wedge \mathsf{id} \notin \mathcal{S}_{\mathsf{cmp}} \wedge (\mathsf{seed}, \mathsf{id}, \mathsf{label}) \in \mathsf{T_{H\text{-}int}}$
70 **if** $\mathsf{label} \in \{\text{“ek”}, \text{“mac”}\}$ **abort** \\ G_2
71 **if** $(\mathsf{seed}, \mathsf{id}, \mathsf{label}) \notin \mathsf{T_H}$
72 $\mathsf{T_H}[\mathsf{seed}, \mathsf{id}, \mathsf{label}] \xleftarrow{\$} \{0,1\}^{\mathsf{kl}}$
73 **return** $\mathsf{T_H}[\mathsf{seed}, \mathsf{id}, \mathsf{label}]$

Fig. 43. Games $\mathsf{G}_0 - \mathsf{G}_2$ for the proof of Theorem 12.

Proof (Claim). The reduction follows a similar approach as the reduction to Auth in the proof of Theorem 10. The differences are as follows. $\mathcal{B}$ needs to simulate a server oracle for the authentication sub protocol which is done via $\mathcal{B}$'s response oracle RESP. Further, the reduction cannot make use of the server secret because compared to INS-OW, $\mathcal{B}$ does not have access to these values in the OUT-OW game for uncompromised users. Since $\mathcal{B}$ can rely on the server response oracle, they do not need the values until a user is compromised. Upon compromise, the server secret k is known to the reduction (Line 31) which is necessary to patch the random oracle correctly. In particular, if the RO is queried on a compromised user, the reduction can check if it is the correct value by executing Auth.Full (Line 81) which is only possible knowing client and server secret. If this is the case, the oracle can be patched on the previously sampled value stored in $\mathsf{T_{H\text{-}int}}$. As shown in the proof of Theorem 10, the simulation is sound and in all cases in which the newly introduced

Adversary $\mathcal{B}^{\text{REG},\text{INIT},\text{RESP},\text{FIN},\text{COMP}_{\mathcal{B}},\text{CHECK}}(\mathsf{pk_S})$

00 $\mathsf{k}_1, \ldots, \mathsf{k}_{Q_{\text{REG}_1}} \leftarrow \bot$ \\ not known
01 $n_c, n_p, n_r \leftarrow 0$
02 $\mathcal{S}_{\mathsf{cmp}}, \mathcal{S}_{\mathsf{ch}}, \mathcal{S}_{\mathsf{chOwn}} \leftarrow \emptyset$
03 $\mathsf{b} \xleftarrow{\$} \{0,1\}$
04 $\mathsf{b}' \leftarrow \mathcal{A}^{\text{O}}(\mathsf{pk_S})$
05 **if** $\exists \mathsf{aid} \in \mathcal{S}_{\mathsf{chOwn}} : \mathsf{aid} \in \mathcal{S}_{\mathsf{cmp}} \vee \mathsf{T_{uaid}}[\mathsf{aid}] = \bot$
06 **return** 0
07 **return** $[\![\mathsf{b}' = \mathsf{b}]\!]$

Oracle $\text{REG}_1(\mathsf{aid})$

08 **require** $\mathsf{T_{uaid}}[\mathsf{aid}] = \bot$
09 n_r++
10 $\mathsf{T_{pw}}[\mathsf{aid}] \leftarrow (\bot, n_r)$
11 $\mathsf{in_C}.\mathsf{aid} \leftarrow \mathsf{aid}$
12 $\mathsf{T_{uaid}}[\mathsf{aid}] \leftarrow \mathsf{false}$
13 $\text{REG}(\mathsf{aid})$ \\ register client
14 **if** $\exists \mathsf{rw} : (\mathsf{rw}, \mathsf{aid}, \cdot) \in \mathsf{T_H} \wedge \text{CHECK}(\mathsf{rw}, \mathsf{aid})$
15 **output** $(\mathsf{rw}, \mathsf{aid})$ \\ win
16 $\mathsf{rw} \leftarrow$ "chl" \\ mark internal RO query
17 $\mathsf{k_{ek}} \leftarrow \mathsf{H}(\mathsf{rw}, \mathsf{aid}, \text{"ek"})$
18 $\mathsf{k_{mac}} \leftarrow \mathsf{H}(\mathsf{rw}, \mathsf{aid}, \text{"mac"})$
19 $\mathsf{seed_{mk}} \xleftarrow{\$} \{0,1\}^{\mathsf{kl}}$
20 $\mathsf{k_{mk}} \leftarrow \mathsf{H}(\mathsf{seed_{mk}}, \mathsf{aid}, \text{"mk"})$
21 $\mathsf{n_{mk}} \xleftarrow{\$} \{0,1\}^{\mathsf{nl}}$
22 $\mathsf{ct_{mk}} \xleftarrow{\$} \mathsf{AEAD.Enc}(\mathsf{k_{ek}}, \mathsf{n_{mk}}, \mathsf{seed_{mk}}, (\mathsf{aid}, \text{"mk"}))$
23 $\mathsf{m_C} \leftarrow (\mathsf{aid}, \mathsf{k}, \mathsf{n_{mk}}, \mathsf{ct_{mk}}, \mathsf{k_{mac}})$
24 n_p++
25 $\mathsf{i_p} \leftarrow n_p$
26 $\mathsf{T_p}[(\Pi, \mathsf{C}, \mathsf{i_p})] \leftarrow (i, \varepsilon, \mathsf{in_C}, 2)$
27 **return** $\mathsf{Chk}(\Pi, \mathsf{i_p}, \mathsf{out_C}, \mathsf{m_C})$

Oracle $\text{COMP}(\mathsf{aid})$

28 **require** $\mathsf{T_{uaid}}[\mathsf{aid}] \neq \bot$
29 $\mathcal{S}_{\mathsf{cmp}} \leftarrow \mathcal{S}_{\mathsf{cmp}} \cup \{\mathsf{aid}\}$
30 $(\cdot, n'_r) \leftarrow \mathsf{T_{pw}}[\mathsf{aid}]$
31 $(\mathsf{pw}, \mathsf{k}_{n'_r}) \xleftarrow{\$} \text{COMP}_{\mathcal{B}}(n'_r)$ \\ comp oracle
32 $\mathsf{T_{pw}}[\mathsf{aid}] \leftarrow (\mathsf{pw}, n'_r)$ \\ store pw to patch
33 **return** $(\mathsf{pw}, \{\mathsf{T_{st_C}}[\mathsf{i_c}] \mid \mathsf{T_u}[\mathsf{i_c}] = \mathsf{aid}\}, \{\mathsf{T_{oob}}[(\mathsf{aid}, \cdot)]\})$

