Modeling and construction of interactive cryptographic protocols for outsourced storage

Maxime Roméas

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2022IPPAX086

Thèse de doctorat

Modélisation et construction de protocoles cryptographiques interactifs pour le stockage distant

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à l'École polytechnique

École doctorale n°626 École doctorale de l'Institut Polytechnique de Paris (EDIPP)
Spécialité de doctorat : Informatique

Thèse présentée et soutenue à Palaiseau, le 29/11/2022, par

Maxime Roméas

Composition du Jury :

Gilles Zémor Professeur, Université de Bordeaux, IMB	Président
Damien Vergnaud Professeur, Sorbonne Université, LIP6	Rapporteur
Pierre Loidreau Chercheur associé, DGA & Université de Rennes 1, IRMAR	Rapporteur
Camilla Hollanti Professeure, Aalto University	Examinatrice
Duong Hieu Phan Professeur, Télécom Paris	Examinateur
Matthieu Rambaud Maître de conférences, Télécom Paris	Examinateur
Françoise Levy-dit-Vehel Enseignante-Chercheuse HDR, ENSTA	Directrice de thèse

Introduction

This thesis deals with the security of the storage, the access and the maintenance of outsourced data. The outsourcing of data on distant servers (storage and computations in the "cloud") is a growing trend for companies and users alike. It reduces the cost of locally storing and maintaining the data. Outsourcing also allows to access the data from anywhere and to use the services of cloud service providers ("Software as a Service"). Moreover, one can archive large amounts of data, that are rarely accessed and pointless to store locally, for many years. Unfortunately, cloud storage raises new threats for users. We focus on the three following issues.

First, if a user rarely accesses its data, how can he be sure that it is effectively stored and that it did not suffer any alterations? Indeed, a distant server can face a hardware problem, loose some of the data and not notify users that only access their data rarely, or never have. A malicious server can even delete rarely accessed files to make room for new customers and increase its profits. The cryptographic schemes that solve this problem are called Proofs of Retrievability (PoR).

Second, when the user of an encrypted database suspects that his cryptographic key has been compromised, how can he remotely update its ciphertexts to a fresh key? By remotely, we mean without downloading and re-uploading the whole database. This process of remotely rotating cryptographic keys can also be useful to implement access control policies when an encrypted database is used by a constantly evolving group of clients. For instance, a company can rotate keys each time an employee leaves. This problem can be dealt with using Updatable Encryption (UE) schemes.

Third, sometimes it is not the data that is confidential but the way in which it is accessed Indeed, knowing which patient's files a doctor accesses or which stock values a trader requests is clearly problematic. How can a client hide the way he accesses remote data? This question can be answered by Private Information Retrieval (PIR) schemes.

The goal of this thesis can be summarize in three steps. In step one, we develop modular security notions and models that closely match the security expectations of real-world solutions for the three above problems. Then, in step two, we check if existing security definitions are sufficient, and sometimes also necessary, to provide the security guarantees identified in step one. Finally, we determine if existing cryptographic schemes reach our security definitions and, if not, we improve them or propose new constructions that do.

We give an overview of the cryptographic protocols we chose to study.

Proofs of Retrievability

The context of PoRs is the following: a client stores a huge file on a distant server and erases its local copy. PoRs feature an audit procedure that allows the client to convince himself, using only a little amount of bandwith, that its file is stored on the server and that it can retrieve it in full (*i.e.* that it is not modified or deleted). After a successful audit, the client can perform the extraction phase of the PoR to retrieve the file. The first formal definition of PoRs was given

by Juels and Kaliski in 2007 [JK07]. They also propose the first PoR scheme. It is based on checking the integrity of some sentinel symbols secretly placed by the client before uploading its file. This scheme has low communication but its drawback is that it is bounded-use only, as the number of possible verifications depends on the number of sentinels. Shacham and Waters [SW08] proposed to correct this drawback by appending some authenticator symbols to the file. Verification consists in checking random linear combinations of file symbols and authenticators. Then comes a few PoR schemes based on codes. Bowers *et al.* [BJO09] proposed a double-layer encoding with the use of an inner code to recover information symbols and an outer code to correct the remaining erasures. Dodis *et al.* [DVW09] formalize the verification process as a request to a code which models the space of possible answers to a challenge. They use Reed-Solomon codes to design their PoR scheme. In 2013, Paterson *et al.* [PSU13] laid the foundation for studying PoR schemes using a coding theoretic framework. Following these ideas, Lavauzelle and Levy-dit-Vehel [LLDV16] used the local structure of the lifted codes introduced by Guo *et al.* [GKS13] to build a PoR scheme, that compares favourably to those presented above with respect to storage overhead.

Updatable Encryption

UE is a variant of symmetric encryption, introduced by Boneh *et al.* in 2013 [BLMR13], which supports key rotation on an outsourced encrypted database while minimizing the bandwith used. To rotate keys, the client generates a single update token using both the old and the new key. Then, he sends this token to the server which is able to update all of its ciphertexts under the new key. Of course, confidentiality of the data must be preserved through all these steps. Unlike symmetric encryption, UE schemes aim at preserving the confidentiality of the data in a setting where secret keys and update tokens can leak. In this thesis, we focus on the *ciphertext-independent* variant of UE, where the update token is unique and independent of the ciphertexts. The huge real-life applications of UE explain the recent renewed interest on the subject [LT18, KLR19, BDGJ20, Jia20, Nis22].

Security notions for UE have evolved a lot since the original proposal of [BLMR13]. Lehmann and Tackmann [LT18] proposed two CPA security notions where the adversary can adaptively corrupt keys and tokens. Their IND-ENC notion requires fresh encryptions to be indistinguishable and their IND-UPD notion asks the same for updated ciphertexts. Klooß *et al.* [KLR19] augmented the previous notions with CCA security and integrity protection. Boyd *et al.* [BDGJ20] introduced the IND-UE notion which is stronger than previous ones and requires fresh encryptions to be indistinguishable from updated ciphertexts. They also show that a CPA UE scheme with added ciphertext integrity (CTXT) is CCA.

As for UE constructions in the classical setting, RISE of [LT18], is an updatable variant of ElGamal where the public key is used in the token. [KLR19] introduced two generic constructions based on encrypt-and-MAC (secure under DDH) and on the Naor-Yung transform (secure under SXDH). Boyd *et al.* [BDGJ20] proposed the permutation-based SHINE schemes, that achieve their stronger detIND-UE-CCA security notion in the ideal cipher model (under DDH).

In the post-quantum setting, Jiang [Jia20] presented the first post-quantum UE scheme LWEUE (secure under LWE). In [Nis22], Nishimaki introduced RtR, another LWE-based UE scheme, which is the first ciphertext-independent UE scheme that prevents the adversary from obtaining the new key from the knowledge of the update token and the old key. Nishimaki showed that UE schemes with this property have stronger security than those without. These two LWE-based schemes use homomorphic operations to re-randomize updated ciphertexts which has two main drawbacks. On one hand, ciphertext noise grows with each key update, which means that these schemes only support a bounded number of updates. On the other

hand, using the homomorphic property and the knowledge of the update token, an adversary can craft ciphertexts of related messages which means that these schemes are not CCA secure (only randIND-UE-CPA).

Private Information Retrieval

PIR, introduced by Chor *et al.* in 1995 [CGKS95], allows a client to retrieve an entry of an outsourced database without revealing its location to the server. A trivial way to do this consists in retrieving the whole database and reading the desired entry locally. Thus, PIR protocols try to minimize the bandwith usage, *i.e.*, they require communication *sublinear* in the size of the database. There is a huge literature surrounding PIR, starting from the seminal work of Chor *et al.* [CGKS95]. PIR comes in different flavors, one- or multi- server, information-theoretically secure PIR (IT-PIR) [Amb97, CG97, BI01, BIKR02, Yek08, Efr12, AdVS14, DG16] and computationally secure PIR (cPIR) [KO97, CMS99, KO00, GR05, OI07]. There are many more variants of PIR such as symmetric PIR [GIKM00] where the client cannot learn any information about the entries of the database other than the one he requested or batch PIR [IKOS04, Hen16] whose aim is to amortize the computational cost of the servers over a batch of queries made by the client. Moreover, the database can be stored on an unique server (1-server PIR) or on multiple servers (k-server PIR with $k > 1$). In the $k > 1$ case, PIR tries to protect the confidentiality of requests against a coalition of $t < k$ servers. These servers can be passive (sometimes called honest-but-curious), they only observe the client's requests to try to break their confidentiality. Otherwise, servers can be active (also called byzantine), they try to break confidentiality by not following the protocol. Finally, PIR has a lot of applications in cryptographic protocols: in private messaging [SCM05, AS16, ACLS18], online anonymity [MOT$^+$11, KLDF16], targeted advertising [Jue01] and many more.

Constructive Cryptography

We choose to phrase our security definitions in the Constructive Cryptography (CC) model introduced by Maurer in 2011 [Mau12]. This model aims at asserting the real security of cryptographic primitives. To do so, it redefines cryptographic protocols as discrete systems of three types: resources (*e.g.* channels, keys, servers...), converters (*e.g.* encryption, hash functions, signatures...) and distinguishers (systems connecting to a resource and outputting a bit after interacting with it).

In this model, starting from a basic resource, a converter aims at constructing an enhanced resource, *i.e.* one with better security guarantees. An example of constructed resource is a confidential server, where the data stored by a client is readable by this client only. The only information that leaks to other parties is its length. This resource does not exist, but it can be emulated by an insecure server on which the client uses a suitable encryption protocol. Furthermore, it is then possible to precisely assess the atomic tools (*e.g.* starting resource and security notions for the converter) needed to perform a given construction.

The CC framework follows a top-down approach, allowing to get rid of useless details and hypotheses made in other models. One of the strengths of CC is its "composability", in the sense that, if two steps (*e.g.* integrity and confidentiality) were proven secure in CC, their composition is also secure. Thus, security proofs are simplified because once a security property of a resource is proven, it transfers naturally when adding another functionality.

Contributions

Interactivity in Constructive Cryptography. We extend the CC model so as to handle interactive protocols. By interactive, we mean any protocol that involves at least two parties who agree to perform calculations specified by the protocol. We design and construct a so-called *Interactive Server-Memory Resource* (ISMR), that is an augmented version of the basic instantiation of a client-server protocol in CC, namely the Server-Memory Resource of [BM18]. We make this modeling of interactivity possible by proposing new ways of using converters and simulators. Regarding converters: in addition to their basic features, they can now be used to modify the number or type of parameters of a query at a given interface. Moreover, such converters have the ability to transiently deactivate capabilities of an interface. We also introduce and model what we call *semi-honest* (commonly named honest-but-curious) interfaces, in which both converters and simulators can be plugged. This permits us to precisely model and describe the behavior of semi-honest adversaries, as well as byzantine ones. Using the flexibility of the CC model equipped with our new tools, we give a composable modeling of PIR protocols, yielding a unique model that unifies different notions of PIR: information theoretic-PIR, computational-PIR and one- or multi- server PIR. We then apply our ISMR construction to UE to obtain the contributions described below.

Composable modeling and security notions for Updatable Encryption. We create the first composable modeling of UE schemes. Indeed, in the only concurrent work we know of that considers UE in the CC context [FMM21], the construction proposed is only valid for particular instantiations of UE schemes. In order to give this generic composable model for UE, we had to provide and refine new and existing ideas. First, we make good use of the abstraction permitted by CC, of our new tools to model interactive protocols as well as of the global event history of [JMM19] to propose a simple modeling of UE, devoid of unnecessary, and potentially troublesome, details. Second, we improve a proof technique of [CMT13] by proposing a double hybrid argument to prove our constructions in CC. Our proof technique is of independent interest since it permits to prove statements about conjunctions of security notion or security notions with challenges of different types. For example, in the IND-UE security notion, the challenge can either belong to the message space or to the ciphertext space. Our double hybrid argument deals with this asymmetry. Last, but not least, we rule out the tedious commitment problem: an impossibility result that usually arises when dealing with key exposures in composable frameworks. To do so, we adapt the interval-wise guarantees of [JM20] to UE by making distinct security statements inside many time intervals. By carefully choosing both statements and intervals, we are able to circumvent impossibility results and still make meaningful security statements.

We use our modeling of UE to study its security notions. These notions are numerous and have constantly evolved since the introduction of UE in 2013. We focus our attention on the IND-ENC notion of [LT18], IND-UPD of [LT18] and IND-UE of [BDGJ20]. With our modeling, we identify two meaningful leakage contexts for UE and study the security notions best suited for both of them. In the *unrestricted* leakage context, the adversary is allowed to leak outsourced ciphertexts as he wants. Meanwhile, in the *restricted* leakage context, the adversary is only allowed to leak a given ciphertext once between each key update. In both contexts, the adversary can freely expose cryptographic keys and update tokens and we precise the post-compromise security guarantees given by each security notions. Furthermore, we show that IND-UE-CPA security is sufficient, and also necessary, to securely construct a confidential ISMR that hides the age of ciphertexts in the restricted leakage context; and IND-ENC-CPA + IND-UPD-CPA security is sufficient, and also necessary, for a secure construction of a confidential ISMR in

case of unrestricted leakage (where the age of ciphertexts is impossible to protect). By age of a ciphertext, we mean the number of times it has been updated. We do the same with CCA security notions when the adversary is allowed to inject arbitrary data. In particular, we show that the IND-UE security notion of [BDGJ20] is not always strictly stronger than the one of [LT18], namely IND-ENC + IND-UPD.

New constructions for post-quantum Updatable Encryption. This thesis solves two open problems in ciphertext-independent post-quantum UE. First, we propose the first post-quantum UE scheme that supports an unbounded number of key updates. Second, the security of our UE scheme comes from the hardness of isogeny-based problems, whereas the two prior post-quantum UE schemes of Jiang [Jia20] and Nishimaki [Nis22] are both based on the Learning with Errors (LWE) problem [Reg05]. Moreover, we show how to build a CCA secure UE scheme using group actions. Unfortunately, we do not know how to instantiate this new construction in the post-quantum setting. Doing so would solve the open problem of building a CCA secure post-quantum UE scheme. We do so by studying the problem of building UE in the group action framework. We propose two new generic constructions of UE in this framework.

First, we introduce a new notion of Mappable Effective Group Action (MEGA) and show that we can build UE from a MEGA by generalizing the SHINE construction of Boyd *et al.* [BDGJ20]. We call this first construction GAINE for Group-Action Ideal-cipher Nonce-based Encryption. As mentioned above, we require our group actions to be *mappable*. This means that there exists an invertible and efficient map going from the space of messages to the set of the group action. Interestingly, apart from being mappable, we require almost nothing from the group action. In particular, we do not need it to be free or transitive, and the group need not even be abelian. In this case, we require the group action to be weak pseudorandom, *i.e.* an adversary cannot distinguish between many samples that or either of the form $(s_i, g \star s_i)$, where g is a random group element and the elements s_i are random set elements, or samples of the form (s_i, t_i) where s_i and t_i are random set elements. Unfortunately, we do not know how to instantiate GAINE in the post-quantum setting. Indeed, there are currently two cryptographic group actions that are mappable and believed to be post-quantum. The first one was introduced by Tang *et al.* [TDJ+22] and uses alternating trilinear forms, while the second one from Ji *et al.* [JQSY19] is based on tensors. These two group actions are not abelian and they were conjectured to be weak pseudorandom but only over a very small number of samples. A recent attack by Beullens [Beu22] disproves this conjecture for the trilinear forms group action. Unfortunately, when working with nonabelian group actions, our security proof for GAINE needs one sample per ciphertext. Thus, we are not currently able to give a practical post-quantum instantiation for GAINE. Finding one would solve the problem left open by Jiang [Jia20] at Asiacrypt2020 to build a CCA post-quantum UE scheme.

Our GAINE construction cannot be instantiated by any group action. Indeed, for the group action to be mappable and cryptographically secure is not trivial at all. In particular, these requirements remove the most obvious way to sample elements in the set by using the group action. Thus, building the invertible map must really depend on the concrete description of the set and how its elements can be represented.

Isogeny-based group actions are the most studied post-quantum group actions. They have been more studied than their multivariate counterpart and while there are on-going discussions regarding the exact level of security reached by these group actions, we can have some confidence in the fact that the underlying problems are hard. Unfortunately, in the case of the CSIDH cryptographic group action [CLM+18], it is notoriously hard to sample elements in the set [BBD+22]. Thus, we cannot really hope to instantiate GAINE with CSIDH or another similar group action from isogenies. However, we still manage to show how to build UE from the CSIDH

group action. We circumvent the mappable requirement by using an idea of Moriya *et al.* for their SIGAMAL encryption scheme [MOT20]. We obtain an analog of CSIDH by considering a set made of elements constituted by a curve and a point (and not just a curve). We refine the idea of Moriya *et al.* to get a scheme that is updatable. The way we circumvent the issue that CSIDH is not mappable could be of independent interest as there are numerous examples of protocols where this proves to be a big obstacle. We extracted an abstract framework of this idea to identify the algebraic structure required by our new UE scheme. This gave us what we call a TOGA for Triple Orbital Group Action. As the name suggests, there are three group actions involved in this scheme, each with a specific role, and we require the three different operations to interact in a very specific way. We call our generic UE scheme based on this concept TOGA-UE.

New and improved constructions for code-based Proofs of Retrievability. First, we show that the security of a code-based PoR of Lavauzelle and Levy-dit-Vehel [LLDV16] was largely overestimated. In order to fix this scheme, we propose a framework for the design of secure and efficient PoR schemes based on Locally Correctable Codes (LCC). We give a first instantiation of our framework using the high rate lifted codes of Guo *et al.* [GKS13]. This yields an infinite family of good PoRs, that can be seen as a secure generalization of the [LLDV16] PoR. Our PoR features low communication complexity, small storage overhead and clear security guarantees. We assert its security by solving a finite geometry problem, giving an explicit formula for the probability of an adversary to fool the client. More precisely, we characterize the adversarial corruption patterns that prevent the client from retrieving its outsourced file. They form what we call d-cover sets of a finite affine space, where the integer d is a parameter of the code used in the PoR. Here is a definition of these sets.

Definition 0.1 (d-cover sets)**.** *Let $\mathbb{F}$ be a finite field and m, d be positive integers. We say that a set $S \subseteq \mathbb{F}^m$ is a d-cover set if S verifies the following property:*

$$\forall s \in S, \forall \text{ line } \ell \subseteq \mathbb{F}^m \text{ going through } s, |S \cap \ell| \geq d$$

Or equivalently, for all lines $\ell \subseteq \mathbb{F}^m, |S \cap \ell| = 0$ or $|S \cap \ell| \geq d$

We find a lower bound on the number of lines intersecting these d-cover sets and use it to estimate the security of our scheme. Then, using the local correctability properties of graph codes [Tan81], we get another instantiation of our framework and derive an analogous formula for the success probability of the audit. This time, given a graph G and an integer parameter d, we show that the adversarial erasure patterns that prevent the client from retrieving its outsourced file correspond to the subgraphs of G of minimum degree d.

Furthermore, using a different approach, we also design a good PoR based on a family of graph codes called expander codes [Tan81, SS96]. We use expander codes based on graphs derived from point-line incidence relations of finite affine planes. Høholdt *et al.* [HJ06, HJ11, BHPJ13] showed that, when using Reed-Solomon codes as inner codes, these codes have good dimension and minimum distance over a relatively small alphabet. Moreover, expander codes possess very efficient unique decoding algorithms [SS96, Zém01]. We take advantage of these results to design a PoR scheme that extracts the outsourced file in quasi-linear time and features better concrete parameters than state-of-the-art schemes with respect to storage overhead and size of the outsourced file. We follow an unbounded-use audit procedure of [JK07, SW08, BM18] to ensure that the extraction of the outsourced file will succeed with high probability. The properties of our expander codes yield an audit with communication complexity comparable to other code-based PoRs.

Composability of Locally Correctable Codes. Contrary to classical error correcting codes, LCCs permit to correct a given location of the codeword by only reading a sublinear number of its symbols. We give a composable modeling of LCCs [KT00] in CC. We obtain an error-resilient SMR where clients can read symbols with sublinear communication. Doing so, we show that the failure probability of the local correcter depends not only on the number of corrupted symbols but also on their locations. Computing this exact failure probability, a problem often overlooked in the literature, is needed to give the security guarantees of our error-resilient SMR and it may be of independent interest. For instance, in our work on code-based PoRs, we showed that having a precise understanding of the fail cases of the local correcter was essential to estimate the security of our schemes. Finally, for the important class of lifted Reed-Solomon codes [GKS13], we show that this failure probability can be computed in polynomial time in the length of the lifted code.

Outline of the thesis

In Chapter 1, we introduce the Constructive Cryptography model, together with the translation of the outsourced storage setting in this model. We then present the syntax and the security definitions of Proofs of Retrievability, Updatable Encryption and Private Information Retrieval. We also give the definition and an example of Locally Correctable Code. In Chapter 2, we present our first results: the post-quantum UE schemes GAINE and TOGA-UE. We detail the algebraic structure needed to instantiate our schemes, prove their correctness and security and discuss how to instantiate them in the post-quantum setting. In Chapter 3, we present our new and improved PoR constructions. First, we propose a framework generalizing the PoR construction of Lavauzelle and Levy-dit-Vehel [LLDV16] and give a detailed security analysis of our generalization. We give two instantiations of our framework. The first one is based on lifted codes and constitutes an improvement over the [LLDV16] PoR, parameters and security levels can be found in Figs. 3.7 and 3.8. The second one is based on graph codes. Then, using a different approach, we propose a PoR scheme based on expander codes that features quasi-linear extraction time and better storage overhead than our previous constructions. We give parameters in Figs. 3.10 and 3.11. In Chapter 4, we extend the CC model so as to handle interactive protocols. We introduce an interactive modeling of the outsourced storage setting in CC. We then use it to give a composable modeling of UE and PIR. As for UE, we compare the security notions of [LT18] and [BDGJ20], showing which one to use in different real-world applications. As for PIR, we verify that the security definitions of [CGKS95] give the expected security guarantees and we propose a unified modeling of computational, information theoretic, one server and multi-server PIR. Finally, we conclude by pointing out some future prospects for our work.

Résumé

Cette thèse concerne la sécurisation du stockage, de l'accès et de la maintenance de données distantes. De plus en plus d'entreprises et de particuliers ont recours à l'externalisation de leurs données sur des serveurs distants (stockage et calcul dans le "cloud") car elle permet notamment de réduire les coûts liés au stockage et à la maintenance de ces données. De plus, cette externalisation permet d'accéder aux données n'importe où et d'utiliser les services proposés par les fournisseurs de stockage distant ("Software as a Service"). Enfin, il devient possible d'archiver des quantités importantes de données, auxquelles on accède rarement, pendant des années. Malheureusement, le stockage de données sur le cloud fait émerger de nouvelles menaces pour ses utilisateurs. Nous nous intéressons aux trois problématiques suivantes.

Premièrement, si un utilisateur n'accède que rarement à ses données, comment peut-il être certain qu'elles sont toujours stockées sur le serveur et qu'elles n'ont pas subies de modifications ? En effet, le serveur pourrait faire face à un problème matériel, perdre une partie des données et ne pas en informer les utilisateurs qui n'accèdent pas souvent, voire jamais, à leurs données. Un fournisseur de stockage malveillant pourrait même supprimer des fichiers rarement consultés pour faire de la place à de nouveaux clients et ainsi accroître ses profits. Les schémas cryptographiques qui apportent une solution à ce problème sont nommés Preuves de Récupérabilité (PoR pour Proof of Retrievability).

Deuxièmement, quand un utilisateur d'une base de données chiffrée suspecte que ses clés cryptographiques ont été compromises, comment peut-il mettre à jour ses chiffrés sous une nouvelle clé à distance ? Par à distance, nous voulons dire sans télécharger puis re-externaliser l'intégralité de la base de données. Cette procédure de rotation de clés cryptographiques peut aussi être très utile pour mettre en place des politiques de contrôle d'accès sur une base de données utilisée par un ensemble de clients en évolution constante. Par exemple, une entreprise pourrait changer de clé à chaque départ d'un salarié. Ce problème peut être résolu en utilisant du Chiffrement avec Mise à Jour (UE pour Updatable Encryption).

Troisièmement, il arrive que ce ne soit pas les données elles-mêmes qui soient confidentielles mais la façon dont on y accède. En effet, savoir à quel dossier médical un médecin accède ou quel cours d'action un trader consulte est clairement problématique. On peut alors se demander : comment un utilisateur peut-il cacher la manière dont il accède à ses données distantes ? Les schémas de Récupération Confidentielle d'Information (PIR pour Private Information Retrieval) permettent de répondre à cette question.

L'objectif de cette thèse peut être résumé en trois étapes. En premier lieu, nous développons des notions et des modèles de sécurité modulaires qui correspondent étroitement aux attentes de sécurité des solutions concrètes pour les trois problèmes ci-dessus. Ensuite, nous vérifions si les notions de sécurité existantes sont suffisantes, et parfois aussi nécessaires, pour procurer les garanties de sécurité identifiées précédemment. Enfin, nous déterminons si les schémas cryptographiques existants atteignent nos notions de sécurité et, si ce n'est pas le cas, nous les améliorons ou nous proposons de nouveaux schémas qui le font.

Nous donnons maintenant un aperçu des protocoles cryptographiques que nous avons choisi

d'étudier.

Preuves de Récupérabilité

Le contexte des PoRs est le suivant : un utilisateur envoie un fichier volumineux sur un serveur distant et efface sa copie locale. Le PoR met à disposition de l'utilisateur une procédure d'audit qui lui permet de se convaincre, en utilisant le moins de bande passante possible, que son fichier est toujours présent sur le serveur et qu'il peut le récupérer en intégralité (sans aucune modification ou suppression). Après un audit réussi, l'utilisateur peut utiliser la procédure d'extraction du PoR pour récupérer son fichier. La première définition formelle d'un PoR a été donnée par Juels et Kaliski en 2007 [JK07]. Ils proposèrent également le premier schéma de PoR. Il consiste à vérifier l'intégrité de symboles sentinelles placés par le client dans son fichier avant de l'externaliser. Ce schéma utilise peu de bande passante mais le fait qu'il ne peut être utilisé qu'un nombre borné de fois (dépendant du nombre de sentinelles) constitue un inconvénient majeur. Shacham et Waters [SW08] ont proposé de corriger cet inconvénient en ajoutant des symboles d'authentification à la fin du fichier. La phase d'audit consiste alors à vérifier des combinaisons linéaire aléatoires des symboles du fichier et des symboles d'authentification. Puis, quelques schémas de PoR basés sur les codes correcteurs d'erreurs ont commencé à apparaître. Bowers *et al.* [BJO09] ont proposé une double couche d'encodage avec utilisation d'un code intérieur pour récupérer les symboles d'information et d'un code extérieur pour corriger les éventuels effacements. Dodis *et al.* [DVW09] formalisent leur phase d'audit comme une requête à un code qui modélise l'espace des réponses possibles à un challenge. Ils utilisent des codes de Reed-Solomon pour construire leur PoR. En 2013, Paterson *et al.* [PSU13] ont posé les fondations de l'étude des PoRs dans le cadre de la théorie des codes correcteurs. Suivant leurs idées, Lavauzelle et Levy-dit-Vehel [LLDV16] ont utilisé les propriétés locales des relèvements de code de Guo *et al.* [GKS13] pour constuire un PoR qui nécessite peu de stockage supplémentaire par rapport aux schémas précédents.

Chiffrement avec Mise à Jour

UE est une variante du chiffrement symétrique, introduite par Boneh *et al.* en 2013 [BLMR13], qui permet de mettre à jour une clé cryptographique sur une base de données chiffrée et distante en minimisant l'utilisation de bande passante. Pour mettre à jour la clé, l'utilisateur génère un token en utilisant à la fois l'ancienne et la nouvelle clé. Puis, il envoie ce token au serveur qui peut l'utiliser pour mettre à jour tous les chiffrés qu'il détient sous la nouvelle clé. Bien entendu, la confidentialité des données doit être préservée durant toutes ces étapes. A la différence du chiffrement symétrique, les schémas UE cherchent à préserver la confidentialité des données dans un cadre où les clés et les tokens peuvent être corrompus. Dans cette thèse, nous nous concentrons sur la variante d'UE dite *indépendante des chiffrés*, dans laquelle le token est unique et indépendant des chiffrés. Les applications réelles multiples et cruciales d'UE expliquent le fort intérêt que ce sujet a généré récemment [LT18, KLR19, BDGJ20, Jia20, Nis22].

Les notions de sécurité d'UE ont beaucoup évolué depuis la proposition originelle de [BLMR13]. Lehmann et Tackmann [LT18] ont proposé deux notions de sécurité CPA (contre les attaques à clair choisi) où l'adversaire peut choisir de corrompre les clés et les tokens de manière adaptative. Leur notion de sécurité IND-ENC impose aux nouveaux chiffrés d'être indistinguables les uns des autres et leur notion IND-UPD demande la même chose pour les chiffrés mis à jour. Klooß *et al.* [KLR19] ont amélioré les notions précédentes en ajoutant une variante CCA (contre les attaques à chiffré choisi) et une protection de l'intégrité. Boyd *et al.* [BDGJ20] ont introduit

la notion IND-UE qui est plus forte que les notions précédentes, elle demande que les nouveaux chiffrés soient indistinguables des chiffrés mis à jour. Ils montrent également qu'un schéma UE sûr contre les attaques CPA et assurant l'intégrité des chiffrés (CTXT) est également sûr contre les attaques CCA.

Concernant les constructions d'UE dans le contexte pré-quantique, RISE de [LT18] est une variante avec mise à jour d'ElGamal où la clé publique est utilisée comme token. [KLR19] introduit deux constructions génériques basé sur encrypt-and-MAC (sûre sous l'hypothèse DDH) et sur la transformation de Naor-Yung (sûre sous l'hypothèse SXDH). Boyd *et al.* [BDGJ20] proposent les schémas SHINE basés sur l'utilisation d'une permutation, ils atteignent leur notion de sécurité detIND-UE-CCA dans le modèle du chiffrement idéal (sous l'hypothèse DDH).

Dans le contexte post-quantique, Jiang [Jia20] présente le premier schéma UE post-quantique appelé LWEUE (sûr sous l'hypothèse LWE). Dans [Nis22], Nishimaki introduit RtR, un autre schéma basé sur LWE, qui est le premier schéma UE qui empêche un adversaire d'obtenir la nouvelle clé s'il connaît déjà l'ancienne clé et le token. Nishimaki prouve également que les schémas UE possédant cette propriété possède une sécurité supérieure à ceux qui ne la possède pas. Ces deux schémas basés sur LWE utilisent des opérations homomorphes pour rafraîchir l'aléa des chiffrés lors de leur mise à jour. Ceci a deux désavantages : d'une part, le bruit augmente à chaque mise à jour de chiffré, ce qui signifie que ces schémas n'autorisent qu'un nombre limité de mises à jour. D'autre part, en utilisant les propriétés homomorphes et la connaissance du token, un adversaire peut forger des chiffrés de nouveaux messages ce qui signifie que ces schémas ne sont pas sûrs au sens CCA (seulement au sens randIND-UE-CPA).

Private Information Retrieval

Le PIR, introduit par Chor *et al.* en 1995 [CGKS95], permet à un client de récupérer une entrée d'un base de donnée distante sans révéler laquelle au serveur. Une manière triviale d'y arriver est de récupérer l'intégralité de la base de donnée et de lire l'entrée cherchée localement. C'est pourquoi les protocoles de PIR essayent de minimiser l'utilisation de bande passante, *i.e.*, on exige que le protocole possède une complexité de communication *sous-linéaire* en la taille de la base de données. Il existe une littérature massive concernant le PIR, qui commence avec l'article fondateur de Chor *et al.* [CGKS95]. Le PIR possède de nombreuses variantes, un seul ou plusieurs serveurs, le PIR sûr au sens de la théorie de l'information (IT-PIR) [Amb97, CG97, BI01, BIKR02, Yek08, Efr12, AdVS14, DG16] et le PIR sûr au sens calculatoire (cPIR) [KO97, CMS99, KO00, GR05, OI07]. On peut également citer le PIR symétrique [GIKM00] dans lequel le client ne peut pas apprendre d'information concernant les entrées de la base de données exceptée celle qu'il a demandée au serveur. Dans le cadre du "batch PIR" [IKOS04, Hen16], l'objectif est d'amortir le coût des calculs effectués par le serveur en traitant les requêtes du client par lots. Pour les protocoles de PIR à serveurs multiples, le protocole doit protéger la confidentialité des requêtes du client contre une coalition de plusieurs serveurs. Ces serveurs peuvent êtres passifs (on dit aussi honnêtes mais curieux), dans ce cas ils se contentent d'observer les actions du client pour mettre à mal sa confidentialité. Sinon, les serveurs peuvent être actifs (on dit aussi byzantins), ils essayent alors de briser la confidentialité en ne suivant pas le protocole. Enfin, le PIR possède de nombreuses applications : dans les systèmes de messagerie sécurisée [SCM05, AS16, ACLS18], pour assurer l'anonymat [MOT+11, KLDF16], pour la publicité ciblée [Jue01] et bien plus encore.

Cryptographie Constructive

Nous avons choisi de donner nos définitions de sécurité dans le modèle Cryptographie Constructive (CC) introduit par Maurer en 2011 [Mau12]. Ce modèle a pour objectif d'identifier la sécurité réelle apportée par les primitives cryptographiques. Pour ce faire, il redéfinit les protocoles cryptographiques comme des systèmes discrets de trois types : les ressources (*e.g.* canaux de communication, clés, serveurs...), les convertisseurs (*e.g.* chiffrement, fonctions de hachage, signatures...) et les distingueurs (des systèmes connectés et interagissant avec une ressource et renvoyant un bit).

Dans ce modèle, partant d'une ressource de départ, un convertisseur a pour but de construire une ressource améliorée, *i.e.* avec de meilleures garanties de sécurité. Un exemple d'une telle ressource est un serveur confidentiel, où les données stockées ne peuvent être lues que par le client. La seule information parvenant à un adversaire étant la taille de ces données. Cette ressource n'existe pas, mais elle peut être émulée par un serveur basique sur lequel le client utilise un protocole de chiffrement adéquat. De plus, il devient alors possible d'identifier les outils atomiques (*e.g.* ressource de départ et notions de sécurité des convertisseurs) requis pour effectuer une construction donnée.

Le modèle CC suit une approche "top-down" qui lui permet de se débarrasser de détails et d'hypothèses inutiles utilisés dans d'autres modèles. Une des forces de CC est sa "composabilité", dans le sens où, si deux étapes (*e.g.* confidentialité et intégrité) sont prouvées sûres dans CC, leur composition est aussi sûre. Ainsi, les preuves de sécurité se trouvent simplifiées car dès qu'une propriété d'une ressource est prouvée, elle se transfère naturellement lors de l'ajout de fonctionnalités supplémentaires.

Contributions

Interactivité dans Cryptographie Constructive. Nous étendons le modèle CC pour le rendre capable de manipuler les protocoles interactifs. Par interactif, nous entendons les protocoles qui impliquent au moins deux participants qui se mettent d'accord pour effectuer des calculs spécifiés par le protocole. Nous concevons et construisons l'*Interactive Server-Memory Resource* (ISMR), qui est une version améliorée de l'instanciation basique des protocoles client-serveur dans CC, appelée Server-Memory Resource dans [BM18]. Nous rendons cette modélisation de l'interactivité possible en proposant de nouvelles manières d'utiliser les convertisseurs et les simulateurs. Concernant les convertisseurs : en plus de leurs capacités de base, ils peuvent maintenant être utilisés pour modifier le nombre de paramètres de leurs requêtes à une interface donnée. De plus, les convertisseurs ont maintenant le pouvoir de désactiver les capacités d'une interface. Nous introduisons et modélisons également les interfaces dites *semi-honest*, dans lesquelles un convertisseur et un simulateur peuvent être branchés. Cela nous permet de modéliser et de décrire précisément le comportement des adversaires honnêtes mais curieux, mais aussi celui des byzantins. En utilisant la flexibilité du modèle CC équipé de nos nouveaux outils, nous donnons une modélisation composable des protocoles de PIR qui unifie ses différentes variantes : IT-PIR, cPIR et PIR à un seul ou plusieurs serveurs. Enfin, nous appliquons notre construction du ISMR à UE pour obtenir les contributions décrites ci-dessous.

Modélisation composable et notions de sécurité du Chiffrement avec Mise à Jour. Nous proposons la première modélisation composable des schémas UE. En effet, dans le seul travail concurrent (à notre connaissance), qui considère UE dans le contexte CC [FMM21], la construction proposée est seulement valide pour des instanciations particulières d'UE. Afin de donner une modélisation composable et générique pour UE, nous avons dû introduire et

raffiner des idées nouvelles et préexistantes. Tout d'abord, nous mettons à profit les abstractions permises par CC, nos nouveaux outils pour modéliser les protocoles interactifs et aussi le "global event history" de [JMM19] pour proposer une modélisation simple d'UE dénuée de détails non nécessaires, voire gênants. Ensuite, nous améliorons une technique de preuve de [CMT13] en proposant un double argument hybride pour prouver la sécurité de nos constructions dans le modèle CC. Notre technique de preuve est intéressante car elle permet de prouver des énoncés relatifs aux conjonctions de notions de sécurité et aux notions de sécurité avec des challenges de types différents. Par exemple, pour la notion de sécurité IND-UE, le challenge peut appartenir soit à l'espace des clairs soit à celui des chiffrés. Notre double argument hybride permet de gérer cette asymétrie. Enfin, nous écartons l'ennuyeux problème du "commitment" : un résultat d'impossibilité qui apparaît quand on s'intéresse aux fuites de clés dans les modèles composables. Pour ce faire, nous adaptons les garanties de sécurité par intervalles de [JM20] à UE en proposant des énoncés de sécurité distincts au sein de plusieurs intervalles de temps. En choisissant soigneusement les énoncés et les intervalles, nous sommes en mesure d'éviter ce résultat d'impossibilité tout en prouvant des énoncés de sécurité significatifs.

Nous utilisons notre modélisation d'UE pour étudier ses différentes notions de sécurité. Celles-ci sont nombreuses et en constante évolution depuis l'introduction d'UE en 2013. Nous nous concentrons sur les notions IND-ENC de [LT18], IND-UPD de [LT18] et IND-UE de [BDGJ20]. Notre modélisation nous permet d'identifier deux contexte de fuite intéressants pour UE et d'étudier la notion de sécurité la plus adaptée à chacun d'eux. Dans le contexte de fuite *sans restrictions*, l'adversaire peut consulter les chiffrés distants comme il l'entend. En revanche, dans le contexte de fuite *restreint*, l'adversaire ne peut consulter un chiffré donné qu'une seule fois entre deux mise à jour de la clé. Dans ces deux contextes, l'adversaire peut compromettre les clés et les tokens librement et nous précisons les garanties de sécurité post-compromission données par chaque notion de sécurité. En outre, nous montrons que la sécurité IND-UE-CPA est suffisante, et aussi nécessaire, pour une construction sûre d'un ISMR confidentiel qui protège l'âge des chiffrés dans le contexte de fuite restreint; et que la notion IND-ENC-CPA + IND-UPD-CPA est suffisante, et aussi nécessaire, pour une construction sûre d'un ISMR confidentiel dans le contexte de fuite sans restrictions (où l'âge des chiffrés est impossible à protéger). Par âge d'un chiffré, nous entendons le nombre de fois où il a été mis à jour. Nous faisons la même chose pour les notions de sécurité CCA où l'adversaire est capable d'injecter des données arbitraires sur le serveur. En particulier, nous montrons que la notion de sécurité IND-UE de [BDGJ20] n'est pas toujours strictement plus forte que celle de [LT18], à savoir IND-ENC + IND-UPD.

Nouvelles constructions de Chiffrement avec Mise à Jour post-quantique. Cette thèse résout deux problèmes ouverts concernant l'UE post-quantique où le token est indépendant des chiffrés. Tout d'abord, nous proposons le premier schéma UE post-quantique qui permette d'effectuer un nombre illimité de mises à jour de clé. Ensuite, la sécurité de notre schéma repose sur un problème basé sur les isogénies, alors que les deux seuls schémas UE post-quantique connus, proposés par Jiang [Jia20] et Nishimaki [Nis22], reposent tous les deux sur la difficulté du problème Learning with Errors (LWE) [Reg05]. Nous y arrivons en étudiant le problème consistant à construire des schémas UE dans le cadre des actions de groupe. Nous proposons deux nouvelles constructions génériques d'UE dans ce cadre. Nous montrons comment utiliser des actions de groupe pour construire un schéma UE post-quantique atteignant le niveau de sécurité CCA.

Pour commencer, nous introduisons la nouvelle notion de Mappable Effective Group Action (MEGA) et nous montrons comment construire un schéma UE avec une MEGA en généralisant la construction SHINE de Boyd *et al.* [BDGJ20]. Nous appelons cette première construction GAINE pour Group-Action Ideal-cipher Nonce-based Encryption. Pour pouvoir effectuer cette

construction, nous demandons à ce qu'il existe une fonction efficace et inversible allant de l'espace des messages vers l'ensemble de l'action de groupe. Autrement, nous n'attendons presque aucune propriété de l'action de groupe. En particulier, nous n'avons pas besoin qu'elle soit libre ou transitive, et le groupe n'a même pas besoin d'être abélien. Malheureusement, nous ne pouvons pas encore instancier GAINE dans le contexte post-quantique. Pouvoir le faire résoudrait le problème ouvert laissé par Jiang à Asiacrypt 2020 [Jia20].

Les actions de groupes utilisant des isogénies sont les actions post-quantiques les plus étudiées. Bien qu'il existe des discussions en cours sur leur niveau de sécurité exact, nous pouvons avoir une certaine confiance dans le fait que les problèmes sous-jacents sont difficiles. Malheureusement, dans le cas de CSIDH [CLM+18], il est notoirement difficile d'échantillonner des éléments dans l'ensemble sur lequel le groupe agit [BBD+22]. Il est donc pour le moment impossible d'instancier GAINE avec CSIDH ou toute autre action de groupe similaire utilisant des isogénies. Toutefois, nous montrons comment il est tout de même possible de construire un schéma UE en utilisant CSIDH. Nous contournons la nécessité d'échantillonner dans l'ensemble en peaufinant une idée de Moriya *et al.* pour leur schéma de chiffrement SIGAMAL [MOT20]. Nous obtenons un analogue de CSIDH en considérant un ensemble constitué d'éléments comprenant une courbe et un point (et non juste une courbe). La manière dont nous contournons cet obstacle est intéressante indépendamment d'UE car ce problème est présent dans de nombreux autres protocoles. Nous avons extrait un cadre abstrait de notre idée en identifiant la structure algébrique requise pour notre nouveau schéma UE. Nous avons obtenu ce que nous appelons TOGA pour Triple Orbital Groupe Action. Comme le nom le suggère, trois actions de groupe sont utilisées dans notre schéma, chacune ayant un rôle spécifique et les trois devant interagir entre elles d'une manière bien spécifique. Nous nommons notre schéma UE générique reposant sur ce concept TOGA-UE.

Constructions nouvelles et améliorées pour les Preuves de Récupérabilité reposant sur les codes. Tout d'abord, nous montrons que la sécurité d'un schéma de PoR basé sur des codes de Lavauzelle et Levy-dit-Vehel [LLDV16] était surestimée. Pour réparer ce schéma, nous proposons un cadre pour concevoir des schémas de PoR efficaces et sûrs en utilisant des codes localement corrigibles (LCC). Nous donnons une première instanciation de notre cadre en utilisant les codes relevés à haut rendement de Guo *et al.* [GKS13]. Nous obtenons une famille infinie de bons PoRs, qui peut être vue comme une généralisation sûre du PoR de [LLDV16]. Notre PoR dispose d'une complexité de communication faible, d'un petit surplus de stockage et de garanties de sécurité claires. Nous estimons sa sécurité en résolvant un problème de géométrie finie. Plus précisément, nous caractérisons les configurations de corruptions qu'un adversaire peut introduire pour empêcher le client de récupérer son fichier. Elles forment ce que nous appelons des ensembles couvrants à l'ordre d d'un espace affine fini, où l'entier d est un paramètre du code utilisé par le PoR. En voici une définition.

Definition 0.2 (Ensembles couvrants à l'ordre d)**.** *Soient $\mathbb{F}$ un corps fini et m, d deux entiers positifs. On dit qu'un sous-ensemble $S \subseteq \mathbb{F}^m$ est un ensemble couvrant à l'ordre d si S vérifie la propriété suivante :*

$$\forall s \in S, \forall \text{ droite } \ell \subseteq \mathbb{F}^m \text{ passant par } s, |S \cap \ell| \geq d$$

Ou de manière équivalente : Pour toute droite $\ell \subseteq \mathbb{F}^m, |S \cap \ell| = 0$ ou $|S \cap \ell| \geq d$.

Nous trouvons une borne inférieure concernant le nombre de droites intersectant ces ensembles couvrants à l'ordre d et nous l'utilisons pour estimer la sécurité de notre schéma. Puis, en utilisant les propriétés locales des codes basés sur les graphes de Tanner [Tan81], nous obtenons

une nouvelle instanciation de notre cadre et nous estimons encore une fois sa sécurité. Cette fois ci, étant donnés un graphe G et un entier d, nous montrons que les configurations de corruptions qui permettent à un adversaire d'empêcher le client de récupérer son fichier correspondent exactement aux sous-graphes de G de degré minimal d.

Par ailleurs, en utilisant une approche différente, nous proposons un PoR efficace basé sur la famille des codes expanseurs [Tan81, SS96]. Nous utilisons des codes expanseurs reposant sur des graphes construits à partir des relations d'incidence point-droite dans les espaces affines finis. Høholdt *et al.* [HJ06, HJ11, BHPJ13] ont montré que, en utilisant un code de Reed-Solomon comme code intérieur, ces codes ont une bonne dimension ainsi qu'une bonne distance minimale tout en ayant un alphabet relativement petit. De plus, les codes expanseurs possèdent des algorithmes de décodage très efficaces [SS96, Zém01]. Nous tirons parti de ces résultats pour proposer un schéma de PoR qui récupère le fichier du client en temps quasi-linéaire et qui affiche des paramètres concrets meilleurs que ceux des schémas de l'état de l'art, notamment en ce qui concerne la taille des fichiers et le surplus de stockage.

Composabilité des Codes Localement Corrigibles. Contrairement aux codes correcteurs d'erreurs classiques, les LCCs permettent de corriger un symbole du mot de code en ne lisant qu'un nombre sous-linéaire de ses symboles. Nous donnons une modélisation composable des LCCs dans CC. Nous obtenons un SMR résistant aux erreurs où le client peut lire des données avec une communication sous-linéaire. Ce faisant, nous montrons que la probabilité d'échec de décodeur local ne dépend pas seulement du nombres de corruptions présentes mais aussi de leur localisation sur le mot de code. Calculer cette probabilité d'échec, un problème souvent négligé dans la littérature, est nécessaire pour donner les garanties de sécurité de notre SMR et il pourrait être intéressant dans d'autres contextes. Par exemple, dans notre travail sur les PoRs à base de codes, nous avons montré qu'une compréhension précise des cas d'échec du décodeur local était essentielle pour estimer la sécurité de nos schémas. Pour finir, pour la classe importante des codes de Reed-Solomon relevés, nous montrons comment cette probabilité d'échec peut être calculée en temps polynomial en la longueur du code.

Contents

Notations

We explain common notations that are not given anywhere else in this document.

- Given a finite set S, sampling uniformly from S is denoted by $\xleftarrow{\$} S$.
- Given a distribution $\mathcal{D}$ on a finite set S, sampling from $\mathcal{D}$ is denoted by $\leftarrow \mathcal{D}$.
- The set of permutations of a finite set S is denoted by $\mathfrak{S}(S)$.
- Given an algorithm $\mathcal{A}$ and an oracle $\mathcal{O}$, $\mathcal{A}$ having access to $\mathcal{O}$ is denoted by $\mathcal{A}^{\mathcal{O}}$.
- We use PPT to mean that an algorithm is probabilistic and runs in polynomial time.
- We use $\mathbb{F}$ to denote finite fields. For example, $\mathbb{F}_q$ is the finite field of size q.
- We use $|\cdot|$ to give the length or size of an object.
- Given a vector $v := (v_1, \ldots, v_n)$ for some integer n and a set $I \subseteq \{1, \ldots n\}$, we use $v_{|I}$ to denote the vector $(v_i)_{i \in I}$.
- We use log for the logarithm in base 2.

Chapter 1

Background

In this chapter, we give precise definitions for the protocols and objects we are going to study. We also present the Constructive Cryptography model in which some of our contributions are going to be phrased.

1.1 Constructive Cryptography

We build upon the presentation of [JM20] to give a comprehensive and up to date overview of the CC model.

1.1.1 Global event history

This thesis uses the globally observable events introduced in [JMM19]. Formally, we consider a *global event history* $\mathcal{E}$ which is a list of event without duplicates. An event is defined by a name n, and triggering the event n corresponds to the action of appending n to $\mathcal{E}$, denoted by $\mathcal{E} \xleftarrow{+} \mathcal{E}_n$. For short, we use the notation $\mathcal{E}_n$ to say that event n happened. Finally, $\mathcal{E}_n \prec \mathcal{E}_{n'}$ means that the event n precedes n' in the event history.

1.1.2 Resources, converters and distinguishers

A *resource* $\mathbf{R}$ is a system that interacts, in a black-box manner, at one or more of its *interfaces*, by receiving an input at a given interface and subsequently sending an output at the same interface. For example, one can imagine a **Coin** resource where a party can send the request `throw` at interface `I` to obtain an output. The nature and the distribution of outputs is given in the resource's specification (often given in pseudo-code). Moreover, parties can append parameters to their requests. We use the notation $(\texttt{request}, p) \in S$ to denote that the request `request` expects an additional parameter p belonging to a set S. Do note that a resource only defines the observable behavior of a system and not how it is defined internally. The behavior of the resource depends on the global event history $\mathcal{E}$ and it can append events to it. We use the notation $[\mathbf{R}_1, \ldots, \mathbf{R}_k]$ to denote the parallel composition of resources. It corresponds to a new resource and, if $\mathbf{R}_1, \ldots, \mathbf{R}_k$ have disjoint interfaces sets, the interface set of the composed resource is the union of those.

In CC, *converters* are used to link resources and reprogram interfaces, thus expressing the local computations of the parties involved. A converter is plugged in a set of interfaces at the inside and provides a set of interfaces at the outside. When it receives an input at its outside interface, the converter uses a bounded number of queries to the inside interface before computing a value and outputting it at its outside interface.