Oracle $\text{STEP}(\mathsf{auth}, \mathsf{S}, \mathsf{i_p}, \mathsf{m})$ \\ only for server round $\mathsf{r} = 1$

34 $(\mathsf{i_c}, \mathsf{st}^{\mathsf{tmp}}, \mathsf{in}, 1) \leftarrow \mathsf{T_p}[(\mathsf{prot}, \mathsf{S}, \mathsf{i_p})]$
35 $(\mathsf{aid}, \mathsf{auth}_1) \leftarrow \mathsf{m}$
36 **if** $\mathsf{aid} \notin \mathsf{st_S}.\mathsf{acc}$: $\mathsf{fail_S}$
37 $\mathsf{sid} \xleftarrow{\$} \mathsf{NewID}$
38 $(\mathsf{k}, \ldots, \mathsf{k_{mac}}) \leftarrow \mathsf{st_S}.\mathsf{acc}[\mathsf{aid}]$
39 $(\cdot, n'_r) \leftarrow \mathsf{T_{pw}}[\mathsf{aid}]$ \\ recover instance
40 $\mathsf{auth}_2 \xleftarrow{\$} \text{RESP}(\mathsf{auth}_1, n'_r)$ \\ resp oracle
41 $\mathsf{m_S} \leftarrow (\mathsf{sid}, \mathsf{auth}_2)$
42 $\mathsf{st}^{\mathsf{tmp}}_{\mathsf{S}} \leftarrow (\mathsf{aid}, \mathsf{k_{mac}}, \mathsf{m}, \mathsf{m_S})$
43 $\mathsf{T_p}[(\mathsf{auth}, \mathsf{S}, \mathsf{i_p})] \leftarrow (\mathsf{i_c}, \mathsf{st}^{\mathsf{tmp}}, \mathsf{in}, 2)$
44 **return** $\mathsf{m_S}$

Oracle $\text{AUTH}_1(\mathsf{aid})$

45 **require** $\mathsf{T_{uaid}}[\mathsf{aid}] = \mathsf{true}$
46 $\mathsf{in_C}.\mathsf{aid} \leftarrow \mathsf{aid}$
47 $\mathsf{in_C}.\mathsf{pw} \leftarrow \mathsf{T_{pw}}[\mathsf{aid}]$
48 n_c++
49 $\mathsf{i_c} \leftarrow n_c$
50 n_p++
51 $\mathsf{i_p} \leftarrow n_p$
52 $\mathsf{T_u}[\mathsf{i_c}] \leftarrow \mathsf{aid}$
53 $\mathsf{T_{st_C}}[\mathsf{i_c}] \leftarrow \varepsilon$
54 $\mathsf{auth}_1 \xleftarrow{\$} \text{INIT}(\mathsf{aid}, \mathsf{i_p})$ \\ init oracle
55 $\mathsf{m_C} \leftarrow (\mathsf{aid}, \mathsf{auth}_1)$
56 $\mathsf{st}^{\mathsf{tmp}}_{\mathsf{C}} \leftarrow (\mathsf{aid}, \mathsf{in_C}.\mathsf{pw}, \mathsf{st}, \mathsf{m_C})$
57 $\mathsf{T_p}[(\mathsf{auth}, \mathsf{C}, \mathsf{i_p})] \leftarrow (\mathsf{i_c}, \mathsf{st}^{\mathsf{tmp}}_{\mathsf{C}}, \mathsf{in_C}, 2)$
58 **return** $\mathsf{Chk}(\mathsf{prot}, \mathsf{i_p}, \varepsilon, \mathsf{m_C})$

Oracle $\text{STEP}(\mathsf{auth}, \mathsf{C}, \mathsf{i_p}, \mathsf{m})$ \\ only for client round $\mathsf{r} = 2$

59 $(\mathsf{i_c}, \mathsf{st}^{\mathsf{tmp}}_{\mathsf{C}}, \mathsf{in_C}, 2) \leftarrow \mathsf{T_p}[(\mathsf{auth}, \mathsf{C}, \mathsf{i_p})]$
60 $(\mathsf{aid}, \mathsf{pw}, \mathsf{st}, \mathsf{m}'_{\mathsf{C}}) \leftarrow \mathsf{st}^{\mathsf{tmp}}_{\mathsf{C}}$
61 $(\cdot, \mathsf{auth}_1) \leftarrow \mathsf{m}'_{\mathsf{C}}$
62 $(\mathsf{sid}, \mathsf{auth}_2) \leftarrow \mathsf{m}$
63 $(\cdot, n'_r) \leftarrow \mathsf{T_{pw}}[\mathsf{aid}]$ \\ recover instance
64 $\mathsf{rw} \xleftarrow{\$} \text{FIN}(\mathsf{auth}_2, n'_r, \mathsf{i_p})$ \\ fin oracle
65 **if** $\mathsf{rw} = \top$ \\ correct execution detected
66 $\mathsf{rw} \leftarrow$ "chl" \\ use consistent RO output
67 $\mathsf{st}^{\mathsf{tmp}}_{\mathsf{C}}.\mathsf{k_{ek}} \leftarrow \mathsf{H}(\mathsf{rw}, \mathsf{aid}, \text{"ek"})$
68 $\mathsf{k_{mac}} \leftarrow \mathsf{H}(\mathsf{rw}, \mathsf{aid}, \text{"mac"})$
69 $\mathsf{m_C} \leftarrow \mathsf{MAC.Tag}(\mathsf{k_{mac}}, (\mathsf{m}'_{\mathsf{C}}, \mathsf{m_S}))$
70 $\mathsf{T_p}[(\mathsf{auth}, \mathsf{C}, \mathsf{i_p})] \leftarrow (\mathsf{i_c}, \mathsf{st}^{\mathsf{tmp}}_{\mathsf{C}}, \mathsf{in_C}, 3)$
71 **return** $\mathsf{Chk}(\mathsf{auth}, \mathsf{i_p}, \varepsilon, \mathsf{m_C})$