A converter π connected to the interface set $\mathcal{I}$ of a resource $\mathbf{R}$ yields a new resource $\mathbf{R}' := \pi^{\mathcal{I}}\mathbf{R}$. The interfaces of $\mathbf{R}'$ inside the set $\mathcal{I}$ are the interfaces emulated by π. A protocol can be modelled as a tuple of converters with pairwise disjoint interface sets.

A *distinguisher* $\mathbf{D}$ is an environment that connects to all interfaces of a resource $\mathbf{R}$ and sends queries to them. $\mathbf{D}$ has access to the global event history and can append events that cannot be added by $\mathbf{R}$. At any point, the distinguisher can end its interaction by outputting a bit. The advantage of a distinguisher is defined as

$$\Delta^{\mathbf{D}}(\mathbf{R}, \mathbf{S}) := |\Pr[\mathbf{D}^{\mathcal{E}}(\mathbf{R}) = 1] - \Pr[\mathbf{D}^{\mathcal{E}}(\mathbf{S}) = 1]|,$$

$\mathbf{D}^{\mathcal{E}}$ meaning that the distinguisher has oracle access to the global event history $\mathcal{E}$.

1.1.3 Specifications

An important concept of CC is the one of *specifications*. Systems are grouped according to desired or assumed properties that are relevant to the user, while other properties are ignored on purpose. A specification $\mathcal{S}$ is a set of resources that have the same interface set and share some properties, for example confidentiality. In order to construct this set of confidential resources, one can use a specification of assumed resources $\mathcal{R}$ and a protocol π, and show that the specification $\pi\mathcal{R}$ satisfies confidentiality. Proving security is thus proving that $\pi\mathcal{R} \subseteq \mathcal{S}$, sometimes written as $\mathcal{R} \xrightarrow{\pi} \mathcal{S}$, and we say that the protocol π constructs the specification $\mathcal{S}$ from the specification $\mathcal{R}$. The composition property of the framework comes from the transitivity of inclusion. Formally, for specifications $\mathcal{R}, \mathcal{S}$ and $\mathcal{T}$ and protocols π for $\mathcal{R}$ and π' for $\mathcal{S}$, we have $\mathcal{R} \xrightarrow{\pi} \mathcal{S} \wedge \mathcal{S} \xrightarrow{\pi'} \mathcal{T} \Rightarrow \mathcal{R} \xrightarrow{\pi' \circ \pi} \mathcal{T}$.

We use the real-world/ideal-world paradigm, and often refer to $\pi\mathcal{R}$ and $\mathcal{S}$ as the real and ideal-world specifications respectively, to understand security statements. Those statements say that the real-world is "just as good" as the ideal one, meaning that it does not matter whether parties interact with an arbitrary element of $\pi\mathcal{R}$ or one of $\mathcal{S}$. This means that the guarantees of the ideal specification $\mathcal{S}$ also apply in the real world where an assumed resource is used together with the protocol.

Since specifications are set of resources, we can consider the intersection $\mathcal{S} \cap \mathcal{T}$ of two specifications $\mathcal{S}$ and $\mathcal{T}$. The resulting specification possesses the guarantees of both $\mathcal{S}$ and $\mathcal{T}$.

Remark 1. In this thesis, we use *simulators*, *i.e.* converters that translate behaviors of the real world to the ideal world, to make the achieved security guarantees obvious. For example, one can model confidential servers as a specification $\mathcal{S}$ that only leaks the data length, combined with an arbitrary simulator σ, and show that $\pi\mathcal{R} \subseteq \sigma\mathcal{S}$. It is then clear that the adversary cannot learn anything more that the data length.

1.1.4 Relaxations

In order to talk about computational assumptions, post-compromise security or other security notions, the CC framework relies on *relaxations* which are mappings from specifications to larger, and thus weaker, *relaxed specifications*. The idea of relaxation is that, if we are happy with constructing specification $\mathcal{S}$ in some context, then we are also happy with constructing its relaxed variant. One common example of this is computational security. Let ϵ be a function that maps distinguishers $\mathbf{D}$ to the winning probability, in $[0, 1]$, of a modified distinguisher $\mathbf{D}'$ (the reduction) on the underlying computational problem. Formally,

Definition 1.1. *Let ϵ be a function that maps distinguishers to a value in $[0,1]$. Then, for a resource $\mathbf{R}$, the reduction relaxation $\mathbf{R}^{\epsilon}$ is defined as*

$$\mathbf{R}^{\epsilon} := \{\mathbf{S} \mid \forall \mathbf{D}, \Delta^{\mathbf{D}}(\mathbf{R}, \mathbf{S}) \leq \epsilon(\mathbf{D})\}$$

This (in fact any) relaxation can be extended to a specification $\mathcal{R}$ by defining $\mathcal{R}^{\epsilon} := \cup_{\mathbf{R}\in\mathcal{R}}\mathbf{R}^{\epsilon}$.

We use the notation $\mathbf{R} \equiv \mathbf{R}'$ to express that the two resources $\mathbf{R}$ and $\mathbf{R}'$ are indistinguishable (the advantage of any distinguisher is 0). The other relaxation that we will use is the *interval-wise relaxation* introduced in [JM20, Definition 9]. Given two predicates $P_1(\mathcal{E})$ and $P_2(\mathcal{E})$ on the global event history, the interval-wise relaxation $\mathbf{R}^{[P_1,P_2]}$ is the set of all resources that must behave like $\mathbf{R}$ in the time interval starting when $P_1(\mathcal{E})$ becomes true and ending when $P_2(\mathcal{E})$ becomes true. Outside this interval, we have no guarantees on how the resources behave.

These two relaxations have nice composition properties, mainly they are compatible together and with parallel and sequential protocol applications, as shown in [JM20, Theorems 13 & 14]. This means that all the constructions presented in this work can be used in a modular fashion inside bigger constructions, without needing to write a new security proof.

1.1.5 Server-memory resources

Since we work in the outsourced storage context, we describe its translation in CC. This translation was given in 2018 by Badertscher and Maurer [BM18].

The key resource is the basic server-memory resource (SMR) denoted by $\mathbf{SMR}_{\Sigma,n}$ where Σ is a finite alphabet and $n \in \mathbb{N}$ is the number of data blocks (the memory size). The **SMR** specification is given in Fig. 1.1. It is a modeling of basic client-server interactions. Indeed, it allows a client to read and write data blocks that are encoded as elements of Σ via interface C. The memory $\mathbb{M}$ is originally filled with the special symbol $\lambda \notin \Sigma$, signifying that it has not been written yet. These symbols can be overwritten at interface C_0 to initialize the memory. The server can be "honest but curious" by obtaining the entire history of accesses made by the client, through the log file HIST, and reading the memory at interface S_H. The server can also be intrusive and overwrite data using its interface S_I when the resource is set into a special write mode (*e.g.* when INTRUSION is set to `true`). This write mode can be toggled by the distinguisher at the world interface W. Throughout this document, we use the notation $\mathcal{P}$ to denote the set of honest parties $\{\mathsf{C}_0, \mathsf{C}\}$. Also, we often use the notation $[n]$ to denote the set $\{1, \ldots, n\}$.

For readers unfamiliar with CC, we explain how to read the construction statements of this model. Lets say that we want to construct an ideal SMR **idSMR** from a basic **SMR** using a converter $\pi := (\pi_1, \pi_2)$. The ideal SMR has augmented security guarantees, for example integrity or confidentiality. This construction translates to the following dummy theorem:

Theorem 1.1 (Dummy theorem)**.** *Let Σ be a finite alphabet and $n \in \mathbb{N}$. The protocol $\pi := (\pi_1, \pi_2)$ constructs the $\mathbf{idSMR}_{\Sigma,n}$ from the basic $\mathbf{SMR}_{\Sigma,n}$, with respect to the simulator sim and the pair $(\mathsf{honSrv}, \mathsf{honSrv})$. More precisely, for all distinguishers $\mathbf{D}$, we have*

$$\Delta^{\mathbf{D}}(\mathsf{honSrv}^{\mathsf{S}}\pi_{\mathcal{P}}\,\mathbf{SMR}_{\Sigma,n}, \mathsf{honSrv}^{\mathsf{S}}\mathbf{idSMR}_{\Sigma,n}) = 0$$
$$\textit{and} \quad \Delta^{\mathbf{D}}(\pi_{\mathcal{P}}\,\mathbf{SMR}_{\Sigma,n}, \mathsf{sim}^{\mathsf{S}}\mathbf{idSMR}_{\Sigma,n}) \leq \epsilon(\mathbf{D})$$

First, honSrv denotes a dummy converter that, when plugged in interface S, deactivates all the capabilities of the adversary. Then, the notation $\pi_{\mathcal{P}}$ denotes that π_1 is plugged in interface C_0 and π_2 is plugged in interface C. The first condition, called *availability*, captures the correctness of the converter π. Indeed, it expresses that the two resources $\pi_{\mathcal{P}}\,\mathbf{SMR}_{\Sigma,n}$

and $\mathbf{idSMR}_{\Sigma,n}$ behave in the exact same way in the absence of an adversary. This rules out unwanted protocols. For example, one could construct a confidential SMR by deleting the client's data and never uploading on the server. This protocol is secure but it does not meet the availability condition. We will sometimes omit this condition when our schemes are proven (or are clearly) correct.

The second condition is called *security*. It is illustrated, and explained, in Fig. 1.2, taken from [BM18]. This time, the adversary is not impeded by the honSrv converter. The simulator sim is usually described in the proof of the theorem. Given a distinguisher $\mathbf{D}$, the quantity $\epsilon(\mathbf{D})$ represents the distance between the ideal (the ideal SMR with the simulator) and real (the basic SMR with the converter) systems. Often, it is an upper bound of the advantage of a modified version of $\mathbf{D}$ in distinguishing both systems. This modification is due to a reduction specified in the proof. When using the theorem in a real use case, it is possible to restrict the class of distinguishers, and thus the range of values taken by $\epsilon(\mathbf{D})$. A common example of this is to only consider polynomial time distinguishers.

Resource $\mathbf{SMR}_{\Sigma,n}$

Initialization

INIT, ACTIVE, INTRUSION $\leftarrow$ `false`
HIST $\leftarrow$ [ ]

Interface C_0

Input: `init`
  if not INIT
    for $i = 1$ to n **do**
      $\mathbb{M}[i] \leftarrow \lambda$
    HIST $\leftarrow$ HIST $||$ $(0, \texttt{init})$
    INIT $\leftarrow$ `true`
Input: $(\texttt{read}, i) \in [1, n]$
  if INIT **and not** ACTIVE
    HIST $\leftarrow$ HIST $||$ $(0, \texttt{R}, i)$
    return $\mathbb{M}[i]$
Input: $(\texttt{write}, i, x) \in [1, n] \times \Sigma$
  if INIT **and not** ACTIVE
    HIST $\leftarrow$ HIST $||$ $(0, \texttt{W}, i, x)$
    $\mathbb{M}[i] \leftarrow x$
Input: `initComplete`
  ACTIVE $\leftarrow$ `true`

Interface S_H

Input: `getHist`
  return HIST
Input: $(\texttt{read}, i) \in [1, n]$
  return $\mathbb{M}[i]$

Interface S_I

Input: $(\texttt{write}, i, x) \in [1, n] \times \Sigma$
  if INTRUSION
    return $\mathbb{M}[i] \leftarrow x$

Interface W

Input: `startWriteMode`
  if ACTIVE
    INTRUSION $\leftarrow$ `true`
Input: `stopWriteMode`
  if ACTIVE
    INTRUSION $\leftarrow$ `false`

Interface C

Input: $(\texttt{read}, i) \in [1, n]$
  if ACTIVE **and not** INTRUSION
    HIST $\leftarrow$ HIST $||$ $(\texttt{R}, i)$
    return $\mathbb{M}[i]$

Input: $(\texttt{write}, i, x) \in [1, n] \times \Sigma$
  if ACTIVE **and not** INTRUSION
    HIST $\leftarrow$ HIST $||$ $(\texttt{W}, i, x)$
    $\mathbb{M}[i] \leftarrow x$

Figure 1.1: Description of the basic server-memory resource of [BM18].

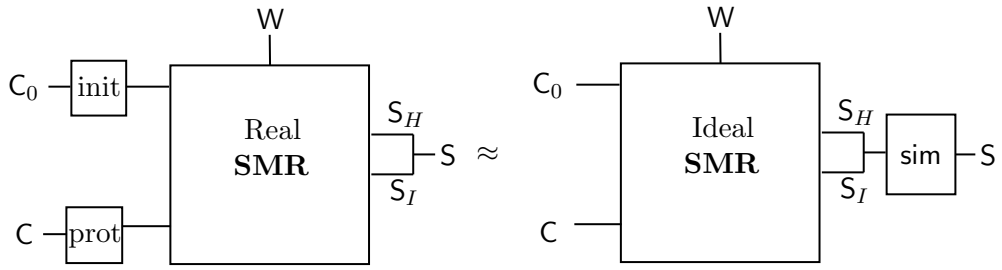

Figure 1.2: Illustration of the construction notion for SMRs. On the left, we have a real **SMR** with a protocol for the client. On the right, we have an ideal **SMR** with stronger security guarantees. The construction is secure if there exists a simulator that makes these two resources indistinguishable.

1.2 Proofs of Retrievability

PoRs are cryptographic protocols whose goal is to guarantee that a file stored by a client on a server remains retrievable in full. PoRs thus involve two parties: a client who owns a file F and a server on which F is stored. We start by giving an overview of the commonly used security model for PoRs by Juels and Kaliski [JK07]. We will then highlight its drawbacks and show how [BM18] corrected them using CC. A PoR scheme is composed of three main procedures:

- *An initialization phase.* The client encodes his file F with an initialization function $\mathbf{Init}(F) = (\tilde{F}, \texttt{data})$. He keeps `data` (*e.g.* keys, etc.) for himself, then he sends $\tilde{F}$ to the server and erases F.
- *A verification phase.* The client produces a challenge c with a randomized **Chall** function and sends it to the server. The latter creates a response $r = \mathbf{Resp}(\tilde{F}, c)$ and sends it back to the client. The client checks if r is correct by running $\mathbf{Verif}(c, r)$, which also access `data`, and outputs `accept` if r is considered correct and `reject` otherwise.
- *An extraction phase.* If the client has been convinced by the verification phase, he can use his **Extract** algorithm to recover his whole file with high probability.

The client wants to use the **Verif** procedure to be sure that he will be able to retrieve his file in full by using the **Extract** procedure. We want to model the fact that, if the server's answers to client's challenges make him look like he owns the file, then the client must be able to recover it entirely.

Definition 1.2 (ϵ-adversary)**.** *Let $\mathcal{P}$ be a PoR system and X be the space of challenges generated by* **Chall***. An ϵ-adversary $\mathcal{A}$ for $\mathcal{P}$ is an algorithm such that, for all files F,*

$$\Pr_{x \in X} \left[\mathbf{Verif}(x, \mathcal{A}(x)) = \texttt{false} \right] \leq \epsilon$$

The client models the server as an ϵ-adversary and uses his verification process to maintain an approximation of ϵ. Depending on this estimate, the client can decide whether his file is retrievable or not. We thus define a way to measure PoRs' security:

Definition 1.3 (PoR security)**.** *Let $\epsilon, \rho \in [0, 1]$. A PoR system is said to be (ϵ, ρ)-sound if, for all ϵ-adversaries $\mathcal{A}$ and for all files F, we have:*

$$\Pr\left[\mathbf{Extract}^{\mathcal{A}} = F\right] \geq \rho$$

where the probability is taken over the internal randomness of $\mathbf{Extract}^{\mathcal{A}}$*.*

As pointed out by Badertscher and Maurer in [BM18], this definition has a major drawback concerning client-side security guarantees. The most important thing for the client, the availability of his data, is conditioned to the execution of the **Extract** algorithm which needs to access the client's private data and the server's strategy (as indicated in the above definition). In practice, no server would reveal its entire state to a client. This problem is addressed in [BM18] by using CC to propose a definition of PoRs that corrects this drawback. In [BM18], the authors introduced an ideal abstraction of PoRs in the form of an ideal SMR that sees the client interface augmented with an `audit` mechanism. On an `audit` request, an `auditReq` request is first sent to interface S_H. If the server answers abort, the audit stops and the resource outputs `reject` at interface C. In the case when the server outputs allow, the following audit is executed. First, the resource checks whether the current memory content is indeed the newest version that the client wrote to the storage. If a single data block has changed, the audit will detect this and output `reject` to the client. In case of a successful audit (returning `accept`), this guarantee holds until the server gains write-access to the storage, in which case a new audit has to reveal whether modifications have been made.

We present the specification of the auditable server-memory resource $\mathbf{SMR}^{\mathsf{audit}}_{\Sigma,n}$ in Fig. 1.3. In addition to the advantages we discussed, we believe that this CC based security model is simpler and more intuitive than the one of ϵ-adversaries.

Resource $\mathbf{SMR}^{\mathsf{audit}}_{\Sigma,n}$

Interface C

Input: audit
  if ACTIVE **and not** INTRUSION
    output `auditReq` to S_H
    Let $d \in \{\text{allow}, \text{abort}\}$ be the result
    if $d = \text{allow}$
      $\mathbb{M}' \leftarrow [\,]$
      for $i = 1$ to n **do**
        if $\exists k, x : \text{HIST}[k] = (\mathtt{W}, i, x)$
          $k_0 \leftarrow \max\{k \mid \exists x : \text{HIST}[k] = (\mathtt{W}, i, x)\}$
          Parse $\text{HIST}[k_0]$ as $(\mathtt{W}, i, x_0)$
          $\mathbb{M}'[i] \leftarrow x_0$
        else
          $\mathbb{M}'[i] \leftarrow \lambda$
      if $\mathbb{M}' = \mathbb{M}$
        return `accept`
      else
        return `reject`
    else
      return `reject`

Figure 1.3: Description of the auditable server-memory resource of [BM18] (only the differences with **SMR** of Fig. 1.1 are shown)

In CC, a PoR scheme is given by a pair of converters $\mathsf{por} := (\mathsf{por}_{\mathsf{init}}, \mathsf{por}_{\mathsf{audit}})$ where $\mathsf{por}_{\mathsf{init}}$ implements the $(\mathtt{write}, F)$ query that uploads the client's file F (or an encoded/encrypted

version of F) on the SMR, and $\mathsf{por}_{\mathsf{audit}}$ implements the `audit` query that returns either `accept` or `reject`, and the `read` query that extracts the file F from the SMR. $\mathsf{por}_{\mathsf{init}}$ is connected to interface C_0 and $\mathsf{por}_{\mathsf{audit}}$ to interface C. A PoR scheme is considered secure if the por converter constructs $\mathbf{SMR}^{\mathtt{audit}}$ from $\mathbf{SMR}$.

1.3 Updatable Encryption

The syntax and security definitions of UE have been in constant evolution since the seminal paper of Boneh *et al.* [BLMR13]. We merge the presentations of [LT18, BDGJ20, Jia20, Nis22] to give up to date and comprehensive definitions for UE.

1.3.1 Syntax and correctness

A UE scheme operates in *epochs*, where an epoch is an index incremented with each key update. Let $n+1$ be the maximum number of epochs (this is only for proof purposes).

Definition 1.4. *An updatable encryption scheme* UE *for message space* $\mathcal{M}$ *consists of a tuple of PPT algorithms* ($\mathsf{UE.Setup}$, $\mathsf{UE.KeyGen}$, $\mathsf{UE.TokenGen}$, $\mathsf{UE.Enc}$, $\mathsf{UE.Dec}$, $\mathsf{UE.Upd}$) *where:*

- $\mathsf{UE.Setup}(1^\lambda) \to \mathsf{pp}$*: The setup algorithm takes as input the security parameter and outputs a public parameter* pp.
- $\mathsf{UE.KeyGen}(\mathsf{pp}) \to k_{\mathsf{e}}$*: The key generation algorithm takes as input the public parameter* pp *and outputs an epoch key* k_{e}.
- $\mathsf{UE.Enc}(k, m) \to c$*: The encryption algorithm takes as input an epoch key* k *and a message* m *and outputs a ciphertext* c.
- $\mathsf{UE.Dec}(k, c) \to m$*: The encryption algorithm takes as input an epoch key* k *and a ciphertext* c *and outputs a message* m *or* $\perp$.
- $\mathsf{UE.TokenGen}(k_{\mathsf{e}}, k_{\mathsf{e}+1}) \to \Delta_{\mathsf{e}+1}$*: The token generation algorithm takes as input two keys of consecutive epochs* e *and* $\mathsf{e}+1$ *and outputs a token* $\Delta_{\mathsf{e}+1}$.
- $\mathsf{UE.Upd}(\Delta_{\mathsf{e}+1}, c_{\mathsf{e}}) \to c_{\mathsf{e}+1}$*: The update algorithm takes as input a token* $\Delta_{\mathsf{e}+1}$ *and a ciphertext* c_{e} *and outputs a ciphertext* $c_{\mathsf{e}+1}$.

Correctness definitions for symmetric encryption can naturally be extended to UE by asking that updated ciphertexts still decrypt to the plaintexts given to Enc when the original ciphertexts were first created.

Definition 1.5 (Correctness)**.** *For any* $m \in \mathcal{M}$*, for* $0 \le \mathsf{e}_1 \le \mathsf{e}_2 \le n+1$*, it holds that*

$$\Pr[\mathsf{UE.Dec}(k_{\mathsf{e}_2}, c_{\mathsf{e}_2}) \neq m] \le \mathsf{negl}(\lambda)$$

where $\mathsf{pp} \leftarrow \mathsf{UE.Setup}(1^\lambda)$*,* $k_{\mathsf{e}_1}, \ldots, k_{\mathsf{e}_2} \leftarrow \mathsf{UE.KeyGen}(\mathsf{pp})$*,* $c_{\mathsf{e}_1} \leftarrow \mathsf{UE.Enc}(k_{\mathsf{e}_1}, m)$*, and* $\Delta_{i+1} \leftarrow \mathsf{UE.TokenGen}(k_i, k_{i+1})$*,* $c_{i+1} \leftarrow \mathsf{UE.Upd}(\Delta_{i+1}, c_i)$ *for* $i \in [\mathsf{e}_1, \mathsf{e}_2 - 1]$.

1.3.2 Security notions

In previous works, the security of UE is described using security games. We will study the $\mathsf{IND\text{-}ENC} + \mathsf{IND\text{-}UPD}$ security notion of [LT18] as well as the $\mathsf{IND\text{-}UE}$ notion of [BDGJ20]. First, we give an informal description of the security games for each notion:

- In $\mathsf{IND\text{-}UE}$ security, when given a plaintext m and a ciphertext c from a previous epoch encrypting a message of length $|m|$, the game challenges the adversary with either an encryption of m or an update of c.

- In IND-ENC security, when given two plaintexts m_0 and m_1 of the same length, the game challenges the adversary with an encryption of one of them.
- In IND-UPD security, when given two old ciphertexts c_0 and c_1 encrypting two messages of the same length, the game challenges the adversary with an update of one of them.

Formally, these notions are defined as follows:

Definition 1.6 (xxIND-yy-atk [BDGJ20])**.** *Let* UE := (UE.Setup, UE.KeyGen, UE.TokenGen, UE.Enc, UE.Dec, UE.Upd) *be an updatable encryption scheme. The* xxIND-yy-atk *advantage, for* $\mathsf{xx} \in \{\mathsf{det}, \mathsf{rand}\}$, $\mathsf{yy} \in \{\mathsf{UE}, \mathsf{ENC}, \mathsf{UPD}\}$ *and* $\mathsf{atk} \in \{\mathsf{CPA}, \mathsf{CCA}\}$ *of an adversary* $\mathcal{A}$ *against* UE *is given by*

$$\mathbf{Adv}^{\mathsf{xxIND\text{-}yy\text{-}atk}}_{\mathsf{UE},\mathcal{A}}(\lambda) := \left|\Pr[\mathbf{Exp}^{\mathsf{xxIND\text{-}yy\text{-}atk\text{-}0}}_{\mathsf{UE},\mathcal{A}} = 1] - \Pr[\mathbf{Exp}^{\mathsf{xxIND\text{-}yy\text{-}atk\text{-}1}}_{\mathsf{UE},\mathcal{A}} = 1]\right|$$

where the confidentiality experiment $\mathbf{Exp}^{\mathsf{xxIND\text{-}yy\text{-}atk}\text{-}b}_{\mathsf{UE},\mathcal{A}}$ *is given in Fig. 1.4.*

$\mathbf{Exp}^{\mathsf{xxIND\text{-}yy\text{-}atk}\text{-}b}_{\mathsf{UE},\mathcal{A}}(\lambda)$

1. **do** $\mathsf{UE.Setup}(1^\lambda)$
2. $\mathtt{ors} \leftarrow \mathcal{O}.\{\mathsf{Enc}, \mathsf{Upd}, \mathsf{Next}, \mathsf{Corr}\}$
3. **if** $\mathsf{atk} = \mathsf{CCA}$
4. $\quad \mathtt{ors} \leftarrow \mathtt{ors} \cup \{\mathcal{O}.\mathsf{Dec}\}$
5. $\mathsf{CHALL} \leftarrow \mathcal{A}^{\mathtt{ors}}(1^\lambda)$
6. $\tilde{C}_\mathsf{e} \leftarrow \mathcal{O}.\mathsf{Chall}(\mathsf{CHALL})$
7. $b' \leftarrow \mathcal{A}^{\mathtt{ors},\mathcal{O}.\mathsf{Upd}\tilde{\mathsf{C}}}(\tilde{C}_\mathsf{e})$
8. **if** $\mathcal{K}^* \cap \mathcal{C}^* \neq \emptyset$ **or** $(\mathsf{xx} = \mathsf{det}$ **and** $\mathcal{I}^* \cap \mathcal{C}^* \neq \emptyset)$
9. $\quad \mathsf{twf} \leftarrow 1$
10. **if** $\mathsf{twf} = 1$
11. $\quad b' \xleftarrow{\$} \{0,1\}$
12. **return** b'

Figure 1.4: Description of the confidentiality experiment $\mathbf{Exp}^{\mathsf{xxIND\text{-}yy\text{-}atk}\text{-}b}_{\mathsf{UE},\mathcal{A}}$ for scheme UE and adversary $\mathcal{A}$, for $\mathsf{xx} \in \{\mathsf{det}, \mathsf{rand}\}$, $\mathsf{yy} \in \{\mathsf{UE}, \mathsf{ENC}, \mathsf{UPD}\}$ and $\mathsf{atk} \in \{\mathsf{CPA}, \mathsf{CCA}\}$. As in [BDGJ20], we do not consider nor define randIND-yy-CCA. The oracles are given in Fig. 1.5 and the respective challenge oracles are given in Fig. 1.6, Fig. 1.7 and Fig. 1.8. Trivial win conditions, *i.e.* deciding the value of twf and computing $\mathcal{K}^*$, $\mathcal{C}^*$, $\mathcal{I}^*$ are discussed in Section 1.3.4 and Section 1.3.5

1.3.3 Ciphertext integrity game: definitions and composition result.

In the ciphertext integrity (CTXT) game, the adversary $\mathcal{A}$ is given access to oracles $\mathcal{O}$.Enc, $\mathcal{O}$.Next, $\mathcal{O}$.Upd and $\mathcal{O}$.Corr. At some point $\mathcal{A}$ attempts to provide a ciphertext forgery via the oracle $\mathcal{O}$.Try defined in Fig. 1.9. $\mathcal{A}$ wins the game if its forgery is valid, *i.e.* if it decrypts to a message and not $\perp$. If $\mathcal{A}$ is allowed to ask a single $\mathcal{O}$.Try query, we speak of the $\mathsf{INT\text{-}CTXT}^\mathsf{s}$ notion. If $\mathcal{A}$ can send multiple $\mathcal{O}$.Try queries, we speak of the INT-CTXT notion instead. $\mathsf{INT\text{-}CTXT}^\mathsf{s}$ and INT-CTXT are proved to be equivalent in [BDGJ20, Lemma 1]. Thus, we only define the $\mathsf{INT\text{-}CTXT}^\mathsf{s}$ advantage in Definition 1.7.

Setup(1^λ)

1. $\mathsf{pp} \leftarrow \mathsf{UE.Setup}(1^\lambda)$
2. $k_0 \leftarrow \mathsf{UE.KeyGen}(\mathsf{pp})$
3. $\Delta_0 \leftarrow \perp$
4. $\mathsf{e}, \mathsf{c} \leftarrow 0$
5. $\mathsf{phase}, \mathsf{twf} \leftarrow 0$
6. $\mathcal{L}, \tilde{\mathcal{L}}, \mathcal{C}, \mathcal{K}, \mathcal{T} \leftarrow \emptyset$

$\mathcal{O}.\mathsf{Enc}(M)$

1. $C \leftarrow \mathsf{UE.Enc}(k_\mathsf{e}, M)$
2. $\mathsf{c} \leftarrow \mathsf{c} + 1$
3. $\mathcal{L} \leftarrow \mathcal{L} \cup \{(\mathsf{c}, C, \mathsf{e})\}$
4. **return** C

$\mathcal{O}.\mathsf{Dec}(C)$

1. **if** $\mathsf{phase} = 1$ **and** $C \in \tilde{\mathcal{L}}$
2. $\quad \mathsf{twf} \leftarrow 1$
3. M **or** $\perp \leftarrow \mathsf{UE.Dec}(k_\mathsf{e}, C)$
4. **return** M **or** $\perp$

$\mathcal{O}.\mathsf{Next}$

1. $\mathsf{e} \leftarrow \mathsf{e} + 1$
2. $k_\mathsf{e} \leftarrow \mathsf{UE.KeyGen}(\mathsf{pp})$
3. $\Delta_\mathsf{e} \leftarrow \mathsf{UE.TokenGen}(k_{\mathsf{e}-1}, k_\mathsf{e})$
4. **if** $\mathsf{phase} = 1$
5. $\quad \tilde{C}_\mathsf{e} \leftarrow \mathsf{UE.Upd}(\Delta_\mathsf{e}, \tilde{C}_{\mathsf{e}-1})$

$\mathcal{O}.\mathsf{Upd}(C_{\mathsf{e}-1})$

1. **if** $(j, C_{\mathsf{e}-1}, \mathsf{e}-1) \notin \mathcal{L}$
2. $\quad$ **return** $\perp$
3. $C_\mathsf{e} \leftarrow \mathsf{UE.Upd}(\Delta_\mathsf{e}, C_{\mathsf{e}-1})$
4. $\mathcal{L} \leftarrow \mathcal{L} \cup \{(j, C_\mathsf{e}, \mathsf{e})\}$
5. **return** C_e

$\mathcal{O}.\mathsf{Corr}(\mathsf{inp}, \hat{\mathsf{e}})$

1. **if** $\hat{\mathsf{e}} > \mathsf{e}$
2. $\quad$ **return** $\perp$
3. **if** $\mathsf{inp} = \mathsf{key}$
4. $\quad \mathcal{K} \leftarrow \mathcal{K} \cup \{\hat{\mathsf{e}}\}$
5. $\quad$ **return** $k_{\hat{\mathsf{e}}}$
6. **if** $\mathsf{inp} = \mathsf{token}$
7. $\quad \mathcal{T} \leftarrow \mathcal{T} \cup \{\hat{\mathsf{e}}\}$
8. $\quad$ **return** $\Delta_{\hat{\mathsf{e}}}$

$\mathcal{O}.\mathsf{Upd}\tilde{\mathsf{C}}$

1. **if** $\mathsf{phase} \neq 1$
2. $\quad$ **return** $\perp$
3. $\mathcal{C} \leftarrow \mathcal{C} \cup \{\mathsf{e}\}$
4. $\tilde{\mathcal{L}} \leftarrow \tilde{\mathcal{L}} \cup \{(\tilde{C}_\mathsf{e}, \mathsf{e})\}$
5. **return** $\tilde{C}_\mathsf{e}$

Figure 1.5: Oracles in security games for UE with deterministic updates. Computing the leakage sets is discussed in Section 1.3.4

$\mathcal{O}.\mathsf{Chall}(\bar{M}, \bar{C})$

1. **if** $\mathsf{phase} \neq 1$
2. $\quad$ **return** $\perp$
3. $\mathsf{phase} \leftarrow 1$
4. $\tilde{\mathsf{e}} \leftarrow \mathsf{e}$
5. **if** $(\cdot, \bar{C}, \mathsf{e}-1) \notin \mathcal{L}$ **or** $|\mathsf{UE.Dec}(k_{\mathsf{e}-1}, \bar{C})| \neq |\bar{M}|$
6. $\quad$ **return** $\perp$
7. **if** $b = 0$
8. $\quad \tilde{C}_\mathsf{e} \leftarrow \mathsf{UE.Enc}(k_\mathsf{e}, \bar{M})$
9. **else** $(b = 1)$
10. $\quad \tilde{C}_\mathsf{e} \leftarrow \mathsf{UE.Upd}(\Delta_\mathsf{e}, \bar{C})$
11. $\mathcal{C} \leftarrow \mathcal{C} \cup \{\mathsf{e}\}$
12. $\tilde{\mathcal{L}} \leftarrow \tilde{\mathcal{L}} \cup \{(\tilde{C}_\mathsf{e}, \mathsf{e})\}$
13. **return** $\tilde{C}_\mathsf{e}$

Figure 1.6: Challenge oracle for IND-UE games.

$\mathcal{O}.\mathsf{Chall}(\bar{M}_0, \bar{M}_1)$

1. **if** $\mathsf{phase} \neq 1$
2. **return** $\perp$
3. $\mathsf{phase} \leftarrow 1$
4. $\tilde{\mathsf{e}} \leftarrow \mathsf{e}$
5. **if** $|\bar{M}_0| \neq |\bar{M}_1|$
6. **return** $\perp$
7. $\tilde{C}_{\mathsf{e}} \leftarrow \mathsf{UE.Enc}(k_{\mathsf{e}}, \bar{M}_b)$
8. $\mathcal{C} \leftarrow \mathcal{C} \cup \{\mathsf{e}\}$
9. $\tilde{\mathcal{L}} \leftarrow \tilde{\mathcal{L}} \cup \{(\tilde{C}_{\mathsf{e}}, \mathsf{e})\}$
10. **return** $\tilde{C}_{\mathsf{e}}$

Figure 1.7: Challenge oracle for IND-ENC games.

$\mathcal{O}.\mathsf{Chall}(\bar{C}_0, \bar{C}_1)$

1. **if** $\mathsf{phase} \neq 1$
2. **return** $\perp$
3. $\mathsf{phase} \leftarrow 1$
4. $\tilde{\mathsf{e}} \leftarrow \mathsf{e}$
5. **if** $(\cdot, \bar{C}_0, \mathsf{e}-1) \notin \mathcal{L}$ **or** $(\cdot, \bar{C}_1, \mathsf{e}-1) \notin \mathcal{L}$ **or** $|\mathsf{UE.Dec}(k_{\mathsf{e}-1}, \bar{C}_0)| \neq |\mathsf{UE.Dec}(k_{\mathsf{e}-1}, \bar{C}_1)|$
6. **return** $\perp$
7. $\tilde{C}_{\mathsf{e}} \leftarrow \mathsf{UE.Upd}(\Delta_{\mathsf{e}}, \bar{C}_b)$
8. $\mathcal{C} \leftarrow \mathcal{C} \cup \{\mathsf{e}\}$
9. $\tilde{\mathcal{L}} \leftarrow \tilde{\mathcal{L}} \cup \{(\tilde{C}_{\mathsf{e}}, \mathsf{e})\}$
10. **return** $\tilde{C}_{\mathsf{e}}$

Figure 1.8: Challenge oracle for IND-UPD games.

$\mathcal{O}.\mathsf{Try}(\tilde{C})$

1. **if** $\mathsf{phase} = 1$
2. **return** $\perp$
3. $\mathsf{phase} \leftarrow 1$
4. **if** $\mathsf{e} \in \mathcal{K}^*$ **or** $\tilde{C} \in \mathcal{L}^*$
5. $\mathsf{twf} \leftarrow 1$
6. M **or** $\perp \leftarrow \mathsf{UE.Dec}(k_{\mathsf{e}}, \tilde{C})$
7. **if** $M \neq \perp$
8. $\mathsf{win} \leftarrow 1$

Figure 1.9: The oracle $\mathcal{O}.\mathsf{Try}$ for the $\mathsf{INT\text{-}CTXT}^{\mathsf{s}}$ security notion.

Definition 1.7 ([BDGJ20]). *Let* $\mathsf{UE} = \{\mathsf{UE.KeyGen}, \mathsf{UE.TokenGen}, \mathsf{UE.Enc}, \mathsf{UE.Dec}, \mathsf{UE.Upd}\}$ *be a UE scheme. The* $\mathsf{INT\text{-}CTXT^s}$ *advantage of an adversary* $\mathcal{A}$ *against* UE *is defined as*

$$\mathbf{Adv}^{\mathsf{INT\text{-}CTXT^s}}_{\mathsf{UE},\mathcal{A}}(\lambda) := \Pr[\mathbf{Exp}^{\mathsf{INT\text{-}CTXT^s}}_{\mathsf{UE},\mathcal{A}} = 1]$$

where the experiment $\mathbf{Exp}^{\mathsf{INT\text{-}CTXT^s}}_{\mathsf{UE},\mathcal{A}}$ *is given in Fig. 1.10.*

$\mathbf{Exp}^{\mathsf{INT\text{-}CTXT^s}}_{\mathsf{UE},\mathcal{A}}(\lambda)$

1. **do** $\mathsf{UE.Setup}(1^\lambda)$
2. $\mathsf{win} \leftarrow 0$
3. $\mathcal{A}^{\mathcal{O}.\mathsf{Enc},\mathcal{O}.\mathsf{Next},\mathcal{O}.\mathsf{Upd},\mathcal{O}.\mathsf{Corr},\mathcal{O}.\mathsf{Try}}(\lambda)$
4. **if** $\mathsf{twf} = 1$
5. $\quad \mathsf{win} \leftarrow 0$
6. **return** win

Figure 1.10: The $\mathsf{INT\text{-}CTXT^s}$ experiment for UE scheme UE and adversary $\mathcal{A}$. Trivial win conditions are discussed in Section 1.3.5.

Composition result for CPA, CTXT and CCA security. In [BDGJ20, Theorem 3], Boyd *et al.* show the following generic composition result for UE: CPA + CTXT ⇒ CCA. We will use this result to prove that our new UE scheme $\mathsf{GAINE0}$ can be made $\mathsf{detIND\text{-}UE\text{-}CCA}$ secure in Chapter 2.

1.3.4 Leakage sets

We follow the bookkeeping technique [LT18, BDGJ20] to maintain the epoch leakage sets.

- $\mathcal{C}$: List of epochs in which the adversary learned an updated version of the challenge ciphertext (from $\mathcal{O}.\mathsf{Chall}$ or $\mathcal{O}.\mathsf{Upd}\tilde{\mathsf{C}}$).
- $\mathcal{K}$: List of epochs in which the adversary corrupted the encryption key.
- $\mathcal{T}$: List of epochs in which the adversary corrupted the update token.

The adversary can also learn the values of ciphertexts and their updates.

- $\mathcal{L}$: List of non-challenge ciphertexts (from $\mathcal{O}.\mathsf{Enc}$ or $\mathcal{O}.\mathsf{Upd}$) with entries of the form $(\mathsf{c}, C, \mathsf{e})$, where c is a counter incremented with each $\mathcal{O}.\mathsf{Enc}$ query.
- $\tilde{\mathcal{L}}$: List of updated versions of challenge ciphertext (created by $\mathcal{O}.\mathsf{Next}$ and returned by $\mathcal{O}.\mathsf{Upd}\tilde{\mathsf{C}}$), with entries of the form $(\tilde{C}, \mathsf{e})$.

1.3.5 Trivial wins

Trivial wins via keys and ciphertexts. The following holds for all security notions. We consider the extended epoch leakage sets $\mathcal{C}^*$, $\mathcal{K}^*$ and $\mathcal{T}^*$ inferred from $\mathcal{C}$, $\mathcal{K}$ and $\mathcal{T}$. These extended sets are used to identify trivial wins, *i.e.* if $\mathcal{C}^* \cap \mathcal{K}^* \neq \emptyset$, then there exists an epoch in which the adversary knows the epoch key and a valid update of the challenge ciphertext. The challenger computes these sets once the adversary has finished running. Using [LT18], we show how to compute the extended epoch leakage sets $\mathcal{C}^*$, $\mathcal{K}^*$ and $\mathcal{T}^*$:

$$
\begin{aligned}
\mathcal{K}^* &\leftarrow \{\mathsf{e} \in \{0,\dots,n\} \mid \mathsf{CorrK}(\mathsf{e}) = \mathsf{true}\} \\
&\mathsf{true} \leftarrow \mathsf{CorrK}(\mathsf{e}) \Leftrightarrow (\mathsf{e} \in \mathcal{K}) \vee (\mathsf{CorrK}(\mathsf{e}-1) \wedge \mathsf{e} \in \mathcal{T}) \vee (\mathsf{CorrK}(\mathsf{e}+1) \wedge \mathsf{e}+1 \in \mathcal{T}) \\
\mathcal{T}^* &\leftarrow \{\mathsf{e} \in \{0,\dots,n\} \mid (\mathsf{e} \in \mathcal{T}) \vee (\mathsf{e} \in \mathcal{K}^* \wedge \mathsf{e}-1 \in \mathcal{K}^*)\} \\
\mathcal{C}^* &\leftarrow \{\mathsf{e} \in \{0,\dots,n\} \mid \mathsf{ChallEq}(\mathsf{e}) = \mathsf{true}\} \\
&\mathsf{true} \leftarrow \mathsf{ChallEq}(\mathsf{e}) \Leftrightarrow (\mathsf{e} = \tilde{\mathsf{e}}) \vee (\mathsf{e} \in \mathcal{C}) \vee (\mathsf{ChallEq}(\mathsf{e}-1) \wedge \mathsf{e} \in \mathcal{T}^*) \vee \\
&\qquad\qquad (\mathsf{ChallEq}(\mathsf{e}+1) \wedge \mathsf{e}+1 \in \mathcal{T}^*)
\end{aligned}
$$

Trivial wins via direct updates. The following holds for detIND-yy-atk notions where $\mathsf{yy} \in \{\mathsf{UE}, \mathsf{UPD}\}$ and $\mathsf{atk} \in \{\mathsf{CPA}, \mathsf{CCA}\}$. Define $\mathcal{I}$ as the set of epochs in which the adversary learned an updated version of the ciphertext given as challenge input ($\bar{C}$). Furthermore, define $\mathcal{I}^*$ to be the extended set in which the adversary has inferred information via token corruption. In the case when the algorithm Upd is deterministic, an updated ciphertext is uniquely determined by a token and a ciphertext. Thus, the adversary trivially wins if $\mathcal{I}^* \cap \mathcal{C}^* \neq \emptyset$. Indeed, there exists an epoch in which the adversary knows the updated ciphertext of the challenge input $\bar{C}$ and a valid challenge-equal ciphertext. Comparing them allows the adversary to win the game.

In [BDGJ20], $\mathcal{I}$ is computed by finding an entry in $\mathcal{L}$ that contains the challenge input $\bar{C}$. Then, note the query identifier c for that entry and scan $\mathcal{L}$ for other entries with this identifier $\mathcal{I} := \{\mathsf{e} \in \{0,\dots,n\} \mid (\mathsf{c}, \cdot, \mathsf{e}) \in \mathcal{L}\}$. We extend $\mathcal{I}$ into $\mathcal{I}^*$:

$$
\begin{aligned}
\mathcal{I}^* &\leftarrow \{\mathsf{e} \in \{0,\dots,n\} \mid \mathsf{ChallInpEq}(\mathsf{e}) = \mathsf{true}\} \\
&\mathsf{true} \leftarrow \mathsf{ChallInpEq}(\mathsf{e}) \Leftrightarrow (\mathsf{e} \in \mathcal{I}) \vee (\mathsf{ChallInpEq}(\mathsf{e}-1) \wedge \mathsf{e} \in \mathcal{T}^*) \vee \\
&\qquad\qquad (\mathsf{ChallInpEq}(\mathsf{e}+1) \wedge \mathsf{e}+1 \in \mathcal{T}^*)
\end{aligned}
$$

Furthermore, for the IND-UPD notion, the adversary uses two ciphertexts $\bar{C}_0, \bar{C}_1$ as challenge inputs. Thus, we can compute the sets $\mathcal{I}_b, \mathcal{I}_b^*$ where $b \in \{0,1\}$ first and use $\mathcal{I}^* = \mathcal{I}_0^* \cup \mathcal{I}_1^*$ to check the trivial win condition. Moreover, this trivial win condition does not exist in IND-ENC as there is no ciphertext in the challenge input of this notion and $\mathcal{I}^* = \emptyset$.

Trivial wins via decryptions. The following holds for detIND-yy-CCA notions where $\mathsf{yy} \in \{\mathsf{UE}, \mathsf{ENC}, \mathsf{UPD}\}$. It follows the analysis of [KLR19]. If the adversary knows a challenge ciphertext $(\tilde{C}, \mathsf{e}_0) \in \tilde{\mathcal{L}}$ and tokens from epoch $\mathsf{e}_0 + 1$ to epoch e, then he can update the challenge ciphertext to epoch e. By querying $\mathcal{O}.\mathsf{Dec}$ on this ciphertext, the adversary learns the underlying message and wins the game. We consider this to be a trivial win and exclude it by defining $\tilde{\mathcal{L}}^*$ to be the extended set of $\tilde{\mathcal{L}}$ in which the adversary has learned or inferred information via token corruption. We give an algorithm of [BDGJ20] to compute $\tilde{\mathcal{L}}^*$ during the game in Fig. 1.11.

Trivial wins in ciphertext integrity games. The following applies to UE schemes with deterministic updates. The adversary can corrupt an epoch key and use it to forge ciphertexts in this epoch. Thus, we exclude this trivial win by setting twf to 1 when the adversary provides a forgery in an epoch in $\mathcal{K}^*$.

Next, suppose that the adversary knows a ciphertext $(C, \mathsf{e}_1) \in \mathcal{L}$ and tokens from epoch $\mathsf{e}_1 + 1$ to epoch e_2. Then, updating C to epoch e_2 provides a forgery in epoch e_2. We exclude this trivial wins by defining $\mathcal{L}^*$ to be the extended set of $\mathcal{L}$ in which the adversary has learned or inferred information via token corruption. If $\mathcal{O}.\mathsf{Try}$ receives a ciphertext of $\mathcal{L}^*$, it sets twf to 1. We give an algorithm of [BDGJ20] to compute $\mathcal{L}^*$ during the game in Fig. 1.12.

Update $\tilde{\mathcal{L}}^*$

1. **if** $\mathcal{O}.\mathsf{Chall}$ **or** $\mathcal{O}.\mathsf{Upd}\tilde{\mathsf{C}}$ happens
2. $\quad \tilde{\mathcal{L}}^* \leftarrow \tilde{\mathcal{L}}^* \cup \{(\tilde{C}, \cdot)\}$
3. **if** $\mathsf{phase} = 1$ **and** $\mathcal{O}.\mathsf{Corr}(\mathsf{token}, \cdot)$ happens
4. $\quad$ **for** $i \in \mathcal{T}^*$ **and** $(\tilde{C}_{i-1}, i-1) \in \tilde{\mathcal{L}}^*$
5. $\quad\quad \tilde{\mathcal{L}}^* \leftarrow \tilde{\mathcal{L}}^* \cup \{(\tilde{C}_i, i)\}$

Figure 1.11: Update procedure of [BDGJ20] for the list $\tilde{\mathcal{L}}^*$.

Update $\mathcal{L}^*$

1. **if** $\mathcal{O}.\mathsf{Enc}$ **or** $\mathcal{O}.\mathsf{Upd}$ happens
2. $\quad \mathcal{L}^* \leftarrow \mathcal{L}^* \cup \{(\cdot, C, \cdot)\}$
3. **if** $\mathcal{O}.\mathsf{Corr}(\mathsf{token}, \cdot)$ happens
4. $\quad$ **for** $i \in \mathcal{T}^*$
5. $\quad\quad$ **for** $(j, C_{i-1}, i-1) \in \mathcal{L}^*$
6. $\quad\quad\quad C_i \leftarrow \mathsf{UE.Upd}(\Delta_i, C_{i-1})$
7. $\quad\quad\quad \mathcal{L}^* \leftarrow \mathcal{L}^* \cup \{(j, C_i, i)\}$

Figure 1.12: Update procedure of [BDGJ20] for list $\mathcal{L}^*$.

1.4 Private Information Retrieval

A PIR scheme involves a client C, who is interested in retrieving the i-th entry of a database $\mathbb{M}$ stored on $k \geq 1$ servers $\mathsf{S}_1, \ldots, \mathsf{S}_k$. The scheme allows the client to retrieve $\mathbb{M}[i]$ while hiding the index i from the servers and using communication which is sublinear in the size of the database n. In the following, we assume that the entries of databases belong to a finite alphabet Σ. Also, let $\mathbb{M}_j$ be the database stored on the server S_j: it can either be a copy of $\mathbb{M}$ or a modified version of it (*e.g.* encrypted, encoded, ...). Formally, we have the following definition given by [CGKS95].

Definition 1.8. *A k-server PIR protocol is a triple $(\mathcal{Q}, \mathcal{A}, \mathcal{R})$ of algorithms such that :*

- *The client samples a random string s with distribution $\mathcal{S}$, we denote this operation by $s \leftarrow \mathcal{S}$. The client can then use the algorithm $\mathcal{Q}$ to generate a k-tuple of queries $\mathcal{Q}(i, s) := (q_1, \ldots, q_k)$ and send each one to the corresponding server.*
- *Each server S_j computes an answer $\mathcal{A}(j, \mathbb{M}_j, q_j) := a_j$ and sends it back to the client.*
- *The client can recover $\mathbb{M}[i]$ by invoking the reconstruction algorithm $\mathcal{R}(a_1, \ldots, a_k, i, s)$*

Furthermore, correctness and privacy can be defined in the following manner:

Definition 1.9 (Correctness)**.** *For all databases $\mathbb{M} \in \Sigma^n$, for every index $1 \leq i \leq n$, we have*

$$\mathcal{R}(\mathcal{A}(1, \mathbb{M}_1, q_1), \ldots, \mathcal{A}(k, \mathbb{M}_k, q_k), i, s) = \mathbb{M}[i]$$

where $(q_1, \ldots, q_k) := \mathcal{Q}(i, s)$ and $s \leftarrow \mathcal{S}$.

Definition 1.10 (Privacy)**.** *Take $1 \leq t \leq k$, we say that a PIR scheme is t-private if, for all databases $\mathbb{M} \in \Sigma^n$, for every index $1 \leq i \leq n$ and for every $1 \leq j_1 \leq \ldots \leq j_t \leq k$, the joint distribution $(q_{j_1}, \ldots, q_{j_t})$ is independent of i, where $(q_1, \ldots, q_k) := \mathcal{Q}(i, s)$ and $s \leftarrow \mathcal{S}$. Probabilities are taken over the choices of s.*

Informally, the above definition means that no coalition of at most t servers can learn any information about i. Definition 1.10 can be relaxed to consider statistically or computationally secure PIR schemes.

1.5 Locally Correctable Codes

When encoding an outsourced file with a classical error correcting code, a client who wants to read a single data block needs to read the entire memory in order for him to run the decoding algorithm of the code to recover (or not) the data block. In this thesis, we show how one can use LCCs, so that one has to read only a small number of memory positions to recover one data block. We also use the locality of LCCs to design protocols with low communication complexity in the outsourced storage context. We now briefly present LCCs, which were formally introduced by Katz and Trevisan [KT00] in 2000.

Definition 1.11 (Locally correctable code)**.** *Let $n, r \in \mathbb{N}$, $\delta \in [0,1]$, $\epsilon : [0,1] \to [0,1]$ and $\mathbb{F}$ be a finite field. A code $\mathcal{C} \subseteq \mathbb{F}^n$ is said to be (r, δ, ϵ)-locally correctable if there exists a probabilistic decoding algorithm $\mathcal{A}$ such that,*

1. *For all $\mathbf{c} \in \mathcal{C}$, for all $i \in [1, n]$ and for all vectors $\mathbf{y} \in \mathbb{F}^n$ with relative Hamming distance $\Delta(\mathbf{c}, \mathbf{y}) \leq \delta$, we have $\Pr[\mathcal{A}^{\mathbf{y}}(i) = \mathbf{c}_i] \geq 1 - \epsilon(\delta)$, where the probability is taken over the internal randomness of $\mathcal{A}^{\mathbf{y}}$.*
2. *The algorithm $\mathcal{A}$ makes at most r queries to the vector $\mathbf{y}$.*

Examples of LCCs include the well-known Hadamard codes (see Ex. 1) and Reed-Muller codes [Ree54, Mul54]. Codewords of Reed-Muller codes are evaluations of multivariate polynomials of bounded total degree. Unfortunately, their rate tends towards zero as their length grows. A major breakthrough in the theory of LCCs is the introduction of a high-rate class of LCCs, called multiplicity codes, by Kopparty *et al.* [KSY11] in 2011. They generalize Reed-Muller codes by evaluating multivariate polynomials as well as their partial derivatives up to some order. Other high-rate constructions are the lifted codes of Guo *et al.* [GKS13] and the expander codes of Hemenway *et al.* [HOW13], both introduced in 2013.

Example 1. Let $n \in \mathbb{N}$. We present the Hadamard code of length $2^n - 1$ over $\mathbb{F}_2$. The coordinates of a codeword are indexed by the non-zero elements of $\mathbb{F}_2^n$. Moreover, each codeword is of the form $(f(\mathbf{u}))_{\mathbf{u} \in \mathbb{F}_2^n \setminus \{\mathbf{0}\}}$, where f is a linear form over $\mathbb{F}_2^n$. Using the fact that $f(\mathbf{u}+\mathbf{v}) = f(\mathbf{u}) + f(\mathbf{v})$, we get the following local correcter. To correct the symbol at coordinate $\mathbf{u} \in \mathbb{F}_2^n \setminus \{\mathbf{0}\}$, the local correcter proceeds as follows: with probability $1 - \frac{1}{2^{n-1}}$, it uniformly samples a coordinate $\mathbf{v} \neq \mathbf{u}$ and returns $f(\mathbf{u}+\mathbf{v}) - f(\mathbf{v})$. Otherwise, with probability $\frac{1}{2^{n-1}}$, the local correcter returns $f(\mathbf{u})$. This local correcter is said to be *smooth* since it queries coordinates uniformly. Moreover, let $\delta < 1/4$ and $n \geq 2$, one can show that the Hadamard code is a $(2, \delta, 2\delta)$-LCC. See [Lav18, Prop. 2.5] for a proof.

In this thesis, we consider LCCs for erasures and we do not use the estimate of their failure probability until Section 3.5.

Chapter 2

New constructions for post-quantum Updatable Encryption

When protecting the confidentiality of outsourced data, it is reasonable to expect that an adversary will craft and inject its own data on the remote server during his attack. It is thus preferable for clients to use UE schemes that are secure against chosen ciphertext attacks (CCA), since they are tailored for this situation. In fact, we will prove in Chapter 4 that the weaker RCCA (where R stands for replayable) security notion is sufficient and necessary to ensure confidentiality against adversarial injections (when the message space is large enough). However, few UE schemes are known to have this level of security. Klooß *et al.* [KLR19] proposed two generic constructions for UE called E&M and NYUE. E&M is based on encrypt-and-MAC and its security relies on the decisional Diffie-Hellman (DDH) assumption. It is worth noting that this scheme does not feature malicious updates resistance, *i.e.* it provides no security guarantees when the server updates a ciphertext injected by the adversary, a situation that is likely going to happen when dealing with injections. The NYUE scheme of [KLR19] uses the Naor-Yung paradigm and its security is based on the symmetric external Diffie-Hellman (SXDH) assumption. It provides plaintext integrity and RCCA security. Finally, the SHINE schemes of Boyd *et al.* [BDGJ20] are CCA secure under the DDH assumption.

Unfortunately, none of these schemes are post-quantum secure since the DDH and SXDH assumptions do not hold against quantum adversaries. Moreover, only two post-quantum UE schemes exist, they are due to Jiang [Jia20] and Nishimaki [Nis22]. These two schemes are not CCA secure (only CPA) and their security relies on the LWE assumption. Moreover, they only support a bounded number of key updates. The open problem of finding a post-quantum CCA secure UE scheme was first formally posed by Jiang [Jia20] at Asiacrypt 2020. In this chapter, we explore how it might be possible to solve this problem by proposing our new GAINE schemes based on group actions. For now, GAINE has no practical post-quantum instantiations. Moreover, we also introduce another group action based scheme, called TOGA-UE, which is post-quantum CPA secure. Our two schemes are also the first post-quantum UE schemes that support an unbounded number of updates.

Disclaimer. The contents of this chapter are the result of a joint-work with Antonin Leroux. One of our contribution consists in instantiating our generic TOGA-UE scheme with isogeny-based objects. Since this part was solely Antonin's work, as unlike him I am no isogeny specialist, I chose to leave it out of this document. Instead, I have replaced isogenies with pre-quantum objects that fit the same role in our construction. I invite the reader interested in isogenies to consult our paper for more detail on how to use them to instatiate TOGA-UE [LR22].

2.1 Preliminaries

2.1.1 Cryptographic group actions

In this section, we give a few reminders about group actions and how they can be endowed with hardness properties for cryptographic use. We use the framework of cryptographic group actions of Alamati *et al.* [ADFMP20].

Definition 2.1 (Group Action)**.** *Let G be a group for a law written multiplicatively and let S be a set. A group action of G on S is an operation $\star : G \times S \to S$ such that*

1. *(Identity) If 1_G is the identity element of G, then for any $s \in S$, we have $1_G \star s = s$.*
2. *(Compatibility) For any $g, h \in G$ and any $s \in S$, we have $(gh) \star s = g \star (h \star s)$.*

We may use the notation $(G, S, \star)$ to denote a group action. We stress that the group actions used in this thesis do not need to be abelian. A group action $(G, S, \star)$ partitions the set S into a disjoint union of *orbits* where the orbit of $s \in S$ is the set $\mathrm{Orb}(s) := \{g \star s \mid g \in G\} \subseteq S$.

Properties of group actions. Our group actions $(G, S, \star)$ can be:

1. Transitive: A group action is *transitive* if it has a single orbit, *i.e.*, if for any $(s_1, s_2) \in S$, there exists $g \in G$ such that $g \star s_1 = s_2$. We can always obtain a transitive group action from any group action. Indeed, take $s \in S$, one can easily verify that $(G, \mathrm{Orb}(s), \star)$ is a transitive group action.
2. Free: A group action is *free* if for all $g \in G$, $g = 1_G$ if and only if there exists $s \in S$ such that $g \star s = s$.

Since we need to define computational assumptions related to group actions, we need a notion of efficiency.

Definition 2.2 (Effective Group Action [ADFMP20])**.** *We say that $(G, S, \star)$ is an effective group action (EGA), with respect to a parameter λ, if the following properties are satisfied:*

1. *The group G is finite and there exist PPT algorithms for:*
   (a) *Membership testing, i.e. to decide if a given bit string represents a valid element in G.*
   (b) *Equality testing, i.e. to decide if two bit strings represent the same group element in G.*
   (c) *Sampling, i.e. to sample an element g from a distribution $\mathcal{D}_G$ on G.*
   (d) *Operation, i.e. to compute gh for any $g, h \in G$.*
   (e) *Inversion, i.e. to compute g^{-1} for any $g \in G$.*
2. *The set S is finite and there exist efficient PPT algorithms for:*
   (a) *Membership testing.*
   (b) *Unique representation, i.e. given any arbitrary set element $s \in S$, compute a string $\hat{s}$ that canonically represents s.*
3. *There exists a distinguished element $s_0 \in S$, called the origin, such that its bit-string representation is known.*
4. *There exists an efficient algorithm that given (some bit-string representations of) any $g \in G$ and any $s \in S$, outputs $g \star s$.*

Definition 2.3 (Group Action Family)**.** *We say that $\mathcal{GA}$ is a group action family if, for a security parameter λ, $\mathcal{GA}(\lambda)$ consists of a group action $(G, S, \star)$ where $|G|, |S| = \mathsf{poly}(\lambda)$.*

$\mathbf{Exp}^{\mathsf{wk\text{-}PR}\text{-}b}_{\mathcal{GA},\mathcal{A}}(\lambda)$:

1. $(G, S, \star) \leftarrow \mathcal{GA}(\lambda)$
2. **if** $b = 0$
3. $\quad g \leftarrow \mathcal{D}_G$
4. $\quad \mathcal{O}.\mathsf{Sample} \leftarrow \pi_g^{\$}$
5. **else** $(b = 1)$
6. $\quad \pi \xleftarrow{\$} \mathfrak{S}(S)$
7. $\quad \mathcal{O}.\mathsf{Sample} \leftarrow \pi^{\$}$
8. $b' \leftarrow \mathcal{A}^{\mathcal{O}.\mathsf{Sample}}(1^\lambda, (G, S, \star))$
9. **if** $b' = b$
10. $\quad$ **return** 1
11. **else**
12. $\quad$ **return** 0

Figure 2.1: Weak pseudorandom group action experiment. Recall that $\mathcal{D}_G$ and $\mathcal{D}_S$ are distributions on G and S respectively. For a permutation $f \in \mathfrak{S}(S)$, we use $f^{\$}$ to denote the randomized oracle that samples $s \leftarrow \mathcal{D}_S$ and outputs $(s, f(s))$.

In the following, let $\mathcal{GA}$ be a group action family. We start by defining one-way group actions:

Definition 2.4 (One-Way Group Action [ADFMP20])**.** *Let $(G, S, \star)$ be $\mathcal{GA}(\lambda)$ for some security parameter λ. Let $\mathcal{D}_G$ and $\mathcal{D}_S$ be distributions on G and S respectively. For $s \in S$, let $f_s : G \to S$ be the function defined by $f_s : g \mapsto g \star s$. We say that $(G, S, \star)$ is $(\mathcal{D}_G, \mathcal{D}_S)$-one-way if, for all PPT adversaries $\mathcal{A}$, we have*

$$\Pr[f_s(\mathcal{A}(s, f_s(g))) = f_s(g)] \leq \mathsf{negl}(\lambda)$$

where $s \leftarrow \mathcal{D}_S$ and $g \leftarrow \mathcal{D}_G$.

Informally, a group action $(G, S, \star)$ is $(\mathcal{D}_G, \mathcal{D}_S)$-one-way if, given a pair of set elements $(s, g \star s)$ where $s \leftarrow \mathcal{D}_S$ and $g \leftarrow \mathcal{D}_G$, there is no PPT adversary that can recover g. If $\mathcal{D}_S$ and $\mathcal{D}_G$ are uniform distributions, then we simply speak of OW group action.

Then, we define weak pseudorandom group actions:

Definition 2.5 (Weak Pseudorandom Group Action [ADFMP20])**.** *Let $(G, S, \star)$ be $\mathcal{GA}(\lambda)$ for some security parameter λ. Let $\mathcal{D}_G$ and $\mathcal{D}_S$ be distributions on G and S respectively. For $g \in G$, let $\pi_g : S \to S$ be the permutation defined by $\pi_g : s \mapsto g \star s$. For a permutation $f \in \mathfrak{S}(S)$, we use $f^{\$}$ to denote the randomized oracle that, when queried, samples $s \leftarrow \mathcal{D}_S$ and outputs $(s, f(s))$. We say that $(G, S, \star)$ is $(\mathcal{D}_G, \mathcal{D}_S)$-weakly pseudorandom if, for all PPT adversaries $\mathcal{A}$, we have*

$$\mathbf{Adv}^{\mathsf{wk\text{-}PR}}_{\mathcal{GA},\mathcal{A}}(\lambda) := \left|\Pr[\mathbf{Exp}^{\mathsf{wk\text{-}PR}\text{-}0}_{\mathcal{GA},\mathcal{A}}(\lambda) = 1] - \Pr[\mathbf{Exp}^{\mathsf{wk\text{-}PR}\text{-}1}_{\mathcal{GA},\mathcal{A}}(\lambda) = 1]\right| \leq \mathsf{negl}(\lambda)$$

where $\mathbf{Exp}^{\mathsf{wk\text{-}PR}\text{-}b}_{\mathcal{GA},\mathcal{A}}(\lambda)$ is the experiment described in Fig. 2.1.

Informally, a group action $(G, S, \star)$ is $(\mathcal{D}_G, \mathcal{D}_S)$-weakly pseudorandom if there is no PPT adversary that can distinguish tuples of the form $(s_i, g \star s_i)$ from (s_i, u_i) where $g \leftarrow \mathcal{D}_G$ and each $s_i, u_i \leftarrow \mathcal{D}_S$. If both distributions are uniform, we omit them and we say that the group action is weakly pseudorandom.

Finally, we define weak unpredictable group actions:

Definition 2.6 (Weak Unpredictable Group Action [ADFMP20])**.** *Let $(G, S, \star)$ be $\mathcal{GA}(\lambda)$ for some security parameter λ. Let $\mathcal{D}_G$ and $\mathcal{D}_S$ be distributions on G and S respectively. For $g \in G$, let $\pi_g : S \to S$ be the permutation defined by $\pi_g : s \mapsto g \star s$. For a permutation $f \in \mathfrak{S}(S)$, we use $f^{\$}$ to denote the randomized oracle that, when queried, samples $s \leftarrow \mathcal{D}_S$ and outputs $(s, f(s))$. We say that $(G, S, \star)$ is $(\mathcal{D}_G, \mathcal{D}_S)$-weakly unpredictable if, for all PPT adversaries $\mathcal{A}$, we have*

$$\mathbf{Adv}^{\mathsf{wk\text{-}UP}}_{\mathcal{GA},\mathcal{A}}(\lambda) := \Pr[\mathcal{A}^{\pi_g^{\$}}(s^*) = \pi_g(s^*)] \leq \mathsf{negl}(\lambda)$$

where $g \leftarrow \mathcal{D}_G$ and $s^* \leftarrow \mathcal{D}_S$. We denote this experiment by $\mathbf{Exp}^{\mathsf{wk\text{-}UP}}_{\mathcal{GA},\mathcal{A}}$.

Informally, $(G, S, \star)$ is $(\mathcal{D}_G, \mathcal{D}_S)$-weakly unpredictable if, given polynomially many tuples of the form $(s_i, g \star s_i)$ where $g \leftarrow \mathcal{D}_G$ and each $s_i \leftarrow \mathcal{D}_S$, there is no PPT adversary that can compute $g \star s^*$ for a given challenge $s^* \leftarrow \mathcal{D}_S$. If both distributions are uniform, we simply speak of a weakly unpredictable group action.

2.2 Updatable Encryption from group actions

We generalize the SHINE (Secure Homomorphic Ideal-cipher Nonce-based Encryption) scheme of Boyd *et al.* [BDGJ20] using the framework of cryptographic group actions of Alamati *et al.* [ADFMP20]. SHINE uses message space $\mathcal{M} := \{0,1\}^m$, nonce space $\mathcal{N} := \{0,1\}^v$ for some integers m and v, ciphertext space H a cyclic group of order q and it assumes the existence of a permutation $\pi : \{0,1\}^{m+v} \to H$. SHINE is presented in Fig. 2.2.

$\mathsf{Setup}(1^\lambda)$:

1. Choose H, q, π, m, v as above
2. $\mathsf{pp} \leftarrow H, q, \pi, m, v$
3. **return** pp

$\mathsf{KeyGen}(\mathsf{pp})$:

1. $k \xleftarrow{\$} \mathbb{Z}_q^*$.
2. **return** k.

$\mathsf{TokenGen}(k_\mathsf{e}, k_{\mathsf{e}+1})$:

1. $\Delta_{\mathsf{e}+1} \leftarrow \frac{k_{\mathsf{e}+1}}{k_\mathsf{e}}$
2. **return** $\Delta_{\mathsf{e}+1}$

$\mathsf{Upd}(\Delta_{\mathsf{e}+1}, C_\mathsf{e})$:

1. $C_{\mathsf{e}+1} \leftarrow C_\mathsf{e}^{\Delta_{\mathsf{e}+1}}$
2. **return** $C_{\mathsf{e}+1}$

$\mathsf{Enc}(k_\mathsf{e}, M)$:

1. $N \xleftarrow{\$} \mathcal{N}$.
2. $C_\mathsf{e} \leftarrow (\pi(N \| M))^{k_\mathsf{e}}$
3. **return** C_e

$\mathsf{Dec}(k_\mathsf{e}, C_\mathsf{e})$:

1. $s \leftarrow \pi^{-1}(C_\mathsf{e}^{1/k_\mathsf{e}})$
2. Parse s as $N' \| M'$
3. **return** M'

Figure 2.2: The SHINE scheme of [BDGJ20].

2.2.1 Generalizing SHINE to group actions

Our initial idea is to see exponentiation, the key operation used to encrypt, update and decrypt ciphertexts in SHINE, as a component of the group action of $\mathbb{Z}_q^*$ on H. Then, our goal is to generalize SHINE to group actions and identify the key properties needed to get a secure UE scheme. Finally, we check if the known post-quantum group actions satisfy our requirements. If they do, we can instantiate our scheme to get a new post-quantum UE scheme. First, we introduce the novel *mappable* EGA (MEGA) definition.

Definition 2.7 (Mappable EGA)**.** *Let $(G, S, \star)$ be an EGA. We say that $(G, S, \star)$ is a mappable EGA if there exists an efficient bijection $\pi : \{0,1\}^N \to S$.*

This new notion can be seen as a strengthened hashable group action. Indeed, [ADFMP20] strengthen the Definition 2.2 of an EGA by replacing the existence of the origin s_0 by the following *Hashing to the set* axiom:

Definition 2.8 (Hashable EGA [ADFMP20])**.** *Let $(G, S, \star)$ be an OW-EGA. We say that $(G, S, \star)$ is a hashable OW-EGA if there exists an efficient sampler $H : \{0,1\}^N \to S$ (where N depends on the security parameter), such that for all PPT adversaries $\mathcal{A}$, we have*

$$\Pr[\mathcal{A}(i, j) \star H(i) = H(j)] \leq \mathsf{negl}(\lambda)$$

where $i, j \xleftarrow{\$} \{0,1\}^N$.

We show that our OW-MEGA is also a hashable OW-EGA.

Proposition 2.1. *Let $(G, S, \star)$ be an OW-MEGA with bijection π. Then, $(G, S, \star)$ is also a hashable OW-EGA with sampler $H := \pi$.*

Proof. Keeping the notations of the proposition, let $\mathcal{A}$ be an adversary that breaks the *Hashing to the set* axiom of the sampler π of $(G, S, \star)$ with probability ϵ. We build the following adversary $\mathcal{B}$ against the one-way property of $(G, S, \star)$.

1. $\mathcal{B}$ receives $(s, f_s(g)) = (s, g \star s)$ such that $s \xleftarrow{\$} S$ and $g \xleftarrow{\$} G$.
2. $\mathcal{B}$ calls $\mathcal{A}$ on input $(\pi^{-1}(s), \pi^{-1}(g \star s))$ and let $h \in G$ be the value returned by $\mathcal{A}$.
3. $\mathcal{B}$ outputs h.

By definition, $\mathcal{A}$ returns h such that $h \star \pi(\pi^{-1}(s)) = \pi(\pi^{-1}(g \star s))$ with probability ϵ. This means that, with probability ϵ, $\mathcal{B}$ outputs h such that $h \star s = f_s(g)$ which is exactly breaking the one-way property of $(G, S, \star)$. □

We present our generalization of the SHINE scheme to group actions, which we call GAINE for Group Action Ideal-cipher Nonce-based Encryption, in Fig. 2.3. Let $\mathcal{GA}$ be a MEGA family and let $(G, S, \star)$ be $\mathcal{GA}(\lambda)$: a MEGA with bijection $\pi : \{0,1\}^{m+v} \to S$, for integers m and v. Let $\mathcal{M} := \{0,1\}^m$ be the message space and $\mathcal{N} := \{0,1\}^v$ be the nonce space. We use group elements as keys and set elements as ciphertexts. Encryption, decryption and updates boil down to a group action computation. Ciphertexts are randomized by adding a random nonce as input to π.

GAINE is correct, even for non-abelian group actions.

Proposition 2.2 (Correctness of updates)**.** *Let $k_\mathsf{e}, k_{\mathsf{e}+1} \in G$ be two keys and $C_\mathsf{e} := k_\mathsf{e} \star s$ for some $s \in S$. If $\Delta_{\mathsf{e}+1} := \mathsf{TokenGen}(k_\mathsf{e}, k_{\mathsf{e}+1})$, then $\mathsf{Upd}(\Delta_{\mathsf{e}+1}, C_\mathsf{e}) = k_{\mathsf{e}+1} \star s$.*

Proof. By definition of TokenGen, we have $\Delta_{\mathsf{e}+1} := k_{\mathsf{e}+1} k_\mathsf{e}^{-1}$. By definition of Upd, we have

$$\mathsf{Upd}(\Delta_{\mathsf{e}+1}, C_\mathsf{e}) = \Delta_{\mathsf{e}+1} \star C_\mathsf{e} = (k_{\mathsf{e}+1} k_\mathsf{e}^{-1}) \star (k_\mathsf{e} \star s) = k_{\mathsf{e}+1} \star s$$

where the last equality holds because of the *compatibility* of the group action (see Definition 2.1) and because k_e^{-1} is the inverse of k_e in the group G. □

Proposition 2.3 (Correctness)**.** *The* GAINE *scheme is correct.*

$\mathsf{Setup}(1^\lambda)$:

1. $(G, S, \star) \leftarrow \mathcal{GA}(\lambda)$
2. Choose π, m, v as above
3. $\mathsf{pp} \leftarrow (G, S, \star), \pi, m, v$
4. **return** pp

$\mathsf{KeyGen}(\mathsf{pp})$:

1. $k \xleftarrow{\$} G$.
2. **return** k.

$\mathsf{TokenGen}(k_\mathsf{e}, k_{\mathsf{e}+1})$:

1. $\Delta_{\mathsf{e}+1} \leftarrow k_{\mathsf{e}+1} k_\mathsf{e}^{-1}$
2. **return** $\Delta_{\mathsf{e}+1}$

$\mathsf{Upd}(\Delta_{\mathsf{e}+1}, C_\mathsf{e})$:

1. $C_{\mathsf{e}+1} \leftarrow \Delta_{\mathsf{e}+1} \star C_\mathsf{e}$
2. **return** $C_{\mathsf{e}+1}$

$\mathsf{Enc}(k_\mathsf{e}, M)$:

1. $N \xleftarrow{\$} \mathcal{N}$.
2. $C_\mathsf{e} \leftarrow k_\mathsf{e} \star \pi(N \| M)$
3. **return** C_e

$\mathsf{Dec}(k_\mathsf{e}, C_\mathsf{e})$:

1. $s \leftarrow \pi^{-1}(k_\mathsf{e}^{-1} \star C_\mathsf{e})$
2. Parse s as $N' \| M'$
3. **return** M'

Figure 2.3: GAINE: our generalization of the SHINE scheme using group actions.

Proof. Let $0 \leq \mathsf{e}_1 \leq \mathsf{e}_2 \leq n+1$ be two epochs and let us consider a ciphertext C_{e_2} updated through the successive tokens Δ_{i+1} for $i \in [\mathsf{e}_1, \mathsf{e}_2 - 1]$ from an initial ciphertext C_{e_1} that is the encryption of a message m under the key $k_{\mathsf{e}-1}$ as in Definition 1.5. By definition of $C_{\mathsf{e}_1} := \mathsf{Enc}(k_{\mathsf{e}_1}, M)$, we have $C_{\mathsf{e}_1} = k_{\mathsf{e}_1} \star \pi(N \| M)$ for some random nonce N. By applying Prop. 2.2 on the updates of C_{e_1}, we have that $C_{\mathsf{e}_2} = k_{\mathsf{e}_2} \star \pi(N \| M)$. Then, we get

$$k_{\mathsf{e}_2}^{-1} \star C_{\mathsf{e}_2} = k_{\mathsf{e}_2}^{-1} \star (k_{\mathsf{e}_2} \star \pi(N \| M)) = 1_G \star \pi(N \| M) = \pi(N \| M)$$

where the penultimate equality holds because of the *compatibility* of the group action and the last equality holds because of the *identity* property of the group action. Finally, we have $\pi^{-1}(k_{\mathsf{e}_2}^{-1} \star C_{\mathsf{e}_2}) = N \| M$ and $\mathsf{Dec}(k_{\mathsf{e}_2}, C_{\mathsf{e}_2})$ returns M. □

2.2.2 Security - GAINE is detIND-UE-CPA secure

In Theorem 2.4, we show that GAINE is detIND-UE-CPA in the ideal cipher model, if the group action is a weakly pseudorandom MEGA. The ideal cipher model, introduced by Shannon [Sha49] and shown to be equivalent to the random oracle model by Coron *et al.* [CPS08], gives all parties access to a permutation chosen randomly from all possible key permutations of appropriate length. The GAINE scheme acts on the outputs of the permutation with the epoch key to encrypt, so our reduction can "program" the transformation from permutation outputs to set elements.

Theorem 2.4 (GAINE is detIND-UE-CPA)**.** *Let* GAINE *be the UE scheme described in Fig. 2.3 for a MEGA family $\mathcal{GA}$. For any ideal cipher model adversary $\mathcal{A}$, there exists a reduction $\mathcal{B}$ such that*

$$\mathbf{Adv}^{\mathsf{detIND\text{-}UE\text{-}CPA}}_{\mathsf{GAINE},\mathcal{A}}(\lambda) \leq \mathcal{O}(1)(n+1)^3 \cdot \mathbf{Adv}^{\mathsf{wk\text{-}PR}}_{\mathcal{GA},\mathcal{B}}(\lambda)$$

We follow the proof strategy of [BDGJ20] and use their hybrid argument across insulated regions. In each hybrid, we can embed at one firewall of the insulated region and simulate all tokens within that insulated region to answer queries to both $\mathcal{O}.\mathsf{Upd}$ and $\mathcal{O}.\mathsf{Upd}\tilde{\mathsf{C}}$. In GAINE, we update a ciphertext from epoch e to epoch $\mathsf{e}+1$ by computing the action of the group element

$k_{\mathsf{e}+1}k_{\mathsf{e}}^{-1}$. Fresh ciphertexts are randomized using a nonce N but updates are deterministic, thus our reduction will need to provide consistent ciphertexts to the adversary, *i.e.* the N value must be consistent.

We give a reduction $\mathcal{B}$ which receives a group action $(G, S, \star)$ and an oracle $\mathcal{O}.\mathsf{Sample}$ that returns either tuples of the form $(s_i, g \star s_i)$ or (s_i, u_i) where $g \xleftarrow{\$} G$ and $s_i, u_i \xleftarrow{\$} S$. $\mathcal{B}$ will use the tuples of $\mathcal{O}.\mathsf{Sample}$ to perfectly simulate the $\mathsf{detIND\text{-}UE\text{-}CPA}$ experiment for GAINE when those tuples are of the form $(s_i, g \star s_i)$ (and a random experiment otherwise). The idea is to embed g to a well chosen epoch key by using s_i as randomness and $g \star s_i$ as ciphertext value. Thus, if we know an efficient adversary $\mathcal{A}$ against the $\mathsf{detIND\text{-}UE\text{-}CPA}$ security of GAINE, using the hybrid argument of [BDGJ20], $\mathcal{B}$ can use $\mathcal{A}$ to break the weak pseudorandomness of $(G, S, \star)$.

Proof. Play hybrid games. We partition the non-corrupted key space as follows:$\{0, \ldots, n\} \setminus \mathcal{K}^* = \cup_{(j,\mathsf{fwl}_j,\mathsf{fwr}_j)\in\mathcal{FW}}\{\mathsf{fwl}_j \ldots \mathsf{fwr}_j\}$, where fwl_i and fwr_i are firewalls of the i-th insulated region. For $b \in \{0, 1\}$, define game $\mathcal{G}_i^b$ as $\mathbf{Exp}^{\mathsf{detIND\text{-}UE\text{-}CPA}\text{-}b}_{\mathsf{GAINE},\mathcal{A}}$ except for:

1. The game randomly picks $\mathsf{fwl}_i, \mathsf{fwr}_i \xleftarrow{\$} \{0, \ldots, n\}$ and if they are not the i-th firewalls, it aborts and returns a random bit b'. This loss is upper-bounded by $(n+1)^2$.
2. For the challenge (made in epoch $\tilde{\mathsf{e}}$ on input $(\bar{M}, \bar{C})$), the game returns an updated version of $\bar{C}$ if $\tilde{\mathsf{e}} < \mathsf{fwl}_i$ and it returns an encryption of $\bar{M}$ if $\tilde{\mathsf{e}} > \mathsf{fwr}_i$. Finally, if $\mathsf{fwl}_i \leq \tilde{\mathsf{e}} \leq \mathsf{fwr}_i$, the game returns an encryption of $\bar{M}$ if $b = 0$ and an updated version of $\bar{C}$ if $b = 1$.
3. After $\mathcal{A}$ outputs b', the game returns b' if $\mathsf{twf} \neq 1$ or some additional trivial win condition triggers.

If fwl_i, fwr_i are the desired values, then $\mathcal{G}_1^0$ is $\mathbf{Exp}^{\mathsf{detIND\text{-}UE\text{-}CPA}\text{-}0}_{\mathsf{GAINE},\mathcal{A}}$, *i.e.* all challenges are encryptions of $\bar{M}$. Let ℓ be the total number of insulated regions (bounded by $n+1$), such that $\mathcal{G}_\ell^1$ is $\mathbf{Exp}^{\mathsf{detIND\text{-}UE\text{-}CPA}\text{-}1}_{\mathsf{GAINE},\mathcal{A}}$, *i.e.* all challenges are updates of $\bar{C}$. Let E be the event that fwl_i and fwr_i are the desired values. By definition, for any $1 \leq i \leq n+1$ and $b \in \{0, 1\}$, we have $\Pr[\mathcal{G}_i^b = 1 \mid \neg E] = 1/2$. Then

$$\begin{aligned}
\Pr[\mathcal{G}_\ell^1 = 1] &= \Pr[\mathcal{G}_\ell^1 = 1 \mid E] \cdot \Pr[E] + \Pr[\mathcal{G}_\ell^1 = 1 \mid \neg E] \cdot \Pr[\neg E] \\
&= \Pr[\mathbf{Exp}^{\mathsf{detIND\text{-}UE\text{-}CPA}\text{-}1}_{\mathsf{GAINE},\mathcal{A}} = 1] \cdot \frac{1}{(n+1)^2} + \frac{1}{2} \cdot (1 - \frac{1}{(n+1)^2}), \text{and} \\
\Pr[\mathcal{G}_1^0 = 1] &= \Pr[\mathbf{Exp}^{\mathsf{detIND\text{-}UE\text{-}CPA}\text{-}0}_{\mathsf{GAINE},\mathcal{A}} = 1] \cdot \frac{1}{(n+1)^2} + \frac{1}{2} \cdot (1 - \frac{1}{(n+1)^2})
\end{aligned}$$

Thus, we have $|\Pr[\mathcal{G}_\ell^1 = 1] - \Pr[\mathcal{G}_1^0 = 1]| = \frac{1}{(n+1)^2} \cdot \mathbf{Adv}^{\mathsf{detIND\text{-}UE\text{-}CPA}}_{\mathsf{GAINE},\mathcal{A}}(\lambda)$.

Notice that the games $\mathcal{G}_{i-1}^1$ and $\mathcal{G}_i^0$ behave in the same way: for the challenge query and $\mathcal{O}.\mathsf{Upd}\tilde{\mathsf{C}}$, in an epoch in the first $i-1$ insulated regions, the reduction returns an update of $\bar{C}$, otherwise it returns an encryption of $\bar{M}$. Thus, for any $\ell \leq n+1$, $|\Pr[\mathcal{G}_\ell^1 = 1] - \Pr[\mathcal{G}_1^0 = 1]| \leq \sum_{i=1}^{\ell} |\Pr[\mathcal{G}_i^1 = 1] - \Pr[\mathcal{G}_i^0 = 1]|$. In the following, we prove that for any $1 \leq i \leq \ell$, $|\Pr[\mathcal{G}_i^1 = 1] - \Pr[\mathcal{G}_i^0 = 1]| \leq 2\,\mathbf{Adv}^{\mathsf{wk\text{-}PR}}_{\mathcal{GA},\mathcal{B}}(\lambda)$ for a reduction $\mathcal{B}$.

In hybrid i. Let $\mathcal{A}_i$ be an adversary trying to distinguish $\mathcal{G}_i^0$ from $\mathcal{G}_i^1$. For all queries concerning epochs outside of the i-th insulated region, the responses of both games are the same. Thus, we assume that $\mathcal{A}_i$ asks for at least one challenge ciphertext in an epoch within the i-th insulated region. This is where we will embed the weak pseudorandom group action samples in our reduction.

We construct a reduction $\mathcal{B}$, presented in Fig. 2.4, that is playing the weak pseudorandom group action game (Definition 2.5) and will simulate the responses of queries made by adversary $\mathcal{A}_i$. Since we do not assume the group action $(G, S, \star)$ to be abelian, we define $(\prod_{i=0}^{n} g_i) \star s := (g_0 g_1 \ldots g_n) \star s$ for $s \in S$ and $g_0, \ldots, g_n \in G$.

Reduction $\mathcal{B}$ playing $\mathbf{Exp}^{\text{wk-PR-}b^*}_{\mathcal{G},\mathcal{A},\mathcal{B}}(\lambda)$

1. **receive** $(G, S, \star)$ and $\mathcal{O}.\mathsf{Sample}$
2. **do Setup**(1^λ)
3. $\bar{M}, \bar{C} \leftarrow \mathcal{A}^{\mathtt{ors}}(\lambda)$
4. $\mathsf{phase} \leftarrow 1$
5. $\tilde{C}_{\tilde{\mathsf{e}}} \leftarrow \mathcal{O}.\mathsf{Chall}(\bar{M}, \bar{C})$
6. $b' \leftarrow \mathcal{A}^{\mathtt{ors}, \mathcal{O}.\mathsf{Upd}\tilde{\mathsf{C}}}(\tilde{C}_{\tilde{\mathsf{e}}})$
7. $\mathsf{twf} \leftarrow 1$ **if**
8. $\quad \mathcal{C}^* \cap \mathcal{K}^* \neq \emptyset$ **or**
9. $\quad \mathcal{I}^* \cap \mathcal{C}^* \neq \emptyset$
10. **if** ABORT occurred **or** $\mathsf{twf} = 1$
11. $\quad b' \xleftarrow{\$} \{0,1\}$
12. $\quad$ **return** b'
13. **if** $(i, \mathsf{fwl}_i, \mathsf{fwr}_i) \notin \mathcal{FW}$
14. $\quad b' \xleftarrow{\$} \{0,1\}$
15. $\quad$ **return** b'
16. **if** $b' = b$
17. $\quad$ **return** 0
18. **else**
19. $\quad$ **return** 1

Setup(1^λ)

1. $b \xleftarrow{\$} \{0,1\}$
2. $\mathsf{pp} \leftarrow \mathsf{GAINE.Setup}(1^\lambda)$
3. $k_0 \leftarrow \mathsf{GAINE.KeyGen}(\mathsf{pp})$
4. $\Delta_0 \leftarrow \perp$
5. $\mathsf{e}, \mathsf{c}, \mathsf{phase}, \mathsf{twf} \leftarrow 0$
6. $\mathcal{L}, \tilde{\mathcal{L}}, \mathcal{C}, \mathcal{K}, \mathcal{T} \leftarrow \emptyset$
7. $\mathsf{fwl}_i, \mathsf{fwr}_i \xleftarrow{\$} \{0, \ldots, n\}$
8. **for** $j \in \{0, \ldots, \mathsf{fwl}_i - 1\}$ **do**
9. $\quad k_j \xleftarrow{\$} G; \Delta_j \leftarrow k_j k_{j-1}^{-1} \bowtie$
10. **for** $j \in \{\mathsf{fwr}_i + 1, \ldots, n\}$ **do**
11. $\quad k_j \xleftarrow{\$} G; \Delta_j \leftarrow k_j k_{j-1}^{-1} \bowtie$
12. **for** $j \in \{\mathsf{fwl}_i + 1, \ldots, \mathsf{fwr}_i\}$ **do**
13. $\quad \Delta_j \xleftarrow{\$} G$

$\mathcal{O}.\mathsf{Enc}(M)$

1. $\mathsf{c} \leftarrow \mathsf{c} + 1$
2. $(\mathsf{inf}_1, \mathsf{inf}_2) \leftarrow \mathcal{O}.\mathsf{Sample}()$
3. $\pi(N \| M) \leftarrow \mathsf{inf}_1$
4. **if** $\mathsf{e} \in \{0, \mathsf{fwl}_i - 1\} \cup \{\mathsf{fwr}_i + 1, \ldots, n\}$
5. $\quad C_{\mathsf{e}} \leftarrow k_{\mathsf{e}} \star \mathsf{inf}_1$
6. **else**
7. $\quad C_{\mathsf{fwl}_i} \leftarrow \mathsf{inf}_2$
8. $\quad$ **for** $j \in \{\mathsf{fwl}_i + 1, \ldots, \mathsf{e}\}$ **do**
9. $\quad\quad C_j \leftarrow \Delta_j \star C_{j-1}$
10. $\mathsf{inf} \leftarrow (\mathsf{inf}_1, \mathsf{inf}_2)$
11. $\mathcal{L} \leftarrow \mathcal{L} \cup \{(\mathsf{c}, C_{\mathsf{e}}, \mathsf{e}; \mathsf{inf})\}$
12. **return** C_{e}

$\mathcal{O}.\mathsf{Next}$

1. $\mathsf{e} \leftarrow \mathsf{e} + 1$

$\mathcal{O}.\mathsf{Upd}(C_{\mathsf{e}-1})$

1. **if** $(\mathsf{c}, C_{\mathsf{e}-1}, \mathsf{e} - 1; \mathsf{inf}) \notin \mathcal{L}$
2. $\quad$ **return** $\perp$
3. **if** $\mathsf{e} \in \{1, \ldots, \mathsf{fwl}_i - 1\} \cup \{\mathsf{fwr}_i + 1, \ldots, n\}$
4. $\quad (\mathsf{inf}_1, \mathsf{inf}_2) \leftarrow \mathsf{inf}$
5. $\quad C_{\mathsf{e}} \leftarrow k_{\mathsf{e}} \star \mathsf{inf}_1$
6. **else**
7. $\quad (\mathsf{inf}_1, \mathsf{inf}_2) \leftarrow \mathsf{inf}$
8. $\quad C_{\mathsf{fwl}_i} \leftarrow \mathsf{inf}_2$
9. $\quad$ **for** $j \in \{\mathsf{fwl}_i + 1, \ldots, \mathsf{e}\}$ **do**
10. $\quad\quad C_j \leftarrow \Delta_j \star C_{j-1}$
11. $\mathcal{L} \leftarrow \mathcal{L} \cup \{(\mathsf{c}, C_{\mathsf{e}}, \mathsf{e}; \mathsf{inf})\}$
12. **return** C_{e}

$\mathcal{O}.\mathsf{Corr}(\mathsf{inp}, \hat{\mathsf{e}})$

1. **do** $\mathsf{Check}(\mathsf{inp}, \hat{\mathsf{e}}; \mathsf{e}; \mathsf{fwl}_i, \mathsf{fwr}_i)$
2. **if** $\mathsf{inp} = \mathsf{key}$
3. $\quad \mathcal{K} \leftarrow \mathcal{K} \cup \{\hat{\mathsf{e}}\}$
4. $\quad$ **return** $k_{\hat{\mathsf{e}}}$
5. **if** $\mathsf{inp} = \mathsf{token}$
6. $\quad \mathcal{T} \leftarrow \mathcal{T} \cup \{\hat{\mathsf{e}}\}$
7. $\quad$ **return** $\Delta_{\hat{\mathsf{e}}}$

$\mathcal{O}.\mathsf{Chall}(\bar{M}, \bar{C})$

1. **if** $(\mathsf{c}, \bar{C}, \tilde{\mathsf{e}} - 1; \mathsf{inf}) \notin \mathcal{L}$
2. $\quad$ **return** ABORT
3. **if** $b = 0$
4. $\quad (s, t) \leftarrow \mathcal{O}.\mathsf{Sample}()$
5. $\quad \pi(N \| \bar{M}) \leftarrow s$
6. $\quad \tilde{C}_{\mathsf{fwl}_i} \leftarrow t$
7. **else**
8. $\quad (\mathsf{inf}_1, \mathsf{inf}_2) \leftarrow \mathsf{inf}$
9. $\quad \pi(N \| \bar{M}) \xleftarrow{\$} S$
10. $\quad \tilde{C}_{\mathsf{fwl}_i} \leftarrow \mathsf{inf}_2$
11. **for** $j \in \{0, \ldots, \mathsf{fwl}_i - 1\}$ **do**
12. $\quad \tilde{C}_j \leftarrow (\prod_{k=j}^{1} \Delta_k)(\prod_{k=1}^{\tilde{\mathsf{e}}-1} \Delta_k^{-1}) \star \bar{C}$ //left
13. **for** $j \in \{\mathsf{fwl}_i + 1, \ldots, \mathsf{fwr}_i\}$ **do**
14. $\quad \tilde{C}_j \leftarrow \Delta_j \star \tilde{C}_{j-1}$ //embed
15. **for** $j \in \{\mathsf{fwr}_i + 1, \ldots, n\}$ **do**
16. $\quad \tilde{C}_j \leftarrow k_j \star \pi(N \| \bar{M})$ //right
17. $\tilde{\mathcal{L}} \leftarrow \cup_{j=0}^{n} \{(\tilde{C}_j, j)$
18. **return** $\tilde{C}_{\mathsf{e}}$

$\mathcal{O}.\mathsf{Upd}\tilde{\mathsf{C}}$

1. $\mathcal{C} \leftarrow \mathcal{C} \cup \{\mathsf{e}\}$
2. **find** $(\tilde{C}_{\mathsf{e}}, \mathsf{e}) \in \tilde{\mathcal{L}}$
3. **return** $\tilde{C}_{\mathsf{e}}$

Figure 2.4: Our reduction $\mathcal{B}$ for proof of Theorem 2.4 in hybrid i. inf encodes fixed programming information: it marks two set elements $(\mathsf{inf}_1, \mathsf{inf}_2)$ sampled with $\mathcal{O}.\mathsf{Sample}$. inf_1 is the randomness used during encryption and inf_2 is the ciphertext value in epoch fwl_i. ors refers to the set $\{\mathcal{O}.\mathsf{Enc}, \mathcal{O}.\mathsf{Next}, \mathcal{O}.\mathsf{Upd}, \mathcal{O}.\mathsf{Corr}\}$. $\bowtie$ indicates that Δ_0 and $\Delta_{\mathsf{fwr}_i+1}$ are skipped in the computation. Comments start with //.

Recall that $\mathcal{A}_i$ is an adversary attempting to distinguish $\mathcal{G}_i^0$ from $\mathcal{G}_i^1$. $\mathcal{B}$ will try to use $\mathcal{A}$ to break the weak pseudorandomness of the group action $(G, S, \star)$. In $\mathbf{Exp}^{\mathsf{wk\text{-}PR}}_{\mathcal{G}\mathcal{A},\mathcal{B}}(\lambda)$, when $\mathcal{O}.\mathsf{Sample}$ returns pairs of the form $(s_j, g \star s_j)$ for $g \xleftarrow{\$} G$ and $s_j \xleftarrow{\$} S$, $\mathcal{B}$ will perfectly simulate the environment of $\mathcal{A}_i$ in $\mathcal{G}_i^b$. When $\mathcal{O}.\mathsf{Sample}$ returns pairs of the form (s_j, t_j) for $s_j, t_j \xleftarrow{\$} S$, $\mathcal{B}$ will give random inputs to $\mathcal{A}_i$ such that $\mathcal{A}_i$ distinguishes $\mathcal{G}_i^0$ from $\mathcal{G}_i^1$ with advantage 0. We explain how our reduction $\mathcal{B}$ does this without knowing which $\mathcal{O}.\mathsf{Sample}$ oracle was provided to it.

The reduction $\mathcal{B}$ receives the oracle $\mathcal{O}.\mathsf{Sample}$, takes $b \xleftarrow{\$} \{0,1\}$ and simulates $\mathcal{G}_i^b$. Whenever the reduction needs to provide an output of $\pi(\cdot)$ to $\mathcal{A}_i$, it chooses some set value $s \in S$ such that $\pi(\cdot) = s$. In this setting, computing π^{-1} is simply a lookup to this mapping of the ideal cipher π. We explain our simulation:

Initially,

1. $\mathcal{B}$ guesses the values of fwl_i and fwr_i.
2. $\mathcal{B}$ generates all keys and tokens except for $k_{\mathsf{fwl}_i}, \ldots, k_{\mathsf{fwr}_i}, \Delta_{\mathsf{fwl}_i}, \Delta_{\mathsf{fwr}_i+1}$. If $\mathcal{A}_i$ corrupts these keys and tokens, this means that the firewall guess is wrong and the reduction aborts the game using the Check algorithm, of [BDGJ20], presented in Fig. 2.5.

$\mathsf{Check}(\mathsf{inp}, \hat{\mathsf{e}}; \mathsf{e}; \mathsf{fwl}, \mathsf{fwr})$

1. **if** $\hat{\mathsf{e}} > \mathsf{e}$
2. **return** $\perp$
3. **if** $\mathsf{inp} = \mathsf{key}$ **and** $\hat{\mathsf{e}} \in \{\mathsf{fwl}, \ldots, \mathsf{fwr}\}$
4. **return** `ABORT`
5. **if** $\mathsf{inp} = \mathsf{token}$ **and** $\hat{\mathsf{e}} \in \{\mathsf{fwl}, \ldots, \mathsf{fwr}+1\}$
6. **return** `ABORT`

Figure 2.5: Algorithm Check of [BDGJ20] used in our proofs. $\hat{\mathsf{e}}$ is the epoch in the adversary's request and e is the current epoch.

$\mathcal{B}$ will operate so as to embed the value g used by $\mathcal{O}.\mathsf{Sample}$ to the key k_{fwl_i} and the value $gk^{-1}_{\mathsf{fwl}_i-1}$ to the token Δ_{fwl_i}. If $\mathcal{O}.\mathsf{Sample}$ returns uniformly distributed pairs of set elements instead, all the ciphertexts inside insulated region i will be random set elements (no key or token could possibly explain these ciphertexts).

To simulate a non-challenge ciphertext that is:

- An $\mathcal{O}.\mathsf{Enc}$ query in epoch $\mathsf{e} \in \{0, \ldots \mathsf{fwl}_i - 1\} \cup \{\mathsf{fwr}_i + 1, \ldots, n\}$: $\mathcal{B}$ queries $\mathcal{O}.\mathsf{Sample}$ to get a pair $(s,t) \in S^2$. $\mathcal{B}$ uses s as a random value by programming $\pi(\cdot) \leftarrow s$ (so the randomness will be consistent with calls that $\mathcal{A}_i$ makes to $\mathcal{O}.\mathsf{Upd}$), computes the ciphertext $C_{\mathsf{e}} = k_{\mathsf{e}} \star s$ (the value of k_{e} is known to $\mathcal{B}$ in these epochs) and stores (s,t) in its memory for later use. To respond to $\mathcal{O}.\mathsf{Upd}$ queries in these epochs, $\mathcal{B}$ computes $C_{\mathsf{e}} = k_{\mathsf{e}} \star s$ using the randomness s generated during the first encryption of the input ciphertext.
- An $\mathcal{O}.\mathsf{Enc}$ query in epoch $\mathsf{e} \in \{\mathsf{fwl}_i, \ldots, \mathsf{fwr}_i\}$: $\mathcal{B}$ queries $\mathcal{O}.\mathsf{Sample}$ to get a pair $(s,t) \in S^2$ and programs $\pi(\cdot) \leftarrow s$. It sets $C_{\mathsf{fwl}_i} = t$ (so that all ciphertexts will be encrypted under the key g in epoch fwl_i if $\mathcal{O}.\mathsf{Sample}$ returns pairs of the form $(s_j, g \star s_j)$) and updates C_{fwl_i} to the right epoch e using its simulated tokens (remember that $\mathcal{B}$ does not know the keys inside the i-th insulated region). To respond to $\mathcal{O}.\mathsf{Upd}$ queries in these epochs, $\mathcal{B}$ uses the value t (if $t = g \star s$ then the randomness will still be consistent) generated during the first encryption of the input ciphertext as ciphertext value in epoch fwl_i and updates t to the right epoch e using its simulated tokens.

During the challenge call, the adversary will provide a ciphertext $\bar{C}$ which was created during the c-th call to $\mathcal{O}.\mathsf{Enc}$. The adversary cannot ask for an update of the c-th encryption in an epoch $\mathsf{e} \geq \mathsf{fwl}_i$, as this would trigger the trivial win condition $[\mathsf{fwl}_i, \mathsf{fwr}_i] \subseteq \mathcal{I}^* \cap \mathcal{C}^* \neq \emptyset$.