Oracle $\mathsf{H}(\mathsf{seed}, \mathsf{aid}, \mathsf{label})$

72 **if** $(\mathsf{seed}, \mathsf{aid}, \mathsf{label}) \notin \mathsf{T_H}$
73 $\mathsf{T_H}[\mathsf{seed}, \mathsf{aid}, \mathsf{label}] \xleftarrow{\$} \{0,1\}^{\mathsf{kl}}$
74 $\mathsf{T_{H\text{-}int}}[\mathsf{seed}, \mathsf{aid}, \mathsf{label}] \leftarrow 1$
75 **return** $\mathsf{T_H}[\mathsf{seed}, \mathsf{aid}, \mathsf{label}]$

Oracle $\text{RO}(\mathsf{seed}, \mathsf{id}, \mathsf{label})$

76 **if** $\mathsf{T_{uaid}}[\mathsf{id}] \neq \bot \wedge (\text{"chl"}, \mathsf{id}, \mathsf{label}) \in \mathsf{T_{H\text{-}int}}$
77 **if** $\mathsf{id} \notin \mathcal{S}_{\mathsf{cmp}} \wedge \mathsf{label} \in \{\text{"ek"}, \text{"mac"}\}$
$\wedge \text{CHECK}(\mathsf{seed}, \mathsf{id}) \wedge (\text{"chl"}, \mathsf{id}, \mathsf{label}) \in \mathsf{T_{H\text{-}int}}$
78 **output** $(\mathsf{seed}, \mathsf{id})$ \\ win
79 **else if** $\mathsf{id} \in \mathcal{S}_{\mathsf{cmp}}$
80 $(\mathsf{pw}, n'_r) \leftarrow \mathsf{T_{pw}}[\mathsf{id}]$
81 **if** $\mathsf{seed} = \mathsf{Auth.Full}(\mathsf{pw}, \mathsf{id}, \mathsf{k}_{n'_r})$
82 $\mathsf{T_H}[\mathsf{seed}, \mathsf{id}, \mathsf{label}] \leftarrow \mathsf{T_{H\text{-}int}}[\text{"chl"}, \mathsf{id}, \mathsf{label}]$ \\ patch RO
83 **if** $\mathsf{T_H}[\mathsf{seed}, \mathsf{id}, \mathsf{label}] = \bot$
84 $\mathsf{T_H}[\mathsf{seed}, \mathsf{id}, \mathsf{label}] \xleftarrow{\$} \{0,1\}^{\mathsf{kl}}$
85 **return** $\mathsf{T_H}[\mathsf{seed}, \mathsf{id}, \mathsf{label}]$

Fig. 44. Adversary $\mathcal{B}$ against OUT-OW security of Auth having access to oracles REG, INIT, RESP, FIN, and CHECK simulating $\mathsf{G}_1/\mathsf{G}_2$ for adversary $\mathcal{A}$.

abort condition triggers, adversary $\mathcal{B}$ wins their OUT-OW game. For the number of queries, note that a step

of some procedure can only be executed if the previous step was executed and further this is only possible once, i.e. it is not possible to branch a protocol and execute the server step of the authentication twice for one client step. Hence, we can upper bound all authentication sub protocol steps by the number of queries to the initiation oracle $\textsc{Auth}_1$. □

Final reduction. Finally, we can apply the same changes as applied in the proof of Theorem 10 between G_2 and the last game and also apply the final reduction. Note that G_2 is exactly in the same state as G_2 of the proof of Theorem 10 except for the server oracles, i.e. queries to $\textsc{Step}$ with $\mathsf{P} = \mathsf{S}$. The server oracles can be simulated trivially because the server is not involved in any nonce sampling or AEAD operations but only stores and forwards ciphertexts. Hence, we can apply the same arguments and obtain the same bound:

$$\begin{aligned}2\,\mathsf{Pr}[\mathsf{G}_2(\mathcal{A}) \Rightarrow 1] - 1 \leq 2\Big(&\mathsf{Adv}^{Q_{\mathcal{C}_1}\text{-}\mathsf{INT\text{-}CTXT}}_{\mathsf{AEAD}}(\mathcal{C}_1) + \mathsf{Adv}^{Q_{\mathcal{D}_1}\text{-}\mathsf{OW\text{-}CPA}}_{\mathsf{AEAD},\mathsf{kl}}(\mathcal{D}_1)\\ &+ \mathsf{Adv}^{Q_{\mathcal{C}_2}\text{-}\mathsf{INT\text{-}CTXT}}_{\mathsf{AEAD}}(\mathcal{C}_2) + \mathsf{Adv}^{Q_{\mathcal{D}_2}\text{-}\mathsf{OW\text{-}CPA}}_{\mathsf{AEAD},\mathsf{kl}}(\mathcal{D}_2)\Big) + \mathsf{Adv}^{Q_{\mathcal{E}}\text{-}\mathsf{IND\text{-}CCA}}_{\mathsf{AEAD}}(\mathcal{E})\\ &+ \frac{(Q_{\textsc{Put}_1} + Q_{\textsc{Acpt}_1})^2 + (Q_{\textsc{Upd}_1} + 1)^2}{2^{\mathsf{nl}+1}}\end{aligned}$$

with

$$\begin{aligned}Q_{\mathcal{C}_1} &= (Q_{\textsc{Reg}_1}, Q_{\textsc{Comp}}, Q_{\textsc{Reg}_1}, Q_{\textsc{Auth}_1}),\\ Q_{\mathcal{D}_1} &= (Q_{\textsc{Reg}_1}, Q_{\textsc{Comp}}, 0, Q_{\textsc{Reg}_1}, Q_{\textsc{Reg}_1} + Q_{\textsc{RO}}),\\ Q_{\mathcal{C}_2} &= (Q_{\textsc{Reg}_1}, Q_{\textsc{Comp}}, Q_{\textsc{Put}_1} + Q_{\textsc{Acpt}_1} + Q_{\textsc{Chl}}, Q_{\textsc{Upd}_1} + Q_{\textsc{Get}_1} + Q_{\textsc{Share}_1} + Q_{\textsc{Chl}}),\\ Q_{\mathcal{D}_2} &= (Q_{\textsc{Reg}_1}, Q_{\textsc{Comp}}, Q_{\textsc{Acpt}_1}, Q_{\textsc{Put}_1} + Q_{\textsc{Acpt}_1} + Q_{\textsc{Chl}}, Q_{\textsc{Put}_1} + Q_{\textsc{Upd}_1} + Q_{\textsc{Get}_1} + Q_{\textsc{Share}_1} + Q_{\textsc{Chl}} + Q_{\textsc{RO}}),\\ Q_{\mathcal{E}} &= (Q_{\textsc{Put}_1} + Q_{\textsc{Chl}}, Q_{\textsc{Put}_1} + Q_{\textsc{Chl}}, Q_{\textsc{Put}_1} + Q_{\textsc{Upd}_1}, Q_{\textsc{Get}_1}, Q_{\textsc{Chl}}).\end{aligned}$$

Collecting the probabilities completes the proof of Theorem 12. □

E.2 Integrity

We restate Theorem 8 using concrete bounds. The proof follows from the combination of other theorems, as shown in Section 6.2.