To simulate challenge-equal ciphertext in an epoch that is:

- To the left of the i-th insulated region: $\mathcal{B}$ simulates $\mathsf{GAINE.Upd}(\bar{C})$ using tokens that it created itself.
- Within the i-th insulated region: $\mathcal{B}$ simulates $\mathsf{GAINE.Upd}(\bar{C})$ if $b = 1$, and simulates $\mathsf{GAINE.Enc}(\bar{M})$ if $b = 0$. More precisely, if $\mathcal{O}.\mathsf{Sample}$ returns pairs of the form $(s_j, g \star s_j)$, $\mathcal{B}$ embeds g to k_{fwl_i} and $gk_{\mathsf{fwl}_i-1}^{-1}$ to Δ_{fwl_i}. If $b = 0$, the reduction samples $(s, t) \leftarrow \mathcal{O}.\mathsf{Sample}()$, gives value s to $\pi(N \| \bar{M})$ and t to $\tilde{C}_{\mathsf{fwl}_i}$ (we want $k_{\mathsf{fwl}_i} = g$) since

$$\tilde{C}_{\mathsf{fwl}_i} = \mathsf{GAINE.Enc}(\bar{M}) = k_{\mathsf{fwl}_i} \star \pi(N \| \bar{M})$$

  If $b = 1$, assume that $\bar{C}$ is an update of $\bar{C}_{\mathsf{e_c}}$, the output of the c-th $\mathcal{O}.\mathsf{Enc}$ query. $\mathcal{B}$ sampled $(s, t) \leftarrow \mathcal{O}.\mathsf{Sample}()$ and used s as randomness to create $\bar{C}_{\mathsf{e_c}}$ and to update it in epochs $\mathsf{e} < \mathsf{fwl}_i$. The reduction gives value t to $\tilde{C}_{\mathsf{fwl}_i}$ (we want $\Delta_{\mathsf{fwl}_i} = gk_{\mathsf{fwl}_i-1}^{-1}$) since

$$\tilde{C}_{\mathsf{fwl}_i} = \mathsf{GAINE.Upd}(\bar{C}) = \Delta_{\mathsf{fwl}_i} \star (k_{\mathsf{fwl}_i-1} \star s)$$

  Furthermore, the reduction uses tokens $\Delta_{\mathsf{fwl}_i+1}, \ldots, \Delta_{\mathsf{fwr}_i}$ to update $\tilde{C}_{\mathsf{fwl}_i}$ to simulate all challenge ciphertexts in epochs within the insulated region.
- To the right of the i-th insulated region: $\mathcal{B}$ simulates $\mathsf{GAINE.Enc}(\bar{M})$ using the keys that it created itself.

Eventually, $\mathcal{B}$ receives the output bit b' from $\mathcal{A}_i$. If $b' = b$, then $\mathcal{B}$ guesses that $\mathcal{O}.\mathsf{Sample}$ returned pairs of the form $(s_j, g \star s_j)$ (returns 0 to the $\mathsf{wk\text{-}PR}$ challenger), otherwise, $\mathcal{B}$ guesses that it has seen uniformly chosen pairs of set elements (returns 1). If $\mathcal{B}$ receives an oracle $\mathcal{O}.\mathsf{Sample}$ that samples pairs of the form $(s_j, g \star s_j)$, then $\mathcal{B}$ perfectly simulates the environment of $\mathcal{A}_i$ in $\mathcal{G}_i^b$. If $\mathcal{B}$ receives an oracle $\mathcal{O}.\mathsf{Sample}$ that samples pairs uniformly at random, then $\mathcal{B}$ wins with probability $1/2$. Thus,

$$\begin{aligned}
\mathbf{Adv}^{\mathsf{wk\text{-}PR}}_{\mathcal{GA},\mathcal{B}}(\lambda) &= \left|1/2 - \Pr[\mathbf{Exp}^{\mathsf{wk\text{-}PR\text{-}0}}_{\mathcal{GA},\mathcal{B}} = 1]\right| \\
&= \left|1/2 - (1/2 \Pr[\mathcal{G}_i^0 = 0] + 1/2 \Pr[\mathcal{G}_i^1 = 1])\right| \\
&= \left|1/2 - 1/2(1 - \Pr[\mathcal{G}_i^0 = 1]) - 1/2 \Pr[\mathcal{G}_i^1 = 1]\right| \\
&= 1/2 \left|\Pr[\mathcal{G}_i^0 = 1] - \Pr[\mathcal{G}_i^1 = 1]\right|
\end{aligned}$$

Finally, we get

$$\begin{aligned}
\frac{1}{(n+1)^2} \mathbf{Adv}^{\mathsf{detIND\text{-}UE\text{-}CPA}}_{\mathsf{GAINE},\mathcal{A}}(\lambda) &\leq \sum_{i=1}^{\ell} |\Pr[\mathcal{G}_i^1 = 1] - \Pr[\mathcal{G}_i^0 = 1]| \\
&= 2\ell \, \mathbf{Adv}^{\mathsf{wk\text{-}PR}}_{\mathcal{GA},\mathcal{B}}(\lambda) \\
&\leq 2(n+1) \, \mathbf{Adv}^{\mathsf{wk\text{-}PR}}_{\mathcal{GA},\mathcal{B}}(\lambda)
\end{aligned}$$

and thus $\mathbf{Adv}^{\mathsf{detIND\text{-}UE\text{-}CPA}}_{\mathsf{GAINE},\mathcal{A}}(\lambda) \leq 2(n+1)^3 \, \mathbf{Adv}^{\mathsf{wk\text{-}PR}}_{\mathcal{GA},\mathcal{B}}(\lambda)$. □

That being said, finding a post-quantum instantiation for $\mathsf{GAINE0}$ would make it the first post-quantum $\mathsf{detIND\text{-}UE\text{-}CCA}$ UE scheme. Indeed, the two LWE-based schemes of Jiang [Jia20] and Nishimaki [Nis22] are only $\mathsf{randIND\text{-}UE\text{-}CPA}$ secure. Finally, it is also worth noting that [BDGJ20, Th 2.2, 2.4 & 2.6] showed that $\mathsf{detIND\text{-}UE\text{-}CCA}$ security implies the $\mathsf{detIND\text{-}ENC\text{-}CCA}$ and $\mathsf{detIND\text{-}UPD\text{-}CCA}$ security of [LT18], so $\mathsf{GAINE0}$ is also CCA secure in the [LT18] sense.

2.2.3 Post-quantum instantiations of GAINE

After proving the security of GAINE, we look for post-quantum weakly pseudorandom MEGAs in the literature to instantiate our scheme. A good candidate is the non-abelian group action of the general linear group on the set of alternating trilinear forms introduced by Tang *et al.* [TDJ+22]. Some definitions are in order.

Let $\mathbb{F}_q$ be the finite field of order q. A trilinear form $\phi : \mathbb{F}_q^n \times \mathbb{F}_q^n \times \mathbb{F}_q^n \to \mathbb{F}_q$ is alternating if ϕ evaluates to 0 whenever two arguments are equal. Let $\mathrm{ATF}(n,q)$ be the set of all alternating trilinear forms defined over $\mathbb{F}_q^n$. The general linear group $\mathrm{GL}(n,q)$ acts on $\mathrm{ATF}(n,q)$ as follows: $A \in \mathrm{GL}(n,q)$ sends ϕ to $A \star \phi := \phi \circ A$, defined as $(\phi \circ A)(u,v,w) := \phi(A^t(u), A^t(v), A^t(w))$. This action defines an equivalence relation $\sim$ on $\mathrm{ATF}(n,q)$, namely $\phi \sim \psi$ if only there exists $A \in \mathrm{GL}(n,q)$, such that $\phi = \psi \circ A$.

$\phi \in \mathrm{ATF}(n,q)$ can be represented as $\sum_{1 \le i < j < k \le n} c_{i,j,k} e_i^* \wedge e_j^* \wedge e_k^*$, where $c_{i,j,k} \in \mathbb{F}_q$, e_i is the i-th standard basis vector, e_i^* is the linear form sending $u = (u_1, \ldots, u_n)^t \in \mathbb{F}_q^n$ to u_i, and $\wedge$ denotes the wedge product (see [TDJ+22] for more details). This representation requires $\binom{n}{3}\lceil \log q \rceil$ bits.

Concretely, $\star$ can be computed as follows. Let $A = (a_{i,j}) \in \mathrm{GL}(n,q)$, it sends $e_i^* \wedge e_j^* \wedge e_k^*$ to $\sum_{1 \le r < s < t \le n} d_{r,s,t} e_r^* \wedge e_s^* \wedge e_t^*$, where $d_{r,s,t} = \det \begin{vmatrix} a_{i,r} & a_{i,s} & a_{i,t} \\ a_{j,r} & a_{j,s} & a_{j,t} \\ a_{k,r} & a_{k,s} & a_{k,t} \end{vmatrix}$. For any $\phi \in \mathrm{ATF}(n,q)$, $A \star \phi$ can be obtained by linearly extending the action of A to each term $e_i^* \wedge e_j^* \wedge e_k^*$.

It is clear that $(\mathrm{GL}(n,q), \mathrm{ATF}(n,q), \star)$ is an EGA. Indeed, membership testing, equality testing, sampling, operation and inversion can all be done efficiently in $\mathrm{GL}(n,q)$. The group action is efficiently computable. Membership testing and unique representation in $\mathrm{ATF}(n,q)$ stem from the algorithmic representation given above. Moreover, we can define an invertible map between binary strings of length $\binom{n}{3}\lceil \log q \rceil$ and elements of $\mathrm{ATF}(n,q)$ using this representation. Thus, $(\mathrm{GL}(n,q), \mathrm{ATF}(n,q), \star)$ is a MEGA and can be considered to instantiate GAINE.

[TDJ+22, Conjecture 1] states that the *alternating trilinear form equivalence* problem (ATFE) is hard and that $(\mathrm{GL}(n,q), \mathrm{ATF}(n,q), \star)$ is weakly pseudorandom (in the post-quantum setting). The ATFE problem is the following: given $\phi, \psi \in \mathrm{ATF}(n,q)$, decide if there exists $A \in \mathrm{GL}(n,q)$ such that $\phi = \psi \circ A$. See [TDJ+22, Section 4 & 5] for an argumentation on why it is reasonable to believe in [TDJ+22, Conjecture 1]. The authors conclude that the best attack against ATFE is in $\mathcal{O}(q^{2n/3} n^{2\omega} \log q)$ where ω is the matrix multiplication exponent.

Another post-quantum weakly pseudorandom MEGA is the general linear group (non-abelian) action on k-tensors (GLAT) of [JQSY19] for $k \ge 3$. For simplicity, we only present the case when $k = 3$ since it is the most studied and believed to be hard. A 3-tensor T is a 3-dimensional array with 3 indices (i_1, i_2, i_3) over a finite field $\mathbb{F}_q$ where $i_j \in \{1, \ldots, d_j\}$. The tensor is said to have order 3 and dimensions (d_1, d_2, d_3) over $\mathbb{F}_q$. The direct product of general linear groups $G := \prod_{i=1}^3 \mathrm{GL}(d_i, q)$ acts on the set of 3-tensors via the following action $\star$ that represents a local change of basis. For $A = (A^{(j)})_{j=1}^k \in G$ and a 3-tensor T,

$$A \star T := \hat{T} \text{ where } \hat{T}_{i_1,i_2,i_3} := \sum_{l_1,l_2,l_3} \left(\prod_{j=1}^{3} A^{(j)}_{i_j,l_j} \right) T_{l_1,l_2,l_3}$$

[JQSY19] also argues that GLAT is weakly-pseudorandom in the post-quantum setting, we defer the argumentation to [JQSY19, Section 5]. This gives us a second post-quantum candidate to instantiate GAINE.

However, neither of those group actions are weak pseudorandom over a large number of samples. Indeed, in both cases, the set S is a vector space of finite dimension, isomorphic to $\mathbb{F}^N$

for a finite field $\mathbb{F}$ and an integer N. Moreover, both actions act linearly on this vector space. This means that the map $f_g : s \mapsto g \star s$ is linear. Given enough pairs of the form $(s, g \star s)$ for a fixed group element g and random set elements s, one can reconstruct f_g and evaluate it on any set element. Since the pairs $(s, g \star s)$ are exactly those of the weak pseudorandom experiment, we see that these two actions can only be conjectured to be weak pseudorandom over a small number of samples. Since our security proof for GAINE uses one such sample per ciphertext, we can only use these two group actions to instantiate GAINE when there are few ciphertexts, which is unpractical. Moreover, a recent attack by Beullens [Beu22] breaks the ATFE problem by finding useful graph theoretic invariants.

2.2.4 On the detIND-UE-CCA security of GAINE

In [BDGJ20, sec. 5.1.1], a variant of SHINE with added ciphertext integrity, called SHINE0, is given by using $N\|M\|0^t$, for some t, as input of the permutation π during encryption and by checking that the 0 string is still present during decryption (if not $\perp$ is returned). This version of SHINE is shown to be detIND-UE-CCA secure under CDH [BDGJ20, th. 5].

We define GAINE0 (see Fig. 2.6) similarly to SHINE0 and prove that it is detIND-UE-CCA secure if the group action is *weakly unpredictable* (see Definition 2.6). Informally, recall that $(G, S, \star)$ is weakly unpredictable if, given polynomially many tuples of the form $(s_i, g \star s_i)$ where $g \xleftarrow{\$} G$ and each $s_i \xleftarrow{\$} S$, there is no PPT adversary that can compute $g \star s^*$ for a given challenge $s^* \xleftarrow{\$} S$. A full proof and precise definitions are given in the following Section 2.2.5 and Section 2.2.6.

This means that finding a post-quantum group action to instantiate GAINE would provide the first CCA-secure post-quantum UE scheme.

2.2.5 GAINE with zeros: GAINE0

We add ciphertext integrity to GAINE using the technique of [BDGJ20, sec. 5.1.1] for their SHINE0 scheme. Take message space $\mathcal{M} = \{0,1\}^m$ and nonce space $\mathcal{N} = \{0,1\}^v$. Let $(G, S, \star)$ be a MEGA with permutation $\pi : \{0,1\}^{m+v+t} \to S$. The encryption algorithm of GAINE0 feeds as input to π the concatenation of the message, the random nonce, and a zero string of length t. The decryption returns $\perp$ if the decrypted value does not end with 0^t. GAINE0 is defined in Fig. 2.6.

2.2.6 GAINE0 is INT-CTXT$^{\mathsf{s}}$

We prove the following theorem.

Theorem 2.5 (GAINE0 is INT-CTXT$^{\mathsf{s}}$)**.** *Let* GAINE0 *be the UE scheme described in Fig. 2.6 for a MEGA family $\mathcal{GA}$. For any ideal cipher model adversary $\mathcal{A}$ that makes at most Q_E encryption queries before calling $\mathcal{O}.\mathsf{Try}$, there exists a reduction $\mathcal{B}$ such that*

$$\mathbf{Adv}^{\mathsf{INT\text{-}CTXT^s}}_{\mathsf{GAINE0},\mathcal{A}}(\lambda) \leq \mathcal{O}(1) Q_E (n+1)^2 \, \mathbf{Adv}^{\mathsf{wk\text{-}UP}}_{\mathcal{GA},\mathcal{B}}(\lambda) + \mathsf{negl}(\lambda)$$

where $\mathbf{Adv}^{\mathsf{wk\text{-}UP}}_{\mathcal{GA},\mathcal{B}}(\lambda)$ is defined in Definition 2.6

Remark 2. Combining the results of [BDGJ20, Theorem 3], which states that a UE scheme that is CPA and CTXT is also CCA, with our Theorem 2.4 and Theorem 2.5, we have that GAINE0 is detIND-UE-CCA.

We follow the proof technique of [BDGJ20] and its presentation.

$\mathsf{Setup}(1^\lambda)$:

1. $(G, S, \star) \leftarrow \mathcal{GA}(\lambda)$
2. Choose π, m, v, t as above
3. $\mathsf{pp} \leftarrow (G, S, \star), \pi, m, v, t$
4. **return** pp

$\mathsf{KeyGen}(\mathsf{pp})$:

1. $k \xleftarrow{\$} G$.
2. **return** k.

$\mathsf{TokenGen}(k_\mathsf{e}, k_{\mathsf{e}+1})$:

1. $\Delta_{\mathsf{e}+1} \leftarrow k_{\mathsf{e}+1} k_\mathsf{e}^{-1}$
2. **return** $\Delta_{\mathsf{e}+1}$

$\mathsf{Upd}(\Delta_{\mathsf{e}+1}, C_\mathsf{e})$:

1. $C_{\mathsf{e}+1} \leftarrow \Delta_{\mathsf{e}+1} \star C_\mathsf{e}$
2. **return** $C_{\mathsf{e}+1}$

$\mathsf{Enc}(k_\mathsf{e}, M)$:

1. $N \xleftarrow{\$} \mathcal{N}$.
2. $C_\mathsf{e} \leftarrow k_\mathsf{e} \star \pi(N \| M \| 0^t)$
3. **return** C_e

$\mathsf{Dec}(k_\mathsf{e}, C_\mathsf{e})$:

1. $s \leftarrow \pi^{-1}(k_\mathsf{e}^{-1} \star C_\mathsf{e})$
2. Parse s as $N' \| M' \| Z$
3. **if** $Z = 0^t$
4. **return** M'
5. **else**
6. **return** $\perp$

Figure 2.6: GAINE0: our generalization of the SHINE0 scheme using group actions.

Proof method. The challenger of the $\mathsf{INT\text{-}CTXT^s}$ game keeps a list of consistent values for ciphertexts, *i.e.*, the underlying permutation output. Let $\tilde{C}$ be a forgery attempt sent to $\mathcal{O}.\mathsf{Try}$ in epoch $\tilde{\mathsf{e}}$ and let $\tilde{c} := k_{\tilde{\mathsf{e}}}^{-1} \star \tilde{C}$ be the underlying permutation output.

1. If $\tilde{c}$ is a new value, since π is a random permutation, then the $\mathsf{INT\text{-}CTXT^s}$ challenger simulates $\pi^{-1}(\tilde{c})$ to be a random string. The probability that this string ends by 0^t is negligible, and this carries over to the probability that the adversary wins the $\mathsf{INT\text{-}CTXT^s}$ game.
2. If $\tilde{c}$ already exists, suppose that this happens with probability p. We construct a reduction $\mathcal{B}$ playing the wk-UP game such that it wins with probability $p/(Q_E(n+1)^2)$. $\mathcal{B}$ guesses the location of the firewalls around the challenge epoch, embeds the wk-UP values and simulates the $\mathsf{INT\text{-}CTXT^s}$ game, using any successfully-forged ciphertext to compute the group action forgery for its wk-UP challenger.

Proof. The following proof is practically the same as in [BDGJ20], we just replaced exponentiations by group actions and CDH by wk-UP. We give the proof of [BDGJ20] for completeness.

Note that the probability of a random string ends by 0^t is $1/2^t$. In the $\mathsf{INT\text{-}CTXT^s}$ game, the adversary ultimately sends a forgery C^* to the $\mathcal{O}.\mathsf{Try}$ oracle. If the trivial win condition does not trigger, then C^* is a new ciphertext to the challenger and there exists an insulated region around the challenge epoch. We split the proof into two parts on if $k_\mathsf{e}^{-1} \star C^*$ is a new value to the challenger:

1. If $k_\mathsf{e}^{-1} \star C^*$ is a new value, the random permutation π^{-1} will pick a random string a as the output of $\pi^{-1}(k_\mathsf{e}^{-1} \star C^*)$. The probability of a ending with 0^t is upper bounded by $1/2^t$.
2. If $k_\mathsf{e}^{-1} \star C^*$ is an existing value, denote this event as E_3, we claim that the probability of E_3 happens is very low. Which means it is hard to provide a valid forgery with a known permutation value. In other words, without the knowledge of the encryption key, it is difficult to provide a correct group action. Formally, we prove the following inequality in

Lemma 2.6 :

$$\Pr[E_3] := \Pr[k_{\mathsf{e}}^{-1} \star C^* \text{ exists}, C^* \text{ is new}] \leq Q_E(n+1)^2 \, \mathbf{Adv}^{\mathsf{wk\text{-}UP}}_{\mathcal{GA}}$$

In order to analyze the security, we define some events:

- $E_1 := \{C^* \text{ is new}\}$,
- $E_2 := \{k_{\mathsf{e}}^{-1} \star C^* \text{ is new}, C^* \text{ is new}\}$,
- Recall $E_3 := \{k_{\mathsf{e}}^{-1} \star C^* \text{ exists}, C^* \text{ is new}\}$.

Denote the experiment $\mathbf{Exp}^{\mathsf{INT\text{-}CTXT^s}}_{\mathsf{GAINE0},\mathcal{A}}$ to be $\mathbf{Exp}$. We have :

- $\Pr[\mathbf{Exp} = 1 \mid \neg E_1] = 0$.
- We proved $\Pr[\mathbf{Exp} = 1 \mid E_2] \leq 1/2^t$ in part 1.
- Events $\neg E_1, E_2, E_3$ are disjoint from each other, so $\Pr[\neg E_1] + \Pr[E_2] + \Pr[E_3] = 1$.
- We prove $\Pr[E_3] \leq Q_E(n+1)^2 \, \mathbf{Adv}^{\mathsf{wk\text{-}UP}}_{\mathcal{GA}}$ in Lemma 2.6.
  Applying the above properties, we can compute the $\mathsf{INT\text{-}CTXT^s}$ advantage

$$\begin{aligned}
\mathbf{Adv}^{\mathsf{INT\text{-}CTXT^s}}_{\mathsf{GAINE0},\mathcal{A}}(\lambda) &= \Pr[\mathbf{Exp} = 1] \\
&= \Pr[\mathbf{Exp} = 1 \mid \neg E_1] \cdot \Pr[\neg E_1] + \Pr[\mathbf{Exp} = 1 \mid E_2] \cdot \Pr[E_2] \\
&\quad + \Pr[\mathbf{Exp} = 1 \mid E_3] \cdot \Pr[E_3] \\
&= \Pr[\mathbf{Exp} = 1 \mid E_2] \cdot \Pr[E_2] + \Pr[\mathbf{Exp} = 1 \mid E_3] \cdot \Pr[E_3] \\
&\leq \Pr[\mathbf{Exp} = 1 \mid E_2] + \Pr[E_3] \\
&\leq 1/2^t + Q_E(n+1)^2 \, \mathbf{Adv}^{\mathsf{wk\text{-}UP}}_{\mathcal{GA}}
\end{aligned}$$

□

Lemma 2.6. *Let $\mathcal{GA}$ be the MEGA family used in* GAINE0*. Let $\mathcal{A}$ be an* $\mathsf{INT\text{-}CTXT^s}$ *adversary against* GAINE0 *that asks at most Q_E queries to $\mathcal{O}$.*Enc *before it sends its $\mathcal{O}$.*Try *query. Suppose that $\tilde{C}$ is a forgery attempt provided by $\mathcal{A}$ and that its corresponding permutation value is $\tilde{c}$. Define E to be the event that $\tilde{c}$ is an existed value but $\tilde{C}$ is a new one. Then, there exists a reduction $\mathcal{B}$ such that*

$$\Pr[E] \leq Q_E(n+1)^2 \, \mathbf{Adv}^{\mathsf{wk\text{-}UP}}_{\mathcal{GA},\mathcal{B}}$$

Proof. Suppose that $\mathcal{A}$ is an adversary against $\mathsf{INT\text{-}CTXT^s}$, and that it can provide a forgery such that $\tilde{C}$ is a new ciphertext but the underlying permutation value is an existing one with probability $\Pr[E]$. We give a reduction $\mathcal{B}$, in Fig. 2.7, that wins the wk-UP game with probability $\Pr[E]/(Q_E(n+1)^2)$.

$\mathcal{B}$ guesses the location of firewalls ($\hat{\mathsf{fwl}}$ and $\hat{\mathsf{fwr}}$) around the epoch when $\mathcal{O}$.Try is queried. Furthermore, it guesses which message (the h-th encryption) will be the underlying message of the forgery. Then, $\mathcal{B}$ receives the wk-UP values $(G, S, \star)$, $\mathcal{O}$.Sample and s^*, where $\mathcal{O}$.Sample returns tuples of the form $(s_i, g \star s_i)$ for a fixed $g \xleftarrow{\$} G$ and $s_i \xleftarrow{\$} S$. $\mathcal{B}$ embeds g (the group element used in $\mathcal{O}$.Sample) to $k_{\hat{\mathsf{fwl}}}$ by using the elements sampled by $\mathcal{O}$.Sample as ciphertexts in epoch $\hat{\mathsf{fwl}}$. On the h-th encryption, $\mathcal{B}$ embeds s^* to $\pi(N \| M \| 0^t)$. When $\mathcal{B}$ receives a forgery $\tilde{C}$ for the h-th encryption in epoch $\tilde{\mathsf{e}} \in \{\hat{\mathsf{fwl}}, \ldots, \hat{\mathsf{fwr}}\}$, it can downgrade $\tilde{C}$ to epoch $\hat{\mathsf{fwl}}$ (where it embedded g to the epoch key). Then, $(\prod_{\mathsf{e}=\tilde{\mathsf{e}}}^{\hat{\mathsf{fwl}}+1} \Delta_{\mathsf{e}}^{-1}) \star \tilde{C} = g \star s^*$ with probability $\Pr[E]/(Q_E(n+1)^2)$, which is the advantage of winning the $\mathbf{Exp}^{\mathsf{wk\text{-}UP}}_{\mathcal{GA},\mathcal{B}}$ game. □

Reduction $\mathcal{B}$ playing $\mathbf{Exp}^{\mathsf{wk\text{-}UP}}_{\mathcal{GA},\mathcal{B}}(\lambda)$

1. **receive** $(G, S, \star)$, $\mathcal{O}.\mathsf{Sample}$ and s^*
2. **do Setup**(1^λ)
3. $\mathcal{A}^{\mathtt{ors}}(\lambda)$
4. **if** ABORT occurred **or** $\mathsf{twf} = 1$
5. $\quad \mathsf{win} \leftarrow 0$
6. **else**
7. $\quad$ **return** win

Setup(1^λ)

1. $\mathsf{pp} \leftarrow \mathsf{GAINE0.Setup}(1^\lambda)$
2. $k_0 \leftarrow \mathsf{GAINE0.KeyGen}(\mathsf{pp})$
3. $\Delta_0 \leftarrow \perp$
4. $\mathsf{e}, \mathsf{c}, \mathsf{phase}, \mathsf{win}, \mathsf{twf} \leftarrow 0$
5. $\mathcal{L}^*, \mathcal{C}, \mathcal{K}, \mathcal{T} \leftarrow \emptyset$
6. $\hat{\mathsf{fwl}}, \hat{\mathsf{fwr}} \xleftarrow{\$} \{0, \ldots, n\}$
7. $\mathsf{h} \xleftarrow{\$} \{1, \ldots, Q_E\}$
8. **for** $j \in \{0, \ldots, \hat{\mathsf{fwl}} - 1\} \cup \{\hat{\mathsf{fwr}} + 1, \ldots, n\}$ **do**
9. $\quad k_j \xleftarrow{\$} G; \Delta_j \leftarrow k_j k_{j-1}^{-1}$ $\bowtie$
10. **for** $j \in \{\hat{\mathsf{fwl}} + 1, \ldots, \hat{\mathsf{fwr}}\}$ **do**
11. $\quad\quad \Delta_j \xleftarrow{\$} G$

$\mathcal{O}.\mathsf{Enc}(M)$

1. $\mathsf{c} \leftarrow \mathsf{c} + 1$
2. **if** $\mathsf{c} = \mathsf{h}$
3. $\quad$ **if** $\mathsf{e} < \hat{\mathsf{fwl}}$
4. $\quad\quad \pi(N \| M \| 0^t) \leftarrow s^*$
5. $\quad\quad C_\mathsf{e} \leftarrow k_\mathsf{e} \star s^*$
6. $\quad$ **else**
7. $\quad\quad$ **return** ABORT
8. **else**
9. $\quad (\mathsf{inf}_1, \mathsf{inf}_2) \leftarrow \mathcal{O}.\mathsf{Sample}()$
10. $\quad$ **if** $\mathsf{e} \in \{0, \hat{\mathsf{fwl}} - 1\} \cup \{\hat{\mathsf{fwr}} + 1, \ldots, n\}$
11. $\quad\quad \pi(N \| M \| 0^t) \leftarrow \mathsf{inf}_1$
12. $\quad\quad C_\mathsf{e} \leftarrow k_\mathsf{e} \star \mathsf{inf}_1$
13. $\quad$ **else**
14. $\quad\quad C_{\hat{\mathsf{fwl}}} \leftarrow \mathsf{inf}_2$
15. $\quad\quad$ **for** $j \in \{\hat{\mathsf{fwl}} + 1, \ldots, \mathsf{e}\}$ **do**
16. $\quad\quad\quad C_j \leftarrow \Delta_j \star C_{j-1}$
17. $\mathsf{inf} \leftarrow (\mathsf{inf}_1, \mathsf{inf}_2)$
18. $\mathcal{L}^* \leftarrow \mathcal{L}^* \cup \{(\mathsf{c}, C_\mathsf{e}, \mathsf{e}; \mathsf{inf})\}^{\Rightarrow}$
19. **return** C_e

$\mathcal{O}.\mathsf{Next}$

1. $\mathsf{e} \leftarrow \mathsf{e} + 1$

$\mathcal{O}.\mathsf{Upd}(C_{\mathsf{e}-1})$

1. **if** $(\mathsf{c}, C_{\mathsf{e}-1}, \mathsf{e} - 1; \mathsf{inf}) \notin \mathcal{L}^*$
2. $\quad$ **return** $\perp$
3. **if** $\mathsf{c} = \mathsf{h}$
4. $\quad$ **if** $\mathsf{e} < \hat{\mathsf{fwl}}$
5. $\quad\quad C_\mathsf{e} \leftarrow \Delta_\mathsf{e} \star C_{\mathsf{e}-1}$
6. $\quad$ **else**
7. $\quad\quad$ **return** ABORT
8. **else**
9. $\quad$ **if** $\mathsf{e} \in \{1, \ldots, \hat{\mathsf{fwl}} - 1\} \cup \{\hat{\mathsf{fwr}} + 1, \ldots, n\}$
10. $\quad\quad (\mathsf{inf}_1, \mathsf{inf}_2) \leftarrow \mathsf{inf}$
11. $\quad\quad C_\mathsf{e} \leftarrow k_\mathsf{e} \star \mathsf{inf}_1$
12. $\quad$ **else**
13. $\quad\quad (\mathsf{inf}_1, \mathsf{inf}_2) \leftarrow \mathsf{inf}$
14. $\quad\quad C_{\hat{\mathsf{fwl}}} \leftarrow \mathsf{inf}_2$
15. $\quad\quad$ **for** $j \in \{\hat{\mathsf{fwl}} + 1, \ldots, \mathsf{e}\}$ **do**
16. $\quad\quad\quad C_j \leftarrow \Delta_j \star C_{j-1}$
17. $\mathcal{L}^* \leftarrow \mathcal{L}^* \cup \{(\mathsf{c}, C_\mathsf{e}, \mathsf{e}; \mathsf{inf})\}^{\Rightarrow}$
18. **return** C_e

$\mathcal{O}.\mathsf{Corr}(\mathsf{inp}, \hat{\mathsf{e}})$

1. **do** $\mathsf{Check}(\mathsf{inp}, \hat{\mathsf{e}}; \mathsf{e}; \hat{\mathsf{fwl}}, \hat{\mathsf{fwr}})$
2. **if** $\mathsf{inp} = \mathsf{key}$
3. $\quad \mathcal{K} \leftarrow \mathcal{K} \cup \{\hat{\mathsf{e}}\}$
4. $\quad$ **return** $k_{\hat{\mathsf{e}}}$
5. **if** $\mathsf{inp} = \mathsf{token}$
6. $\quad \mathcal{T} \leftarrow \mathcal{T} \cup \{\hat{\mathsf{e}}\}$
7. $\quad$ **for** $i \in \mathcal{T}^*$ **do**
8. $\quad\quad$ **for** $(j, C_{i-1}, i - 1; \mathsf{inf}) \in \mathcal{L}^*$ **do**
9. $\quad\quad\quad C_i \leftarrow \mathcal{O}.\mathsf{Upd}(C_{i-1})$
10. $\quad\quad\quad \mathcal{L}^* \leftarrow \mathcal{L}^* \cup \{(j, C_i, i; \mathsf{inf})\}$
11. $\quad$ **return** $\Delta_{\hat{\mathsf{e}}}$

$\mathcal{O}.\mathsf{Try}(\tilde{C})$

1. **if** $\mathsf{phase} = 1$
2. $\quad$ **return** $\perp$
3. $\mathsf{phase} \leftarrow 1$
4. **if** $\tilde{\mathsf{e}} \in \mathcal{K}^*$ **or** $\tilde{C} \in \mathcal{L}^*$
5. $\quad \mathsf{twf} \leftarrow 1$
6. **if** $\tilde{\mathsf{e}} \notin \{\hat{\mathsf{fwl}}, \ldots, \hat{\mathsf{fwr}}\}$
7. $\quad \mathsf{twf} \leftarrow 1$
8. $y \leftarrow (\prod_{\mathsf{e}=\tilde{\mathsf{e}}}^{\hat{\mathsf{fwl}}+1} \Delta_\mathsf{e}^{-1}) \star \tilde{C}$ //$\tilde{\mathsf{e}} \geq \hat{\mathsf{fwl}}$
9. **output** y **to** $\mathbf{Exp}^{\mathsf{wk\text{-}UP}}_{\mathcal{GA},\mathcal{B}}$; **get** b
10. $\mathsf{win} \leftarrow b$

Figure 2.7: Our reduction $\mathcal{B}$ for proof of Lemma 2.6. ors refers to the set $\{\mathcal{O}.\mathsf{Enc}, \mathcal{O}.\mathsf{Next}, \mathcal{O}.\mathsf{Upd}, \mathcal{O}.\mathsf{Corr}, \mathcal{O}.\mathsf{Try}\}$. $\bowtie$ indicates that Δ_0 and $\Delta_{\hat{\mathsf{fwr}}+1}$ are skipped in the computation. $\Rightarrow$ indicates that inf is empty when $\mathsf{c} = \mathsf{h}$. Comments start with //.

2.2.7 Dealing with bad ciphertext expansion

The problematic ciphertext expansion of SHINE0 is reduced to almost nothing in the construction of OCBSHINE of [BDGJ20, sec. 5.1.3] which is inspired by the authenticated encryption scheme OCB [RBBK01]. Once again, we can adapt this construction and proof to GAINE0 to get the OCBGAINE variant. See [BDGJ20, Figure 27 & 28] for more details.

2.3 Updatable Encryption from triple orbital group actions

We acknowledge that the weak pseudorandomness of the two MEGAs used to instantiate GAINE relies on two non-standard and recent assumptions. A well established source of post-quantum weak pseudorandom group actions comes from isogeny-based cryptography. However, we cannot directly instantiate GAINE with isogenies because it is notoriously hard [BBD+22] to hash into the set of supersingular elliptic curves, which is a necessary requirement to get a MEGA (see Prop. 2.1). In order to build post-quantum UE from well established assumptions, we present a new abstract algebraic structure that we call Triple Orbital Group Action (TOGA). This construction circumvents the need for a MEGA when designing UE from group actions.

Let us start with a quick overview. A TOGA is made of three group actions, each with a distinct role. The main group action, that we write $(A, S, \star_A)$, is our starting point. The main ingredient to get a TOGA from the simple group action $\star_A$ is a congruence relation $\sim_A$. This relation allows us to derive a second group action $(A/\sim_A, S/\sim_S, \star_G)$, called the induced group action, of the quotient group $G := A/\sim_A$ on the quotient set $T := S/\sim_S$ (see Definition 2.9 for $\sim_S$). Of course, this induced group action is not mappable as we would not need a TOGA to build UE in this case. This time, we consider messages as group elements of a third group action $(H, S, \star_H)$. For decryption to be possible, we assume that this action is efficiently invertible. We want $\star_H$ to commute with $\star_A$ but also that the orbits of $\star_H$ are exactly the classes of equivalences of $S/\sim_S$, which is what we call to be *orbital*. For a visualization of the interaction between the three group actions of a TOGA, see Fig. 2.8.

2.3.1 The algebraic structure

Let us assume that we have a group action $(A, S, \star_A)$ for an abelian multiplicative group A and a set S. We write 1_A for the identity element of A. If there exists a congruence relation $\sim_A$ on A (we recall that a *congruence* on a set with an intern law is an equivalence relation compatible with the law, *i.e.*, such that if $a_1 \sim_A a_2$ and $b_1 \sim_A b_2$ we have $a_1 b_1 \sim_A a_2 b_2$), then we get that $G := A/\sim_A$ is an abelian group for the law naturally derived from the multiplication in A.

Definition 2.9. *Let A be an abelian group and let $\sim_A$ be a congruence relation on A. Let S be a set and let $\star_A$ be a group action of A on S. The relation $\sim_S$ induced by $\sim_A$ and $\star_A$ is*

$$s_1 \sim_S s_2 \iff \exists a_1, a_2 \in A \text{ with } a_1 \sim_A a_2 \text{ such that } a_1 \star_A s_1 = a_2 \star_A s_2$$

Proposition 2.7. *Keeping the notations of Definition 2.9, we have that $\sim_S$ is an equivalence relation and $\star_A$ induces a group action $\star_G$ of $G := A/\sim_A$ on $T := S/\sim_S$.*

Proof. The relation $\sim_S$ is clearly reflexive and symmetric. For transitivity let us take $s_1, s_2, s_3 \in S$ with $s_1 \sim_S s_2$ and $s_2 \sim_S s_3$, we have $a_1 \star_A s_1 = a_2 \star_A s_2$ and $b_2 \star_A s_2 = b_3 \star_A s_3$, thus $a_1 b_2 \star_A s_1 = a_2 b_3 \star_A s_3$ and $a_1 b_2 \sim_A a_2 b_3$ since $\sim_A$ is a congruence. Let us write $G = A/\sim_A$ and $T = S/\sim_S$. First, we need to verify that the operation $\star_A$ is well-defined on the quotients. To see that, we need to verify that $a_1 \star_A s_1 \sim_S a_2 \star_A s_2$ when $a_1 \sim_A a_2$ and $s_1 \sim_S s_2$. This is true because we have $b_1 \sim_A b_2$ such that $b_1 \star_A s_1 = b_2 \star_A s_2$, and so $(a_2 b_1) \star_A (a_1 \star_A s_1) = (a_1 b_2) \star_A (a_2 \star_A s_2)$

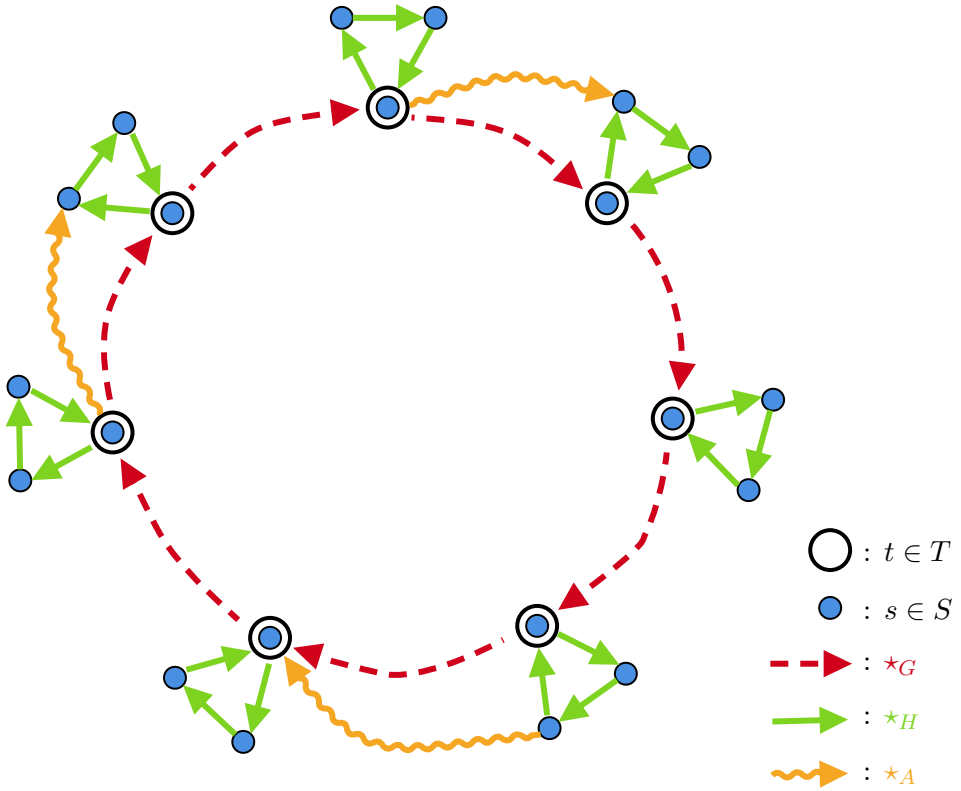

Figure 2.8: Diagram for a TOGA $A, H, S, \star_A, \sim_A, \star_H$.

with $a_2 b_1 \sim_A a_1 b_2$ because $\sim_A$ is a congruence. Then, we need to show that $\star_A : G \times T \to T$ verifies the usual group action properties from Definition 2.1. First, let us take $a \sim_A 1_A$. We must have $a \star_A s \sim_S s$ for any $s \in S$, which is clearly the case. Then, for any $a_1, a_2 \in A$, $s \in S$, we have the equality $(a_1 a_2) \star_A s = a_1 \star_A (a_2 \star_A s)$ and this equality remains true when considering the quotients G, T. □

Definition 2.10. *Given $A, S, \star_A, \sim_A$ as in Prop. 2.7, the group action $\star_G$ of $A/\sim_A$ on $S/\sim_S$ is called the group action induced by $A, \star_A, \sim_A$ (or induced by A when it is clear from the context) and $(A, S, \star_A)$ is called the main group action.*

We obtain a third group action (hence the name of triple group action) by looking at the classes of equivalence of S. We want to consider these classes as the orbits of a third group action $\star_H : H \times S \to S$ for another abelian group H. By that we mean that, for any $s \in S$ and $h \in H$, we have $s \sim_S h \star_H s$ and that, for all $s' \sim_S s$, there exists $h \in H$ with $s' = h \star_H s$. Additionally, we need the group action $(H, S, \star_H)$ to be *free* because we will need to invert $\star_H$. When these constraints are respected we qualify the group action $(H, S, \star_H)$ to be *orbital*.

Finally, we want that $\star_A$ and $\star_H$ commute and that for any $a_1, a_2 \in A$ such that $a_1 a_2 \sim_A 1_A$, there exists a unique element $h(a_1, a_2) \in H$ such that $(a_1 a_2) \star_A s = h(a_1, a_2) \star_H s$ for any $s \in S$. With Prop. 2.8, we give a useful reformulation that will prove useful for the correctness of our scheme.

Proposition 2.8. *For any $a, b \in A$ with $a \sim_A b$, we have $a \star_A s = (h(a, c) h(b, c)^{-1}) \star_H (b \star_A s)$ for any $c \in A$ with $ac \sim_A 1_A$ and $s \in S$.*

Proof. We have $h(a, c) \star_H (b \star_A s) = (ac) \star_A (b \star_A s) = (abc) \star_A s = (bc) \star_A (a \star_A s) = h(b, c) \star_H (a \star_A s)$. □

Definition 2.11 (TOGA)**.** *When $A, H, S, \star_A, \sim_A, \star_H$ satisfy all the above properties we say that we have a Triple Orbital Group Action (TOGA).*

A visualisation of a TOGA is given in Fig. 2.8. We give an example of a (pre-quantum) TOGA.

Example 2 (Pre-quantum TOGA)**.** Let $S := U \times V$, where $U := \langle u \rangle$ is a cyclic (multiplicative) group of prime order q and $V := \langle v \rangle$ is a cyclic (multiplicative) group of order 2^n for some integer n. Take $A := (\mathbb{Z}/q\mathbb{Z} \times \mathbb{Z}/2^n\mathbb{Z}, +)$. A acts on S through

$$\forall (a,b) \in A, \ \forall (x,y) \in S, \quad (a,b) \star_A (x,y) := (xu^a, yv^b)$$

one can easily verify that $(A, S, \star_A)$ is a group action. We define the following relation $\sim_A$ on A:

$$\forall (a_1,b_1),(a_2,b_2) \in A, \quad (a_1,b_1) \sim_A (a_2,b_2) \Leftrightarrow a_1 = a_2$$

one can easily verify that $\sim_A$ is a congruence relation on A. We have $G := A/\sim_A \simeq (\mathbb{Z}/q\mathbb{Z}, +)$. We recall the equivalence relation $\sim_S$ used in TOGA:

$$\forall s_1, s_2 \in S, \quad s_1 \sim_S s_2 \Leftrightarrow \exists c_1, c_2 \in A \text{ s.t. } c_1 \sim_A c_2 \text{ and } c_1 \star_A s_1 = c_2 \star_A s_2$$

Thus, for all $s_1, s_2 \in S$ such that $s_1 := (x_1, y_1)$ and $s_2 := (x_2, y_2)$, we have

$$\begin{aligned} s_1 \sim_S s_2 &\Leftrightarrow \exists a \in \mathbb{Z}/q\mathbb{Z}, b_1, b_2 \in \mathbb{Z}/2^n\mathbb{Z} \text{ s.t. } (a,b_1) \star_A s_1 = (a,b_2) \star_A s_2 \\ &\Rightarrow (x_1 u^a, y_1 v^{b_1}) = (x_2 u^a, y_2 v^{b_2}) \\ &\Rightarrow x_1 = x_2 \end{aligned}$$

Thus $T := S/\sim_S \simeq U$. Now take $H := (\mathbb{Z}/2^n\mathbb{Z}, +)$ and define $h \star_H (x,y) := (x, yv^h)$ for all $h \in H$ and $(x,y) \in S$. Clearly $\star_A$ and $\star_H$ commute, $\star_H$ is free and it is efficiently invertible using the Pohlig-Hellman algorithm for computing discrete logarithms. Moreover, it is also clear that each equivalence class of S is an orbit of $\star_H$. There remains one condition to check. Take $a_1, a_2 \in A$ such that $a_1 + a_2 \sim_A 1_A$, *i.e.* there exists $a \in \mathbb{Z}/q\mathbb{Z}$ and $b_1, b_2 \in \mathbb{Z}/2^n\mathbb{Z}$ such that $a_1 = (a, b_1)$, $a_2 = (-a, b_2)$ and $-a_2 = (a, -b_2)$. Then,

$$\forall (x,y) \in S, \quad (a_1 + a_2) \star_A (x,y) = (x, yv^{b_1+b_2}) = (b_1 + b_2) \star_H (x,y)$$

Moreover, $b_1 + b_2$ is the unique element of H satisfying the above equality. To conclude, we showed that $A, H, S, \star_A, \sim_A, \star_H$ satisfy Definition 2.11 and is thus a TOGA. Concretely, we can instantiate this TOGA by finding a prime $p = 2^n \cdot q + 1$, where q is a large prime, taking $S := \mathbb{F}_p^*$, $u := g^{2^n}$ and $v := g^q$ where g is a generator of S.

Remark 3. Note that A being a group is not really necessary for the UE scheme that we will introduce below. In fact, we only need that A is a monoid and that the quotient $A/\sim_A$ is a group. We only assumed that A is a group for simplicity.

2.3.2 Computational model

As for group actions, we define an ETOGA as an Effective TOGA:

Definition 2.12 (ETOGA)**.** *A TOGA $A, H, S, \star_A, \sim_A, \star_H$ is effective if:*

1. *The group action $(H, S, \star_H)$ is an Effective and Easy Group Action (EEGA):*
   (a) *The group action $(H, S, \star_H)$ is a free EGA.*

(b) *There is a PPT inversion algorithm* $\mathsf{Invert}_H : S^2 \to \{\bot\} \cup H$ *taking two elements* s_1, s_2 *and that outputs either* $\bot$ *when* $s_1 \not\sim_S s_2$ *or the element* $h \in H$ *such that* $s_1 = h \star_H s_2$.

2. *There exists a finite subset* $A' \subset A$ *such that:*
   (a) *The class of equivalence of* A' *form a generating set of* G, *i.e.,* $G := A'/\sim_A$.
   (b) *There is a PPT algorithm to compute* $a' \star_A s$ *for any* $s \in S$ *and* $a' \in A'$.
   (c) *There exists a PPT algorithm* $\mathsf{Reduce}_A : A \to A'$ *that takes an element* $a \in A$ *and outputs* $a' \sim_A a$.
   (d) *There exists a PPT algorithm to sample from* A' *in a distribution statistically close to the uniform distribution, we write* $a' \xleftarrow{\$} A'$ *for elements sampled in that manner.*
   (e) *The distribution* $\mathcal{D}_G$ *that samples* $a' \xleftarrow{\$} A'$ *and returns the class of* $\mathsf{Reduce}_A(a')$ *in* G *is statistically close to the uniform distribution.*
3. *There exists a deterministic PPT algorithm* Reduce_S *to compute a canonical representative for equivalence classes in* $S/\sim_S$.

Remark 4. Note that the Reduce_A algorithm may or may not be deterministic. For efficiency, it is interesting to try to select the element a' in the class of a that minimizes the computation cost of $a' \star_A s$ for any $s \in S$.

Note that when $a_1 a_2 \sim 1_A$, we have $h(a_1, a_2) = \mathsf{Invert}_H((a_1 a_2) \star_A s, s)$ for any $s \in S$. Thus, we can define a PPT algorithm to compute $h(a_1, a_2)$ from Invert_H. We abuse notations and write h for this algorithm.

Since the function Reduce_S is deterministic, we can abuse notations and assimilate $T := S/\sim_S$ and $\mathsf{Reduce}_S(S)$ by identifying the elements of T to their canonical representative in S through Reduce_S. Using this, we sometimes apply the action $\star_A$ on the elements of T (it suffices to compose $\star_A$ with Reduce_S to obtain the canonical representative afterwards).