Theorem 13 (Net-Int). *For any password distribution $\mathcal{PW}$ and adversary $\mathcal{A}$ against the* Net-Int *security of* $\mathsf{CS} \coloneqq \mathsf{CS}[\mathsf{Auth}, \mathsf{AEAD}, \mathsf{MAC}]$ *there exist an* OUT-OW *adversary $\mathcal{B}$ against* Auth, INT-CTXT *adversaries $\mathcal{C}_1, \mathcal{C}_2$ against* AEAD, OW-CPA *adversaries $\mathcal{D}_1, \mathcal{D}_2$ against* AEAD, *and an* INT-CTXT *adversary $\mathcal{E}$ against* AEAD *with $t_{\mathcal{A}} \approx t_{\mathcal{B}} \approx t_{\mathcal{C}_1} \approx t_{\mathcal{D}_1} \approx t_{\mathcal{C}_2} - Q_{\textsc{Comp}} \cdot Q_{\textsc{Auth}_1} \approx t_{\mathcal{D}_2} - Q_{\textsc{Comp}} \cdot Q_{\textsc{Auth}_1} \approx t_{\mathcal{E}}$ such that*

$$\begin{aligned}\mathsf{Adv}^{Q_{\mathcal{A}}\text{-}\mathsf{Net\text{-}Int}}_{\mathsf{CS},\mathcal{PW}}(\mathcal{A}) \leq\ &\mathsf{Adv}^{Q_{\mathcal{B}}\text{-}\mathsf{OUT\text{-}OW}}_{\mathsf{Auth},\mathcal{PW}}(\mathcal{B}) + \mathsf{Adv}^{Q_{\mathcal{C}_1}\text{-}\mathsf{INT\text{-}CTXT}}_{\mathsf{AEAD}}(\mathcal{C}_1) + \mathsf{Adv}^{Q_{\mathcal{D}_1}\text{-}\mathsf{OW\text{-}CPA}}_{\mathsf{AEAD},\mathsf{kl}}(\mathcal{D}_1)\\ &+ \mathsf{Adv}^{Q_{\mathcal{C}_2}\text{-}\mathsf{INT\text{-}CTXT}}_{\mathsf{AEAD}}(\mathcal{C}_2) + \mathsf{Adv}^{Q_{\mathcal{D}_2}\text{-}\mathsf{OW\text{-}CPA}}_{\mathsf{AEAD},\mathsf{kl}}(\mathcal{D}_2) + \mathsf{Adv}^{Q_{\mathcal{E}}\text{-}\mathsf{INT\text{-}CTXT}}_{\mathsf{AEAD}}(\mathcal{E})\\ &+ Q_{\textsc{Auth}_1} \cdot \delta_{\mathsf{Auth}} + \frac{(Q_{\textsc{Put}_1} + Q_{\textsc{Acpt}_1})^2 + (Q_{\textsc{Upd}_1} + 1)^2}{2^{\mathsf{nl}+1}}\end{aligned}$$

with

$$\begin{aligned}Q_{\mathcal{A}} &= (Q_{\textsc{Step}}, Q_{\textsc{StepS}_1}, Q_{\textsc{Reg}_1}, Q_{\textsc{Auth}_1}, Q_{\textsc{Put}_1}, Q_{\textsc{Upd}_1}, Q_{\textsc{Get}_1}, Q_{\textsc{Share}_1}, Q_{\textsc{Acpt}_1}, Q_{\textsc{Comp}}, Q_{\textsc{RO}}),\\ Q_{\mathcal{B}} &= (Q_{\textsc{Reg}_1}, Q_{\textsc{Comp}}, Q_{\textsc{Auth}_1}, Q_{\textsc{Auth}_1}, Q_{\textsc{Auth}_1}, Q_{\textsc{Reg}_1} + Q_{\textsc{RO}}),\\ Q_{\mathcal{C}_1} &= (Q_{\textsc{Reg}_1}, Q_{\textsc{Comp}}, Q_{\textsc{Reg}_1}, Q_{\textsc{Auth}_1}),\\ Q_{\mathcal{D}_1} &= (Q_{\textsc{Reg}_1}, Q_{\textsc{Comp}}, 0, Q_{\textsc{Reg}_1}, Q_{\textsc{Reg}_1} + Q_{\textsc{RO}}),\\ Q_{\mathcal{C}_2} &= (Q_{\textsc{Reg}_1}, Q_{\textsc{Comp}}, Q_{\textsc{Put}_1} + Q_{\textsc{Acpt}_1}, Q_{\textsc{Upd}_1} + Q_{\textsc{Get}_1} + Q_{\textsc{Share}_1}),\\ Q_{\mathcal{D}_2} &= (Q_{\textsc{Reg}_1}, Q_{\textsc{Comp}}, Q_{\textsc{Acpt}_1}, Q_{\textsc{Put}_1} + Q_{\textsc{Acpt}_1}, Q_{\textsc{Put}_1} + Q_{\textsc{Upd}_1} + Q_{\textsc{Get}_1} + Q_{\textsc{Share}_1} + Q_{\textsc{RO}}),\\ Q_{\mathcal{E}} &= (Q_{\textsc{Put}_1}, Q_{\textsc{Comp}}, Q_{\textsc{Put}_1} + Q_{\textsc{Upd}_1}, Q_{\textsc{Get}_1}).\end{aligned}$$

E.3 Explicit Authentication

We restate Theorem 9 using concrete bounds and provide the full proof.