2.3.3 The Updatable Encryption scheme

Let $\mathcal{TOGA}$ be an ETOGA family and let $(A, H, S, \star_A, \sim_A, \star_H)$ be $\mathcal{TOGA}(1^\lambda)$ for some λ. We fix a starting element $s_0 \in S$, and we also assume the existence of an invertible map $\psi : \mathcal{M} \to H$ where $\mathcal{M}$ is the space of the messages. We will use the function ψ to send the messages in the group H before encrypting them with $\star_H$. Then, decryption will rely on Invert_H. This operation is efficient by definition of an ETOGA. This principle basically solves the problem of needing our group action $\star_A$ to be mappable. The rest of our scheme follows the framework of GAINE with keys being elements of $A \times H$ and updates being obtained by applying $\star_A$ and $\star_H$. The security relies on the fact that the induced group action $(G, T, \star_G)$ is weakly pseudorandom. Our UE scheme TOGA-UE is given in Fig. 2.9.

Proposition 2.9 (Correctness of updates)**.** *Let* $k_\mathsf{e}, k_{\mathsf{e}+1} := (a_\mathsf{e}, h_\mathsf{e}), (a_{\mathsf{e}+1}, h_{\mathsf{e}+1})$ *be two keys and* $C_\mathsf{e} := h_\mathsf{e} \star_H (a_\mathsf{e} \star_A s)$ *for some* $s \in S$. *If* $\Delta_{\mathsf{e}+1} := \mathsf{TokenGen}(k_\mathsf{e}, k_{\mathsf{e}+1})$, *then* $\mathsf{Upd}(\Delta_{\mathsf{e}+1}, C_\mathsf{e}) = h_{\mathsf{e}+1} \star_H (a_{\mathsf{e}+1} \star_A s)$.

Proof. We reuse the notation of TokenGen, we have for $c_\mathsf{e} = \mathsf{Reduce}_A(a_\mathsf{e}^{-1} a_{\mathsf{e}+1})$. Since $c_\mathsf{e} \sim_A a_{\mathsf{e}+1} a_\mathsf{e}^{-1}$, we have that $c_\mathsf{e} \star_A (a_\mathsf{e} \star_A s') = (h(c_\mathsf{e}, c_\mathsf{e}^{-1}) h(a_{\mathsf{e}+1} a_\mathsf{e}^{-1}, c_\mathsf{e}^{-1})^{-1}) \star_H ((a_{\mathsf{e}+1} a_\mathsf{e}^{-1} a_\mathsf{e}) \star_A s')$ by Prop. 2.8. The proof is completed by the fact that $(a_{\mathsf{e}+1} a_\mathsf{e}^{-1} a_\mathsf{e}) \star_A s' = a_{\mathsf{e}+1} \star_A s'$ and $h(c_\mathsf{e}, c_\mathsf{e}^{-1}) = 1_H$. □

Proposition 2.10 (Correctness)**.** *The* TOGA-UE *scheme is correct.*

$\mathsf{Setup}(1^\lambda)$:

1. $(A, H, S, \star_A, \sim_A, \star_H) \leftarrow \mathcal{TOGA}(\lambda)$
2. Choose ψ, s_0 as above
3. $\mathsf{pp} \leftarrow (A, H, S, \star_A, \sim_A, \star_H, \psi, s_0)$
4. **return** pp

$\mathsf{KeyGen}(\mathsf{pp})$:

1. $a' \xleftarrow{\$} A'$
2. $h \xleftarrow{\$} H$
3. **return** $\mathsf{Reduce}_A(a'), h$

$\mathsf{TokenGen}(k_\mathsf{e}, k_{\mathsf{e}+1})$:

1. $(a_\mathsf{e}, h_\mathsf{e}) \leftarrow k_\mathsf{e}$
2. $(a_{\mathsf{e}+1}, h_{\mathsf{e}+1}) \leftarrow k_{\mathsf{e}+1}$
3. $c_\mathsf{e} \leftarrow \mathsf{Reduce}_A(a_\mathsf{e}^{-1} a_{\mathsf{e}+1})$
4. Compute $h = h(a_\mathsf{e}^{-1} a_{\mathsf{e}+1}, c_\mathsf{e}^{-1})$
5. **return** $c_\mathsf{e}, h h_{\mathsf{e}+1} h_\mathsf{e}^{-1}$

$\mathsf{Upd}(\Delta_{\mathsf{e}+1}, C_\mathsf{e})$:

1. $a, h \leftarrow \Delta_{\mathsf{e}+1}$
2. **return** $h \star_H (a \star_A C_\mathsf{e})$

$\mathsf{Enc}(k_\mathsf{e}, M)$:

1. $r' \xleftarrow{\$} A'$
2. $r \leftarrow \mathsf{Reduce}_A(r')$
3. $s = \mathsf{Reduce}_S(r \star_A s_0)$
4. $(a_\mathsf{e}, h_\mathsf{e}) \leftarrow k_\mathsf{e}$
5. **return** $(\psi(M) h_\mathsf{e}) \star_H (a_\mathsf{e} \star_A s)$

$\mathsf{Dec}(k_\mathsf{e}, C_\mathsf{e})$:

1. $(a_\mathsf{e}, h_\mathsf{e}) \leftarrow k_\mathsf{e}$
2. $b_\mathsf{e} \leftarrow \mathsf{Reduce}_A(a_\mathsf{e}^{-1})$
3. $h' \leftarrow h(a_\mathsf{e}, b_\mathsf{e})$
4. $s' \leftarrow (h_\mathsf{e} h')^{-1} \star_H (b_\mathsf{e} \star_A C_\mathsf{e})$
5. $s \leftarrow \mathsf{Reduce}_S(s')$
6. $M' \leftarrow \psi^{-1}(\mathsf{Invert}_H(s', s))$
7. **return** M'

Figure 2.9: TOGA-UE: UE from ETOGA.

Proof. Let $\mathsf{e}_1 \leq \mathsf{e}_2 \leq n+1$ be two epochs and let us consider a ciphertext c_{e_2} updated through the successive tokens Δ_{i+1} for $i \in [\mathsf{e}_1, \mathsf{e}_2 - 1]$ from an initial ciphertext c_{e_1} that is the encryption of a message m under the key $k_{\mathsf{e}-1}$ as in Definition 1.5. Each key k_i can be decomposed as $a_i, h_i \in A \times H$. By definition of $c_{\mathsf{e}_1} = \mathsf{Enc}(k_{\mathsf{e}_1}, m)$, we have $c_{\mathsf{e}_1} = (h_{\mathsf{e}_1} \psi(m)) \star_H a_{\mathsf{e}_1} \star_A \mathsf{Reduce}_S(s_1)$ for some $s_1 \in S$. By applying Prop. 2.9 on $s = \psi(m) \star_H \mathsf{Reduce}_S(s_1)$, we have that $c_{\mathsf{e}_2} = h_{\mathsf{e}_2} \psi(m) \star_A a_{\mathsf{e}_2} \star_A \mathsf{Reduce}_S(s_1)$ since $\star_A$ and $\star_H$ commute. Then, let us take any $b_{\mathsf{e}_2} \in A'$ such that $a_{\mathsf{e}_2} b_{\mathsf{e}_2} \sim_A 1_A$. By definition of h we know that $(b_{\mathsf{e}_2} a_{\mathsf{e}_2}) \star_A x = h(b_{\mathsf{e}_2}, a_{\mathsf{e}_2}) \star_H x$ for any $x \in S$. Thus, $s' = (h_{\mathsf{e}_2} h(b_{\mathsf{e}_2}, a_{\mathsf{e}_2}))^{-1} \star_H b_{\mathsf{e}_2} \star_A c_{\mathsf{e}_2} = \psi(M) \star_H \mathsf{Reduce}_S(s_1)$. Then, since the orbits of $\star_H$ are exactly the equivalence classes of S, we have $s' \sim_S s_1$ and so $\mathsf{Reduce}_S(s') = \mathsf{Reduce}_S(s_1)$. Thus, when we compute $\mathsf{Invert}_H(s', \mathsf{Reduce}_S(s'))$ we obtain $\psi(m)$ and the message is recovered by applying ψ^{-1}. □

2.3.4 Security - TOGA-UE is detIND-UE-CPA secure

Let $A, H, S, \star_A, \sim_A, \star_H$ be an ETOGA, fix $s_0 \in S$ and let $(G, T, \star_G)$ be the group action induced by A (as in Definition 2.10) where $G := A/\sim_A$ and $T := S/\sim_S$. In Theorem 2.11, we show that our UE scheme TOGA-UE (described in Fig. 2.9) is detIND-UE-CPA secure if the group action $(G, T, \star_G)$ is weakly pseudorandom. We sample uniformly in T by sampling $g \xleftarrow{\$} G$ and returning the equivalence class of $g \star s_0$ in T.

Theorem 2.11 (TOGA-UE is detIND-UE-CPA)**.** *Let* TOGA-UE *be the UE scheme described in Fig. 2.9 for an ETOGA family $\mathcal{TOGA}$. We define a group action family $\mathcal{GA}$, where $\mathcal{GA}(1^\lambda)$ is $(G, T, \star_G)$, the group action induced by $A \in \mathcal{TOGA}(1^\lambda)$ (as in Definition 2.10). For any*

adversary $\mathcal{A}$, there exists a reduction $\mathcal{B}$ such that

$$\mathbf{Adv}^{\mathsf{detIND\text{-}UE\text{-}CPA}}_{\mathsf{TOGA\text{-}UE},\mathcal{A}}(\lambda) \leq \mathcal{O}(1)(n+1)^3 \cdot \mathbf{Adv}^{\mathsf{wk\text{-}PR}}_{\mathcal{GA},\mathcal{B}}(\lambda)$$

Proof. The proof uses the same hybrid argument as the one of Theorem 2.4, thus we only point out the differences between both proofs. Contrary to the proof of Theorem 2.4, we do not need to use the ideal cipher model. Indeed, in TOGA-UE, randomization of ciphertexts is not done through the permutation ψ. Thus, we do not need to "program" ψ to get consistent randomness throughout our reduction.

Our reduction $\mathcal{B}$, given in Fig. 2.10, starts by receiving a group action $(G, T, \star_G)$ and an oracle $\mathcal{O}.\mathsf{Sample}$ that returns either tuples of the form $(t_i, g \star_G t_i)$ or (t_i, u_i) where $g \xleftarrow{\$} G$ and $t_i, u_i \xleftarrow{\$} T$. We use the same hybrid argument over insulated regions as in Theorem 2.4. $\mathcal{B}$ will use the tuples of $\mathcal{O}.\mathsf{Sample}$ to perfectly simulate the detIND-UE-CPA experiment for TOGA-UE when those tuples are of the form $(t_i, g \star_G t_i)$. Thus, if we know an efficient adversary $\mathcal{A}$ against the detIND-UE-CPA security of TOGA-UE, using the hybrid argument of Theorem 2.4, $\mathcal{B}$ can use $\mathcal{A}$ to break the weak pseudorandomness of $(G, T, \star_G)$.

Our reduction $\mathcal{B}$ uses the following notations. Given a ciphertext C_{e} and a token Δ_{e}, we can downgrade C_{e} to epoch $\mathsf{e}-1$ like so:

1. $(c, h) \leftarrow \Delta_{\mathsf{e}}$
2. $b \leftarrow \mathsf{Reduce}_A(c^{-1})$
3. $h' \leftarrow h(b, c)$
4. $C_{\mathsf{e}-1} \leftarrow (hh')^{-1} \star_H (b \star_A C_{\mathsf{e}})$
5. **return** $C_{\mathsf{e}-1}$

For readability, we will use the (abuse of) notation $\Delta_{\mathsf{e}}^{-1} \star C_{\mathsf{e}}$ to denote this downgrade. Similarly, if $\Delta_{\mathsf{e}} := (c, h)$, we will use the notation $\Delta_{\mathsf{e}} \star C_{\mathsf{e}-1}$ to denote the update $h \star_H (c \star_A C_{\mathsf{e}-1})$.

In TOGA-UE, a ciphertext is of the form $C_{\mathsf{e}} := h_{\mathsf{e}} \star_H (a_{\mathsf{e}} \star_G r)$ with $k_{\mathsf{e}} := (a_{\mathsf{e}}, h_{\mathsf{e}})$, where $a_{\mathsf{e}} \xleftarrow{\$} G$, $h_{\mathsf{e}} \xleftarrow{\$} H$ and $r \xleftarrow{\$} T$ is the randomness used during the first encryption. Reduction $\mathcal{B}$ will try to embed the $\mathcal{O}.\mathsf{Sample}$ tuples in the i-th insulated region $[\mathsf{fwl}_i, \mathsf{fwr}_i]$. If $(r, s) \leftarrow \mathcal{O}.\mathsf{Sample}()$, $\mathcal{B}$ uses r as randomness for new ciphertexts. When updating ciphertext $C_{\mathsf{fwl}_i - 1} := h_{\mathsf{fwl}_i - 1} \star_H (a_{\mathsf{fwl}_i - 1} \star_G r)$ to epoch fwl_i, $\mathcal{B}$ sets $C_{\mathsf{fwl}_i} := h_{\mathsf{fwl}_i} \star_H s$ where h_{fwl_i} is simulated by $\mathcal{B}$. If (r, s) is of the form $(r, g \star_G r)$, $\mathcal{B}$ has embedded g into $k_{\mathsf{fwl}_i} := (g, h_{\mathsf{fwl}_i})$ and the randomness of the ciphertext stays consistent because of Prop. 2.9. Else, if (r, s) is a tuple of random elements of T, the ciphertexts inside the i-th insulated region are all random (there is no consistent key or randomness linking them).

Recall that a token $\Delta_{\mathsf{e}+1} := (c_{\mathsf{e}}, h h_{\mathsf{e}+1} h_{\mathsf{e}}^{-1})$ where $h_{\mathsf{e}}, h_{\mathsf{e}+1}$ are part of the epoch keys k_{e} and $k_{\mathsf{e}+1}$ and c_{e}, h are computed by Upd using those keys. When both keys are unknown (like in the i-th insulated region), c_{e} is uniformly distributed in G by Definition 2.12 item 2e. Recall that $h \in H$ is useful for the correction of updates (see Prop. 2.9) and that it is not independent of c_{e}. However, h_{e} and $h_{\mathsf{e}+1}$ are sampled uniformly in H and are not used in the computations of c_{e} and h. Since h_{e} and $h_{\mathsf{e}+1}$ are unknown to the adversary in the i-th insulated region, $h h_{\mathsf{e}+1} h_{\mathsf{e}}^{-1}$ is uniformly distributed in H and reduction $\mathcal{B}$ can perfectly simulate tokens inside the i-th insulated region.

Because of the correctness of updates in TOGA-UE (see Prop. 2.9) and of the observations above, when $\mathcal{O}.\mathsf{Sample}$ returns tuples of the form $(t_i, g \star_G t_i)$, the reduction $\mathcal{B}$ perfectly simulates the environment of the adversary $\mathcal{A}$ and we get a similar result as the one of Theorem 2.4. □

Reduction $\mathcal{B}$ playing $\mathbf{Exp}^{\text{wk-PR-}b^*}_{\mathcal{GA},\mathcal{B}}(\lambda)$

1. **receive** $(G, T, \star_G)$ and $\mathcal{O}.\mathsf{Sample}$
2. **do Setup**(1^λ)
3. $\bar{M}, \bar{C} \leftarrow \mathcal{A}^{\mathtt{ors}}(\lambda)$
4. $\mathsf{phase} \leftarrow 1$
5. $\tilde{C}_{\tilde{\mathsf{e}}} \leftarrow \mathcal{O}.\mathsf{Chall}(\bar{M}, \bar{C})$
6. $b' \leftarrow \mathcal{A}^{\mathtt{ors},\mathcal{O}.\mathsf{Upd}\tilde{\mathsf{C}}}(\tilde{C}_{\tilde{\mathsf{e}}})$
7. $\mathsf{twf} \leftarrow 1$ **if**
8. $\quad \mathcal{C}^* \cap \mathcal{K}^* \neq \emptyset$ **or**
9. $\quad \mathcal{I}^* \cap \mathcal{C}^* \neq \emptyset$
10. **if** ABORT occurred **or** $\mathsf{twf} = 1$
11. $\quad b' \xleftarrow{\$} \{0,1\}$
12. $\quad$ **return** b'
13. **if** $(i, \mathsf{fwl}_i, \mathsf{fwr}_i) \notin \mathcal{FW}$
14. $\quad b' \xleftarrow{\$} \{0,1\}$
15. $\quad$ **return** b'
16. **if** $b' = b$
17. $\quad$ **return** 0
18. **else**
19. $\quad$ **return** 1

Setup(1^λ)

1. $b \xleftarrow{\$} \{0,1\}$
2. $\mathsf{pp} \leftarrow \mathsf{TOGA\text{-}UE.Setup}(1^\lambda)$
3. $k_0 \leftarrow \mathsf{TOGA\text{-}UE.KeyGen}(\mathsf{pp})$
4. $\Delta_0 \leftarrow \perp$
5. $\mathsf{e}, \mathsf{c}, \mathsf{phase}, \mathsf{twf} \leftarrow 0$
6. $\mathcal{L}, \tilde{\mathcal{L}}, \mathcal{C}, \mathcal{K}, \mathcal{T} \leftarrow \emptyset$
7. $\mathsf{fwl}_i, \mathsf{fwr}_i \xleftarrow{\$} \{0, \ldots, n\}$
8. **for** $j \in \{0, \ldots, \mathsf{fwl}_i - 1\} \cup \{\mathsf{fwr}_i + 1, \ldots, n\}$ **do**
9. $\quad a_j \xleftarrow{\$} G, h_j \xleftarrow{\$} H; k_j \leftarrow (a_j, h_j)$
10. $\quad \Delta_j \leftarrow \mathsf{TOGA\text{-}UE.TokenGen}(k_j, k_{j+1})^{\bowtie}$
11. **for** $j \in \{\mathsf{fwl}_i + 1, \ldots, \mathsf{fwr}_i\}$ **do**
12. $\quad c_j \xleftarrow{\$} G, h_j \xleftarrow{\$} H; \Delta_j \leftarrow (c_j, h_j)$
13. $h_{\mathsf{fwl}_i} \xleftarrow{\$} H$

$\mathcal{O}.\mathsf{Enc}(M)$

1. $\mathsf{c} \leftarrow \mathsf{c} + 1$
2. $(\mathsf{inf}_1, \mathsf{inf}_2) \leftarrow \mathcal{O}.\mathsf{Sample}()$
3. **if** $\mathsf{e} \in \{0, \mathsf{fwl}_i - 1\} \cup \{\mathsf{fwr}_i + 1, \ldots, n\}$
4. $\quad (a_\mathsf{e}, h_\mathsf{e}) \leftarrow k_\mathsf{e}$
5. $\quad C_\mathsf{e} \leftarrow (\psi(M)h_\mathsf{e}) \star_H (a_\mathsf{e} \star_G \mathsf{inf}_1)$
6. **else**
7. $\quad C_{\mathsf{fwl}_i} \leftarrow (\psi(M)h_{\mathsf{fwl}_i}) \star_H \mathsf{inf}_2$
8. $\quad$ **for** $j \in \{\mathsf{fwl}_i + 1, \ldots, \mathsf{e}\}$ **do**
9. $\quad\quad C_j \leftarrow \Delta_j \star C_{j-1}$
10. $\mathsf{inf} \leftarrow (\mathsf{inf}_1, \mathsf{inf}_2, M)$
11. $\mathcal{L} \leftarrow \mathcal{L} \cup \{(\mathsf{c}, C_\mathsf{e}, \mathsf{e}; \mathsf{inf})\}$
12. **return** C_e

$\mathcal{O}.\mathsf{Next}$

1. $\mathsf{e} \leftarrow \mathsf{e} + 1$

$\mathcal{O}.\mathsf{Upd}(C_{\mathsf{e}-1})$

1. **if** $(\mathsf{c}, C_{\mathsf{e}-1}, \mathsf{e}-1; \mathsf{inf}) \notin \mathcal{L}$
2. $\quad$ **return** $\perp$
3. **if** $\mathsf{e} \in \{1, \ldots, \mathsf{fwl}_i - 1\} \cup \{\mathsf{fwr}_i + 1, \ldots, n\}$
4. $\quad (\mathsf{inf}_1, \mathsf{inf}_2, M) \leftarrow \mathsf{inf}$
5. $\quad (a_\mathsf{e}, h_\mathsf{e}) \leftarrow k_\mathsf{e}$
6. $\quad C_\mathsf{e} \leftarrow (\psi(M)h_\mathsf{e}) \star_H (a_\mathsf{e} \star_G \mathsf{inf}_1)$
7. **else**
8. $\quad (\mathsf{inf}_1, \mathsf{inf}_2, M) \leftarrow \mathsf{inf}$
9. $\quad C_{\mathsf{fwl}_i} \leftarrow (\psi(M)h_{\mathsf{fwl}_i}) \star_H \mathsf{inf}_2$
10. $\quad$ **for** $j \in \{\mathsf{fwl}_i + 1, \ldots, \mathsf{e}\}$ **do**
11. $\quad\quad C_j \leftarrow \Delta_j \star C_{j-1}$
12. $\mathcal{L} \leftarrow \mathcal{L} \cup \{(\mathsf{c}, C_\mathsf{e}, \mathsf{e}; \mathsf{inf})\}$
13. **return** C_e

$\mathcal{O}.\mathsf{Corr}(\mathsf{inp}, \hat{\mathsf{e}})$

1. **do** $\mathsf{Check}(\mathsf{inp}, \hat{\mathsf{e}}; \mathsf{e}; \mathsf{fwl}_i, \mathsf{fwr}_i)$
2. **if** $\mathsf{inp} = \mathsf{key}$
3. $\quad \mathcal{K} \leftarrow \mathcal{K} \cup \{\hat{\mathsf{e}}\}$
4. $\quad$ **return** $k_{\hat{\mathsf{e}}}$
5. **if** $\mathsf{inp} = \mathsf{token}$
6. $\quad \mathcal{T} \leftarrow \mathcal{T} \cup \{\hat{\mathsf{e}}\}$
7. $\quad$ **return** $\Delta_{\hat{\mathsf{e}}}$

$\mathcal{O}.\mathsf{Chall}(\bar{M}, \bar{C})$

1. **if** $(\mathsf{c}, \bar{C}, \tilde{\mathsf{e}} - 1; \mathsf{inf}) \notin \mathcal{L}$
2. $\quad$ **return** ABORT
3. **if** $b = 0$
4. $\quad (s, t) \leftarrow \mathcal{O}.\mathsf{Sample}()$
5. $\quad r \leftarrow s$
6. $\quad \tilde{C}_{\mathsf{fwl}_i} \leftarrow (\psi(\bar{M})h_{\mathsf{fwl}_i}) \star_H t$
7. **else**
8. $\quad (\mathsf{inf}_1, \mathsf{inf}_2, M) \leftarrow \mathsf{inf}$
9. $\quad r \xleftarrow{\$} T$
10. $\quad \tilde{C}_{\mathsf{fwl}_i} \leftarrow (\psi(M)h_{\mathsf{fwl}_i}) \star_H \mathsf{inf}_2$
11. **for** $j \in \{0, \ldots, \mathsf{fwl}_i - 1\}$ **do**
12. $\quad \tilde{C}_j \leftarrow (\prod_{k=j}^{1} \Delta_k)(\prod_{k=1}^{\tilde{\mathsf{e}}-1} \Delta_k^{-1}) \star \bar{C}$ //left
13. **for** $j \in \{\mathsf{fwl}_i + 1, \ldots, \mathsf{fwr}_i\}$ **do**
14. $\quad \tilde{C}_j \leftarrow \Delta_j \star \tilde{C}_{j-1}$ //embed
15. **for** $j \in \{\mathsf{fwr}_i + 1, \ldots, n\}$ **do**
16. $\quad (a_j, h_j) \leftarrow k_j$
17. $\quad \tilde{C}_j \leftarrow (\psi(\bar{M})h_j) \star_H (a_j \star_G r)$ //right
18. $\tilde{\mathcal{L}} \leftarrow \cup_{j=0}^{n} \{(\tilde{C}_j, j)$
19. **return** $\tilde{C}_\mathsf{e}$

$\mathcal{O}.\mathsf{Upd}\tilde{\mathsf{C}}$

1. $\mathcal{C} \leftarrow \mathcal{C} \cup \{\mathsf{e}\}$
2. **find** $(\tilde{C}_\mathsf{e}, \mathsf{e}) \in \tilde{\mathcal{L}}$
3. **return** $\tilde{C}_\mathsf{e}$

Figure 2.10: Our reduction $\mathcal{B}$ for proof of Theorem 2.11 in hybrid i. inf encodes fixed programming information: it marks two set elements $(\mathsf{inf}_1, \mathsf{inf}_2)$ sampled with $\mathcal{O}.\mathsf{Sample}$ and a plaintext M. inf_1 is the randomness used during encryption, inf_2 is used to compute the ciphertext value in epoch fwl_i and M is the plaintext. ors refers to the set $\{\mathcal{O}.\mathsf{Enc}, \mathcal{O}.\mathsf{Next}, \mathcal{O}.\mathsf{Upd}, \mathcal{O}.\mathsf{Corr}\}$. $\bowtie$ indicates that Δ_0 and $\Delta_{\mathsf{fwr}_i+1}$ are skipped in the computation. Comments start with //.

2.3.5 On the CCA security of TOGA-UE

Unlike GAINE, making TOGA-UE CCA secure appears to be hard. Indeed, our construction has a pretty clear malleability property: let M, M' be two distinct messages, under the ψ map we get two elements $h := \psi(M), h' := \psi(M')$. Then, for any encryption c of the message M, we compute $h'h^{-1} \star_H c$ to obtain a valid encryption of M'. We leave the problem of making TOGA-UE CCA secure open for future work.

2.3.6 Instantiating TOGA-UE

In our [LR22] paper, we show how to build a post-quantum ETOGA from the CSIDH (Commutative Supersingular Isogeny Diffie–Hellman) group action [CLM$^+$18]. Then, we show how to instantiate TOGA-UE using this ETOGA. For the readers unfamiliar with isogenies, we can give a pre-quantum instantiation of TOGA-UE using the ETOGA of Ex. 2.

Chapter 3

New and improved constructions for Proofs of Retrievability

We start this chapter by proposing an authentication protocol tailored for code-based applications in the outsourced storage setting. Then, we present a framework for the design of secure and efficient PoRs based on LCCs. We study a generalization of a PoR scheme of Lavauzelle and Levy-dit-Vehel [LLDV16] based on lifted codes in our framework and we give a new security analysis and new sets of parameters for their scheme. We find out that the security of [LLDV16] was overestimated and we recommend to use our scheme and our parameters in its place. Moreover, we show that our scheme produces less false positive audits than the [LLDV16] one. An audit is a false positive when it answers reject whereas the outsourced file is still retrievable in full. We also give another instantiation of our framework using graph codes.

Then, we use a different approach to design a new PoR scheme based on expander codes. By exploiting the local properties of these codes, a fast erasure decoder of [SS96, Zém01] and the excellent properties of point-line incidence graphs [HJ06, HJ11, BHPJ13] we are able to design a PoR scheme featuring a quasi-linear time extraction phase as well as better concrete parameters than our previous constructions.

Finally, we conclude this chapter by modeling LCCs in the CC model. In doing so, we show that understanding the behavior of the local decoder of such codes can be essential. In particular, we show how an adversary can target specific memory locations by carefully placing corruptions that make the local decoder more likely to fail on some locations rather than on others.

3.1 Preliminaries

3.1.1 Message authentication codes in Constructive Cryptography

Our protocols will use message authentication codes (MAC). Thus, we recall the notations along with a description of the security condition for MAC in the CC model, given in [BM18].

We consider MAC functions f with message space $\mathcal{M}$, tag space $\mathcal{T}$ and key space $\mathcal{K}$. The security condition for MAC function f states that no efficient adversary can win the following game $\mathbf{G}^{\mathsf{MAC}}$ better than with negligible probability. In CC, games are represented as resources. In our case, the game $\mathbf{G}^{\mathsf{MAC}}$ chooses a secret key $sk \xleftarrow{\$} \mathcal{K}$. Then, it acts as a signing oracle by receiving messages $m \in \mathcal{M}$ at its interface and responding with $f_{sk}(m)$. At any point, the adversary can make a forging attempt by providing a message m' and a tag t' to the game. The game is won if and only if $f_{sk}(m') = t'$ and m' has never been signed by the game before. The probability of adversary $\mathcal{A}$ to win the game is denoted by $\Gamma^{\mathcal{A}}(\mathbf{G}^{\mathsf{MAC}})$.

3.1.2 Lifted Reed-Solomon Codes

We introduce a very interesting class of LCCs, namely the high rate lifted Reed-Solomon (RS) codes of Guo *et al.* [GKS13]. In the following, let $\mathbb{F}_q$ be the finite field with q elements and m be a positive integer. The set of affine lines in $\mathbb{F}_q^m$ is denoted by $\mathcal{L}_m := \{(at+b)_{t\in\mathbb{F}_q} \mid a, b \in \mathbb{F}_q^m\}$. $\mathrm{RS}_q[q, d]$ is the q-ary RS code of length q, minimum distance d and dimension $k := q - d + 1$.

Definition 3.1 (Lifted Reed-Solomon Code [GKS13])**.** *Let $\mathbb{F}_q$ be a finite field. Let $d, m \in \mathbb{N}^*$. The m-lift of $\mathrm{RS}_q[q, d]$ is $\mathrm{Lift}_m(\mathrm{RS}_q[q, d]) := \{w \in (\mathbb{F}_q)^{q^m} \mid \forall$ line $\ell \subseteq \mathbb{F}_q^m, w_{|\ell} \in \mathrm{RS}_q[q, d]\}$.*

As we are using an **aSMR**, codewords can only be affected by potential erasures. A codeword of the RS base code $\mathrm{RS}_q[q, d]$ is the vector of evaluations of a polynomial f of degree strictly less than $k = q - d + 1$. Thus, if there are at most $d - 1$ erasures, we can always recover the codeword, *i.e.* the polynomial f, by interpolating on $k > \deg f$ points. Therefore, if we want to correct a coordinate $x \in \mathbb{F}_q^m$ of the $\mathrm{Lift}_m(\mathrm{RS}_q[q, d])$ code, we can pick a random line going through x and run the aforementioned local decoding algorithm.

3.1.3 Expander codes

We give an overview of the expander codes of Tanner [Tan81] and Sipser and Spielman [SS96]. We follow the presentation of [RZWZ20]. Let $G := (V, E)$ be an undirected d-regular graph on n vertices. The *expansion* of G is $\lambda := \max\{\lambda_2, |\lambda_n|\}$, where $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$ are the eigenvalues of the adjacency matrix of G. We say that G is a *Ramanujan* graph if $\lambda \leq 2\sqrt{d-1}$.

Remark 5. For a d-regular graph G on n vertices, we always have $\lambda_1 = d$. Indeed, every line of its adjacency matrix A possesses exactly d ones and $n-d$ zeros. Thus, the vector $(1, 1, \ldots, 1)$ is an eigenvector of A associated with eigenvalue d and $\lambda_1 \geq d$. Let $v = (v_1, \ldots, v_n)$ be an eigenvector of A associated with eigenvalue λ. Without loss of generality, we can suppose that v is such that $|v_i| \leq 1$, for all $i \in [1, n]$ and $|v_k| = 1$ for some $k \in [1, n]$. Let $A := (a_{i,j})_{1\leq i,j\leq n}$. We have that $|\lambda| = |\lambda v_k| = |\sum_{i=1}^n a_{k,i}v_i| \leq \sum_{i=1}^n |a_{k,i}||v_i| \leq \sum_{i=1}^n |a_{k,i}| = d$. Thus, $\lambda_1 \leq d$ and, finally, $\lambda_1 = d$.

The eigenvalue λ_1 is relatively uninteresting (see Remark 5) and graph theory focuses more on the gap between λ_1 and the second largest eigenvalue, a small gap meaning that the graph is well connected. A solid reference on expander graphs is the survey of Hoory *et al.* [HLW06].

For a vertex $v \in V$, let $\Gamma(v)$ be the set of vertices adjacent to v. Let $\mathcal{C}_0 \subseteq \mathbb{F}_q^d$ be a linear code, called the *inner code*. Fix an order on the edges incident to each vertex of G, and let $\Gamma_i(v)$ be the i-th neighbor of v.

Using the graph G and the inner code $\mathcal{C}_0$ we can construct a new code, called an *expander code*. The expander code $\mathcal{C} := \mathcal{C}(G, \mathcal{C}_0)$ is defined as the set of all labelings of the edges of G that respect the inner code $\mathcal{C}_0$. It has length $nd/2$. More precisely, we have the following definition.

Definition 3.2 (Expander Code)**.** *Let $\mathcal{C}_0 \subseteq \mathbb{F}_q^d$ be a linear code, and let $G := (V, E)$ be a d-regular expander graph on n vertices. The expander code $\mathcal{C}(G, \mathcal{C}_0) \subseteq \mathbb{F}_q^E$ is a linear code of length $nd/2$, so that for $c \in \mathbb{F}_q^E$, $c \in \mathcal{C}$ if and only if, for all $v \in V$,*

$$\left(c_{(v,\Gamma_1(v))}, \ldots, c_{(v,\Gamma_d(v))}\right) \in \mathcal{C}_0$$

One can easily check, by counting constraints, that if $\mathcal{C}_0$ has rate R_0, then $\mathcal{C}(G, \mathcal{C}_0)$ has rate at least $2R_0 - 1$ (see [SS96, Th. 7] for a proof). Moreover, these codes possess some sort of local correction. Indeed, to correct an edge e incident to a vertex v, one can retrieve the vector $(c_{(v,\Gamma_1(v))}, \ldots, c_{(v,\Gamma_q(v))})$ of labels of edges incident to v and correct it using the decoder of $\mathcal{C}_0$.

We say that an undirected graph $G := (L \cup R, E)$ is bipartite if, for all vertices $v \in L$, we have $\Gamma(v) \cap L = \emptyset$ and, for all vertices $v \in R$, we have $\Gamma(v) \cap R = \emptyset$. Let $G := (V, E)$ be an undirected d-regular graph on n vertices with expansion λ. From G, we can construct a d-regular graph $\tilde{G}$ on $2n$ vertices with expansion λ in the following way. The *double-cover* of G is the bipartite graph $\tilde{G} := (L \cup R, \tilde{E})$ defined as follows; let L and R be two copies of V. There is an edge between $u \in L$ and $v \in R$ if and only if $(u, v) \in E$.

3.2 An authentic server-memory resource tailored for code-based protocols

In this chapter, we focus on schemes based on erasure capabilities of error correcting codes. Thus, we need a setting where the actions of adversaries only lead to introducing erasures, instead of errors, in the outsourced data. The SMR security guarantees can be augmented to provide authenticity by using a suitable protocol. This new SMR is called authentic SMR, denoted by **aSMR**, and is introduced in [BM18]. In **aSMR**, the behavior of the server at interface S_I is weakened as the server cannot modify the content of data blocks but is limited to either delete or restore previously deleted data blocks. A deleted data block is indicated by the special symbol ϵ.

In this thesis, we use a different **aSMR** specification that the one used in [BM18]. We modify the `restore` behavior to only restore data blocks that were deleted after the last client update of the database. We introduce a version number that tracks the number of said updates in the history of the **aSMR** and the client is now allowed to overwrite corrupted data blocks. Our changes to the **aSMR** yield substantial improvements for the parameters of our code-based PoR schemes, we explain why in detail below. The **aSMR** specification of [BM18] is given in Fig. 3.1 while our take on the **aSMR** resource is described in Fig. 3.2.

In [BM18, Sec. 3.1], Badertscher and Maurer present a protocol that constructs an **aSMR** using a MAC function, timestamps and a tree structure on the outsourced data. Their construction of the **aSMR** has the following features:

1. The **aSMR** of size n with alphabet Σ is constructed from an **SMR** of size $2n-1$, alphabet $\Sigma \times \mathbb{Z}_q \times \mathcal{T}$ and a local memory of constant size for the client. $\mathcal{T}$ is the tag space of the MAC function used.
2. To read or write one memory cell on **aSMR**, the protocol of [BM18] produces $O(\log n)$ `read` and `write` queries to **SMR**.

This thesis focuses on PoR schemes where the client uploads a very large encoded file to an outsourced server. In this context, the logarithm of the size of the alphabet Σ is an order of magnitude smaller than the length of the MAC tags. The **aSMR** construction of [BM18] is not suited for this kind of application (even though they propose one such PoR scheme in [BM18, Sec. 7.1]). Its issues are threefold. First, since the file size is huge, a factor of 2 in the storage overhead is a big problem. Second, the $O(\log n)$ communication complexity for write operations is of no use to us since we will be working on encoded data and updating a codeword requires anyway to read a linear number of symbols. Third, the verification phase of PoRs often consists in probing as few symbols as possible to ensure that the outsourced file is retrievable in full. Having a $O(\log n)$ read communication complexity is a problem in this context.

With these observations, we now present a different protocol that constructs an **aSMR** with good features for our context:

1. Our **aSMR** of size n with alphabet Σ is constructed using an **SMR** of size n, alphabet $\Sigma \times \mathcal{T}$ and a local memory of constant size for the client.

aSMR$_{\Sigma,n}$ resource of [BM18]

The **aSMR** definition is identical to **SMR** except for the influence of an adversary at interface S_I and the reaction on writing to a corrupted memory location.

Interface C

Input: $(\texttt{read}, i) \in [1, n]$
  if ACTIVE **and not** INTRUSION
    HIST $\leftarrow$ HIST $||$ $(\texttt{R}, i)$
    return $\mathbb{M}[i]$

Input: $(\texttt{write}, i, x) \in [1, n] \times \Sigma$
  if ACTIVE **and not** INTRUSION
    if $\mathbb{M}[i] \neq \epsilon$
      HIST $\leftarrow$ HIST $||$ $(\texttt{W}, i, x)$
      $\mathbb{M}[i] \leftarrow x$
    else
      HIST $\leftarrow$ HIST $||$ $(\texttt{Fail}, i, x)$
      return ϵ

Interface S_I

Input: $(\texttt{delete}, i) \in [1, n]$
  if INTRUSION
    $\mathbb{M}[i] \leftarrow \epsilon$

Input: $(\texttt{restore}, i) \in [1, n]$
  if INTRUSION
    if $\exists k, x$: HIST$[k] = (\texttt{W}, i, x)$
      $k_0 \leftarrow \max\{k \mid \exists x$: HIST$[k] = (\texttt{W}, i, x)\}$
      Parse HIST$[k_0]$ as $(\texttt{W}, i, x_0)$
      $\mathbb{M}[i] \leftarrow x_0$
    else
      $\mathbb{M}[i] \leftarrow \lambda$

Figure 3.1: The authentic SMR of [BM18] (only the differences with **SMR** are shown).

Our aSMR$_{\Sigma,n}$ resource

The **aSMR** definition is identical to **SMR** except for the influence of an adversary at interface S_I and the addition of a version number ctr.

Interface C

Input: $(\texttt{read}, i) \in [1, n]$
  if ACTIVE **and not** INTRUSION
    HIST $\leftarrow$ HIST $||$ $(\texttt{R}, i)$
    return $\mathbb{M}[i]$

Input: $(\texttt{write}, i, x) \in [1, n] \times \Sigma$
  if ACTIVE **and not** INTRUSION
    $ctr \leftarrow ctr + 1$
    HIST $\leftarrow$ HIST $||$ $(\texttt{W}, i, x, ctr)$
    $\mathbb{M}[i] \leftarrow x$

Interface S_I

Input: $(\texttt{delete}, i) \in [1, n]$
  if INTRUSION
    $\mathbb{M}[i] \leftarrow \epsilon$

Input: $(\texttt{restore}, i) \in [1, n]$
  if INTRUSION
    if $\exists k, x$: HIST$[k] = (\texttt{W}, i, x, ctr)$
      $\mathbb{M}[i] \leftarrow x$

Figure 3.2: Our new authentic SMR (only the differences with **SMR** are shown).

2. A `read` request to our **aSMR** produces only one `read` request to **SMR**.
3. A `write` request to our **aSMR** produces at most $2n - 1$ `read` and `write` requests to **SMR**.

This way, we minimize the storage overhead and the communication complexity of read requests on the one hand. On the other hand, the increased communication complexity for write requests does not matter since our PoR schemes use only one such request. We sketch our protocol.

In the following, let n be the size of the **SMR**, $f_{sk}(\cdot)$ be a MAC function with tag space $\mathcal{T}$ and Σ be a finite alphabet. The protocol auth starts with the client choosing a secret key sk for the MAC function, setting a version number ctr to 0. The main idea is the following: if the i-th cell is supposed to store the data $x \in \Sigma$, the protocol will store the pair $(x, f_{sk}(x, ctr, i)) \in \Sigma \times \mathcal{T}$ instead. Do note that the version number ctr is incremented with every write request. This also means that every valid tag needs to be updated with every write request. Intuitively, this protocol prevents the adversary from:

1. Replacing the data x with $y \neq x$ since this would make the tag invalid.
2. Moving the data stored in location i to location $j \neq i$ since this would make the tag invalid.
3. Replaying an older value since the version numbers would not match and the tag would thus be invalid.

Now, we formally describe our protocol. In the following, let n be the size of the **SMR**, $f_{sk}(\cdot)$ be a MAC function with tag space $\mathcal{T}$ and Σ be a finite alphabet. The client will also have read and write access to a local memory resource denoted by $\mathbf{L}$. The protocol starts with the client choosing a secret key sk for the MAC function, setting a version number ctr to 0 and storing both of them in its local memory $\mathbf{L}$.

The protocol is formally depicted in Fig. 3.4, as a converter auth_{RW} to be plugged in the client's interface C. The generation of cryptographic keys and the initialization of the local and outsourced memories are presented in the initialization converter $\mathsf{init}_{\mathsf{auth}}$ for interface C_0, in Fig. 3.3.

From now on, when we speak of **aSMR**, we always refer to our specification of it (the one of Fig. 3.2).

Theorem 3.1. *Let $k, n \in \mathbb{N}$ and let $\Sigma_1 := \Sigma \times \mathcal{T}$ for some finite alphabet Σ. The protocol $\mathsf{auth} := (\mathsf{init}_{\mathsf{auth}}, \mathsf{auth}_{RW})$ described in Fig. 3.3 and Fig. 3.4 constructs the authentic SMR, say $\mathbf{aSMR}_{\Sigma,n}$, from $\mathbf{SMR}_{\Sigma_1,n}$ and a local memory $\mathbf{L}$ of constant size with respect to the simulator $\mathsf{sim}_{\mathsf{auth}}$ described in Fig. 3.5 and the pair $(\mathsf{honSrv}, \mathsf{honSrv})$. More precisely, for all distinguishers $\mathbf{D}$, we have*

$$\Delta^{\mathbf{D}}(\mathsf{honSrv}^{\mathsf{S}}\mathsf{auth}_{\mathcal{P}}[\mathbf{L}, \mathbf{SMR}_{\Sigma_1,n}], \mathsf{honSrv}^{\mathsf{S}}\,\mathbf{aSMR}_{\Sigma,n}) = 0$$

$$\textit{and} \quad \Delta^{\mathbf{D}}(\mathsf{auth}_{\mathcal{P}}[\mathbf{L}, \mathbf{SMR}_{\Sigma_1,n}], \mathsf{sim}^{\mathsf{S}}_{\mathsf{auth}}\,\mathbf{aSMR}_{\Sigma,n}) \leq \Gamma^{\mathbf{DC}}(\mathbf{G}^{\mathsf{MAC}})$$

Proof. The correctness condition is clear for the auth protocol so we only prove the security condition. We analyze the behavior of the real and the ideal systems on every possible input at their interfaces.

Upon `init`, `initComplete` at interface C_0: Upon the `init` query, the real system $\mathsf{auth}_{\mathcal{P}}[\mathbf{L}, \mathbf{SMR}_{\Sigma_1,n}]$ samples a new MAC key and initializes the version number ctr with the value 0. Then, it writes the value $(\lambda, f_{sk}(\lambda, i, ctr))$ at location i, for $i \in [1, n]$ and λ a fixed value in Σ. This adds $n + 1$ entries to the history which reads $(0, \texttt{init}) \,||\, (0, \texttt{W}, 1, f_{sk}(\lambda, 1, 0)) \,||\, \ldots (0, \texttt{W}, n, f_{sk}(\lambda, n, 0))$.

Converter $\mathsf{init}_{\mathsf{auth}}$ for interface C_0

Interface `out`

Input: `init`
  $sk \xleftarrow{\$} \mathcal{K}, ctr \leftarrow 0$
  output $(\texttt{write}, 1, sk)$ at interface `in` of **L**
  output $(\texttt{write}, 2, ctr)$ at interface `in` of **L**
  output `init` at interface C_0 of **SMR**
  for $i = 1$ **to** n **do**
    $v \leftarrow (\lambda, f_{sk}(\lambda, i, ctr))$
    output $(\texttt{write}, i, v)$ at interface C_0 of **SMR**

Input: $(\texttt{read}, i)$ or $(\texttt{write}, i, x)$
  Defined in the same way as for auth_{RW} in Fig. 3.4

Input: `initComplete`
  output `initComplete` at interface C_0 of **SMR**

Figure 3.3: The initialization protocol $\mathsf{init}_{\mathsf{auth}}$ for the construction of our **aSMR**.

Converter auth_{RW} for interface C

Interface `out`

Input: $(\texttt{read}, i) \in [1, n]$
  output $(\texttt{read}, 1)$ at interface `in` of **L**
  Let sk be the result
  output $(\texttt{read}, 2)$ at interface `in` of **L**
  Let ctr be the result
  output $(\texttt{read}, i)$ at interface C of **SMR**
  Let (x, tag) be the result
  if $f_{sk}(x, i, ctr) = tag$
    return x
  else
    return ϵ

Input: $(\texttt{write}, i, x) \in [1, n] \times \Sigma$
  output $(\texttt{read}, 1)$ at interface `in` of **L**
  Let sk be the result
  output $(\texttt{read}, 2)$ at interface `in` of **L**
  Let ctr be the result
  $ctr \leftarrow ctr + 1$
  $v \leftarrow (x, f_{sk}(x, i, ctr))$
  output $(\texttt{write}, i, v)$ at int. C of **SMR**
  for $j = 1, \ldots, i-1, i+1, \ldots, n$ **do**
    output $(\texttt{read}, j)$ at int. C of **SMR**
    Let (y, tag) be the result
    if $f_{sk}(y, j, ctr - 1) = tag$
      $v \leftarrow (y, f_{sk}(y, j, ctr))$
      output $(\texttt{write}, j, v)$ at int. C of **SMR**
  output $(\texttt{write}, 2, ctr)$ at int. `in` of **L**

Figure 3.4: The protocol auth_{RW} for the construction of our **aSMR**.

Simulator $\mathsf{sim}_{\mathsf{auth}}$

Initialization

$sk \xleftarrow{\$} \mathcal{K}, ctr \leftarrow 0$
$\mathbb{M}_{sim} \leftarrow (\lambda, f_{sk}(\lambda, 1, ctr)) \| \ldots \| (\lambda, f_{sk}(\lambda, n, ctr))$
$H_{sim} \leftarrow (0, \texttt{init}) \| (0, \texttt{W}, 1, (\lambda, f_{sk}(\lambda, 1, ctr))) \| \ldots \| (0, \texttt{W}, n, (\lambda, f_{sk}(\lambda, n, ctr)))$
$pos \leftarrow |H_{sim}| + 1$

Interface S_H

Input: `getHist`
  UpdateLog
  return H_{sim}
Input: $(\texttt{read}, i) \in [1, n]$
  UpdateLog
  return $\mathbb{M}_{sim}[i]$

Interface S_I (Intrusion = `true`)

Input: $(\texttt{write}, i, (v, tag)) \in [1, n] \times (\Sigma \times \mathcal{T})$
  UpdateLog
  $\mathbb{M}_{sim}[i] \leftarrow (v, tag)$
  Determine the last entry in H_{sim} that wrote value (v', tag') to location i.
  if $v = v'$ and $tag = tag'$
    output $(\texttt{restore}, i)$ at interface S_I of **aSMR**
  else
    output $(\texttt{delete}, i)$ at interface S_I of **aSMR**

  procedure UpdateLog
    output `getHist` at interface `in` of **aSMR**
    Let Hist be the returned value
    for $j = pos$ to $|\text{Hist}|$ **do**
      if $\text{Hist}[j] = (\texttt{R}, i)$
        $H_{sim} \leftarrow H_{sim} \,||\, (\texttt{R}, i)$
      else if $\text{Hist}[j] = (\texttt{W}, i, x, ctr')$
        if $ctr' > ctr$
          $ctr \leftarrow ctr'$
          $\mathbb{M}_{sim}[i] \leftarrow (x, f_{sk}(x, i, ctr))$
          $H_{sim} \leftarrow H_{sim} \,||\, (\texttt{W}, i, (x, f_{sk}(x, i, ctr)))$
          for $\ell = 1$ **to** n, $\ell \neq i$ **do**
            $H_{sim} \leftarrow H_{sim} \,||\, (\texttt{R}, \ell)$
            $y, tag \leftarrow \mathbb{M}_{sim}[\ell]$
            if $tag = f_{sk}(y, \ell, ctr - 1)$
              $\mathbb{M}_{sim}[\ell] \leftarrow (y, f_{sk}(y, \ell, ctr))$
              $H_{sim} \leftarrow H_{sim} \,||\, (\texttt{W}, \ell, (y, f_{sk}(y, \ell, ctr)))$
    $pos \leftarrow |\text{Hist}| + 1$

Figure 3.5: The simulator of the construction of our **aSMR**.

In the ideal system $\mathsf{sim}^{\mathsf{S}}_{\mathsf{auth}}\,\mathbf{aSMR}_{\Sigma,n}$, the query initializes the memory with the value λ. The simulator samples a MAC key sk and initializes the version number ctr to 0. Then, it initializes its simulated history H_{sim} with $n+1$ entries just like above.

Finally, on entry `initComplete` both systems deactivate interface C_0 and the other client interfaces become available.

Upon $(\texttt{read}, i)$ **at interface** C**:** On this query, auth_{RW} reads the i-th memory cell. The history is thus increased by the value $(\texttt{R}, i)$. Then, the protocol checks the validity of the tag. If it succeeds, the last value written to this location x_i is returned. Otherwise, ϵ is returned.
In the ideal system, the **aSMR** directly returns the last value written at location i if this cell's content has not been deleted. Otherwise, ϵ is returned. The simulator emulates this view by simulating the memory of the real world and sending `delete` (resp. `restore`) requests when the adversary writes values that would fail (resp. pass) the real world check. If this simulation is perfect, the behavior of the ideal system will be the same as the ideal one. We will discuss this when we analyze the `write` requests at interface S_I. Additionally, the next time the simulator is activated, it will update its simulated history H_{sim} using its procedure UpdateLog. If the read request $(\texttt{R}, i)$ is the next entry in the history Hist of the **aSMR**, the simulator increases its simulated history with the value $(\texttt{R}, i)$ which perfectly matches the real world behavior.