Theorem 14 (Explicit Authentication). *For any password distribution $\mathcal{PW}$ and adversary $\mathcal{A}$ against the* Exp-Auth *security of* $\mathsf{CS} := \mathsf{CS}[\mathsf{Auth}, \mathsf{AEAD}, \mathsf{MAC}]$ *there exist an* OUT-OW *adversary $\mathcal{B}$ against* Auth *and an* SUF *adversary $\mathcal{C}$ against* MAC *with $t_{\mathcal{A}} \approx t_{\mathcal{B}} \approx t_{\mathcal{C}}$ such that*

$$\mathsf{Adv}^{Q_{\mathcal{A}}\text{-}\mathsf{Exp}\text{-}\mathsf{Auth}}_{\mathsf{CS},\mathcal{PW}}(\mathcal{A}) \leq \mathsf{Adv}^{Q_{\mathcal{B}}\text{-}\mathsf{OUT}\text{-}\mathsf{OW}}_{\mathsf{Auth},\mathcal{PW}}(\mathcal{B}) + \mathsf{Adv}^{Q_{\mathcal{C}}\text{-}\mathsf{SUF}}_{\mathsf{MAC}}(\mathcal{C}) + Q_{\textsc{Auth}_1} \cdot \delta_{\mathsf{Auth}}$$

with

$$\begin{aligned}
Q_{\mathcal{A}} &= (Q_{\textsc{Step}}, Q_{\textsc{StepS}_1}, Q_{\textsc{Reg}_1}, Q_{\textsc{Auth}_1}, Q_{\textsc{Put}_1}, Q_{\textsc{Upd}_1}, Q_{\textsc{Get}_1}, Q_{\textsc{Share}_1}, Q_{\textsc{Acpt}_1}, Q_{\textsc{Comp}}, Q_{\textsc{RO}}),\\
Q_{\mathcal{B}} &= (Q_{\textsc{Reg}_1}, Q_{\textsc{Comp}}, Q_{\textsc{Auth}_1}, Q_{\textsc{Auth}_1}, Q_{\textsc{Auth}_1}, Q_{\textsc{Reg}_1} + Q_{\textsc{RO}}),\\
Q_{\mathcal{C}} &= (Q_{\textsc{Reg}_1}, Q_{\textsc{Comp}}, Q_{\textsc{Auth}_1}, Q_{\textsc{Auth}_1}).
\end{aligned}$$

Proof. We prove the theorem by a sequence of games.

Game G_0. This is the explicit authentication game in which we plug in construction CS. As in the proof of Theorem 10, we distinguish two random oracles. One can be called by the adversary, which is denoted by RO as usual, and one is an internal random oracle, denoted by H, which is only called from other oracles. The change is conceptual since both oracles represent the same function and maintain the same table for lazy sampling. Additionally, we introduce a table $\mathsf{T}_{\mathsf{H\text{-}int}}$ which indicates if a value was queried to the internal RO; this will be used in later changes. It holds that

$$\mathsf{Pr}[\mathsf{G}_0(\mathcal{A}) \Rightarrow 1] = \mathsf{Adv}^{Q_{\mathcal{A}}\text{-}\mathsf{Exp}\text{-}\mathsf{Auth}}_{\mathsf{CS},\mathcal{PW}}(\mathcal{A}),$$

where $Q_{\mathcal{A}}$ is the same as in the theorem statement.

Game G_1. This is the same as game G_2 from the proof of Theorem 12 as depicted in Figure 43. In particular, the game aborts if there are any direct random oracle queries corresponding to master key or mac key of uncompromised users and include the correct raw secret rw. This means that at this point all encryption keys and mac keys of uncompromised users are uniformly random. Since the changes are exactly the same as in the proof of Theorem 12, we obtain the same bound:

$$\mathsf{Pr}\left[\mathsf{G}_0(\mathcal{A}) \Rightarrow 1\right] - \mathsf{Pr}\left[\mathsf{G}_1(\mathcal{A}) \Rightarrow 1\right] \leq \mathsf{Adv}^{Q_{\mathcal{B}}\text{-}\mathsf{OUT}\text{-}\mathsf{OW}}_{\mathsf{Auth},\mathcal{PW}}(\mathcal{B}) + Q_{\textsc{Auth}_1} \cdot \delta_{\mathsf{Auth}}$$

with number of queries denoted in the theorem bound.

Final reduction. We can reduce G_1 to the SUF of MAC.

Claim. There exists an adversary $\mathcal{C}$ against SUF such that

$$\mathsf{Pr}[\mathsf{G}_1(\mathcal{A}) \Rightarrow 1] \leq \mathsf{Adv}^{Q\text{-}\mathsf{SUF}}_{\mathsf{MAC}}(\mathcal{C}),$$

with $Q = (Q_{\textsc{Reg}_1}, Q_{\textsc{Comp}}, Q_{\textsc{Auth}_1}, Q_{\textsc{Auth}_1})$.

Proof (Claim). The formal reduction can be found in Figure 45 showing only the oracles that cannot trivially (by just executing as they are) be simulated. In the registration oracle, $\mathcal{C}$ starts a new instance for their SUF game and stores them for the queried aid. The output of the oracle is the length of the message which can be simulated since all values of the message are known to the reduction except for the MAC key $\mathsf{k}_{\mathsf{mac}}$ which has a fixed length of kl (Line 35). The server's side of the registration protocol is simple to simulate since the server does not output any value. The reduction can therefore store the relevant instance Q' in the server state representing the MAC key.

If the authentication protocol is queried for the second step of the client, the reduction can look up the MAC instance and compute the correct tag using their tag oracle (Line 61). This is only done for the

Adversary $\mathcal{C}^{\text{TAG},\text{VER},\text{COMP}_\mathcal{C}}$

00 $\mathsf{H} \xleftarrow{\$} \mathcal{OS}$
01 $(n_c, n_p, n_r) \leftarrow (0,0,0)$
02 $(\mathsf{pw}_1, \ldots, \mathsf{pw}_{Q_{\text{REG}_1}}) \xleftarrow{\$} \mathcal{PW}$
03 $\mathcal{S}_{\mathsf{cmp}} \leftarrow \emptyset$
04 $(\mathsf{pk}_\mathsf{S}, \mathsf{sk}_\mathsf{S}) \xleftarrow{\$} \mathsf{Setup}$
05 $\mathsf{st}_\mathsf{S}.\mathsf{sk} \leftarrow \mathsf{sk}_\mathsf{S}$
06 $\mathsf{win} \leftarrow \mathsf{false}$
07 $\mathcal{A}^{\mathcal{O}}(\mathsf{pk}_\mathsf{S})$
08 **return** win

Procedure $\mathsf{Step}_1(\mathsf{prot}, \mathsf{i_c}, \mathsf{i_r}, \mathsf{in_C})$

09 $(\mathsf{T}_{\mathsf{st_C}}[\mathsf{i_c}], \mathsf{st}_\mathsf{C}^{\mathsf{tmp}}, \mathsf{out_C}, \mathsf{m}^\star) \leftarrow \Pi^{\mathsf{C:1}}_{\mathsf{prot}}[\mathsf{H}](\mathsf{T}_{\mathsf{st_C}}[\mathsf{i_c}], \varepsilon, \mathsf{in_C}, \varepsilon)$
10 n_p++
11 $\mathsf{i_p} \leftarrow n_p$
12 $\mathsf{T_p}[(\mathsf{prot}, \mathsf{C}, \mathsf{i_p})] \leftarrow (\mathsf{i_c}, \mathsf{st}_\mathsf{C}^{\mathsf{tmp}}, \mathsf{in_C}, 2)$
13 **if** prot = auth: $\mathsf{T}_{\mathsf{Client}}[\mathsf{i_p}] \leftarrow (\mathsf{m}^\star)$
14 **return** $\mathsf{m}^\star$

Oracle $\text{REG}_1(\mathsf{aid}, \mathsf{pk_S})$

15 **require** $\mathsf{T}_{\mathsf{uaid}}[\mathsf{aid}] = \bot$
16 n_r++
17 $\mathsf{in_C}.\mathsf{pw} \leftarrow \mathsf{pw}_{n_r}$
18 $\mathsf{T}_{\mathsf{pw}}[\mathsf{aid}] \leftarrow \mathsf{in_C}.\mathsf{pw}$
19 $\mathsf{in_C}.\mathsf{aid} \leftarrow \mathsf{aid}$
20 $\mathsf{in_C}.\mathsf{pk} \leftarrow \mathsf{pk_S}$
21 $\mathsf{k} \xleftarrow{\$} \mathsf{Auth}.\mathcal{D}_\mathsf{S}$
22 $\mathsf{rw} \leftarrow \mathsf{Auth.Full}[\mathsf{H}_{\mathsf{Auth}}](\mathsf{pk_S}, \mathsf{in_C}.\mathsf{pw}, \mathsf{aid}, \mathsf{k})$
23 $\mathsf{T}_{\mathsf{rw}}[\mathsf{aid}] \leftarrow (\mathsf{rw}, \mathsf{k})$
24 **if** $(\mathsf{rw}, \mathsf{aid}, \cdot) \in \mathsf{T_H}$: **abort**
25 Q++ ⟍ new instance
26 $\mathsf{T}_{\mathsf{key}}[\mathsf{aid}] \leftarrow Q$ ⟍ store instance
27 $(\mathsf{n_{mk}}, \mathsf{seed_{mk}}) \xleftarrow{\$} \{0,1\}^{\mathsf{nl}} \times \{0,1\}^{\mathsf{kl}}$
28 $\mathsf{k_{mk}} \leftarrow \mathsf{H}(\mathsf{seed_{mk}}, \mathsf{aid}, \text{“mk”})$
29 $\mathsf{ct_{mk}} \xleftarrow{\$} \mathsf{AEAD.Enc}(\mathsf{k_{mk}}, \mathsf{n_{mk}}, \mathsf{seed_{mk}}, (\mathsf{aid}, \text{“mk”}))$
30 n_p++
31 $\mathsf{i_p} \leftarrow n_p$
32 $\mathsf{T_p}[(\mathsf{areg}, \mathsf{i_p})] \leftarrow (\varepsilon, \mathsf{st}_\mathsf{C}^{\mathsf{tmp}}, \mathsf{in_C}, 2)$
33 $\mathsf{T}_{\mathsf{uaid}}[\mathsf{in_C}.\mathsf{aid}] \leftarrow \mathsf{true}$ ⟍ reg completed
34 $\mathsf{T}_{\mathsf{areg}}[\mathsf{i_p}] \leftarrow (\mathsf{aid}, \mathsf{k}, \mathsf{n_{mk}}, \mathsf{ct_{mk}}, Q')$
35 $|\mathsf{m_C}| \leftarrow |\mathsf{aid}| + |\mathsf{k}| + \mathsf{nl} + |\mathsf{ct_{mk}}| + \mathsf{kl}$ ⟍ simulate message length
36 **return** $|\mathsf{m_C}|$

Oracle $\text{STEPS}_1(\mathsf{areg}, \mathsf{i_p} \in [n_p], \mathsf{m_C}, \mathsf{in_S})$ ⟍ only areg

37 $(\mathsf{aid}, \mathsf{k}, \mathsf{n_{mk}}, \mathsf{ct_{mk}}, Q') \leftarrow \mathsf{T}_{\mathsf{areg}}[\mathsf{i_p}]$
38 $\mathsf{st_S}.\mathsf{acc}[\mathsf{aid}] \leftarrow (\mathsf{k}, \mathsf{n_{mk}}, \mathsf{ct_{mk}}, Q')$
39 $\mathsf{T_p}[(\mathsf{areg}, \mathsf{S}, \mathsf{i_p})] \leftarrow (\varepsilon, \varepsilon, \mathsf{in_S}, 2)$
40 $\mathsf{success_S}$

Oracle $\text{COMP}(\mathsf{aid})$

41 **require** $\mathsf{T}_{\mathsf{uaid}}[\mathsf{aid}] \neq \bot$
42 $\mathcal{S}_{\mathsf{cmp}} \leftarrow \mathcal{S}_{\mathsf{cmp}} \cup \{\mathsf{aid}\}$
43 $Q' \leftarrow \mathsf{T}_{\mathsf{key}}[\mathsf{aid}]$
44 $\mathsf{k}_{Q'} \leftarrow \text{COMP}_\mathcal{C}(Q')$ ⟍ Comp oracle
45 $(\mathsf{rw}, \cdot) \leftarrow \mathsf{T}_{\mathsf{rw}}[\mathsf{aid}]$
46 $\mathsf{T_H}[\mathsf{rw}, \mathsf{aid}, \text{“mac”}] \leftarrow \mathsf{k}_{Q'}$ ⟍ patch RO
47 **return** $(\mathsf{T}_{\mathsf{pw}}[\mathsf{aid}], \{\mathsf{T}_{\mathsf{st_C}}[\mathsf{i_c}] \mid \mathsf{T_u}[\mathsf{i_c}] = \mathsf{aid}\}, \{\mathsf{T}_{\mathsf{oob}}[(\mathsf{aid}, \cdot)]\})$