Upon $(\texttt{write}, i, x)$ **at interface** C_k**:** On a write request, the protocols start by incrementing the version number ctr and writing the value x, together with its tag t, to location i. The value $(\texttt{write}, i, (x, t))$ is thus appended to the history. Then, they read the content of each other location (in ascending order) and check their tag. If it is correct, a new tag is computed to account for the version number increase, and the value is written back together with the new tag. For $j = 1, \ldots, i-1, i+1, \ldots, n$, the history is increased by $(k, \texttt{R}, j)\ ||\ (k, \texttt{W}, j, (x_j, tag_j))$ if the check succeeds and by $(k, \texttt{R}, j)$ if the check fails.
In the ideal world, on a $(\texttt{write}, i, x)$ request, the i-th memory cell is updated with the value x and $(k, \texttt{W}, i, x, ctr)$ is appended to the history of **aSMR**, where ctr is the current version number. On the entry $(k, \texttt{W}, i, x, ctr)$ of the history, UpdateLog will increase the simulated history and update its simulated memory with the values listed above, using the version number ctr and its simulated key sk to check and produce the appropriate tags. Thus, the simulated history and memory perfectly match the ones of the real world.

Upon `getHist` **at interface** S_H **:** In the real system, the output is the history of **SMR**. By the above analysis, an inductive argument shows that in the ideal system, the simulated history H_{sim}, which is returned upon this query, perfectly emulates the real-world one.

Upon $(\texttt{write}, i, (x, tag))$ **at interface** S_I**:** In the real world, an adversarial write request is a simple replacement of the memory cell i of **SMR**. If (x, tag) corresponds to the last honest value written to this cell, then this cell might become valid again[1]. This is perfectly simulated in the ideal-world since the simulator can update its simulated memory and parse the history to check if (x, tag) is indeed the last honest value written to cell i. If it is the case, the simulator sends a $(\texttt{restore}, i)$ request to **aSMR** which makes the cell valid again if it should be.
Now, let's study the case where the pair (x, tag) is not the same as the last honest one written to this cell. In the real world, the content of the memory cell i is just replaced. In the ideal world, the simulator sends a $(\texttt{delete}, i)$ request to **aSMR** and assigns the value (x, tag) to the i-th cell of its simulated memory. If the pair (x, tag) fails the tag check, the simulation is perfect since the content of the i-th cell is deemed invalid in both worlds, making subsequent read requests return ϵ. However the bad event, denoted by BAD, occurs if the pair (x, tag) passes the verification (recall that this pair differs from the last honest value written to cell i). Indeed, in the real world, the check would succeed and the value x would be returned on

[1] if the version number has not increased

subsequent read requests. Meanwhile, in the ideal world, the value of cell i would be deleted and ϵ would be returned on subsequent read requests. Hence, the real and ideal systems are identical until event BAD occurs.

Upon $(\mathtt{read}, i)$ **at interface** S_H**:** In the real system, this query returns the value at location i of **SMR**. In the ideal system, the value at location i of the simulated memory is returned instead. The simulator updates its simulated memory on each activation. As we discussed above, the simulated memory perfectly emulates the real one in all cases. The behavior of both worlds are thus identical.

We conclude that the real and ideal systems are identical until the BAD event occurs. The occurrence of BAD implies a successful forgery against the MAC function f_{sk}. We use the same reduction $\mathbf{C}$ as in [BM18] from a distinguisher $\mathbf{D}$ to an adversary $\mathbf{A} := \mathbf{DC}$ against the game $\mathbf{G}^{\mathsf{MAC}}$ (defined in Section 3.1.1). $\mathbf{C}$ simulates the real system, but evaluates the MAC function using oracle queries to $\mathbf{G}^{\mathsf{MAC}}$. If $\mathbf{D}$ issues a write query at interface S_I that provokes event BAD, $\mathbf{C}$ issues this value as a forgery to the game. Hence, we conclude by noting that

$$\begin{aligned}\Delta^{\mathbf{D}}(\mathsf{auth}_{\mathcal{P}}[\mathbf{L}, \mathbf{SMR}_{\Sigma_1,n}], \mathsf{sim}_{\mathsf{auth}^{\mathsf{S}}}\, \mathbf{aSMR}_{\Sigma,n}) &\leq \Pr^{\mathbf{D}(\mathsf{auth}_{\mathcal{P}}[\mathbf{L},\mathbf{SMR}_{\Sigma_1,n}])}[\mathsf{BAD}] \\ &\leq \Gamma^{\mathbf{DC}}(\mathbf{G}^{\mathsf{MAC}})\end{aligned}$$

□

3.3 A framework for secure and efficient Proofs of Retrievability from codes with locality

We describe our framework which derives PoR schemes from a given LCC $\mathcal{C}$. In all our PoRs, the client's file is encoded as a codeword of $\mathcal{C}$ and uploaded to the server. We want to protect the client from an adversary able to introduce corruptions on the outsourced file. To do so, we need to describe an audit that probes a few symbols of the outsourced file and accepts if it thinks that the corruptions can all be corrected. Recall that, in the CC definition, an audit is considered secure if it only succeeds when the outsourced file is retrievable in full, without modifications. If we want to derive PoR schemes from an LCC $\mathcal{C}$ in CC, we thus need to do the following three things:

1. Give an extraction procedure that aims at retrieving the outsourced file while correcting any corruption encountered.
2. Characterize the configurations of corruptions that are uncorrectable by this extraction procedure.
3. Give an audit procedure that is able to detect those configurations of uncorrectable corruptions on the outsourced file.

Since a good PoR scheme must have low communication complexity, we want to exploit the locality of LCCs to design our audit procedure. We choose our extraction procedure as an iteration of the local correction algorithm of the LCC. This means that our schemes will try to locally correct any corruption encountered during the extraction. Thus, we need a way to identify those corruptions. Using the composability of the CC framework, we will place ourselves in a setting where adversaries can only introduce erasures on the outsourced file. We can design our PoR schemes with this assumption and we will use our **aSMR** construction of Section 3.2 to fulfill it. Our blueprint becomes:

1. Give an extraction procedure that aims at correcting erasures by using the local correctability of $\mathcal{C}$.

2. Characterize the configurations of erasures that are uncorrectable by this extraction procedure.
3. Our audit is the following: try to locally correct a random position of the outsourced file, if the correction is impossible return `reject`, else return `accept`.

In step 2, we identify the configurations of erasures that are unrecoverable when iterating the local correction of $\mathcal{C}$. We find a lower bound on the number of local correction queries that would fail if such a configuration of erasures existed. When instantiating our framework in Section 3.3.1, we shall see that this problem is, in practice, much easier than giving a lower bound on the minimum size of such a configuration of unrecoverable erasures. In the CC model of security for PoRs, the advantage of the adversary in breaking the security of the scheme is the probability that the audit accepts while the file is not retrievable. In our case, our audit consists in checking if a random local correction query succeeds. Our file is not retrievable if there exists a configuration of unrecoverable erasures. Thus, the lower bound we computed above is all we need to assess the security of the PoR. We give a complete proof when instantiating our framework, see Theorem 3.2 of Section 3.3.1.

More precisely, let $\mathcal{C}$ be an erasure code of length n, alphabet Σ and erasure symbol $\perp$. Suppose that $\mathcal{C}$ possesses a local erasure decoder $\mathcal{L}$ with query space $\mathcal{Q} \subseteq 2^{[1,n]}$. On query $q \in \mathcal{Q}$ and input $w \in (\Sigma \cup \{\perp\})^n$ such that there exists $c \in \mathcal{C}$ such that for any $i \in [1,n]$, $w_i \neq \perp$ implies $w_i = c_i$, $\mathcal{L}$ probes the symbols $w_{|q} := (w_i)_{i \in q}$ of w and attempts to correct its erasures if they exist. We can define a global decoder $\mathcal{G}$ for $\mathcal{C}$ by iterating $\mathcal{L}$ until no erasures remain. Let P be a predicate on $\cup_{i=0}^{n}(\Sigma \cup \{\perp\})^i$, *i.e.*, for $w \in (\Sigma \cup \{\perp\})^n$ and $q \in \mathcal{Q}$, $P(w_{|q}) \in \{\mathsf{true}, \mathsf{false}\}$. Let $0 \leq \epsilon \leq 1$ and suppose that we have the following property:

$$\forall w \in (\Sigma \cup \{\perp\})^n, \text{if at least one erasure of } w \text{ cannot be corrected by } \mathcal{G} \text{ then } \Pr_{q \in \mathcal{Q}}[P(w_{|q}) = \mathsf{false}] \geq 1 - \epsilon \tag{3.1}$$

We define our general PoR scheme $\mathsf{por} := (\mathsf{por}_{\mathsf{init}}, \mathsf{por}_{\mathsf{audit}})$, where:

1. On input $(\mathtt{write}, F)$, $\mathsf{por}_{\mathsf{init}}$ encodes F into a codeword $\tilde{F}$ of $\mathcal{C}$ and writes $\tilde{F}$ in the **aSMR** memory.
2. On input `audit`, $\mathsf{por}_{\mathsf{audit}}$ samples a query $q \in \mathcal{Q}$ uniformly. If w is the file stored in the SMR, $\mathsf{por}_{\mathsf{audit}}$ retrieves $w_{|q}$ with `read` queries. Then, the converter returns `accept` if $P(w_{|q}) = \mathsf{true}$ and returns `reject` otherwise.
3. On input (`read`), the converter $\mathsf{por}_{\mathsf{audit}}$ tries to extract the file F using the global decoder $\mathcal{G}$ of $\mathcal{C}$.

Recall that in the CC model of security for PoRs, the advantage of the adversary in breaking the security of por is the probability that the audit accepts while the file is not retrievable. In our case, this advantage is upper bounded by ϵ (see Eq. (3.1)). We believe our security model for PoRs to be cleaner, simpler and to give clearer security guarantees than the ϵ-adversary model.

3.3.1 Instantiation with Lifted Reed-Solomon codes

In this section, we use our PoR framework to design a secure and efficient PoR scheme using lifted RS erasure codes that can be seen as a generalization of the PoR of [LLDV16]. We call this scheme *lifted RS PoR scheme*. We build our PoR for an **aSMR**, using the composability of CC, so we only have to deal with potential erasures instead of errors. Using our blueprint, we need to do the following:

1. Give a global decoding algorithm for lifted RS codes using their local correctability.
2. Characterize the configurations of erasures that are unrecoverable by this algorithm.
3. Give an audit procedure that is able to detect those configurations of uncorrectable corruptions on the outsourced file.

We start with the global decoding algorithm. For the lifted RS code $\mathrm{Lift}_m(\mathrm{RS}_q[q,d])$, our global decoder works as follows. For each erasure, the decoding algorithm corrects it by finding, if it exists, a line going through the erasure and containing less than $d-1$ other erasures (for example using interpolation as in Section 3.1.2). If one or more erasures have been corrected during this step, the algorithm tries to correct the remaining erasures using the same method. Indeed, since some erasures were corrected, there exists lines with less erasures than before. If, during one iteration, no erasures have been corrected, the algorithm stops and returns the current vector. We give a pseudo-code description of this algorithm in Fig. 3.6.

Input: The encoded file V with potential erasures
Output: The encoded file $\tilde{F}$.
repeat
  $E := \emptyset$
  for an erased position $x \in \mathbb{F}_q^m$ **do**
    if there exists a line $\ell \subseteq \mathbb{F}_q^m$ going through x with strictly less than d erasures.
      Use the global decoder of $\mathrm{RS}_q[q,d]$ on the restriction of the file to ℓ.
      We have corrected all the erasures on that line, x included.
      $E = E \cup \{x\}$
      Modify V accordingly.
until $E = \emptyset$
return V

Figure 3.6: Our global decoding algorithm for lifted Reed-Solomon codes.

We now study the fail cases of the global decoding algorithm. Let $\mathrm{Lift}_m(\mathrm{RS}_q[q,d])$ be a lifted RS code. For an erased position $s \in \mathbb{F}_q^m$ to be unrecoverable, it is necessary that each line going through s possesses at least d erasures. However, it is not sufficient. Indeed, suppose that there exists a line ℓ going through s with exactly d erasures. If there exists an erasure position s' on the line ℓ and a line ℓ' going through s' with at most $d-1$ erasures then the symbol erased at position s' can be recovered using the $\mathrm{RS}_q[q,d]$ decoder. Since, s' lies on ℓ, this means that ℓ now contains only $d-1$ erasures and they all can be corrected, the one at s included.

In order to capture a set of unrecoverable erasures for our global decoding algorithm, we introduce the following property:

Definition 3.3 (d-cover sets)**.** *Let $\mathbb{F}_q$ be a finite field and m, d be positive integers. We say that a set $S \subseteq \mathbb{F}_q^m$ is a d-cover set if S verifies the following property:*

$$\forall s \in S, \forall \text{ line } \ell \subseteq \mathbb{F}_q^m \text{ going through } s, |S \cap \ell| \geq d$$

Or equivalently, for all lines $\ell \subseteq \mathbb{F}_q^m, |S \cap \ell| = 0$ or $|S \cap \ell| \geq d$

Since the d-cover subsets of $\mathbb{F}_q^m$ represent the unrecoverable erasure patterns, we want to find an audit procedure that can detect their existence with high probability and low communication complexity. We propose the following audit:

1. The client randomly chooses a line $\ell \subseteq \mathbb{F}_q^m$.
2. The client retrieves the restriction of the outsourced file to the chosen line.

3. If it contains d or more erasures, reject, if not, accept.

The next step is to determine the probability that this audit detects a set of unrecoverable erasures if one exists. Let $S \subseteq \mathbb{F}_q^m$ be a non-empty d-cover set. Then, there exists $s \in S$ and, for each line ℓ going through s, we have $|\ell \cap S| \geq d$. We also know that for any line $\ell \subseteq \mathbb{F}_q^m$, either $|\ell \cap S| = 0$ or $|\ell \cap S| \geq d$.

Recall that $L := (q^m - 1)/(q - 1)$ is the number of lines going through a point in $\mathbb{F}_q^m$ and that $q^{m-1}L$ is the total number of lines in $\mathbb{F}_q^m$. Let ℓ be the randomly chosen line for the audit and s be an element of S. We have:

$$\Pr[|\ell \cap S| \neq 0] = \frac{L}{q^{m-1}L} \cdot 1 + \left(1 - \frac{1}{q^{m-1}}\right) \cdot \Pr[|\ell \cap S| \neq 0 \mid s \notin \ell]$$

Let E be the event $\{|\ell \cap S| \neq 0 \mid s \notin \ell\}$. For each point $x \in \ell$, there is a unique line (xs) going through x and s. Since $s \in S$, this line contains at least d erased points in S, one being s. Since lines in $\mathbb{F}_q^m$ have q points, the probability that $x \in S$ is at least $(d-1)/(q-1)$. Moreover, if at least $q - d + 1$ points of ℓ do not belong to S we immediately know that $\ell \cap S = \emptyset$ since, by definition of S, either $|\ell \cap S| = 0$ or $|\ell \cap S| \geq d$. Thus, $\Pr[E] \geq 1 - (1 - (d-1)/(q-1))^{q-d+1}$.

$$\text{Therefore,} \quad \Pr[|\ell \cap S| \neq 0] \geq 1 - \left(1 - \frac{1}{q^{m-1}}\right)\left(1 - \frac{d-1}{q-1}\right)^{q-d+1}.$$

The calculation we just made is essential. Indeed, since we supposed $S \neq \emptyset$, the event $\neg\{|\ell \cap S| \neq 0\}$ can be interpreted as 'on probed line ℓ, the audit accepts although the file is not retrievable'. In the CC security model for PoR, this is exactly the advantage of the distinguisher, *i.e.* the security of the scheme. In other words, we just upper-bounded the security of our PoR scheme.

We now formally prove the security of our PoR in the CC framework. We quickly describe the converters $\mathsf{lift\_rs\_por}_{\mathsf{init}}$ and $\mathsf{lift\_rs\_por}_{\mathsf{audit}}$. Both use the encoder and global decoder for lifted RS codes. On input $(\texttt{read}, i)$, both converters retrieve the whole memory using `read` requests, then they call the global decoder on the obtained word (corrupted values ϵ are replaced with erasures) and return the i-th symbol of the output if decoding succeeds. On input $(\texttt{write}, i, x)$, both converters retrieve the whole memory with read requests and decode it like before. If decoding succeeds, they replace the i-th symbol by x, encode the whole word and store it on the SMR using `write` requests.

On input `audit`, $\mathsf{lift\_rs\_por}_{\mathsf{audit}}$ chooses a random line $\ell \subseteq \mathbb{F}_q^m$ and retrieves the restriction of the outsourced file to ℓ through `read` requests. If the restriction contains d or more erasures, it returns `reject`. If not, it returns `accept`.

Theorem 3.2. *Let $d, m, \ell \in \mathbb{N}$ and $\mathbb{F}_q$ be a finite field. The protocol* $\mathsf{lift\_rs\_por} := (\mathsf{lift\_rs\_por}_{\mathsf{init}}, \mathsf{lift\_rs\_por}_{\mathsf{audit}})$ *for the lifted RS code* $\mathrm{Lift}_m(\mathrm{RS}_q[q,d])$ *of dimension ℓ constructs the auditable and authentic SMR, say* $\mathbf{aSMR}^{\mathsf{audit}}_{\Sigma,\ell}$*, from* $\mathbf{aSMR}_{\Sigma,q^m}$*, with respect to the simulator* $\mathsf{sim}_{\mathsf{audit}}$ *and the dummy converter* honSrv*. More precisely, for all distinguishers* $\mathbf{D}$ *making at most r audits, we have*

$$\Delta^{\mathbf{D}}(\mathsf{honSrv}^{\mathsf{S}}\mathsf{lift\_rs\_por}_{\mathcal{P}}\,\mathbf{aSMR}_{\Sigma,q^m}, \mathsf{honSrv}^{\mathsf{S}}\,\mathbf{aSMR}^{\mathsf{audit}}_{\Sigma,\ell}) = 0 \quad \textit{and}$$

$$\Delta^{\mathbf{D}}(\mathsf{lift\_rs\_por}_{\mathcal{P}}\,\mathbf{aSMR}_{\Sigma,q^m}, \mathsf{sim}^{\mathsf{S}}_{\mathsf{audit}}\,\mathbf{aSMR}^{\mathsf{audit}}_{\Sigma,\ell}) \leq r \cdot \left(1 - \frac{1}{q^{m-1}}\right)\left(1 - \frac{d-1}{q-1}\right)^{q-d+1}$$

Proof. Since our scheme is clearly correct (*i.e.* the client can always retrieve its file when there is no adversary), we do not prove the availability condition. We prove security by comparing the behaviors of the audit of the real system (the **aSMR** with the protocol) with that of the

ideal one (the $\mathbf{aSMR}^{\mathsf{audit}}$ with the simulator). We describe the simulator $\mathsf{sim}_{\mathsf{auth}}$. It maintains a simulated memory, emulating the real world memory, using the history of the ideal resource. On $(\texttt{delete}, i)$, the simulator replaces the i-th entry of its simulated memory by ϵ. On $(\texttt{restore}, i)$, the simulator restores the content of the i-th entry of its simulated memory to the last value written there. The simulator maintains a simulated history using the ideal history of the $\mathbf{aSMR}^{\mathsf{audit}}$.

If, after a `delete` request, the set of corrupted locations of the simulated memory contains a d-cover subset of $\mathbb{F}_q^m$, the simulator deletes the whole ideal memory by sending `delete` requests to $\mathbf{aSMR}^{\mathsf{audit}}$. Similarly, if after a `restore` request, the set of corrupted locations of the simulated memory does not contain a d-cover subset of $\mathbb{F}_q^m$, the simulator restores the whole ideal memory by sending `restore` requests to $\mathbf{aSMR}^{\mathsf{audit}}$.

On an `audit` request, the simulator chooses a random line $\ell \subseteq \mathbb{F}_q^m$, adds the entries $(\texttt{read}, i)$ for $i \in \ell$ to its simulated history. Then, if the restriction of its simulated memory to ℓ contains strictly less than d corrupted cells, the simulator sends `allow` to $\mathbf{aSMR}^{\mathsf{audit}}$. Else, it instructs the $\mathbf{aSMR}^{\mathsf{audit}}$ to output `reject`.

Upon `auditReq` at interface S_H: Recall that d-cover sets are the sets of unrecoverable erasures for our global decoder of lifted RS codes. Suppose that a subset of the corrupted cells forms a d-cover set. In order to run the audit, the converter chooses a random line $\ell \subseteq \mathbb{F}_q^m$, retrieves the restriction of the memory to this line through `read` requests and adds the corresponding entries to its simulated history. We showed, see Section 3.3.1, that the probability that this restriction contains strictly less than d erasures, *i.e.*, that the audit is successful, is less than $(1 - 1/q^{m-1})(1 - (d-1)/(q-1))^{q-d+1}$.

The simulation is perfect unless the following BAD event occurs: *having simulated a real audit, the simulator answers* allow *(audit should succeed) whereas a d-cover subset of corrupted cells exists.* In that case, the simulator has chosen a restriction of the memory to a line ℓ that contains strictly less than d corrupted cells, and has written the corresponding `read` requests to its simulated history. Note that the distinguisher has access to the simulated history. Then, the simulator outputs allow to the ideal resource, that runs the ideal audit. Since there exists a d-cover set of corrupted memory cells, the file is unretrievable so the ideal audit fails and the client receives `reject`. The distinguisher thus observes the following incoherence: `reject` is returned while the (simulated) history contains the trace of a valid audit. The adversary knows that he is interacting with the ideal system. We give a detailed explanation of the BAD event in Remark 6.

To sum up, the only observable difference from a distinguisher point of view lies in the audit procedure. The overall distinguishing probability is thus the one of distinguishing a real audit from a simulated one. As we saw, if the distinguisher runs r audits, this probability is less than $r \cdot (1 - 1/q^{m-1}) \cdot (1 - (d-1)/(q-1))^{q-d+1}$, yielding the aforementioned result. □

Remark 6. We detail the interactions between the audit requests, the simulator, the distinguisher and the ideal resource, during an audit. It is essential to notice that the simulator is only connected to the interfaces S_I and S_H of the server, and has no interaction with the client. As `audit` is a functionality available at the client's interface, it follows that an audit request is not directly treated by the simulator. The following steps are done when the ideal resource $\mathbf{aSMR}^{\mathsf{audit}}$ receives an `audit` request at the client's interface C:

1. The resource sends `auditReq` at interface S_H.
2. Via its interface S_H, the simulator either responds `allow` or `abort`.
3. If `allow`, the audit is run and the resource sends back to the client either `accept` or `reject`, depending on whether the ideal audit has succeeded or failed.
4. If `abort`, the ideal resource always sends `reject` back to the client.

Thus, in the proof, the simulator cannot directly control the outcome of the ideal audit, in the sense that it cannot decide whether the ideal resource would send `accept` or `reject` back to the client. On the other hand, as it receives the `auditReq` request at interface S_H, the simulator can choose to answer `abort` or `allow` to the ideal resource. Let us now examine how the simulator behaves in both cases:

1. The simulator answers `abort`:
   It has received an audit request via `auditReq`. It then runs a simulation of a real audit (the one of the protocol) on its simulated memory. In this case, this test fails and the simulator is convinced that the real audit has to fail too. It thus decides to answer the ideal resource `abort`. Consequently, the ideal resource does not run an ideal audit, but rather answers the client `reject`. The client is in fact controlled by the distinguisher in the ideal setting. Looking at the simulated history, it (the distinguisher) believes to interact with the real resource.
2. The simulator answers `allow`:
   It has received an audit request via `auditReq`. It then runs a simulation of a real audit (the one of the protocol) on its simulated memory. In this case, this test succeeds and the simulator is convinced of the success of the real audit, as it simulated it with success. It thus sends `allow` to the ideal resource. The subtlety lies here: the ideal resource receives `allow` but it does not imply that it will send `accept` back to the client. It will run its ideal audit, and send the outcome of this audit to the client. The point is, when the simulator sends its answer (here `allow`) to the ideal resource, it already has simulated the real audit (here with success), and written the corresponding entries in its simulated history. Being not connected to the client's interface, the simulator does not "see" the result of the ideal audit (`accept` or `reject`) sent by the ideal resource to the client. Thus it cannot *a posteriori* modify its audit simulation to comply with the response of the ideal resource. The distinguisher, being connected to server's and client's interfaces, has access to the ideal audit response and, by comparing it to the simulated history, can see the incoherence; (the trace of the audit in the simulated history corresponds to one which should succeed). This incoherence never happens in the real resource, thus the distinguisher has distinguished between the two systems.

3.3.2 Instantiation with graph codes

We give another instantiation of our framework using graph codes. We described these codes in Section 3.1.3. In the following, let G be a regular graph and let d be the minimum distance of the inner code $\mathcal{C}_0$. Again, using the composability of CC, we only have to deal with potential erasures. Following our framework, we start by sketching our global decoder. In the following, we say that an edge is *erased* when the label of that edge is erased. Similarly, we say that we *correct* an edge if we correct the label of that edge.

Assume that we want to correct an erasure on an edge e incident to a vertex v. If v is incident to less than $d-1$ erased edges, we can use the erasure decoding of $\mathcal{C}_0$ to correct all the edges incident to v, e included. Otherwise, v is incident to $k > d-1$ erased edges. Pick an erased edge incident to v. This edge is also incident to a vertex $v' \neq v$. If v' is incident to less than $d-1$ erased edges, we can correct them all and v is now incident to $k-1$ erased edges. If $k-1 \leq d-1$ we can correct the edge e. Else, we iterate the process on v and its neighbors.

Now, we have to characterize the configurations of erased edges that are unrecoverable for our decoding algorithm. We claim that these unrecoverable configurations correspond to subgraphs of G of minimum degree d. Indeed, these are the graph analogues of the d-cover sets for lifted RS codes. We prove our claim: suppose that the subgraph formed by the unrecoverable edges

possesses a vertex v incident to less than $d-1$ unrecoverable edges. Then, by iterating the local decoding algorithm, we can recover the other edges incident to v so that only these unrecoverable edges remain erased. Then, since there are less than $d-1$ erased edges incident to v and since the minimum distance of the inner code is d, we can correct all the edges incident to v using the decoder of the inner code. This is in contradiction with these edges being unrecoverable.

Finally, the audit consists in randomly choosing a vertex v and retrieving the vector $w := (c_{(v,\Gamma_1(v))}, \ldots, c_{(v,\Gamma_q(v))})$ of labeling of edges incident to v. If w contains d or more erasures, the audit rejects. Else, it accepts.

The security of the audit depends on the graph G and the minimum distance d of the inner code $\mathcal{C}_0$. The bigger the minimum subgraphs of G with minimum degree d are, the better the security of the PoR will be. Indeed, let s be the minimum size (number of vertices) of a subgraph of G with minimum degree d. For a configuration of unrecoverable erasures to exist, we thus need at least s vertices of G with at least d erased edges. Recall that our audit chooses a random vertex of G and accepts if and only if this vertex is incident to less than $d-1$ erased edges. Thus, the probability that our audit accepts when there exists an unrecoverable set of erased edges is less than $1 - s/|V|$. In our framework, this is exactly the advantage of the adversary in breaking the security of our PoR. A similar proof and simulator to the ones of Theorem 3.2 yield the following theorem:

Theorem 3.3. *Let $G := (V, E)$ be a q-regular graph with $|V| := n$ and let $\mathcal{C}_0 \subseteq \mathbb{F}^q$ be a linear code with minimum distance d and rate R. Let s be the minimum size (number of vertices) of a subgraph of G with minimum degree d. The protocol $\mathsf{graph\_por} := (\mathsf{graph\_por}_{\mathsf{init}}, \mathsf{graph\_por}_{\mathsf{audit}})$ for an expander code $\mathcal{C}(G, \mathcal{C}_0)$ of length $nq/2$ and rate at least $2R-1$ that:*

1. *Starts by encoding the file and uploads it to the server.*
2. *On an* `audit` *request, chooses a random vertex $v \in V$ and accepts if and only if v is incident to less than $d-1$ erased edges.*
3. *Extracts the file using the algorithm sketched above.*

constructs the auditable and authentic SMR, say $\mathbf{aSMR}^{\mathsf{audit}}_{\mathbb{F},(2R-1)nq/2}$, from $\mathbf{aSMR}_{\mathbb{F},nq/2}$, with respect to the simulator $\mathsf{sim}_{\mathsf{audit}}$ and the dummy converter honSrv. More precisely, for all distinguishers $\mathbf{D}$ making at most r audits, we have

$$\Delta^{\mathbf{D}}(\mathsf{honSrv}^{\mathsf{S}}\mathsf{graph\_por}_{\mathcal{P}}\,\mathbf{aSMR}_{\mathbb{F},nq/2}, \mathsf{honSrv}^{\mathsf{S}}\,\mathbf{aSMR}^{\mathsf{audit}}_{\mathbb{F},(2R-1)nq/2}) = 0 \text{ and}$$

$$\Delta^{\mathbf{D}}(\mathsf{graph\_por}_{\mathcal{P}}\,\mathbf{aSMR}_{\mathbb{F},nq/2}, \mathsf{sim}^{\mathsf{S}}_{\mathsf{audit}}\,\mathbf{aSMR}^{\mathsf{audit}}_{\mathbb{F},(2R-1)nq/2}) \leq r \cdot \left(1 - \frac{s}{n}\right)$$

3.3.3 Parameters

The impact of the choice of the lifted RS code on the parameters of our lifted RS PoR scheme are highlighted in Fig. 3.8. The grey line gives a choice of parameters with a storage overhead of 13.9% and total communication of 0.01% of the file size. Increasing the length q of the RS base code decreases the storage overhead and increasing the lifting parameter m increases the size of the file stored. Exact formulae for the parameters of our scheme is given in Fig. 3.7.

Let us compare our parameters with the ones of [LLDV16]. First, in both schemes, the client's file is encoded using a lifted RS code and the audit consists in probing the restriction of this codeword to a random affine line. In our case, we authenticate the data using our MAC based authentication protocol (see Section 3.2) whereas [LLDV16] binds data to a specific location by using an encryption scheme. Let κ be the computational security parameter of both schemes and Σ be the alphabet of the code. Our scheme stores a code symbol along with a MAC tag, that is $\kappa + \log|\Sigma|$ bits, in each memory location of the server whereas [LLDV16]

stores a ciphertext, that is κ bits, in each memory location. Since $\log|\Sigma| \ll \kappa$ (we have $\kappa = 128$ and $|\Sigma| = q$ in Fig. 3.8), our scheme and the one of [LLDV16] have very close storage overhead and communication complexity. In [LLDV16], the minimum distance of the code d is chosen to be equal to 2. Using our security analysis of Theorem 3.2, we show that the [LLDV16] scheme has only 1.44 bits of statistical security, when $d = 2$, whereas state-of-the-art schemes expect at least 40. See Fig. 3.8 for our recommended parameters.

A major benefit of our scheme is that our audit produces less "false positives" than the one of [LLDV16]. For PoRs, a false positive occurs when an audit rejects while the file is still retrievable. In other words, the client thinks that he lost his file, but it is still retrievable in full. The number of false positive audits has no influence on the security of the PoR but, in practice, it is a situation that we absolutely wish to avoid. The audit of [LLDV16] rejects if the restriction of the file to an affine line does not belong to the RS base code. In other words, if there is at least one corruption on the line probed by the audit, it deems the file unretrievable. If the adversary introduces at least one erasure on every line of the space, the audit would always reject independently of the correction capability (*i.e.* the minimum distance) of the code. Using our framework and our authentication protocol, we are able to fix this problem. Indeed, our audit deems the file unretrievable only if the probed line contains at least d erasures, where d is the minimum distance of the RS base code. This means that we drastically decrease the number of false positive audits, making our scheme much more reliable and usable in practice.

For example, suppose that the outsourced file is encoded using a lifted RS code over $\mathbb{F}_q^2$ with minimum distance $d \geq 3$. Let ℓ_1, ℓ_2 be two intersecting lines in $\mathbb{F}_q^2$. Suppose that an adversary erases all the file's symbols at the locations given by ℓ_1 and ℓ_2 and no other symbols. Of course, the file is still retrievable since the local decoder can correct all the erasures of $\ell_1 \setminus \ell_2$ by querying all the lines parallel to ℓ_2 (these lines contain only one erasure and $d \geq 3$). Then, the local decoder can correct all the erasures of ℓ_2 by querying any line intersecting ℓ_2. Unfortunately, in this situation, the audit of [LLDV16] rejects with probability 1. Indeed, their audit chooses a random line ℓ in $\mathbb{F}_q^2$ and rejects if ℓ contains at least one erasure. This is always the case here since, either ℓ intersects ℓ_1 or, ℓ is parallel to ℓ_1 and is thus intersecting ℓ_2. This is not the case with our audit. Indeed, since only two lines of $\mathbb{F}_q^2$ have d or more erasures, our audit rejects with probability $2/(q^2+q)$ since there are q^2+q lines in $\mathbb{F}_q^2$.

	Exact value	Asymptotics
C. storage overhead	κ	$\mathcal{O}(1)$
S. storage overhead	$(\frac{1}{R} - 1)\|F\| + q^m\kappa$	$\mathcal{O}(\|F\|)$
C. $\rightarrow$ S.	$2m \log q$	$\mathcal{O}(\|F\|)$
S. $\rightarrow$ C.	$q(\kappa + \log q)$	$\mathcal{O}(\|F\|^{1/m})$

Figure 3.7: The exact parameters of our scheme. $|F|$ denotes the file size in bits, κ the security parameter of the MAC, q the field size and $m \geq 2$ the lifting parameter. We have $Rq^m \log q = |F|$.

Remark 7. Unfortunately, we are not yet able to propose parameters for our graph code PoR scheme. Indeed, we need to find regular graphs for which the size of the minimum subgraph of minimum degree d is as large as possible. This is the MSMD (minimum subgraph of minimum degree) problem. This problem was introduced by Erdös *et al.* in 1990 [EFRS90]. In 2011, Peleg *et al.* [PSS11] showed that the best known algorithms for this problem are not polynomial time, even when accepting an approximated solution. They observe that this problem appears to be difficult since the performance of their algorithms is not far from the best approximation algorithms for other very hard graph optimization problems like maximum clique, chromatic

PoR param.			Results					
m	q	d	$\lvert F\rvert$ (MB)	$\frac{1}{R}-1$	comm. C. $\to$ S. (bits)	comm. S. $\to$ C. (bits)	comm./$\lvert F\rvert$	Statistical Security
	256		0.03	1.056	32	2048	0.0081	2^{-42}
	512		0.18	0.631	36	4608	0.0032	2^{-43}
	1024	32	0.93	0.409	40	10240	0.0013	2^{-44}
	2048		4.50	0.279	44	22528	0.0006	2^{-44}
2	4096		21.0	0.195	48	49152	0.0003	2^{-44}
	8192		95.7	0.139	52	106496	0.0001	2^{-44}
	1024		0.79	0.641	40	10240	0.0016	2^{-88}
	2048	64	4.07	0.415	44	22528	0.0007	2^{-89}
	4096		19.6	0.282	48	49152	0.0003	2^{-90}
	8192		91.1	0.197	52	106496	0.0001	2^{-90}
3	512	32	35.9	3.197	54	4608	10^{-5}	2^{-43}

Figure 3.8: Different choices of lifted Reed-Solomon codes for our PoR scheme.

number or longest path problems.

3.4 Efficient Proofs of Retrievability from expander codes

Our new security analysis of the [LLDV16] PoR scheme has considerably worsened its concrete parameters, as shown in Fig. 3.8. The motivation of this section is to find a new code-based PoR scheme able to store very large files (up to hundreds of GB) with low storage overhead. We also try to make the extraction phase of the PoR as fast as possible.

3.4.1 Minimum distance and decoding of expander codes

We give some useful results, due to Sipser, Spielman and Zémor, about the minimum distance and fast erasure decoding for expander codes. First, it is known that expander codes constructed from bipartite graphs have good distance [SS96, Zém01]:

Proposition 3.4. *Let $\mathcal{C}_0 \subseteq \mathbb{F}_q^d$ be a linear code with relative distance δ, and let $G := (L \cup R, E)$ be a d-regular bipartite expander graph with expansion λ. Then the expander code $\mathcal{C}(G, \mathcal{C}_0)$ has distance at least $\delta(\delta - \lambda/d)$.*

Moreover, these codes can be efficiently uniquely decoded up to this fraction of erasures.

Proposition 3.5. *Let $\mathcal{C}_0 \subseteq \mathbb{F}_q^d$ be a linear code with relative distance δ. Let $\mathcal{D}(d)$ be the time needed to uniquely decode $\mathcal{C}_0$ from $\delta - 1/d$ erasures. Let $G := (L \cup R, E)$ be a d-regular bipartite expander graph on n vertices with expansion λ. Let $\epsilon > 0$ and suppose that $\frac{\lambda}{d} < \frac{\delta}{2}$. Then, there is an algorithm which uniquely decodes the expander code $\mathcal{C}(G, \mathcal{C}_0)$ from up to $(1-\epsilon)\delta(\delta - \lambda/d)$ erasures in time $n \cdot \mathcal{D}(d)/\epsilon$.*

We use the algorithm given in [RZWZ20] and we follow their presentation and proof. This algorithm is well-known and follows from [SS96, Zém01], we describe it in Fig. 3.9. Since we will use it in the extraction phase of our PoR, we prove its correction and complexity for completeness. We will need to use the expander mixing lemma.

Algorithm UniqueDecode

Input: A description of the graph $G := (L \cup R, E)$ of degree d and of the code $\mathcal{C}_0 \subseteq \Sigma^d$ of relative minimum distance δ, and $z \in (\Sigma \cup \{\perp\})^E$.

Output: The unique $c \in \mathcal{C}(G, \mathcal{C}_0)$ so that c agrees with z on all un-erased positions.

1: $E_1 := \{e \in E \mid z_e \neq \perp\}$
2: $P_0 := \{v \in R \mid v \text{ is incident to an edge } e \in E \setminus E_1\}$
3: $P_1 := \{v \in L \mid v \text{ is incident to an edge } e \in E \setminus E_1\}$
4: **for** $t = 2, 3, \ldots$ **do**
5: **if** $P_{t-1} = \emptyset$
6: **return** the fully labeled codeword.
7: $P_t \leftarrow \emptyset$ and $E_t \leftarrow E_{t-1}$.
8: **for** each vertex $v \in P_{t-1}$ so that $|(\{v\} \times \Gamma(v)) \cap E_{t-1}| > (1-\delta)d$ **do**
9: Run $\mathcal{C}_0$'s erasure-correction algorithm to assign labels to the edges incident to v.
10: Remove v from P_{t-1}.
11: For any $(v, u) \notin E_{t-1}$, add (v, u) to E_t.
12: For each vertex $v \in P_{t-1}$, for any $(v, u) \notin E_{t-1}$, add u to P_t.

Figure 3.9: Unique expander code erasure decoder from up to $\delta(\delta - \lambda/d)(1-\epsilon)$ erasures where λ is the expansion of the graph G and $\epsilon > 0$.

Theorem 3.6 (Expander Mixing Lemma, see [HLW06])**.** *Suppose that $G := (L \cup R, E)$ is the double cover of a d-regular expander graph on n vertices with expansion λ. Then, for any $S \subseteq L$ and $T \subseteq R$,*

$$\left| E(S,T) - \frac{d}{n}|S||T| \right| \leq \lambda \sqrt{|S||T|}$$

where $E(S,T)$ denotes the set of edges with endpoints in $S \cup T$.

First, notice that on any iteration $t \geq 2$, E_{t-1} is the subset of edges that have already been labeled before this iteration, and $P_{t-2} \cup P_{t-1}$ is the set of vertices touching an edge in $E \setminus E_{t-1}$ that is still erased. The following lemma bounds the size of P_t, and thus the number of steps the algorithm takes before terminating on line 6.

Lemma 3.7 ([RZWZ20])**.** *The following holds:*

1. *For any $t \geq 1$, $|P_{t+1}| \leq (1-\epsilon)(\delta - \frac{\lambda}{d})n$*
2. *For any $t \geq 2$, $|P_{t+1}| \leq (\frac{1}{1+\epsilon})^2 |P_t|$*

Proof. For any $t \geq 1$, let $B_{t-1} \subseteq P_{t-1}$ be the subset of vertices $v \in P_{t-1}$ that are incident to less than $(1-\delta)d$ un-erased edges in E_{t-1}. We have

$$P_{t+1} \subseteq B_{t-1} \subseteq P_{t-1} \tag{3.2}$$

since all vertices $v \in P_{t-1} \setminus B_{t-1}$ are removed from P_{t-1} on line 10, and consequently will not be in P_{t+1}.

For the first item, we have $|P_3| \leq |B_1|$ by (Eq. (3.2)), and that $|B_1| \leq (\delta - \lambda/d)(1-\epsilon)n$ since there are at most $(1-\epsilon)\delta(\delta - \lambda/d)nd$ erasures to begin with. Moreover, we have that $|P_3| \geq |P_5| \geq |P_7| \geq \ldots$, thus $|P_{t+1}| \leq (1-\epsilon)(\delta - \lambda/d)n$ for any even $t \geq 1$. Using a similar technique, one can show that this holds for any odd $t \geq 1$.

For the second item, the expander mixing lemma implies, for $t \geq 2$,

$$\delta d|B_{t-1}| \leq |E(B_{t-1}, P_t)| \leq \frac{d}{n}|B_{t-1}||P_t| + \lambda\sqrt{|B_{t-1}||P_t|}$$

as any vertex $v \in B_{t-1}$ has at least δd erased incident edges, and those edges are incident to P_t. After some rewriting, we have

$$|B_{t-1}| \leq \left(\frac{\lambda/d}{\delta - |P_t|/n}\right)^2 |P_t| \leq \left(\frac{1}{1+\epsilon}\right)^2 |P_t|$$

where the last inequality follows from the assumption that $\lambda/d \leq \delta/2$ and from $|P_t|/n \leq (1-\epsilon)(\delta - \lambda/d)$ by the first item. Using (Eq. (3.2)) we finally get

$$|P_{t+1}| \leq |B_{t-1}| \leq \left(\frac{1}{1+\epsilon}\right)^2 |P_t|$$

□

Using the above lemma, we conclude that after $\mathcal{O}((\log n)/\epsilon)$ iterations, P_{t-1} is empty and the algorithm terminates. Moreover, the amount of work done is at most

$$\mathcal{D}(d) \cdot \sum_{t=1}^{\infty} |P_t| = \mathcal{D}(d) \cdot n \sum_{t=1}^{\infty} \left(\frac{1}{1+\epsilon}\right)^{2t} = \mathcal{D}(d) \cdot \frac{n}{\epsilon},$$

where $\mathcal{D}$ is the complexity of $\mathcal{C}_0$'s erasure decoder, which proves Prop. 3.5.

3.4.2 Generic audit for erasure codes

Our scheme will use a generic audit for erasure codes translated in CC by Badertscher and Maurer in [BM18]. First, we give notation for erasure codes. An (n, k, d) erasure code over a finite alphabet Σ, with erasure symbol $\perp \notin \Sigma$, is a pair of algorithms $(\mathsf{enc}, \mathsf{dec})$ that satisfy: for all $F \in \Sigma^k$, let $\bar{F} := \mathsf{enc}(F) \in \Sigma^n$ and define the set

$$E := \{W \in (\Sigma \cup \{\perp\})^n \mid \forall i, W_i \in \{\bar{F}_i, \perp\} \text{ and at most } d-1 \text{ positions of } W \text{ are equal to } \perp\}$$

Then, for all $W \in E$, we have $\mathsf{dec}(W) = F$.

We describe how Badertscher and Maurer implemented the ideas of [JK07, SW08] to construct the $\mathbf{aSMR}^{\mathsf{audit}}_{\Sigma^k,1}$ of Fig. 1.3 from our $\mathbf{aSMR}_{\Sigma,n}$ of Fig. 3.2. Let $(\mathsf{enc}, \mathsf{dec})$ be an (n, k, d) erasure code and $F \in \Sigma^k$ be the client's file. We describe the PoR scheme $\mathsf{ecPor} := (\mathsf{ecInit}, \mathsf{ecAudit})$ for erasure codes.

On input `init` to ecInit, the converter sends `init` to $\mathbf{aSMR}_{\Sigma,n}$ and computes the encoding $\bar{F} := \mathsf{enc}(F) \in \Sigma^n$. Then, for each location $i \in [n]$, the converter sends $(\texttt{write}, i, \bar{F}_i)$ to $\mathbf{aSMR}_{\Sigma,n}$.

On input $(\texttt{read}, k)$ to either ecInit or $\mathsf{ecAudit}$, the converter retrieves the whole memory content via $(\texttt{read}, i)$ requests and obtains for each location, either a symbol $v_i \in \Sigma$ or the erasure symbol $\perp$. If v_i is returned, set $W_i := v_i$ else set $W_i := \perp$. If $|\{i \in [n] \mid W_i = \perp\}| > d-1$, the converter outputs ϵ at its outside interface, otherwise it computes $F := \mathsf{dec}(W)$, and outputs F_k.

Finally, on input `audit` to $\mathsf{ecAudit}$, the converter chooses a random subset $S \subseteq [n]$ of size t and outputs $(\texttt{read}, i)$ to $\mathbf{aSMR}$ for each $i \in S$ to retrieve the memory content at location i. If all read instructions for $i \in S$ returned a non-erased symbol, the converter outputs `accept`. Otherwise, it outputs `reject`. The integer t is chosen according to the security level we want to achieve. The security of the scheme is given by:

Theorem 3.8 ([BM18])**.** *Let $n, k, d \in \mathbb{N}$. Let* $(\mathsf{enc}, \mathsf{dec})$ *be an* (n, k, d) *erasure code for alphabet* Σ *and erasure symbol* $\perp$*. Let* $\rho := 1 - \frac{d-1}{n}$ *be the minimum fraction of symbols needed to recover the file. Then, the above protocol* $\mathsf{ecPor} := (\mathsf{ecInit}, \mathsf{ecAudit})$ *that chooses a random subset of size* t *during the audit, constructs the* $\mathbf{aSMR}^{\mathtt{audit}}_{\Sigma^k,1}$ *from the* $\mathbf{aSMR}_{\Sigma,n}$ *with respect to the simulator* sim *(described in the proof). More specifically, for all distinguishers* $\mathbf{D}$ *performing at most* q *audits,*

$$\Delta^{\mathbf{D}}(\mathsf{ecPor}_{\mathcal{P}}\, \mathbf{aSMR}_{\Sigma,n}, \mathsf{sim}^{\mathsf{S}}\, \mathbf{aSMR}^{\mathtt{audit}}_{\Sigma^k,1}) \leq q \cdot \rho^t$$

Proof. (*sketch* [BM18]) We only compare the behaviors of the audit of the real system (the **aSMR** with the protocol) and of the ideal one (the $\mathbf{aSMR}^{\mathtt{audit}}$ with the simulator). The two systems behave in the same way in every other case, the reader can refer to [BM18] for a full proof.

Assume that a fraction α of cells of the real world **aSMR** have been deleted such that a $\beta := 1 - \alpha$ fraction is still available. A standard bound for binomial coefficients ensures that the probability of selecting a subset of memory locations containing no erased symbol during the audit is

$$\frac{\binom{\beta \cdot n}{|S|}}{\binom{n}{|S|}} \leq \beta^{|S|}$$

In the case when decoding would not be possible, *i.e.* if $\beta < \rho$, we see that the probability that the audit succeeds in the real world is no larger than $\rho^{|S|}$.

We describe the simulator sim. It maintains a simulated memory, emulating the real world memory, using the history of the ideal resource. On $(\mathtt{delete}, i)$, the simulator replaces the i-th entry of its simulated memory by ϵ. On $(\mathtt{restore}, i)$, the simulator restores the content of the i-th entry of its simulated memory to the last value written there. The simulator maintains a simulated history using the (ideal) history of the $\mathbf{aSMR}^{\mathsf{audit}}$.

If, after a `delete` request, the simulated memory contains d or more erased locations, the simulator deletes the whole ideal memory by sending `delete` requests to $\mathbf{aSMR}^{\mathsf{audit}}$. Similarly, if after a `restore` request, the simulated memory contains $d - 1$ or less erased locations, the simulator restores the whole ideal memory by sending `restore` requests to $\mathbf{aSMR}^{\mathsf{audit}}$.

On an audit request, the simulator simulates the random locations probed during the audit by adding the appropriate `read` requests to its simulated history and evaluates if the audit succeeds by checking that none of the simulated locations are deleted. If so, it outputs allow to instruct the ideal resource to output the right result to the client, and otherwise it instructs the resource to output `reject`. The simulation is perfect until the following BAD event occurs : *having simulated a real audit, the simulator answers* allow *(audit should succeed) whereas d or more memory locations are currently erased.* It is the only case when simulation can differ from real execution. Thus, the probability of distinguishing can be upper bounded by the probability that event BAD happens in an execution. As discussed above, this happens with probability no larger than $q \cdot \rho^{|S|}$ over q audits. □

3.4.3 Proofs of Retrievability from expander codes: the general case

Let $\mathcal{C}_0$ be a linear code of length d, relative distance δ_0 and rate R_0. Using the Singleton bound, we have $\delta_0 \leq 1 + \frac{1}{d} - R_0$. Let G be a d-regular bipartite graph on n vertices with expansion λ. We are going to instantiate the PoR scheme $\mathsf{ecPor} := (\mathsf{ecInit}, \mathsf{ecAudit})$ with the expander code $\mathcal{C}(G, \mathcal{C}_0)$.

In the following, we determine the number t, of edges probed during the audit, needed to reach a given security level. If we suppose that $\frac{\lambda}{d} < \frac{\delta_0}{2}$, using the Singleton bound, we must

have $R_0 < 1 + \frac{1}{d} - \frac{2\lambda}{d}$. Moreover, if $\mathcal{C}_0$ is Maximum Distance Separable, this implication becomes an equivalence. This is why, from now on, we will suppose that the inner code $\mathcal{C}_0$ is MDS.

We took G to be a bipartite expander graph with expansion λ such that $\frac{\lambda}{d} < \frac{\delta_0}{2}$. Using Prop. 3.4, the minimum distance $\delta_{\mathcal{C}}$ of $\mathcal{C}(G, \mathcal{C}_0)$ is at least

$$\delta_0(\delta_0 - \frac{\lambda}{d}) > \frac{2\lambda^2}{d^2}$$

Let $\epsilon > 0$. If we want to correct a $(1-\epsilon)\delta_{\mathcal{C}}$ fraction of erasures, the minimum fraction of valid edges needed to recover our file is

$$\rho = 1 + \frac{1}{nd} - (1-\epsilon)\delta_{\mathcal{C}} \leq 1 + \frac{1}{nd} - (1-\epsilon)\frac{2\lambda^2}{d^2}$$

Let σ be a statistical security parameter and t be the number of edges probed during the audit. Our scheme is considered secure if $\rho^t \leq 2^{-\sigma}$. We want to choose t such that $t \geq -\sigma / \log \rho$. *Approximation* : If $\frac{1}{nd} - (1-\epsilon)\frac{2\lambda^2}{d^2} \approx 0$, we have

$$\begin{aligned} \frac{-\sigma}{\log \rho} &\approx \frac{nd^2\sigma}{2(1-\epsilon)n\lambda^2 - d} \\ &= \frac{d^2\sigma}{2(1-\epsilon)\lambda^2 - \frac{d}{n}} \end{aligned}$$

Moreover, if G is Ramanujan, we have $\lambda \leq 2\sqrt{d-1}$ and

$$\frac{-\sigma}{\log \rho} \approx \frac{d\sigma}{8(1-\epsilon)}$$

If G has expansion $\sqrt{d}$ instead, we have

$$\frac{-\sigma}{\log \rho} \approx \frac{d\sigma}{2(1-\epsilon)}$$

Note that our scheme requires the adversary to only introduce erasures (and not errors). We enforce this using our authentic server-memory resource **aSMR**. After a successful audit, the client can extract its file by running the decoder of Prop. 3.5 which runs in time $\mathcal{O}(n \cdot \mathcal{D}(d)/\epsilon)$, where $\mathcal{D}(d)$ is the complexity of $\mathcal{C}_0$'s decoder.

3.4.4 Instantiation with the point-line incidence graph of the plane

Let Γ be the point-line incidence graph of the affine plane over $\mathbb{F}_q$ without the vertical lines. This graph is q-regular, has $2q^2$ vertices and expansion $\sqrt{q}$ (see the work of Tanner [Tan84, Section 3]). A line of work of Høholdt *et al.* [HJ06, HJ11, BHPJ13] extensively studies the application of this graph to coding theory. We have $\Gamma := (V_1 \cup V_2, E)$ where

$$V_1 := \{(x, y) \mid x, y \in \mathbb{F}_q\}, \quad V_2 := \{(a, b) \mid a, b \in \mathbb{F}_q\}$$

and

$$E := \{((x, y), (a, b)) \mid (x, y) \in V_1, (a, b) \in V_2, ax + b - y = 0\}$$

This graph is an excellent choice for our PoR scheme. Recall that the rate of the inner code is upper bounded by $1 + \frac{1}{d} - \frac{2\lambda}{d}$ and the rate of the expander code is lower bounded by $2R_0 - 1$. Thus, going from expansion $2\sqrt{q-1}$ (the expansion of a q-regular Ramanujan graph) to $\sqrt{q}$

permits us to decrease the storage overhead of our scheme. This comes at the cost of increasing the number of edges probed during the audit - and thus the communication complexity of our PoR - by a factor of 4. See Section 3.4.5 for a comparison of the parameters of our PoR using the graph Γ and a Ramanujan graph.

The graph Γ also has a good ratio between its regularity q and its number of edges q^3. Since we need to probe a number of edges linear in q, this ensures that our PoR scheme has communication complexity of order cubic root of the size of the outsourced file. This is in line or even better than other code-based PoR schemes, such as [LLDV16] (which has communication complexity of order square root of the file size for $m = 2$). In Section 3.4.5, we will show that, at a given security level, our PoR scheme stores much larger files than the lifted code-based PoR scheme of [LLDV16].

Our inner code $\mathcal{C}_0$ will be a Reed-Solomon code of rate $R_0 < 1 + \frac{1}{d} - \frac{2\lambda}{d}$. This code is MDS and thus, we can use the decoder of Prop. 3.5 for our extraction phase. Moreover, because our inner code is a Reed-Solomon code, we can use the following result of Beelen *et al.* [BHPJ13].

Let $\mathbb{F}_q := \{\alpha_1, \alpha_2, \ldots, \alpha_q\}$. We use the following labeling (of [BHPJ13]) for the edges of Γ: if $(x, y) \in V_1$, $\Phi_{(x,y)}(i) := (x, y, \alpha_i, y - x\alpha_i)$ and, if $(a, b) \in V_2$, $\Phi_{(a,b)}(i) := (\alpha_i, a\alpha_i + b, a, b)$.

When q is a power of 2 or a prime, Beelen *et al.* [BHPJ13, Theorem 6] showed that when using this labeling on the graph Γ with a Reed-Solomon code of rate $1/2 < R_0 \leq 1$ as inner code, we obtain an expander code of rate *exactly* $R := R_0^3 + R_0(1 - R_0)(2R_0 - 1)$.

3.4.5 Parameters

Let σ be the statistical security parameter (usually $\sigma = 40$ in the PoR literature) and κ be the computational security parameter[2] ($\kappa = 128$). Set q, a power of 2. Let Γ be the q-regular point-line incidence graph on the affine plane $\mathbb{F}_q^2$. Recall that this graph has $2q^2$ vertices, q^3 edges and expansion $\lambda := \sqrt{q}$.

Let the inner code $\mathcal{C}_0$ be a Reed-Solomon code of length q and rate $R_0 = \max\{\frac{k}{q} \mid k \in \mathbb{N} \text{ and } \frac{k}{q} < 1 + \frac{1}{q} - \frac{2\lambda}{q}\}$. We take R_0 to be as big as possible (to reduce the storage overhead of the PoR) while still having a quasi-linear time decoder for the expander code. Indeed, since q is a power of 2, $\mathcal{C}_0$ can be erasure decoded in time $\mathcal{O}(q \log^2 q)$ thanks to the decoder of Tang and Lin [TL20].

Our expander code $\mathcal{C}(\Gamma, \mathcal{C}_0)$ has length q^3, rate $R := R_0^3 + R_0(1 - R_0)(2R_0 - 1)$ and alphabet $\mathbb{F}_q$. Let $|F|$ be the size of the outsourced file in bits. It is such that $|F| = Rq^3 \log(q)$. Using Prop. 3.5, we get an erasure decoder for $\mathcal{C}(\Gamma, \mathcal{C}_0)$ (and thus an extraction phase) running in time $\mathcal{O}(2q^3 \log^2 q)$ which is quasi-linear in the input size $q^3 \log q$. The storage overhead is given by $1/R - 1$, which is the redundancy of the code. The parameters of our PoR scheme and their asymptotic behavior are given in Fig. 3.10. Even though our PoR has the same asymptotic behavior than the PoR of [LLDV16], we will show below that we get much better parameters in practice.

In Fig. 3.11, we give the concrete parameters of our PoR scheme for different values of q. Let us compare our scheme with our generalization of the PoR of [LLDV16]. For $q = 512$, Fig. 3.8 gives a lifted code-based PoR with codewords of length q^3, maximum file size of 35.9MB and storage overhead of 319%. In Fig. 3.11, we see that for $q = 512$ and codeword length q^3, our PoR has maximum file size 124MB, storage overhead of 21%, communication complexity of the same order of magnitude and a quasi-linear time extraction phase. Overall, our expander code PoR scheme stores much larger files with lesser storage overhead than our secure generalization of the [LLDV16] PoR scheme.

[2] of the MAC used to construct the **aSMR**

	Exact value	Asymptotics ($\lvert F\rvert \to \infty$)
C. storage overhead	κ	$\mathcal{O}(1)$
S. storage overhead	$(\frac{1}{R}-1)\lvert F\rvert + q^3\kappa$	$\mathcal{O}(\lvert F\rvert)$
comm. C. $\to$ S.	$\frac{q\sigma}{2}\log(q^3)$	$\mathcal{O}(\lvert F\rvert^{\frac{1}{3}}\log\lvert F\rvert)$
comm. S. $\to$ C.	$\frac{q\sigma}{2}(\kappa+\log q)$	$\mathcal{O}(\lvert F\rvert^{\frac{1}{3}}\log\lvert F\rvert)$

Figure 3.10: The parameters of our scheme when using the point-line incidence graph over $\mathbb{F}_q^2$ and a Reed-Solomon code as inner code. $|F|$ denotes the file size in bits, κ the security parameter of the MAC, σ the statistical security parameter and R the rate of the code. We have $Rq^3\log(q) = |F|$.

q	R_0	R	$2q^2$	$\lvert F\rvert$	$\frac{1}{R}-1$	comm./$\lvert F\rvert$
256	0.878	0.758	131,072	12MB	0.320	2×10^{-4}
512	0.913	0.827	524,288	124MB	0.210	6×10^{-5}
1024	0.938	0.876	2,097,152	1.176GB	0.141	1×10^{-5}
2048	0.956	0.912	8,388,608	10.772GB	0.096	3×10^{-6}
4096	0.968	0.936	33,554,432	96.485GB	0.068	8×10^{-7}
8192	0.978	0.956	134,217,728	854.055GB	0.046	2×10^{-7}

Figure 3.11: Effective parameters of our PoR using the point-line incidence graph over $\mathbb{F}_q^2$ for different values of q and Reed-Solomon codes as inner code. The graph is q-regular with $2q^2$ vertices. We choose the largest possible rate yielding a quasi-linear time decoder. The statistical security parameter is 40.

To give some perspective on the impact of the choice of the point-line incidence graph on the parameters of the PoR, we give the parameters of our scheme using a Ramanujan graph of the same size with expansion $2\sqrt{q-1}$ instead of $\sqrt{q}$. These parameters can be found in Fig. 3.12.

q	R_0	R	$\lvert F\rvert$	$\frac{1}{R}-1$	comm./$\lvert F\rvert$
256	0.754	0.509	8MB	0.965	1×10^{-4}
512	0.825	0.651	98MB	0.537	2×10^{-5}
1024	0.876	0.752	1.009GB	0.330	4×10^{-6}
2048	0.912	0.824	9.735GB	0.213	1×10^{-6}
4096	0.938	0.875	90.246GB	0.142	2×10^{-7}
8192	0.956	0.912	814.614GB	0.097	5×10^{-8}

Figure 3.12: Effective parameters of our PoR using a q-regular Ramanujan graph with $2q^2$ vertices and expansion $2\sqrt{q-1}$ for different values of q and a Reed-Solomon code as inner code. The graph is q-regular with $2q^2$ vertices. We choose the largest possible rate yielding a quasi-linear time decoder. The statistical security parameter is 40.

3.5 On the composability of locally correctable codes

In the following, we model LCCs in the CC framework. We obtain an error-resilient SMR where clients can read symbols with sublinear communication. Our SMR can then be used when modeling any communication channel that relies on the correction capabilities of LCCs.

3.5.1 aSMR with locally correctable codes

We describe our LCC protocol for the **aSMR** by specifying the client's $\mathsf{lcc}_{\mathsf{init}}$ and lcc_{RW} converters. We denote by $(\mathsf{enc}, \mathsf{localdec}, \mathsf{globaldec})$ an $(r, \delta, \tilde{\epsilon})$-LCC where enc is the encoder, $\mathsf{localdec}$ is the local decoder and where the global decoder $\mathsf{globaldec}$ tolerates up to $d-1$ erasures (denoted by the symbol $\perp$). Let ℓ' be the length of the code, and ℓ its dimension. To correct the i-th symbol of a codeword, the local correcter reads[3] $r' \leq r$ symbols chosen according to distribution[4] $\mathcal{D}_r(i)$.

On input `init`, the converter $\mathsf{lcc}_{\mathsf{init}}$ sends `init` to the resource and computes the encoding $F' \leftarrow \mathsf{enc}(\lambda^\ell) \in \Sigma^{\ell'}$, where λ is an arbitrary element of Σ. Finally, the converter stores F'_i at each address i of the $\mathbf{aSMR}_{\Sigma,\ell'}$, for $i \in [\ell']$.

Given $(\texttt{read}, i)$ as input, the converter lcc_{RW} samples r' positions according to the distribution $\mathcal{D}_r(i)$. For each position i_j, $1 \leq j \leq r'$, it gets - via $(\texttt{read}, i_j)$ requests - either a value $v_{i_j} \in \Sigma$ or the error symbol ϵ of the **aSMR**. If ϵ has been returned, the converter sets $x_{i_j} \leftarrow \perp$. Otherwise, it sets $x_{i_j} \leftarrow v_{i_j}$. Having done this for all r' positions, the converter then computes $x \leftarrow \mathsf{localdec}(x_{i_1}, \ldots, x_{i_{r'}})$ and, if it succeeds, returns the result. If the decoding fails ($x = \perp$), it returns ϵ.

Given $(\texttt{write}, i, F'_i)$ (where $F'_i \in \Sigma$) as input, the converter lcc_{RW} starts by retrieving the whole memory via $(\texttt{read}, i)$ requests to **aSMR**, and obtains either a value $v_i \in \Sigma$ or an error ϵ. If ϵ has been returned, the converter sets $\bar{F}_i \leftarrow \perp$. Otherwise, it sets $\bar{F}_i \leftarrow v_i$. If $|\{i \in [\ell'] \mid \bar{F}_i = \perp\}| \geq d$, the converter returns ϵ at its outer interface. Otherwise, it computes $F \leftarrow \mathsf{globaldec}(\bar{F})$, updates $F_i \leftarrow F'_i$ and re-encodes the new memory content F' as $\bar{F}' \leftarrow \mathsf{enc}(F')$. Finally, it sends $(\texttt{write}, i, \bar{F}'_i)$ for all $i \in [\ell']$ to **aSMR**.

We now have to specify the ideal resource this construction is supposed to achieve, write an appropriate simulator, and check if the ideal resource equipped with it behaves like the constructed one, with respect to any distinguisher.

3.5.2 First attempt: aSMR with uniform pollution factor

An intuitive way to define the ideal resource is to let an adversary set a pollution threshold α on the **aSMR**. When the client asks to read any memory cell, it fails with probability α. This follows the definition of LCCs, where the bound on the decoding failure probability is independent from the considered cell (it only depends on the number of erasures). Unfortunately, in this setting, it is possible for an adversary to choose the location of the erasures so that the decoding of some cells fail less (or more) often than others. This permits to distinguish between the ideal resource and the real **aSMR** resource with LCC. We give an example of this using the Hadamard code of length 7.

Example 3. Recall that the coordinates of an Hadamard codeword of length 7 are indexed by the non-zero elements of $\mathbb{F}_2^3$, and that each codeword is of the form $(f(\mathbf{u})_{\mathbf{u} \in \mathbb{F}_2^3 \setminus \{\mathbf{0}\}})$ where f is a linear form over $\mathbb{F}_2^3$. The local decoder proceeds as follows to decode the value of coordinate $\mathbf{u}$: with probability $\frac{3}{4}$, it uniformly chooses a coordinate $\mathbf{v} \neq \mathbf{u}$ and returns $f(\mathbf{u}+\mathbf{v}) - f(\mathbf{v})$ which, by linearity, is equal to $f(\mathbf{u})$ if the $\mathbf{u}+\mathbf{v}$ and $\mathbf{v}$ coordinates are not erased, and with probability $\frac{1}{4}$, it returns $f(\mathbf{u})$, which is correct if this coordinate is not erased. One can check that each coordinate is chosen uniformly by the local decoder. We give an example of erasure pattern together with the decoding probabilities of each coordinate of the corrupted codeword

[3] r' depends on the probabilistic choices of the algorithm.

[4] This distribution depends on the local decoding algorithm.

for this decoder:

Coordinate	$(0,0,1)$	$(0,1,0)$	$(0,1,1)$	$(1,0,0)$	$(1,0,1)$	$(1,1,0)$	$(1,1,1)$
Value	$\perp$	$\perp$	$f(0,1,1)$	$f(1,0,0)$	$f(1,0,1)$	$f(1,1,0)$	$f(1,1,1)$
Failure probability	$1/2$	$1/2$	$1/4$	$1/2$	$1/2$	$1/2$	$1/2$

To prove the security of a construction in CC, we have to evaluate the probability, for an adversary having access to the real resource equipped with the protocol, and to the ideal resource equipped with the simulator, to distinguish between the two resources. In the above example, the adversary (the distinguisher) can erase coordinates $(0,0,1)$ and $(0,1,0)$ on both resources. Then, with `read` requests, it can estimate the failure probability of the decoder of each resource on coordinate $(0,1,1)$. On the real resource, the decoder fails with probability $1/4$, as shown above. On the ideal resource, due to the introduction of the two erasures by the distinguisher, the simulator will assign α an upper bound on the decoding failure probability, namely $1/2$ here. This difference in behavior allows the distinguisher to know with which resource it interacts.

Remark 8. It has to be noted that a negative result in the CC model only proves that a protocol does not construct the targeted ideal resource. The protocol might construct another ideal resource, or the construction might become secure by adding hypotheses.

To construct a resource that is secure in the CC sense, we give another abstraction of LCCs.

3.5.3 aSMR with independent pollution factors

Starting from an **aSMR**, the converters described in Section 3.5.1 constructs the resource **aSMR_LCC** of Fig. 3.13.

Resource aSMR_LCC$_{\Sigma,n}$

Initialization

INIT, ACTIVE, INTRUSION $\leftarrow$ `false`
$\alpha_i \leftarrow 0$ for all i; HIST $\leftarrow [\,]$
write_mode = `true`

Interface S_I

Input: $(\texttt{setPollution}, \beta, i) \in [0,1] \times [n]$
  if INTRUSION
    $\alpha_i \leftarrow \beta$
Input: $(\texttt{setWriteMode}, b) \in \{\texttt{true}, \texttt{false}\}$
  if INTRUSION
    write_mode $\leftarrow b$

Interface C

Input: $(\texttt{read}, i) \in [n]$
  if ACTIVE **and not** INTRUSION
    $Z \leftarrow \mathsf{Bernoulli}(\alpha_i)$
    if $Z = 0$
      HIST $\leftarrow$ HIST$\|(\texttt{R}, i)$
      return $\mathbb{M}[i]$
    else
      HIST $\leftarrow$ HIST$\|(\texttt{Failure}, i)$
      return ϵ

Input: $(\texttt{write}, i, x) \in [n] \times \Sigma$
  if ACTIVE **and not** INTRUSION
    if write_mode
      HIST $\leftarrow$ HIST$\|(\texttt{W}, i, x)$
      $\mathbb{M}[i] = x$
    else
      HIST $\leftarrow$ HIST$\|(\texttt{Failure}, i, x)$
      return ϵ

Figure 3.13: Our **aSMR** with independent pollution factors (interfaces C_0 and S_H are the same as in the **aSMR**).

In this new resource, the adversary is given the ability to choose a failure probability α_i independently for each location i, $1 \leq i \leq \ell'$, that applies during $(\texttt{read}, i)$ requests. In the LCC setting, an adversary can freely choose the number and locations of erasures so as to make decoding failure happen less (or more) often for some coordinates than for others. Each configuration of erasures comes with its own set of failure probabilities $(\alpha_1, \ldots, \alpha_{\ell'})$; see the example with the Hadamard code in Ex. 3. This means that the capabilities of our adversary are stronger than in reality as he can choose any set of failure probabilities, not only the ones associated to a given configuration of erasures. In the CC language, this means that we have chosen to use a larger (and thus simpler to describe) specification for our modeling of LCCs. If one wishes to precisely match the behavior of LCCs, he can weaken our adversary by only allowing him to choose the failure probabilities $(\alpha_1, \ldots, \alpha_{\ell'})$ from a specific set: the set of failure probabilities of the LCC's local decoder on any configuration of erasures.

Additionally, the adversary can control the ability of the client to update the memory content, by means of a boolean variable `write_mode`. Indeed, when using LCCs, the adversary can make the global decoding impossible by introducing too many erasures. Thus preventing the client from updating its outsourced file.

In the following, the passive server (given by the honSrv converter) never puts any nonzero pollution factor on `read` requests, and never prevents the clients from updating the memory. In this setting, we make the following Hypothesis 3.1 and Hypothesis 3.2:

Hypothesis 3.1. *There exists a polynomial time algorithm* check *that, given a partial codeword* $(x_{i_1}, \ldots, x_{i_{r'}}) \in (\Sigma \cup \{\perp\})^{r'}$ *and* i*, returns* `Success` *if the local decoder will always succeed in decoding cell* i*, and* `Failure` *otherwise.*

In the above hypothesis, $(x_{i_1}, \ldots, x_{i_{r'}})$ are the symbols queried by the local decoding algorithm to try to retrieve the i-th symbol of the codeword. It is clear that a deterministic local decoding algorithm satisfies hypothesis Hypothesis 3.1. This will be the case for all the codes considered in this thesis.

Remark 9. In the following, we assume that LCCs satisfy Hypothesis 3.1. Thus, we describe an LCC with the four algorithms (enc, localdec, globaldec, check) where enc is the encoder, localdec is the local decoder, globaldec is the global decoder and check is the algorithm of Hypothesis 3.1.

Hypothesis 3.2. *Given an erasure pattern, it is possible to compute, in polynomial time, the exact failure probability of the decoder at each coordinate.*

We will use these assumptions to make the previous distinguishing strategy not possible anymore.

Remark 10. Hypothesis 3.2 is fulfilled for any deterministic local decoder that has a polynomial number of possible queries. Indeed, the probabilities can then be computed by running the local decoder on each possible query and counting the number of failures. This is the case for the Hadamard code and we show that it is also true for at least one class of high-rate LCC, namely the lifted Reed-Solomon codes.

Let $\mathbb{F}_q$ be a finite field, $m, d \in \mathbb{N}^*$ and consider the lifted Reed-Solomon code $\mathrm{Lift}_m(\mathrm{RS}_q[q, d])$. There are $L := (q^m - 1)/(q - 1)$ lines passing through any point of $\mathbb{F}_q^m$. In order to compute the failure probability of the local decoder on a point $x \in \mathbb{F}_q^m$, we can compute the ratio of lines passing through x that have d or more erased points. If we do this for every point of $\mathbb{F}_q^m$, we can compute the exact failure probability of the local decoder on each coordinate of the lifted code. There are $q^{m-1}L$ distinct lines in $\mathbb{F}_q^m$ so this computation is polynomial in the length of the lifted code q^m.

In the following, let $\tilde{\epsilon}_i$ be the function that associates, to an erasure pattern, the failure probability of the local decoder on the i-th coordinate. Thanks to our Hypothesis 3.2, all of these functions are efficiently computable. We now give a more formal description of the **aSMR_LCC** resource we obtain.

Theorem 3.9. *Let $\ell, \ell', d, r \in \mathbb{N}$, $\delta \in [0,1]$, $\tilde{\epsilon} : [0,1] \to [0,1]$ and $(\tilde{\epsilon}_i : (\Sigma \cup \{\perp\})^{\ell'} \to [0,1])_{1 \leq i \leq \ell}$. Let $(\mathsf{enc}, \mathsf{localdec}, \mathsf{globaldec}, \mathsf{check})$ be an $(r, \delta, \tilde{\epsilon})$-LCC over the alphabet Σ, with error symbol $\perp$, dimension ℓ', length ℓ and minimum distance d. The decoding failure probabilities of its decoder are given by the functions $(\tilde{\epsilon}_i)_{1 \leq i \leq \ell}$. The protocol $\mathsf{lcc} := (\mathsf{lcc}_{\mathsf{init}}, \mathsf{lcc}_{RW})$ described in Section 3.5.1 constructs the authentic SMR with LCC, say $\mathbf{aSMR\_LCC}_{\Sigma,\ell}$, from $\mathbf{aSMR}_{\Sigma,\ell'}$ with respect to the simulator $\mathsf{simind}_{\mathsf{LCC}}$, described in Fig. 3.14 and Fig. 3.15, and the pair $(\mathsf{honSrv}, \mathsf{honSrv})$. More precisely, for all distinguishers $\mathbf{D}$, we have*

$$\Delta^{\mathbf{D}}(\mathsf{honSrv}^{\mathsf{S}} \mathsf{lcc}_{\mathcal{P}}\, \mathbf{aSMR}_{\Sigma,\ell'}, \mathsf{honSrv}^{\mathsf{S}}\, \mathbf{aSMR\_LCC}_{\Sigma,\ell}) = 0$$

$$\textit{and} \quad \Delta^{\mathbf{D}}(\mathsf{lcc}_{\mathcal{P}}\, \mathbf{aSMR}_{\Sigma,\ell'}, \mathsf{simind}^{\mathsf{S}}_{\mathsf{LCC}}\, \mathbf{aSMR\_LCC}_{\Sigma,\ell}) = 0$$

Proof. We only prove the security condition. The availability condition is plainly satisfied, as the server never puts erasures on the real resource, and always leaves the probabilities α_i to 0 on the ideal resource. To prove the security, we analyze the behavior of each system on all possible queries at each of their interfaces. The simulator is described in Fig. 3.14 and Fig. 3.15.

Upon `init`, `initComplete` queries at interface C_0: upon `init`, the $\mathsf{lcc}_{\mathsf{init}}$ converter of the real system $\mathsf{lcc}_{\mathcal{P}}\, \mathbf{aSMR}_{\Sigma,\ell'}$ initializes the **aSMR** resource and encodes the message λ^{ℓ}. Let $(F'_1, \ldots, F'_{\ell'})$ be the resulting codeword. The converter then writes this codeword in memory, which adds ℓ' entries to the history HIST of **aSMR**. This initialization phase being done, HIST is of the form $(0, \texttt{init}) \| (0, \texttt{W}, 1, F'_1, 1) \| \ldots \| (0, \texttt{W}, \ell', F'_{\ell'}, \ell')$.
In the ideal system $\mathsf{simind}^{\mathsf{S}}_{\mathsf{LCC}}\, \mathbf{aSMR\_LCC}_{\Sigma,\ell}$, `init` initializes the memory with the message λ^{ℓ} and adds the entry $(0, \texttt{init})$ to the simulated history. As it is the first entry of the history, the simulator replaces it by its local simulated list L_{init}, that contains the above ℓ' entries. Note that while the **aSMR_LCC** does not use version numbers in its history, the **aSMR** does. Thus, a simulated version number *ctr* will be updated throughout the simulation and it will be added to `write` entries of the simulated history.
Finally, upon `initComplete`, both systems deactivate their C_0 interface. The client's interface C is thus active for the remaining of the protocol.

Upon $(\texttt{delete}, i)$ at interface S_I: in the real system, this query erases the i-th symbol of the codeword. This means that the local decoding failure probability is potentially raised on some coordinates of the stored codeword. Furthermore, if the number of erasures becomes greater than or equal to d, the corrupted codeword can no longer be decoded by the global decoder. Thus, the converter will fail subsequent `write` queries.
In the ideal system, upon $(\texttt{delete}, i)$, the simulator removes the content of address i of its simulated memory. Then it modifies the reading failure probabilities α_j $(1 \leq j \leq \ell)$ with the failure probabilities of the local decoder according to the functions $\tilde{\epsilon}_j$ called on its simulated memory. If the number of erased memory cells exceeds d, the simulator deactivates the write mode of the client, thus preventing him from modifying the memory content. This way, the simulator perfectly emulates the consequences of putting an erasure in the real world.

Upon $(\texttt{restore}, i)$ at interface S_I: in the real system, this query makes a cell recover its value before deletion. In the case when the client has not modified the memory content of the cell during the interval between deletion and restore, the version number of the cell is still valid, and the local decoding failure probability will potentially decrease on some coordinates of the stored word. The number of invalid cells can then become strictly less than d, thus allowing the global decoder to work again.

In the ideal system, the simulator restores the content of address i of its local simulated memory with the last written value at this address (found in its simulated history). Then it modifies the reading failure probabilities α_j $(1 \leq j \leq \ell)$ with the failure probability of the local decoder given by the function $\tilde{\epsilon}_j$ called on its simulated memory. If the number of invalid memory cells becomes strictly less than d, the simulator reactivates the write mode of the client, thus allowing him to modify the memory content. This way, the simulator perfectly emulates the consequences of attempting to restore a cell in the real world.

Upon $(\texttt{read}, i)$ **at interface** C**:** the protocol lcc_{RW} chooses $r' \leq r$ coordinates $(i_1, \ldots, i_{r'})$ of the codeword according to distribution $\mathcal{D}_r(i)$, and reads their content on the server. Thus, $(\mathtt{R}, i_1) \| \ldots \| (\mathtt{R}, i_{r'})$ are added to HIST. The converter then runs the local decoder for coordinate i, from the r' results given by the server. The local decoding failure probability, taken over the random choices of the r' coordinates according to distribution $\mathcal{D}_r(i)$, is given by the $\tilde{\epsilon}_i$ function. But, upon each `delete` or `restore` request to the ideal system, the simulator updates the reading failure probabilities α_j for all j, with the results of the $\tilde{\epsilon}_j$ functions. This way, the reading failure probabilities are the same in the two systems.

Upon $(\texttt{write}, i, x)$ **at interface** C**:** by means of `read` requests, the real system retrieves the whole memory, thus adding ℓ' entries $(\mathtt{R}, j)$ to its history. Then the converter checks if the number of invalid coordinates still permits the global decoder to run properly. If not, the protocol returns an error and stops. If yes, the real system puts the value x at coordinate i, encodes the modified message and rewrites the whole codeword in memory.
On the ideal resource, the simulator emulates the effect of the $(\texttt{write}, i, x)$ request when it updates its simulated history (via the function UPDATELOG). It starts by adding to its simulated history the entries corresponding to a read of the whole simulated memory. If, following a `delete` request, the number of invalid coordinates in the ideal resource has become too large, the simulator has then deactivated the possibility to write on the server, thus yielding an error upon the `write` request, as in the real world. The ideal resource thus emulates perfectly the behavior of the real resource upon `write` requests.

Upon (`getHist`) **at interface** S_H**:** in the real system, the output is the history of the **aSMR**. In the ideal world, the simulator runs the UPDATELOG procedure every time it processes a query. This procedure parses the history of the resource. On `read` entries for location i, the simulator samples r' locations according to the distribution $\mathcal{D}_r(i)$, the distribution followed by the real world protocol. If the read request was a success (resp. a failure), the simulator uses the `check` algorithm to make sure that the local decoder does not always fail (resp. succeed) when querying the sampled locations. This ensures that the simulation matches what actually happened in the real world. If the condition is fulfilled, the sampled locations are added to the simulated history. If not, new locations are sampled until the condition is met. If the history's entry is a successful `write`, the function UPDATELOG increments the version number ctr, corrects the content of the the simulated memory, assigns the value x at coordinate i and writes the encoding of the modified message in its simulated memory. The corresponding entries are added to the simulated history. Thus, the simulated history perfectly emulates the real world history.

Upon $(\texttt{read}, i)$ **at interface** S_H**:** the real system outputs the content of address i of its memory. The ideal system outputs the content of address i of its simulated memory. As it uses the same encoding as the converter upon `write` requests, and perfectly emulates the `delete` and `restore` requests on the content of its simulated memory, the value output to the server by the ideal system perfectly emulates the real world view.

Finally, we showed that the real and ideal worlds have the same behavior. It is thus impossible, even for a computationally unbounded adversary, to tell them apart. □

Simulator $\mathsf{simind}_{\mathsf{LCC}}$

Initialization

$(F_1, \ldots, F_{\ell'}) \leftarrow \mathsf{enc}(\lambda^\ell)$, $\mathbb{M}_{sim} \leftarrow F_1 \| \ldots \| F_{\ell'}$
$L_{init} \leftarrow (0, \mathtt{init}) \| (0, \mathtt{W}, 1, F_1, 1) \| \ldots \| (0, \mathtt{W}, \ell', F_{\ell'}, \ell')$
$pos \leftarrow 1$, $ctr \leftarrow \ell'$, $L \leftarrow [\,]$

Interface S_H

Input: `getHist`
UpdateLog
return L

Input: $(\mathtt{read}, i) \in [\ell']$
UpdateLog
return $\mathbb{M}_{sim}[i]$

Interface S_I

Input: $(\mathtt{delete}, i) \in [\ell']$
UpdateLog
$\mathbb{M}_{sim}[i] \leftarrow \epsilon$
for $j = 1$ to ℓ **do**
output$(\mathtt{setPollution}, \tilde{\epsilon}_j(\mathbb{M}_{sim}[1], \ldots, \mathbb{M}_{sim}[\ell']), j)$ to interface S_I of **aSMR_LCC**
$e \leftarrow |\{1 \le j \le \ell' \mid \mathbb{M}_{sim}[j] = \epsilon\}|$
if $e \ge d$
output $(\mathtt{setWriteMode}, \mathtt{false})$ to interface S_I of **aSMR_LCC**

Input: $(\mathtt{restore}, i) \in [\ell']$
UpdateLog
if $\exists k, x : L[k] = (\mathtt{W}, i, x, ctr)$
$\mathbb{M}_{sim}[i] \leftarrow x$
for $j = 1$ to ℓ **do**
output$(\mathtt{setPollution}, \tilde{\epsilon}_j(\mathbb{M}_{sim}[1], \ldots, \mathbb{M}_{sim}[\ell']), j)$ to interface S_I of **aSMR_LCC**
$e \leftarrow |\{1 \le j \le \ell' \mid \mathbb{M}_{sim}[j] = \epsilon\}|$
if $e < d$
output $(\mathtt{setWriteMode}, \mathtt{true})$ to interface S_I of **aSMR_LCC**

Figure 3.14: The simulator used in the construction of the **aSMR** with independent pollution factors. The procedure UpdateLog can be found in Fig. 3.15

Simulator $\mathsf{simind}_{\mathsf{LCC}}$ (part 2)

procedure UPDATELOG
  output `getHist` to `in` of **aSMR_LCC**
  Let HIST be the returned value
  for $j = pos$ to $|\text{HIST}|$ **do**
    if $\text{HIST}[j] = (0, \texttt{init})$
      $L \leftarrow L_{init}$
    else if $\text{HIST}[j] = (\texttt{R}, i)$
      $(i_1, \ldots, i_{r'}) \leftarrow \mathcal{D}_r(i)$
      while $\mathsf{check}(\mathbb{M}_{sim}[i_1], \ldots, \mathbb{M}_{sim}[i_{r'}]) =$ `Failure` **do**
        $(i_1, \ldots, i_{r'}) \leftarrow \mathcal{D}_r(i)$
      for $l = 1$ to r' **do**
        $L \leftarrow L \| (\texttt{R}, i_l)$
    else if $\text{HIST}[j] = (\texttt{Failure}, i)$
      $(i_1, \ldots, i_{r'}) \leftarrow \mathcal{D}_r(i)$
      while $\mathsf{check}(\mathbb{M}_{sim}[i_1], \ldots, \mathbb{M}_{sim}[i_{r'}]) =$ `Success` **do**
        $(i_1, \ldots, i_{r'}) \leftarrow \mathcal{D}_r(i)$
      for $l = 1$ to r' **do**
        $L \leftarrow L \| (\texttt{R}, i_l)$
    else if $\text{HIST}[j] = (\texttt{op}, i, x)$
      for $l = 1$ to ℓ' **do**
        $L \leftarrow L \| (\texttt{R}, l)$
        $x_l \leftarrow \mathbb{M}_{sim}[l]$
        if $x_l = \epsilon$
          $x_l \leftarrow \perp$
      if $\texttt{op} = \texttt{W}$
        $(F_1, \ldots, F_\ell) \leftarrow \mathsf{globaldec}(x_1, \ldots, x_{\ell'})$
        $F_i \leftarrow x$
        $(\bar{F}_1, \ldots, \bar{F}_{\ell'}) \leftarrow \mathsf{enc}(F_1, \ldots, F_\ell)$
        for $l = 1$ to ℓ' **do**
          $\mathbb{M}_{sim}[l] = \bar{F}_l$
          $ctr \leftarrow ctr + 1$
          $L \leftarrow L \| (\texttt{W}, l, \bar{F}_l, ctr)$
  $pos \leftarrow |\text{HIST}| + 1$

Figure 3.15: The simulator used in the construction of the **aSMR** with independent pollution factors (part 2).

Chapter 4

Interactivity in Constructive Cryptography

In this chapter, we extend the CC model so as to handle interactive protocols. By interactive, we mean any protocol where the honest parties need the agreement of an untrusted party to deploy the protocol. In the outsourced storage setting, the honest parties usually are a set of clients and the untrusted party is a distant server. We use our extension of CC to model two interactive protocols for outsourced storage.

First, we give the first composable modeling of UE. We then use our modeling to compare the latest security notions for UE. In particular, we show that the IND-UE notion of [BDGJ20] is not always strictly stronger than the IND-ENC + IND-UPD notion of [LT18]. We also show that RCCA is the correct security notion to consider for real-world applications of UE. A concurrent work of Fabrega *et al.* [FMM21] proposed a modeling of UE in CC, but it was not composable as they needed a new proof for each particular UE scheme. Their treatment of interactivity is quite technical, which makes resources and proofs a lot harder to follow and understand. Our own treatment of interactivity follows a different approach. Indeed, we believe that resources should capture and enable the possibility of interactivity while converters should handle the technical part of how interactions happen in practice.

The modeling of UE of [FMM21] allowed them to consider a new attack vector for UE: the adversarial control of randomness. They show that it is preferable to use the randomness to compute the update token, and then use the knowledge of the old key and token to compute the new key. Indeed, if the adversary controls the randomness, using it to compute the new key would immediately compromise it. We could include this insight in our modeling of UE by explicitly modeling randomness sources.

Second, we model PIR protocols. With this modeling, we verify that the security definitions of [CGKS95] for PIR bring the expected security guarantees. Then, we extend our modeling to give a composable modeling of PIR that unifies computational PIR, information theoretic PIR, one server PIR and multi-server PIR.

4.1 Preliminaries

4.1.1 RCCA security for Updatable Encryption

Section 4.3 of this chapter will heavily focus on the different security notions for UE. Recall that these notions are precisely described in Section 1.3 of Chapter 1. Following [KLR19], we present a relaxed variant, in the UE context, of CCA security called RCCA (where R stands for replayable). In differs from the CCA notion in that, on a $\mathcal{O}.\mathsf{Dec}$ request, the oracle answers

`invalid` if the ciphertext decrypts to one of the two messages used in the inputs of the challenge call. This includes ciphertexts that are not updates of the challenge ciphertext and that do not come from a game oracle. In particular, this definition of RCCA security holds even for UE schemes with randomized updates. In the RCCA setting, we make the following additions to the CCA oracle definitions:

1. For $\mathcal{O}.\mathsf{Chall}$, create a variable $\overline{\mathcal{M}}$ and assign it:
   (a) the set $\{\bar{M}_0, \bar{M}_1\}$ in Fig. 1.7 of the IND-ENC game.
   (b) the set $\{\mathsf{UE.Dec}(k_{\mathsf{e}-1}, \bar{C}_0), \mathsf{UE.Dec}(k_{\mathsf{e}-1}, \bar{C}_1)\}$ in Fig. 1.8 of the IND-UPD game.
   (c) the set $\{\bar{M}, \mathsf{UE.Dec}(k_{\mathsf{e}-1}, \bar{C})\}$ in Fig. 1.6 of the IND-UE game.
2. Define a predicate called IsChall that takes as inputs a key k and a ciphertext C and that is true iff $\mathsf{UE.Dec}(k, C) \in \overline{\mathcal{M}}$.
3. For $\mathcal{O}.\mathsf{Dec}(C)$, if $\mathsf{IsChall}(k_{\mathsf{e}}, C)$ is false return $\mathsf{UE.Dec}(k_{\mathsf{e}}, C)$, else return `invalid`.
4. For $\mathcal{O}.\mathsf{Upd}(C_{\mathsf{e}-1})$, we give a full description of this oracle in the randomized update setting in Fig. 4.1.

$\mathcal{O}.\mathsf{Upd}(C_{\mathsf{e}-1})$

1. $C_{\mathsf{e}} \leftarrow \mathsf{UE.Upd}(\Delta_{\mathsf{e}}, C_{\mathsf{e}-1})$
2. **if** $(j, C_{\mathsf{e}-1}, \mathsf{e}-1) \in \mathcal{L}$
3. $\quad \mathcal{L} \leftarrow \mathcal{L} \cup \{(j, C_{\mathsf{e}}, \mathsf{e})\}$
4. **else if** $\mathsf{IsChall}(k_{\mathsf{e}-1}, C_{\mathsf{e}-1})$
5. $\quad \mathcal{C} \leftarrow \mathcal{C} \cup \{\mathsf{e}\}$
6. **return** C_{e}

Figure 4.1: The $\mathcal{O}.\mathsf{Upd}$ oracle used in the RCCA setting for UE with randomized updates.

4.2 The Interactive Server Memory Resource

In [BM18], Maurer and Badertscher propose an instantiation in CC of the client-server setting for non-interactive protocols. They notably introduce the basic **SMR**, where an honest client can read and write data on an outsourced memory owned by a potentially dishonest server. We extend their work to interactive protocols in the client-server setting by introducing a different type of **SMR**, namely the Interactive Server-Memory Resource (**ISMR** for short). By interactive, we mean any protocol where the client sends a query and expects that the server will perform a given computation upon reception. For example, in UE, the server is expected to use the $\mathsf{Upd}(\Delta_{\mathsf{e}}, \cdot)$ algorithm on each ciphertext on an update request to epoch e.

One of the main differences with **SMR** is that, in **ISMR**, the server is now considered as semi-honest (through its interface S), since it has to participate in the interactive protocol when receiving orders from the client. In [BM18], the client was the only party to take part in the protocol, the role of the server being limited to storing data. In our case, the server agrees to apply a converter which carries out the computations requested by the client. We gather these capabilities in a sub-interface of S denoted by S.1. Since the server is only semi-honest, we have no guarantees about how it will use its remaining capabilities. We gather those under the sub-interface S.2.

In **ISMR**, we model interactions by letting the client interact with the server, through its interface C, according to a boolean value NeedInteraction. The server, through its sub-interface S.1, is given the capability of checking this boolean to see if it needs to take actions, act

if necessary and then switch the boolean value back. While **ISMR** is waiting for an interaction to end, *i.e.* as long as NeedInteraction is set to `true`, the capabilities of the client at its interface are disabled. Our approach is general enough for **ISMR** to be used when modeling all kinds of interactive protocols in the outsourced storage setting. We can model interactive protocols that are information-theoretically, statistically or computationally secure, multi-client and multi-server protocols as well as passive or active adversaries. We showcase this by modeling UE schemes in Section 4.3 and PIR schemes in Section 4.4. A formal description of **ISMR** is given in Fig. 4.2. We are able to model all these different protocols by taking advantage of the power and the flexibility of the CC framework together with our new **ISMR**. A key point of our work is to propose new ways to use converters and simulators in CC.

Resource ISMR$_{\Sigma,n}$

Initialization Initialization

NeedInteraction $\leftarrow$ `false`, $\mathbb{M} \leftarrow [\,]$

Interface C

Input: $(\texttt{read}, i) \in [n]$
  if not NeedInteraction
    return $\mathbb{M}[i]$

Input: $(\texttt{write}, i, x) \in [n] \times \Sigma$
  if not NeedInteraction
    $\mathbb{M}[i] \leftarrow x$

Input: `askInteraction`
  if not NeedInteraction
    NeedInteraction $\leftarrow$ `true`

Input: `getStatus`
  return needInteraction

Sub-Interface S.1 of **Interface** S

Input: $(\texttt{read}, i) \in [n]$
  return $\mathbb{M}[i]$

Input: $(\texttt{write}, i, x) \in [n] \times \Sigma$
  $\mathbb{M}[i] \leftarrow x$

Input: `interact`
  if needInteraction
    needInteraction $\leftarrow$ `false`

Sub-Interface S.2 of **Interface** S

Input: $(\texttt{leak}, i) \in [n]$ // other adversarial capabilities can be added
  return $\mathbb{M}[i]$

Input: `getStatus`
  return needInteraction

Figure 4.2: The interactive Server-Memory Resource with finite alphabet Σ and size n. The sub-interface S.1 guarantees that the server follows a protocol through the application of a converter. On the contrary, no guarantees are given at the sub-interface S.2.

First, we show how one can use a converter to modify the number and/or the types of arguments required when sending a query. For example, in PIR, a client wants to retrieve the i-th entry of a database without revealing i to the server. Thus, the client will not send i directly

to the server. Instead, the client will perform a computation involving i and some secret and send the result q to the server where q belongs to some set S. The real resource must account for this by implementing a query $(\texttt{query}, q) \in S$ at the client's interface. The issue is that the final goal of the protocol, modeled in the ideal resource, is to retrieve the i-th entry of a database of size n, which needs to be modeled as a query $(\texttt{query}, i) \in [1, n]$. We bridge the gap between the real and ideal resources by allowing the client's converter (used in the real-world) to reprogram the $(\texttt{query}, q) \in S$ query into a $(\texttt{query}, i) \in [1, n]$ one. Moreover, we also allow the converter to increase or decrease the number of arguments of a query. See Section 4.3 for more details.

Secondly, we show how one can use a converter to disable a capability at an interface. This is particularly useful when dealing with semi-honest adversaries. Indeed, such an adversary participates in the protocol by carrying computations for the client. This is modeled by the application of a converter. For example, in UE schemes, the server is expected to update ciphertexts for the client. To carry out its computation, the server must be able to retrieve the update token sent by the client. This is modeled by a $(\texttt{fetchToken}, e)$ query. In UE schemes, this operation is not considered to be malicious. Thus, we add a $(\texttt{leakToken}, e)$ query to model a malicious access to the update token (with a different behavior than the $(\texttt{fetchToken}, e)$ query). Then, we need to disable the $(\texttt{fetchToken}, e)$ query to prevent an adversary from accessing the update token without querying $(\texttt{leakToken}, e)$. Since our modeling of interactivity allows us to use converters on semi-honest interfaces, we can use a converter to disable the $(\texttt{fetchToken}, e)$ query at the outside interface of the server. See Section 4.4 for more details.

To build an **ISMR** with stronger security guarantees, we will use the construction notion of [BM18] given in Fig. 1.2. One difference with the work of [BM18] is that, although there is a protocol plugged in interface S, this interface is only semi-honest and doesn't belong to the so-called "honest parties". We thus need to plug a simulator at this interface in the ideal world if we hope to achieve any meaningful construction, as illustrated in Fig. 4.3. One way to see this is to consider the `read` capabilities at interfaces C and S when an encryption scheme, such as UE, is used. Since the client holds the decryption key, the interface C is able to retrieve the plaintexts corresponding to the ciphertexts stored in memory. This is not true for the interface S since it does not have access to the key under normal circumstances. Since there is no encryption in the ideal world, we need a simulator to simulate ciphertexts when S sends `read` requests at its interface, as otherwise distinguishing between the two worlds would be trivial.

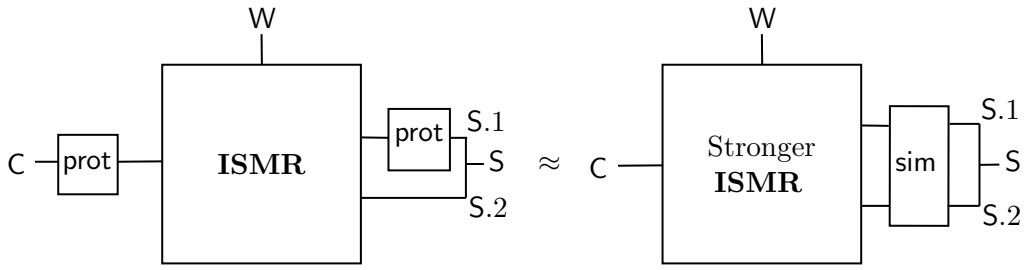

Figure 4.3: The construction notion for **ISMR**. On the left, the plain **ISMR** equipped with a protocol. On the right, the stronger **ISMR** equipped with a simulator. The construction is deemed secure if there exists a simulator that makes the two systems indistinguishable.

For the sake of clarity, **ISMR** will be renamed **USMR** (for Updatable Server Memory-Resource) when modeling UE schemes, in Section 4.3. When modeling PIR schemes, **ISMR** will be renamed **DB** (for Database), see Section 4.4. We will also rename NeedInteraction, `askInteraction` and `interact`.

4.3 A composable treatment of Updatable Encryption

4.3.1 Instantiation of the interactive SMR to Updatable Encryption

We recall that **ISMR** is renamed **USMR** in this section. Moreover, at interface S, we also distinguish between honestly reading the memory - through `read` requests at sub-interface $\mathsf{S}.1$ - to update a ciphertext without trying to use this information against the client; and maliciously reading (we prefer to say *leaking*) the memory - through `leak` requests at sub-interface $\mathsf{S}.2$ - to gain information and try to break the confidentiality guarantees of the client. Sending a $(\texttt{leak}, i)$ request triggers the event $\mathcal{E}^{\texttt{leaked}}_{\mathsf{Data},\mathsf{e},\ell_i,i}$ where e is the number of server updates and ℓ_i is the number of times entry i leaked since the latest update. A full description of **USMR** is given in Fig. 4.4.

4.3.2 The updatable key resource

We want to apply a UE scheme to strengthened the client's security guarantees of **USMR**. To do so, we need to model the use of cryptographic keys and update tokens needed by UE schemes. This is why we introduce an Updatable Key Resource, called **UpdKey**, whose role is to model the existence, the operations and the availability of keys as well as update tokens. In the following, let $\mathcal{K}$ be the key space of UE schemes. Given k and k' two keys in $\mathcal{K}$, the notation $\Delta \leftarrow \mathcal{T}(k, k')$ denotes the assignation, to the variable Δ, of a token that updates ciphertexts encrypted under the key k to ones encrypted under k'.

In **UpdKey**, the `fetchToken` request is always accessible at sub-interface $\mathsf{S}.1$, since the protocol used at this interface will prevent the information it provides to be maliciously used, whereas the request `leakToken` at interface $\mathsf{S}.2$ requires that a special event $\mathcal{E}^{\texttt{leaked}}_{\mathsf{Token},i}$ has been triggered before returning the update token to epoch i, which can be used for malicious purposes. These events are triggered by the environment which, in CC, can be given an interface that is usually denoted by W for world interface.

This separation between `read`/`leak` in **USMR** and `fetchToken`/`leakToken` requests in **UpdKey** is important because it allows us to describe the security guarantees of the system more precisely. Indeed, since the server needs to retrieve the token to update the ciphertexts, if we consider that this token "leaked", it becomes impossible to express the post-compromise security guarantees brought by UE schemes. This is because if all tokens leak, a single key exposure compromises the confidentiality of ciphertexts for all subsequent epochs.