Oracle $\text{STEPS}_1(\mathsf{auth}, \mathsf{i_p} \in [n_p], \mathsf{m}, \mathsf{in_S})$ ⟍ only auth

48 $(\mathsf{st_S}, \mathsf{st}^{\mathsf{tmp}}, \mathsf{out}, \mathsf{m}^\star) \leftarrow \Pi^{\mathsf{S:1}}_{\mathsf{auth}}[\mathsf{H}](\mathsf{st_S}, \varepsilon, \mathsf{in_S}, \mathsf{m})$
49 $\mathsf{T_p}[(\mathsf{auth}, \mathsf{S}, \mathsf{i_p})] \leftarrow (\varepsilon, \mathsf{st}^{\mathsf{tmp}}, \mathsf{in_S}, 2)$
50 $\mathsf{T}_{\mathsf{Server}}[\mathsf{i_p}] \leftarrow (\mathsf{m}, \mathsf{m}^\star)$
51 **return** $\mathsf{m}^\star$

Oracle $\text{STEP}(\mathsf{auth}, \mathsf{C}, \mathsf{i_p} \in [n_p], \mathsf{m_S})$ ⟍ only for C, r = 2

52 **require** $\mathsf{T_p}[(\mathsf{auth}, \mathsf{C}, \mathsf{i_p})] \neq \bot$
53 $(\mathsf{i_c}, \mathsf{st}_\mathsf{C}^{\mathsf{tmp}}, \mathsf{in}, 2) \leftarrow \mathsf{T_p}[(\mathsf{auth}, \mathsf{C}, \mathsf{i_p})]$
54 $(\mathsf{aid}, \mathsf{pw}, \mathsf{pk_S}, \mathsf{st}, \mathsf{m}'_\mathsf{C}) \leftarrow \mathsf{st}_\mathsf{C}^{\mathsf{tmp}}$
55 $(\mathsf{sid}, \mathsf{auth}_2) \leftarrow \mathsf{m_S}$
56 $\mathsf{rw} \leftarrow \mathsf{Auth.Fin}[\mathsf{H}_{\mathsf{Auth}}](\mathsf{pk_S}, \mathsf{auth}_2, \mathsf{pw}, \mathsf{aid}, \mathsf{st})$
57 $\mathsf{st}_\mathsf{C}^{\mathsf{tmp}}.\mathsf{k}_{\mathsf{ek}} \leftarrow \mathsf{H}(\mathsf{rw}, \mathsf{aid}, \text{“ek”})$
58 $(\mathsf{rw}', \cdot) \leftarrow \mathsf{T}_{\mathsf{rw}}[\mathsf{aid}]$
59 **if** $\mathsf{rw} = \mathsf{rw}'$ ⟍ correct raw secret
60 $\quad Q' \leftarrow \mathsf{T}_{\mathsf{key}}[\mathsf{aid}]$ ⟍ recover instance
61 $\quad \mathsf{m}^\star \xleftarrow{\$} \text{TAG}(Q', (\mathsf{m}'_\mathsf{C}, \mathsf{m_S}))$ ⟍ tag oracle
62 **else**
63 $\quad \mathsf{k_{mac}} \leftarrow \mathsf{H}(\mathsf{rw}, \mathsf{aid}, \text{“mac”})$
64 $\quad \mathsf{m}^\star \leftarrow \mathsf{MAC.Tag}(\mathsf{k_{mac}}, (\mathsf{m}'_\mathsf{C}, \mathsf{m_S}))$
65 $\mathsf{T_p}[(\mathsf{auth}, \mathsf{C}, \mathsf{i_p})] \leftarrow (\mathsf{i_c}, \mathsf{st}^{\mathsf{tmp}}, \mathsf{in}, 3)$
66 $\mathsf{T}_{\mathsf{Client}}[\mathsf{i_p}].\mathsf{append}(\mathsf{m_S}, \mathsf{m}^\star)$
67 **return** $\mathsf{m}^\star$

Oracle $\text{STEP}(\mathsf{auth}, \mathsf{S}, \mathsf{i_p} \in [n_p], \mathsf{m_C})$ ⟍ only for S, r = 2

68 **require** $\mathsf{T_p}[(\mathsf{auth}, \mathsf{S}, \mathsf{i_p})] \neq \bot$
69 $(\mathsf{i_c}, \mathsf{st}^{\mathsf{tmp}}, \mathsf{in}, 2) \leftarrow \mathsf{T_p}[(\mathsf{auth}, \mathsf{S}, \mathsf{i_p})]$
70 $(\mathsf{st_S}, \mathsf{st}^{\mathsf{tmp}}, \mathsf{out_S}, \mathsf{m}^\star) \leftarrow \Pi^{\mathsf{S:2}}_{\mathsf{auth}}[\mathsf{H}](\mathsf{st_S}, \mathsf{st}^{\mathsf{tmp}}, \mathsf{m})$
71 $(\mathsf{aid}, Q', \mathsf{n_{mk}}, \mathsf{ct_{mk}}, \mathsf{m}'_\mathsf{C}, \mathsf{m_S}) \leftarrow \mathsf{st}^{\mathsf{tmp}}$
72 $b \leftarrow \text{VER}(Q', (\mathsf{m}'_\mathsf{C}, \mathsf{m_S}), \mathsf{m_C})$ ⟍ ver oracle
73 **if** $b = 0$
74 $\quad \mathsf{fail_S}$
75 $\mathsf{T_p}[(\mathsf{auth}, \mathsf{S}, \mathsf{i_p})] \leftarrow (\mathsf{i_c}, \mathsf{st}^{\mathsf{tmp}}, \mathsf{in}, 3)$
76 $\mathsf{T}_{\mathsf{Server}}[\mathsf{i_p}].\mathsf{append}(\mathsf{m_C})$
77 $\mathsf{m}^\star \leftarrow (\mathsf{n_{mk}}, \mathsf{ct_{mk}})$
78 **return** $\mathsf{Chk}(\mathsf{i_p}, \mathsf{out_S}, \mathsf{m}^\star)$

Procedure $\mathsf{Chk}(\mathsf{i_p}, \mathsf{out_S}, \mathsf{m}^\star)$

79 **if** $\mathsf{out_S}.\mathsf{decision} \wedge \mathsf{m}^\star \neq \bot$ ⟍ execution succeeded
80 $\quad$ **if** $\mathsf{T}_{\mathsf{Client}}[\mathsf{i_p}] \neq \mathsf{T}_{\mathsf{Server}}[\mathsf{i_p}]$ ⟍ messages fresh
81 $\quad\quad (\mathsf{i_c}, \cdot, \cdot, \cdot) \leftarrow \mathsf{T_p}[(\mathsf{auth}, \mathsf{C}, \mathsf{i_p})]$
82 $\quad\quad$ **if** $\mathsf{T_u}[\mathsf{i_c}] \notin \mathcal{S}_{\mathsf{cmp}}$ ⟍ client uncompromised
83 $\quad\quad\quad \mathsf{win} \leftarrow \mathsf{true}$
84 $\mathsf{T}_{\mathsf{Server}}[\mathsf{i_p}].\mathsf{append}(\mathsf{m}^\star)$
85 **return** $\mathsf{m}^\star$

Oracle $\text{STEP}(\mathsf{auth}, \mathsf{C}, \mathsf{i_p} \in [n_p], \mathsf{m_S})$ ⟍ only C, r = 3

86 **require** $\mathsf{T_p}[(\mathsf{auth}, \mathsf{C}, \mathsf{i_p})] \neq \bot$
87 $(\mathsf{i_c}, \mathsf{st}^{\mathsf{tmp}}, \mathsf{in}, 3) \leftarrow \mathsf{T_p}[(\mathsf{auth}, \mathsf{C}, \mathsf{i_p})]$
88 $(\mathsf{n_{mk}}, \mathsf{ct_{mk}}) \leftarrow \mathsf{m_S}$
89 $\mathsf{k_{mk}} \leftarrow \mathsf{AEAD.Dec}(\mathsf{st}^{\mathsf{tmp}}.\mathsf{k_{ek}}, \mathsf{n_{mk}}, \mathsf{ct_{mk}}, (\mathsf{st}^{\mathsf{tmp}}.\mathsf{aid}, \text{“mk”}))$
90 $\mathsf{T}_{\mathsf{st_C}}[\mathsf{i_c}] \leftarrow (\mathsf{st}^{\mathsf{tmp}}.\mathsf{aid}, \mathsf{st}^{\mathsf{tmp}}.\mathsf{sid}, \mathsf{k_{mk}})$
91 $\mathsf{T_p}[(\mathsf{auth}, \mathsf{C}, \mathsf{i_p})] \leftarrow (\mathsf{i_c}, \mathsf{st}^{\mathsf{tmp}}, \mathsf{in}, 4)$
92 $\mathsf{success_C}$

Fig. 45. Adversary $\mathcal{C}$ against SUF security of MAC having access to oracles $\text{TAG}, \text{VER}, \text{COMP}_\mathcal{C}$ simulating G_1 for adversary $\mathcal{A}$.

correct raw secret which can be checked via table T_{rw} and since the Auth computation is perfectly correct due to the previous changes. Otherwise $\mathcal{C}$ can just compute the tag themselves because the MAC key does not correspond to one of their challenge instances. The second server step can be simulated by using the reduction's verification oracle (Line 72).

It remains to show that the simulation of $\mathcal{C}$ is sound with respect to consistent random oracle outputs. The keys for the MAC are not known to $\mathcal{C}$ until a party is compromised. This also means that the output of the random oracle with input $(\mathsf{rw}, \mathsf{aid}, \text{“mac”})$, where rw is the correct Auth output computed during registration, is not known to the reduction. However, due to the abort in the RO introduced in the previous games, G_1 aborts whenever the adversary makes such a query on an uncompromised user. That allows $\mathcal{C}$ to patch the random oracle upon a corruption query because they learn the secret $\mathsf{k}_{Q'}$. An additional challenge could be that the reduction does not only have to make the random oracle consistent but also potential states or temporary states. However, the persistent or temporary client states never contain MAC keys and the server states are not revealed upon compromise.

Eventually, we argue why $\mathcal{C}$ wins their SUF game if $\mathcal{A}$ wins. Note that the Chk subroutine is only accessed if the verification oracle in Line 72 returns 1 implying the first winning condition of $\mathcal{C}$. Further, the flag win of $\mathcal{A}$ is only set if the corresponding user was not compromised implying the second winning condition of $\mathcal{C}$. For the third condition, we need to show that the message/tag pair was never queried for that key to the tag oracle which output the same tag.

For a proof by contradiction, assume that $\mathcal{C}$ queried the message, i.e. $(\mathsf{m}'_{\mathsf{C}}, \mathsf{m}_{\mathsf{S}})$, to their oracle Tag and obtained output m_{C}. Note that these messages must belong to the same protocol session i_{p} because they are taken from the temporary client's and server's state which the adversary cannot modify for uncompromised parties. Hence, the client's message m'_{C} was added to $\mathsf{T}_{\mathsf{Client}}$ in Auth_1 (more precisely in Step_1, Line 13). Since we assumed that m'_{C} also occurs as the first part of the message to Ver, it originates from the server's temporary state and was therefore added in StepS_1 which implies that it was also added to $\mathsf{T}_{\mathsf{Server}}$ (Line 50). The same holds for m_{S} which was added in the same query (Line 50). By assumption, this is again the same value as the second part of the message queried to Tag and was also added to $\mathsf{T}_{\mathsf{Client}}$ (Line 66) together with the output tag m_{C}. Lastly, the tag m_{C} is also added to $\mathsf{T}_{\mathsf{Server}}$ (Line 76). This means it holds $\mathsf{T}_{\mathsf{Client}} = \mathsf{T}_{\mathsf{Server}}$ and $\mathcal{A}$ cannot win their $\mathsf{Exp\text{-}Auth}$ game concluding the contradiction argument. □

Collecting the probabilities completes the proof of Theorem 14. □