4.3.3 An Updatable Encryption protocol

We have to define an updatable encryption protocol for **USMR** (and **UpdKey**) and study its effects. Since we work with CC, we describe our protocol as a pair of converters $(\mathsf{ue}_{\mathsf{cli}}, \mathsf{ue}_{\mathsf{ser}})$ where $\mathsf{ue}_{\mathsf{cli}}$ will be plugged in interface C and $\mathsf{ue}_{\mathsf{ser}}$ in sub-interface $\mathsf{S}.1$ of **USMR** and **UpdKey**. A formal description of $\mathsf{ue}_{\mathsf{cli}}$ (resp. $\mathsf{ue}_{\mathsf{ser}}$) can be found in Fig. 4.6 (resp. Fig. 4.7).

4.3.4 The confidential and updatable SMR

The security guarantees of **USMR** can be improved by requiring confidentiality for the client's data. The resulting resource is called confidential **USMR** and we will refer to it as **cUSMR**. In practice, this means that, on a $(\texttt{leak}, i)$ request at interface S, only the length of $\mathbb{M}[i]$ is returned to the adversary and not the i-th entry itself. The `read` and `write` capabilities of sub-interface $\mathsf{S}.1$ are removed. The resource **cUSMR** is described in Fig. 4.8.

We will show that the $\mathsf{IND\text{-}UE}$ security notion of Boyd *et al.* [BDGJ20] is not always better than the $\mathsf{IND\text{-}ENC} + \mathsf{IND\text{-}UPD}$ notion of [LT18], in that it hides the age of ciphertexts. By

Resource $\mathrm{USMR}_{\Sigma,n}$

Initialization

NeedUpdate $\leftarrow$ `false`, $\mathsf{e} \leftarrow 0$

$\mathbb{M} \leftarrow [\,]$, $\forall 1 \leq i \leq n$, $\ell_i \leftarrow 0$

Interface C

Input: $(\mathtt{read}, i) \in [n]$

  if not NeedUpdate

    return $\mathbb{M}[i]$

Input: $(\mathtt{write}, i, x) \in [n] \times \Sigma$

  if not NeedUpdate

    $\mathbb{M}[i] \leftarrow x$

Input: `askUpdate`

  if not NeedUpdate

    NeedUpdate $\leftarrow$ `true`

Input: `getStatus`

  return needUpdate

Sub-Interface $\mathsf{S}.1$ of **Interface** S

Input: $(\mathtt{read}, i) \in [n]$

  return $\mathbb{M}[i]$

Input: $(\mathtt{write}, i, x) \in [n] \times \Sigma$

  $\mathbb{M}[i] \leftarrow x$

Input: `update`

  if needUpdate

    for $i = 1$ to n **do**

      $\ell_i \leftarrow 0$

    $\mathsf{e} \leftarrow \mathsf{e} + 1$

    needUpdate $\leftarrow$ `false`

Sub-Interface $\mathsf{S}.2$ of **Interface** S

Input: $(\mathtt{leak}, i) \in [n]$

  $\ell_i \leftarrow \ell_i + 1$

  $\mathcal{E} \xleftarrow{+} \mathcal{E}^{\mathsf{leaked}}_{\mathsf{Data},\mathsf{e},\ell_i,i}$

  return $\mathbb{M}[i]$

Input: `getStatus`

  return needUpdate

Figure 4.4: The **ISMR** viewed as an updatable server-memory resource **USMR** with finite alphabet Σ and memory size n. Interface S guarantees that it will endorse an honest behavior, through the application of a converter, at its sub-interface $\mathsf{S}.1$. However, no such guarantees are offered at its sub-interface $\mathsf{S}.2$.

Resource UpdKey

Initialization

$\mathsf{e} \leftarrow 1, \mathbb{K} \leftarrow [\,], \mathbb{T} \leftarrow [\perp]$
$k \xleftarrow{\$} \mathcal{K}, \mathbb{K} \leftarrow \mathbb{K} \,||\, k$
$\mathcal{E} \xleftarrow{+} \mathcal{E}_{\mathsf{e}}^{\texttt{epoch}}$

Interface C

Input: `fetchKey`
return $\mathbb{K}[\mathsf{e}]$

Input: `nextEpoch`
$k_{\mathsf{e}+1} \xleftarrow{\$} \mathcal{K}$
$\mathbb{K} \leftarrow \mathbb{K} \,||\, k_{\mathsf{e}+1}$
$\Delta_{\mathsf{e}+1} \leftarrow \mathcal{T}(k_{\mathsf{e}}, k_{\mathsf{e}+1})$
$\mathbb{T} \leftarrow \mathbb{T} \,||\, \Delta_{\mathsf{e}+1}$
$\mathsf{e} \leftarrow \mathsf{e} + 1$
$\mathcal{E} \xleftarrow{+} \mathcal{E}_{\mathsf{e}}^{\texttt{epoch}}$
return $k_{\mathsf{e}+1}$

Sub-Interface Sub-Interface S.1 of **Interface** S

Input: $(\texttt{fetchToken}, i)$
if $2 \leq i \leq \mathsf{e}$
return $\mathbb{T}[i]$
else
return $\perp$

Sub-Interface Sub-Interface S.2 of **Interface** S

Input: $(\texttt{leakKey}, i)$
if $i \leq \mathsf{e}$ **and** $\mathcal{E}_{\mathsf{Key},i}^{\texttt{leaked}}$
return $\mathbb{K}[i]$
else
return $\perp$

Input: $(\texttt{leakToken}, i)$
if $2 \leq i \leq \mathsf{e}$ **and** $\mathcal{E}_{\mathsf{Token},i}^{\texttt{leaked}}$
return $\mathbb{T}[i]$
else
return $\perp$

Figure 4.5: The updatable key (with its associated token) resource **UpdKey**. For interface S, we use the same distinction between its sub-interfaces S.1 and S.2 as in the **USMR** description.

Converter $\mathsf{ue}_{\mathsf{cli}}$

Initialization

$k \leftarrow$ `fetchKey` at interface C of **UpdKey**

Interface `out`

Input: $(\texttt{read}, i) \in [n]$

$c \leftarrow (\texttt{read}, i)$ at interface C of **USMR**

if $c \neq \perp$

$m \leftarrow \mathsf{Dec}_k(c)$

if $m \neq \perp$

return m

Input: $(\texttt{write}, i, x) \in [n] \times \Sigma$

$c \leftarrow \mathsf{Enc}_k(x)$

Send $(\texttt{write}, i, c)$ at interface C of **USMR**

Input: `askUpdate`

$u \leftarrow$ `getStatus` at interface C of **USMR**

if not u

$k \leftarrow$ `nextEpoch` at interface C of **UpdKey**

Send `askUpdate` at interface C of **USMR**

Figure 4.6: The client's converter $\mathsf{ue}_{\mathsf{cli}}$ for UE scheme $(\mathsf{KeyGen}, \mathsf{TokenGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Upd})$ with decryption error symbol $\perp$.

Converter $\mathsf{ue}_{\mathsf{ser}}$

Initialization

$\Delta \leftarrow \perp$, $\mathsf{e} \leftarrow 1$

Interface `out`

Input: `update`

$u \leftarrow$ `getStatus` at interface S of **USMR**

if u

$\mathsf{e} \leftarrow \mathsf{e} + 1$

$\Delta \leftarrow (\texttt{fetchToken}, \mathsf{e})$ at interface S of **UpdKey**

for $i = 1 \ldots n$ **do**

$c \leftarrow (\texttt{read}, i)$ at interface S of **USMR**

Send $(\texttt{write}, i, \mathsf{Upd}_\Delta(c))$ at interface S of **USMR**

Send `update` at interface S of **USMR**

Input: $(\texttt{read}, i)$, $(\texttt{write}, i, x)$, or $(\texttt{fetchToken}, \mathsf{e})$

return $\perp$

Figure 4.7: The server's converter $\mathsf{ue}_{\mathsf{ser}}$ for UE scheme $(\mathsf{KeyGen}, \mathsf{TokenGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Upd})$. Since the server is semi-honest, the converter monitors the behavior of the requests at sub-interface $\mathsf{S}.1$. The server guarantees that updates are done correctly (using the UE scheme) through the `update` request, and that the requests `read`, `write` and `fetchToken` are only used to update the ciphertexts and not to gain information to break the confidentiality of the data. These requests are thus disabled at its interface `out` but they can still be used internally by the converter.

Resource cUSMR

Sub-Interface S.1 of **Interface** S

Input: `update`
  if NeedUpdate
    NeedUpdate $\rightarrow$ `false`

Sub-Interface S.2 of **Interface** S

Input: $(\texttt{leak}, i) \in [n]$
  return $|\mathbb{M}[i]|$

Input: `getStatus`
  return NeedUpdate

Figure 4.8: The confidential and updatable server-memory resource **cUSMR**. Only differences with **USMR** are shown.

age, we mean the last epoch in which the ciphertext was freshly written to the database. We show that this is only true when ciphertexts can leak at most one time per epoch. Indeed, if a ciphertext can leak at least two times per epoch, the adversary can use its first (resp. second) leak at the start (resp. end) of each epoch. If the ciphertext has changed between the two leaks, it must have been rewritten during this epoch and its age is now 0. If it has not changed, then its age is incremented. We see that in this setting, the age of ciphertexts cannot be protected.

In the rest of this work, we will distinguish between resources that only allow one leak per ciphertext per epoch, denoted by a 1 in the exponent (*e.g.* $\mathbf{USMR}^1$), and resources that allow any number of leaks per ciphertext per epoch, denoted by a + in the exponent (*e.g.* $\mathbf{USMR}^+$). If the number of leaks does not matter we will omit the exponent. We give a detailed description of both resources in the following paragraphs.

The USMR with unrestricted leakage. In this paragraph, we consider an adversary that keeps track of all the client's actions inside a log file Hist. More precisely, on each $(\texttt{read}, i)$, $(\texttt{write}, i, x)$ or `askUpdate` queries from the client, for some i and x, the adversary appends the corresponding entry to its log file Hist. The adversary can access Hist using the `getHist` query at interface S.2. These capabilities are formally given in $\mathbf{USMR}^+$ of Fig. 4.9.

The USMR with restricted leakage. In $\mathbf{USMR}^1$, we consider an adversary that is only able to leak database entries once per epoch. We enforce this by using the leakage counters ℓ_i. Because of this restriction, the adversary is no longer able to maintain a log file of all client's actions like in $\mathbf{USMR}^+$. The $\mathbf{USMR}^1$ is formally presented in Fig. 4.10.

We use this restriction to model the protection of the age of database entries. Indeed, with only one leak per entry per epoch, an adversary cannot tell apart fresh entries from old updated ones. One may wonder why we do not simply track the age of data entries with counters in the real resource and hide their value in the ideal one. Unfortunately, we cannot provably construct this ideal resource in a composable manner. Indeed, during the proof, the adversary controls every interface of the resources. This means that he knows the age of every database entry in both the real and ideal worlds (since he issues all the write and update requests). The age being hidden in the ideal world, we have to find a simulator that can emulate the exact age of every database entries at any time if we want to prevent the adversary from distinguishing both worlds. Of course, this is impossible and, in order to still produce meaningful composable

Resource $\mathbf{USMR}^{+}_{\Sigma,n}$

Initialization

HIST $\leftarrow [\,]$

Interface C

Input: $(\texttt{read}, i) \in [n]$
  if not NEEDUPDATE
    HIST $\leftarrow$ HIST$\|(\texttt{read}, i)$
    return $\mathbb{M}[i]$

Input: $(\texttt{write}, i, x) \in [n] \times \Sigma$
  if not NEEDUPDATE
    HIST $\leftarrow$ HIST$\|(\texttt{write}, i, x)$
    $\mathbb{M}[i] \leftarrow x$

Input: `askUpdate`
  if not NEEDUPDATE
    HIST $\leftarrow$ HIST$\|(\texttt{askUpdate})$
    NEEDUPDATE $\leftarrow$ `true`

Sub-Interface S.2 of **Interface** S

Input: `getHist`
  return HIST

Figure 4.9: The $\mathbf{USMR}^{+}$ where the adversary maintains a log file of all client's actions (only the differences with **USMR** of Fig. 4.4 are shown).

statements for UE, we choose to introduce the leakage contexts of $\mathbf{USMR}^{+}$ and $\mathbf{USMR}^{1}$ even though they are unusual in the composable setting.

4.3.5 Handling post-compromise security guarantees

The goal of this section is to give an exact description of the post-compromise security guarantees given by UE schemes. Said differently, we want to explain how the security guarantees evolve after a key exposure. When dealing with situations such as key exposures, composable frameworks usually stumble on an impossibility result called the *commitment problem*. This problem is the following: given a message m, how can an online simulator explain a simulated ciphertext c, generated without knowledge of m, with a key k such that c decrypts to m under this key. Thanks to a recent work of Jost *et al.* [JM20], the CC framework is well equipped to deal with this impossibility result. This is done through the use of interval-wise security guarantees. In CC, the interval-wise relaxation describes security guarantees within an interval delimited by predicates on the global event history. For example, we can describe security guarantees before and after the key leaks to circumvent the commitment problem.

In UE schemes, it is clear that the confidentiality of the user data is lost when an epoch key leaks. This security loss remains in subsequent epochs if the keys continue to leak or if successive update tokens leak. However, as soon as we encounter an epoch where neither the key nor the update token leaks, the confidentiality is restored. This remains true until a future epoch, where either a key leaks or consecutive update tokens leak until a key is finally exposed. This is due to the fact that ciphertexts can be upgraded and downgraded with update tokens to an epoch where a key is exposed.

Resource $\mathbf{USMR}^1_{\Sigma,n}$

Interface C

Input: $(\texttt{read}, i) \in [n]$
  if not NeedUpdate
    return $\mathbb{M}[i]$

Input: $(\texttt{write}, i, x) \in [n] \times \Sigma$
  if not NeedUpdate
    $\mathbb{M}[i] \leftarrow x$

Input: `askUpdate`
  if not NeedUpdate
    NeedUpdate $\leftarrow$ `true`

Sub-Interface S.2 of **Interface** S

Input: $(\texttt{leak}, i) \in [n]$
  $\ell_i \leftarrow \ell_i + 1$
  if $\ell_i = 1$
    $\mathcal{E} \xleftarrow{+} \mathcal{E}^{\texttt{leaked}}_{\mathsf{Data},\mathsf{e},\ell_i,i}$
    return $\mathbb{M}[i]$

Figure 4.10: The $\mathbf{USMR}^1$ where the adversary can only leak entries of the memory once per epoch (only differences with $\mathbf{USMR}^+$ of Fig. 4.9 are shown).

In the epoch timeline, the areas where confidentiality is preserved are called *insulated regions*. They have been studied and used in previous works [LT18, KLR19, BDGJ20, Jia20]. We describe those regions with their extreme left and right epochs. These pairs of epochs are called *firewalls*. We recall the definition used in [BDGJ20].

Definition 4.1. *An insulated region with firewalls* fwl *and* fwr *is a consecutive sequence of epochs* $(\mathsf{fwl}, \ldots, \mathsf{fwr})$ *for which:*

1. *No key in the sequence of epochs* $(\mathsf{fwl}, \ldots, \mathsf{fwr})$ *is corrupted.*
2. *The tokens* Δ_{fwl} *and* $\Delta_{\mathsf{fwr}+1}$ *are not corrupted, if they exist.*
3. *All tokens* $(\Delta_{\mathsf{fwl}+1}, \ldots, \Delta_{\mathsf{fwr}})$ *are corrupted.*

The set of all firewall pairs is denoted by $\mathcal{FW}$*. The set of all insulated regions is denoted by* $\mathcal{IR} := \cup_{(\mathsf{fwl},\mathsf{fwr})\in\mathcal{FW}}\{\mathsf{fwl}, \ldots, \mathsf{fwr}\}$.

The epochs where the confidentiality guarantees do not hold are the ones not found in $\mathcal{IR}$. The set of firewalls $\mathcal{FW}$ can easily be described using predicates on the global event history. This is done in the following manner.

$$\begin{aligned}\mathcal{FW} := \{(\mathsf{fwl}, \mathsf{fwr}) \mid & \mathsf{fwl} \leq \mathsf{fwr}, \\ & \forall \mathsf{e} \in \{\mathsf{fwl}, \ldots, \mathsf{fwr}\}, \neg\mathcal{E}^{\texttt{leaked}}_{\mathsf{Key},\mathsf{e}}, \\ & \neg\mathcal{E}^{\texttt{leaked}}_{\mathsf{Token},\mathsf{fwl}} \text{ and } \neg\mathcal{E}^{\texttt{leaked}}_{\mathsf{Token},\mathsf{fwr}+1}, \\ & \forall \mathsf{e} \in \{\mathsf{fwl}+1, \ldots, \mathsf{fwr}\}, \mathcal{E}^{\texttt{leaked}}_{\mathsf{Token},\mathsf{e}}\}\end{aligned}$$

Remark 11. In 2020, in his "The Direction of Updatable Encryption does not Matter Much" paper, Jiang [Jia20] initiated the study of the directions of key updates for UE schemes. Recall

that the update token $\Delta_{\mathsf{e}+1}$ is computed by the $\mathsf{TokenGen}$ algorithm using the old key k_{e} and the new key $k_{\mathsf{e}+1}$. Jiang studied these two types of UE schemes (the terminology was later introduced by Nishimaki [Nis22] in 2022):

1. In UE with *forward-leak uni-directional key updates*, given the new key $k_{\mathsf{e}+1}$ and the token $\Delta_{\mathsf{e}+1}$, the adversary cannot infer the old key k_{e}. However, given k_{e} and $\Delta_{\mathsf{e}+1}$, the adversary can infer $k_{\mathsf{e}+1}$.
2. In UE with *bi-directional key updates*, given either the old key or the new key and the update token, the adversary can recover the other key.

Jiang showed that the security notions of these two settings were equivalent, and concluded that the directions of updates did not matter much. At the time, all UE schemes belonged to one of these two settings. However, in is 2022 paper "The Direction of Updatable Encryption Does Matter", Nishimaki [Nis22] introduced a third setting:

1. In the *backward-leak uni-directional key updates* setting, given the old key k_{e} and the token $\Delta_{\mathsf{e}+1}$, the adversary cannot infer $k_{\mathsf{e}+1}$. However, given $k_{\mathsf{e}+1}$ and $\Delta_{\mathsf{e}+1}$, the adversary can infer k_{e}.

Nishimaki showed that UE schemes with backward-leak uni-directional key updates have strictly stronger security that those without. The intuition for this is that, in UE, we need to protect the latest key since the reason why we update keys is that the current one might have leaked. When the adversary can learn the new key from the token and old key, we must protect older keys to prevent newer keys from leaking *even if older ciphertexts are deleted*. In the setting of [Nis22], we only need to protect the new key if old ciphertexts are properly deleted. Nishimaki also proposed RtR, the first UE scheme with backward-leak uni-directional key updates. The UE schemes we presented in Chapter 2 all feature bi-directional key updates.

The work presented in this chapter was done right after Jiang's paper but before Nishimaki's. Thus, we only consider UE schemes in the bi-directional key updates setting. We could include Nishimaki's findings by using its backward-leak uni-directional definitions for leakage sets [Nis22, Def. 3.4, 3.5, 3.6 & 4.1], confidentiality games [Nis22, Def. 3.7], firewalls [Nis22, Def. 4.2] and insulated regions [Nis22, Def. 4.3].

We say that the i-th entry of a database $\mathbb{M}$ is compromised if an adversary gets an encryption of $\mathbb{M}[i]$ in an epoch e that does not belong to an insulated region. Formally, we introduce the predicate

$$\begin{aligned} P_{\mathtt{compromised},i}(\mathcal{E}) := \exists \mathsf{e} : \mathcal{E}_{\mathsf{e}}^{\mathtt{epoch}} \prec \mathcal{E}_{\mathsf{Data},\mathsf{e},\cdot,i}^{\mathtt{leaked}} \prec \mathcal{E}_{\mathsf{e}+1}^{\mathtt{epoch}} \quad \wedge \\ \forall (\mathsf{fwl},\mathsf{fwr}) \in \mathcal{FW}, \neg(\mathcal{E}_{\mathsf{fwl}}^{\mathtt{epoch}} \prec \mathcal{E}_{\mathsf{e}}^{\mathtt{epoch}} \prec \mathcal{E}_{\mathsf{fwr}+1}^{\mathtt{epoch}}) \end{aligned}$$

The right side of the conjunction means 'the epoch e does not belong to an insulated region'. Recall that the event $\mathcal{E}_{\mathsf{e}}^{\mathtt{epoch}}$ is triggered by **UpdKey** on a `nextEpoch` request and the event $\mathcal{E}_{\mathsf{Data},\mathsf{e},\cdot,i}^{\mathtt{leaked}}$ is triggered by **USMR** on a $(\mathtt{leak}, i)$ request.

Then, we introduce the event $\mathcal{E}_j^{\mathtt{insec}}$ which indicates that the j-th entry of the database is not confidential. This event can only be triggered by the environment. Now, we can modify our definition of **cUSMR** in the following way: on a $(\mathtt{leak}, j)$ request, this resource now returns $\mathbb{M}[j]$ if $\mathcal{E}_j^{\mathtt{insec}}$ has been triggered and $|\mathbb{M}[j]|$ otherwise. For $1 \leq i \leq n$, we introduce the predicate

$$P_{\mathtt{only},i}(\mathcal{E}) := \bigwedge_{j \in \{1,\ldots,n\} \setminus \{i\}} \mathcal{E}_j^{\mathtt{insec}}$$

It formalizes that we do not consider the confidentiality of plaintexts other than the i-th one.

Following the notation of [JM20], we introduce our main theorem. Let n be the number of entries stored in the server. Our construction is an intersection of n specifications. For $i \in \{1, \ldots, n\}$, we assume that an adversary knows every entry of the database except the i-th one. Then, the i-th specification guarantees the confidentiality of the i-th entry until it trivially leaks because an adversary gained access to an encryption of this plaintext under an exposed epoch key.

Theorem 4.1. *Let Σ be a finite alphabet and $n \in \mathbb{N}$. There exists a sequence of simulators $(\sigma_i)_{1 \leq i \leq n}$ such that the protocol $\pi_{\mathsf{UE}} := (\mathsf{ue}_{\mathsf{cli}}, \mathsf{ue}_{\mathsf{ser}})$, described in Fig. 4.6 and Fig. 4.7, based on an* IND-UE-CPA *secure UE scheme constructs the* $\mathbf{cUSMR}^1_{\Sigma,n}$ *from the* $\mathbf{USMR}^1_{\Sigma,n}$ *and* **UpdKey**

$$[\mathbf{USMR}^1_n, \mathbf{UpdKey}] \xrightarrow{\pi_{\mathsf{UE}}} \bigcap_{1 \leq i \leq n} (\sigma_i \, \mathbf{cUSMR}^1_n)^{[P_{\mathtt{only},i}(\mathcal{E}), P_{\mathtt{compromised},i}(\mathcal{E})] : \epsilon_{\mathsf{CPA}}}$$

where ϵ_{CPA} denotes our reduction, given in Theorem 4.2, from distinguishing between the $\mathbf{cUSMR}^1$ *with our simulator and the* $\mathbf{USMR}^1$ *with our protocol, to winning the* IND-UE-CPA *game.*

In the above theorem, we can replace ($\mathbf{USMR}^1$, $\mathbf{cUSMR}^1$, ϵ_{CPA}, IND-UE-CPA) with ($\mathbf{USMR}^+$, $\mathbf{cUSMR}^+$, $\epsilon_{\mathsf{CPA}^+}$, IND-ENC-CPA + IND-UPD-CPA) when we deal with unrestricted leakage. In the context of adversarial injections in the database, we can replace it by ($\mathbf{iUSMR}^1$, $\mathbf{ciUSMR}^1$, ϵ_{RCCA}, IND-UE-RCCA) in the restricted leakage setting and by ($\mathbf{iUSMR}^+$, $\mathbf{ciUSMR}^+$, $\epsilon_{\mathsf{RCCA}^+}$, IND-ENC-RCCA + IND-UPD-RCCA) when we deal with unrestricted leakage, where the **USMR** with injections **iUSMR** is described in Fig. 4.11. This includes all our results.

Proof. Since we consider an intersection of n specification, we need to prove n constructions. For $i \in \{1, \ldots, n\}$, we need to prove that there exists a simulator σ_i such that, in the interval $[P_{\mathtt{only},i}(\mathcal{E}), P_{\mathtt{compromised},i}(\mathcal{E})]$, the protocol π_{UE} constructs the $\mathbf{cUSMR}^1_{\Sigma,n}$ from the $\mathbf{USMR}^1_{\Sigma,n}$ and **UpdKey** with respect to σ_i. This construction is formalized and proven in Theorem 4.2 of Section 4.3.6 where we give a detailed reduction from breaking our construction to winning the IND-UE-CPA game.

When replacing ($\mathbf{USMR}^1$, $\mathbf{cUSMR}^1$, ϵ_{CPA}, IND-UE-CPA) with ($\mathbf{USMR}^+$, $\mathbf{cUSMR}^+$, $\epsilon_{\mathsf{CPA}^+}$, IND-ENC-CPA + IND-UPD-CPA) in the above theorem, we need two prove n more constructions. We formalize and prove these constructions in Theorem 4.4 of Section 4.3.6 where we give a detailed reduction from breaking our construction to winning the IND-ENC-CPA + IND-UPD-CPA games. We do the same in the context of adversarial injections in Theorem 4.5 of Section 4.3.6. □

4.3.6 Exact security of Updatable Encryption schemes

At most one leak per entry per epoch: the CPA case

In this section, we work with a $\mathbf{USMR}^1$ of size n where the attacker can leak entries of the database at its interface S. This capability allows the distinguisher (which is connected to every interface of the system) to build an encryption oracle. Indeed, the distinguisher can use the client's interface C to write messages of its choice into the database, and then leak the ciphertexts associated to these messages by sending a `leak` request at interface S. This fact motivates the use of a CPA security notion since it is tailored to bring security in the presence of such an encryption oracle.

The simulator $\sigma_{k,\mathsf{CPA}}$. Since we are trying to prove our Theorem 4.1, we place ourselves in the context of this theorem: we take $k \in \{1,\dots,n\}$ and we place ourselves in the interval $[P_{\mathtt{only},k(\mathcal{E})}, P_{\mathtt{compromised},k(\mathcal{E})}]$ where we do not consider the confidentiality of plaintexts other than the k-th one. In this case, the simulator $\sigma_{k,\mathsf{CPA}}$ works as follows. It simulates the epoch keys and tokens and, on a $(\mathtt{leakKey}, i)$ or $(\mathtt{leakToken}, i)$ request, it checks if the event $\mathcal{E}^{\mathtt{leaked}}_{\mathsf{Key},i}$ (respectively $\mathcal{E}^{\mathtt{leaked}}_{\mathsf{Token},i}$) exists in the Global Event History, and leaks the corresponding epoch key (respectively token) to the adversary if it is the case, and $\perp$ otherwise. On a $(\mathtt{leak}, k)$ request, the simulator forwards it to the ideal resource to get a length ℓ and returns a fresh encryption of a random plaintext of length ℓ under the current epoch key. Finally, on a $(\mathtt{leak}, i)$ request, $i \neq k$, the simulator forwards it to the ideal resource to get a plaintext x and returns a fresh encryption of x under the current epoch key.

IND-UE-CPA security is sufficient for a secure construction of cUSMR that hides the age

The fact that IND-UE-CPA is sufficient to construct $\mathbf{cUSMR}^1$ from $\mathbf{USMR}^1$ and $\mathbf{UpdKey}$ is expressed in Theorem 4.1 through an intersection of specifications. The following theorem shows how we construct each of those specifications.

Theorem 4.2. *Let Σ be a finite alphabet, $n \in \mathbb{N}$ and $k \in \{1,\dots,n\}$. The protocol $\mathsf{ue} := (\mathsf{ue}_{\mathsf{cli}}, \mathsf{ue}_{\mathsf{ser}})$ described in Fig. 4.6 and Fig. 4.7 based on a UE scheme constructs the $\mathbf{cUSMR}^1_{\Sigma,n}$ from the basic $\mathbf{USMR}^1_{\Sigma,n}$ and $\mathbf{UpdKey}$ inside the interval $[P_{\mathtt{only},k}(\mathcal{E}), P_{\mathtt{compromised},k}(\mathcal{E})]$, with respect to the simulator $\sigma_{k,\mathsf{CPA}}$ described in Section 4.3.6 and the dummy converter honSrv (that disables any adversarial behavior). More specifically, we construct reductions such that, for all distinguishers $\mathbf{D}$ in a set of distinguishers $\mathcal{D}$,*

$$\Delta^{\mathbf{D}^{\mathcal{E}}}(\mathsf{honSrv}^{\mathsf{S}}\mathsf{ue}^{\mathsf{C}}_{\mathsf{cli}}\mathsf{ue}^{\mathsf{S}}_{\mathsf{ser}}[\mathbf{USMR}^1_{\Sigma,n}, \mathbf{UpdKey}], \mathsf{honSrv}^{\mathsf{S}}\,\mathbf{cUSMR}^1_{\Sigma,n}) = 0$$

$$\Delta^{\mathbf{D}^{\mathcal{E}}}(\mathsf{ue}^{\mathsf{C}}_{\mathsf{cli}}\mathsf{ue}^{\mathsf{S}}_{\mathsf{ser}}[\mathbf{USMR}^1_{\Sigma,n}, \mathbf{UpdKey}], \sigma^{\mathsf{S}}_{k,\mathsf{CPA}}\,\mathbf{cUSMR}^1_{\Sigma,n}) \leq (2q+r) \cdot \sup_{\mathbf{D}' \in \mathcal{D}} \Delta^{\mathbf{D}'^{\mathcal{E}}}(\mathbf{G}^{\mathsf{IND\text{-}UE\text{-}CPA}}_{\mathsf{UPD}}, \mathbf{G}^{\mathsf{IND\text{-}UE\text{-}CPA}}_{\mathsf{ENC}})$$

where q (resp. r) is an upper bound on the number of writes (resp. updates) made by the distinguisher to the memory location k.

The first condition, called *availability*, checks if the two systems behave in the same way when no adversary is present. It rules out trivial protocols that would ensure confidentiality by not writing data in memory for example. In all of this chapter, *availability* follows from the correctness of the schemes used. For clarity and conciseness, we will omit it in the proofs.

Proof. Let $\mathbf{R} := \mathsf{ue}^{\mathsf{C}}_{\mathsf{cli}}\mathsf{ue}^{\mathsf{S}}_{\mathsf{ser}}[\mathbf{USMR}^1, \mathbf{UpdKey}]$ be the the real system and $\mathbf{I} := \sigma^{\mathsf{S}}_{k,\mathsf{CPA}}\,\mathbf{cUSMR}^1$ be the ideal system. The two systems behave in the same way except when leaking the content of $\mathbb{M}[k]$: $\mathbf{R}$ leaks an encryption of $\mathbb{M}[k]$ while $\mathbf{I}$ leaks an encryption of a random plaintext of length $|\mathbb{M}[k]|$. In order to determine the advantage of a distinguisher in distinguishing $\mathbf{R}$ from $\mathbf{I}$, denoted by $\Delta^{\mathbf{D}}(\mathbf{R}, \mathbf{I})$, we proceed with a sequence of systems. We introduce a hybrid system $\mathbf{S}$, then we determine the distinguishing advantages $\Delta^{\mathbf{D}}(\mathbf{R}, \mathbf{S})$ and $\Delta^{\mathbf{D}}(\mathbf{S}, \mathbf{I})$, the triangular inequality allowing us to bound $\Delta^{\mathbf{D}}(\mathbf{R}, \mathbf{I})$ by the sum of those two advantages.

• Let $\mathbf{S}$ be a resource that behaves just like $\mathbf{R}$ except on query $(\mathtt{leak}, k)$ where it leaks an encryption of a random plaintext of length $|\mathbb{M}[k]|$ instead of an encryption of $\mathbb{M}[k]$, if $\mathbb{M}[k]$ contains a fresh encryption and not an updated one. This happens if a query $(\mathtt{write}, k, x)$ has

been issued by the client in the current epoch. In the case when $\mathbb{M}[k]$ contains an updated version of a ciphertext, the two resources behave in the exact same way.

Let q be an upper bound on the number of $(\texttt{write}, k, .)$ queries issued to the systems. We define a hybrid resource $\mathbf{H}_i$ that behaves just like $\mathbf{R}$ on the first i $(\texttt{write}, k, .)$ queries and like $\mathbf{S}$ afterwards. Then we define a reduction $\mathbf{C}_i$ that behaves like $\mathbf{H}_i$ except it uses the game $\mathbf{G}_b^{\mathsf{IND\text{-}ENC\text{-}CPA}}$ oracles instead of doing the UE operations by itself and on the i-th $(\texttt{write}, k, .)$ request (of the form $(\texttt{write}, k, x)$) it challenges the game with input $(x, \bar{x})$, with $\bar{x}$ random of length $|x|$, to receive the ciphertext. We have

$$\mathbf{R} \equiv \mathbf{H}_q \text{ and } \mathbf{S} \equiv \mathbf{H}_0$$

and

$$\mathbf{H}_i \equiv \mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}CPA}}\mathbf{C}_i \equiv \mathbf{G}_1^{\mathsf{IND\text{-}ENC\text{-}CPA}}\mathbf{C}_{i+1}$$

Indeed, this can be seen on the following timeline (Table 4.1).

j-th $(\texttt{write}, k, .)$ query	$j < i$	$j = i$	$j = i + 1$	$j > i + 1$
$\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}CPA}}\mathbf{C}_i$	$\mathsf{Enc}(x)$	$\mathsf{Enc}(x)$	$\mathsf{Enc}(\bar{x})$	$\mathsf{Enc}(\bar{x})$
$\mathbf{G}_1^{\mathsf{IND\text{-}ENC\text{-}CPA}}\mathbf{C}_{i+1}$	$\mathsf{Enc}(x)$	$\mathsf{Enc}(x)$	$\mathsf{Enc}(\bar{x})$	$\mathsf{Enc}(\bar{x})$

Table 4.1: Leakage behavior of both systems for each $(\texttt{write}, k, .)$ request.

Let $\mathbf{C}_I$ be a reduction that samples $i \in [1, q]$ at random and behaves like $\mathbf{C}_i$ and define $\mathbf{D}' := \mathbf{D}\mathbf{C}_I$. We have,

$$\Pr[\mathbf{D}'(\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}CPA}}) = 1] = \frac{1}{q}\sum_{i=1}^{q} \Pr[\mathbf{D}(\mathbf{C}_i\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}CPA}}) = 1]$$

and

$$\begin{aligned}\Pr[\mathbf{D}'(\mathbf{G}_1^{\mathsf{IND\text{-}ENC\text{-}CPA}}) = 1] &= \frac{1}{q}\sum_{i=1}^{q} \Pr[\mathbf{D}(\mathbf{C}_i\mathbf{G}_1^{\mathsf{IND\text{-}ENC\text{-}CPA}}) = 1] \\ &= \frac{1}{q}\sum_{i=0}^{q-1} \Pr[\mathbf{D}(\mathbf{C}_i\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}CPA}}) = 1]\end{aligned}$$

Finally, the advantage of the distinguisher in distinguishing system $\mathbf{R}$ from $\mathbf{S}$ is

$$\begin{aligned}\Delta^{\mathbf{D}}(\mathbf{R}, \mathbf{S}) &= \Delta^{\mathbf{D}}(\mathbf{H}_q, \mathbf{H}_0) \\ &= \Delta^{\mathbf{D}}(\mathbf{C}_q\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}CPA}}, \mathbf{C}_0\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}CPA}}) \\ &= |\Pr[\mathbf{D}(\mathbf{C}_q\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}CPA}}) = 1] - \Pr[\mathbf{D}(\mathbf{C}_0\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}CPA}}) = 1]| \\ &= |\sum_{i=1}^{q} \Pr[\mathbf{D}(\mathbf{C}_i\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}CPA}}) = 1] - \sum_{i=0}^{q-1} \Pr[\mathbf{D}(\mathbf{C}_i\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}CPA}}) = 1]| \\ &= q \cdot |\Pr[\mathbf{D}'(\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}CPA}}) = 1] - \Pr[\mathbf{D}'(\mathbf{G}_1^{\mathsf{IND\text{-}ENC\text{-}CPA}}) = 1]| \\ &= q \cdot \Delta^{\mathbf{D}'}(\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}CPA}}, \mathbf{G}_1^{\mathsf{IND\text{-}ENC\text{-}CPA}})\end{aligned}$$

• Let us consider the systems $\mathbf{S}$ and $\mathbf{I}$. By definition, $\mathbf{S}$ behaves just like $\mathbf{I}$ except on query $(\texttt{leak}, k)$ where it leaks an updated ciphertext of an encryption of $\mathbb{M}[k]$ (instead of a fresh encryption of a random $\bar{x}$ of length $|\mathbb{M}[k]|$ in the ideal system) if $\mathbb{M}[k]$ contains an updated encryption and not a fresh one. In the case when $\mathbb{M}[k]$ contains a fresh ciphertext, the two

resources behave in the exact same way. Namely, they leak an encryption of a random $\bar{x}$ of length $|\mathbb{M}[k]|$.

Let r be an upper bound on the number of update queries issued to the systems. We define a hybrid resource $\mathbf{H}'_i$ that behaves just like $\mathbf{S}$ on the first i update queries and like $\mathbf{I}$ afterwards. Then we define a reduction $\mathbf{C}'_i$ that behaves like $\mathbf{H}'_i$ except it uses the game $\mathbf{G}^{\mathsf{IND\text{-}UE\text{-}CPA}}_{\mathsf{xxx}}$, where $\mathsf{xxx} \in \{\mathsf{ENC}, \mathsf{UPD}\}$, oracles instead of doing the UE operations by itself and on the i-th update computation for the encryption c of $\mathbb{M}[k]$, it challenges the game with input $(\bar{x}, c)$ to receive either a fresh encryption of the random plaintext $\bar{x}$ (of length $\mathbb{M}[k]$) or the updated version of the ciphertext c. We have

$$\mathbf{S} \equiv \mathbf{H}'_r \text{ and } \mathbf{I} \equiv \mathbf{H}'_0$$

and

$$\mathbf{H}'_i \equiv \mathbf{G}^{\mathsf{IND\text{-}UE\text{-}CPA}}_{\mathsf{UPD}}\mathbf{C}'_i \equiv \mathbf{G}^{\mathsf{IND\text{-}UE\text{-}CPA}}_{\mathsf{ENC}}\mathbf{C}'_{i+1}$$

Indeed, this can be seen on the following timeline (Table 4.2)

j-th update query for $\mathbb{M}[k]$	$j < i$	$j = i$	$j = i+1$	$j > i+1$
$\mathbf{G}^{\mathsf{IND\text{-}UE\text{-}CPA}}_{\mathsf{UPD}}\mathbf{C}'_i$	$\mathsf{Upd}(c)$	$\mathsf{Upd}(c)$	$\mathsf{Enc}(\bar{x})$	$\mathsf{Enc}(\bar{x})$
$\mathbf{G}^{\mathsf{IND\text{-}UE\text{-}CPA}}_{\mathsf{ENC}}\mathbf{C}'_{i+1}$	$\mathsf{Upd}(c)$	$\mathsf{Upd}(c)$	$\mathsf{Enc}(\bar{x})$	$\mathsf{Enc}(\bar{x})$

Table 4.2: Leakage behavior of both systems for each update request ($\bar{x}$ is always a random plaintext of length $\mathbb{M}[k]$).

Let $\mathbf{C}'_I$ be a reduction that samples $i \in [1, r]$ at random and behaves like $\mathbf{C}'_i$ and define $\mathbf{D}'' := \mathbf{D}\mathbf{C}'_I$. We have,

$$\Pr[\mathbf{D}''(\mathbf{G}^{\mathsf{IND\text{-}UE\text{-}CPA}}_{\mathsf{UPD}}) = 1] = \frac{1}{r}\sum_{i=1}^{r}\Pr[\mathbf{D}(\mathbf{C}'_i\mathbf{G}^{\mathsf{IND\text{-}UE\text{-}CPA}}_{\mathsf{UPD}}) = 1]$$

and

$$\Pr[\mathbf{D}''(\mathbf{G}^{\mathsf{IND\text{-}UE\text{-}CPA}}_{\mathsf{ENC}}) = 1] = \frac{1}{r}\sum_{i=0}^{r-1}\Pr[\mathbf{D}(\mathbf{C}'_i\mathbf{G}^{\mathsf{IND\text{-}UE\text{-}CPA}}_{\mathsf{UPD}}) = 1]$$

Finally, the advantage of the distinguisher in distinguishing system $\mathbf{S}$ from $\mathbf{I}$ is

$$\begin{aligned}
\Delta^{\mathbf{D}}(\mathbf{S}, \mathbf{I}) &= \Delta^{\mathbf{D}}(\mathbf{H}'_r, \mathbf{H}'_0) \\
&= \Delta^{\mathbf{D}}(\mathbf{C}'_r\mathbf{G}^{\mathsf{IND\text{-}UE\text{-}CPA}}_{\mathsf{UPD}}, \mathbf{C}'_0\mathbf{G}^{\mathsf{IND\text{-}UE\text{-}CPA}}_{\mathsf{UPD}}) \\
&= |\Pr[\mathbf{D}(\mathbf{C}'_r\mathbf{G}^{\mathsf{IND\text{-}UE\text{-}CPA}}_{\mathsf{UPD}}) = 1] - \Pr[\mathbf{D}(\mathbf{C}'_0\mathbf{G}^{\mathsf{IND\text{-}UE\text{-}CPA}}_{\mathsf{UPD}}) = 1]| \\
&= |\sum_{i=1}^{r}\Pr[\mathbf{D}(\mathbf{C}'_i\mathbf{G}^{\mathsf{IND\text{-}UE\text{-}CPA}}_{\mathsf{UPD}}) = 1] - \sum_{i=0}^{r-1}\Pr[\mathbf{D}(\mathbf{C}'_i\mathbf{G}^{\mathsf{IND\text{-}UE\text{-}CPA}}_{\mathsf{UPD}}) = 1]| \\
&= r \cdot |\Pr[\mathbf{D}''(\mathbf{G}^{\mathsf{IND\text{-}UE\text{-}CPA}}_{\mathsf{UPD}}) = 1] - \Pr[\mathbf{D}''(\mathbf{G}^{\mathsf{IND\text{-}UE\text{-}CPA}}_{\mathsf{ENC}}) = 1]| \\
&= r \cdot \Delta^{\mathbf{D}''}(\mathbf{G}^{\mathsf{IND\text{-}UE\text{-}CPA}}_{\mathsf{UPD}}, \mathbf{G}^{\mathsf{IND\text{-}UE\text{-}CPA}}_{\mathsf{ENC}})
\end{aligned}$$

• We use the triangular inequality to conclude. Let q be our upper bound on the number of writes and r be our upper bound on the number of updates. The advantage of the distinguisher

in distinguishing the real system $\mathbf{R}$ from the ideal one $\mathbf{I}$ is

$$\begin{aligned}\Delta^{\mathbf{D}}(\mathbf{R},\mathbf{I}) &\leq \Delta^{\mathbf{D}}(\mathbf{R},\mathbf{S}) + \Delta^{\mathbf{D}}(\mathbf{S},\mathbf{I}) \\ &= q \cdot \Delta^{\mathbf{D}'}(\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}CPA}}, \mathbf{G}_1^{\mathsf{IND\text{-}ENC\text{-}CPA}}) + r \cdot \Delta^{\mathbf{D}''}(\mathbf{G}_{\mathsf{UPD}}^{\mathsf{IND\text{-}UE\text{-}CPA}}, \mathbf{G}_{\mathsf{ENC}}^{\mathsf{IND\text{-}UE\text{-}CPA}}) \\ &= 2q \cdot \Delta^{\mathbf{D}'\mathbf{C}''}(\mathbf{G}_{\mathsf{UPD}}^{\mathsf{IND\text{-}UE\text{-}CPA}}, \mathbf{G}_{\mathsf{ENC}}^{\mathsf{IND\text{-}UE\text{-}CPA}}) + r \cdot \Delta^{\mathbf{D}''}(\mathbf{G}_{\mathsf{UPD}}^{\mathsf{IND\text{-}UE\text{-}CPA}}, \mathbf{G}_{\mathsf{ENC}}^{\mathsf{IND\text{-}UE\text{-}CPA}}) \\ &\leq (2q + r) \cdot \Delta^{\mathcal{D}}(\mathbf{G}_{\mathsf{UPD}}^{\mathsf{IND\text{-}UE\text{-}CPA}}, \mathbf{G}_{\mathsf{ENC}}^{\mathsf{IND\text{-}UE\text{-}CPA}})^1\end{aligned}$$

Where we use the notation $\Delta^{\mathcal{D}}(\mathbf{X},\mathbf{Y}) := \sup_{\mathbf{D}\in\mathcal{D}} \Delta^{\mathbf{D}}(\mathbf{X},\mathbf{Y})$ and the reduction $\mathbf{C}''$ is given by Boyd *et al.* in [BDGJ20, Theorem 2.3] to prove the following:

Proposition 4.3. *Let* UE *be a UE scheme. For any* IND-ENC-CPA *adversary* $\mathbf{A}$ *against* UE*, there exists a reduction* $\mathbf{C}''$ *such that*

$$\Delta^{\mathbf{A}}(\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}CPA}}, \mathbf{G}_1^{\mathsf{IND\text{-}ENC\text{-}CPA}}) \leq 2 \cdot \Delta^{\mathbf{A}\mathbf{C}''}(\mathbf{G}_{\mathsf{ENC}}^{\mathsf{IND\text{-}UE\text{-}CPA}}, \mathbf{G}_{\mathsf{UPD}}^{\mathsf{IND\text{-}UE\text{-}CPA}})$$

□

Remark 12. We point out that our interval choice, our simulator and our reduction circumvent the commitment problem. Indeed, when a key exposure makes us leave an insulated region, the adversary can:

1. decrypt the content of ciphertexts stored at location $i \neq k$. These ciphertexts are perfectly simulated since we do not consider the confidentiality of their plaintexts in our interval. Thus, there is no commitment problem.
2. decrypt the content of the k-th ciphertext before its content has been used to produce the CPA game challenge. This closes our interval since the k-th plaintext is no longer confidential.
3. decrypt the content of the k-th ciphertext after its content has been used to produce the CPA game challenge. This triggers a trivial win condition in the CPA game, the adversary thus loses the game and our interval closes like above.

IND-UE-CPA security is necessary for a secure construction of cUSMR

Recall that we are working in the context of restricted leakage, where the adversary is only allowed to leak each entry of the database at most one time per epoch. In order to show that the IND-UE-CPA security notion is necessary to securely construct the $\mathbf{cUSMR}^1$ from a $\mathbf{USMR}^1$ and $\mathbf{UpdKey}$ using a UE scheme, we are going to use a technique of Coretti *et al.* in [CMT13]. Keeping our notation, we will start from the real system $\mathbf{R} := \mathsf{ue}_{\mathsf{cli}}^{\mathsf{C}}\mathsf{ue}_{\mathsf{ser}}^{\mathsf{S}}[\mathbf{USMR}^1, \mathbf{UpdKey}]$ and the ideal one $\mathbf{I} := \sigma^{\mathsf{S}}\,\mathbf{cUSMR}^1$, where this time an arbitrary simulator σ is used, and use reductions to construct the resources $\mathbf{G}_{\mathsf{xxx}}^{\mathsf{IND\text{-}UE\text{-}CPA}}$ implementing the IND-UE-CPA games. Formally, we will describe two reductions $\mathbf{C}_0$ and $\mathbf{C}_1$ such that

$$\mathbf{C}_0\mathbf{R} \equiv \mathbf{G}_{\mathsf{ENC}}^{\mathsf{IND\text{-}UE\text{-}CPA}},\ \mathbf{C}_1\mathbf{R} \equiv \mathbf{G}_{\mathsf{UPD}}^{\mathsf{IND\text{-}UE\text{-}CPA}} \text{ and } \mathbf{C}_0\mathbf{I} \equiv \mathbf{C}_1\mathbf{I}.$$

Let $\mathbf{A}$ be an adversary against the IND-UE-CPA security notion. Using the triangular inequality, we have

[1]Since we can also split the games the other way and consider IND-UPD-CPA security first, we can replace $2q + r$ with $\min\{2q + r, q + 2r\}$.

$$\begin{aligned}\Delta^{\mathbf{A}}(\mathbf{G}^{\mathsf{IND\text{-}UE\text{-}CPA}}_{\mathsf{ENC}}, \mathbf{G}^{\mathsf{IND\text{-}UE\text{-}CPA}}_{\mathsf{UPD}}) &= \Delta^{\mathbf{A}}(\mathbf{C}_0\mathbf{R}, \mathbf{C}_1\mathbf{R}) \\ &\leq \Delta^{\mathbf{A}}(\mathbf{C}_0\mathbf{R}, \mathbf{C}_0\mathbf{I}) + \underbrace{\Delta^{\mathbf{A}}(\mathbf{C}_0\mathbf{I}, \mathbf{C}_1\mathbf{I})}_{=0} + \quad \Delta^{\mathbf{A}}(\mathbf{C}_1\mathbf{I}, \mathbf{C}_1\mathbf{R}) \\ &= \Delta^{\mathbf{A}\mathbf{C}_0}(\mathbf{R}, \mathbf{I}) + \Delta^{\mathbf{A}\mathbf{C}_1}(\mathbf{R}, \mathbf{I})\end{aligned}$$

Thus, if a UE scheme securely construct (through a protocol) $\mathbf{I}$ from $\mathbf{R}$, then for any distinguisher $\mathbf{D}$, the distinguishing advantage $\Delta^{\mathbf{D}}(\mathbf{R}, \mathbf{I})$ is "small" and using the above inequality, the advantage of any adversary $\mathbf{A}$ in distinguishing the $\mathsf{IND\text{-}UE\text{-}CPA}$ games will also be "small". We conclude that, under the above hypothesis, the UE scheme will be $\mathsf{IND\text{-}UE\text{-}CPA}$ secure.

• We are going to describe two reductions $\mathbf{C}_0$ and $\mathbf{C}_1$ that connect to a $[\mathbf{USMR}^1, \mathbf{UpdKey}]$ (or $\mathbf{cUSMR}^1$) resource and implement the oracles of the $\mathbf{G}^{\mathsf{IND\text{-}UE\text{-}CPA}}_{xxx}$ games. Let n be an upper bound on the number of encryption queries issued to the games by the adversary. In the following, we consider resources of memory size $n+1$. This is done to have enough space to store the ciphertexts returned by the encryption oracle as well as the challenge in case the challenger asks for updates later on. Here is how the reductions implement the game oracles:

- On the i-th $\mathcal{O}.\mathsf{Enc}(x)$ query: $\mathbf{C}_b$ sends $(\texttt{write}, i, x)$ at interface C, then sends $(\texttt{leak}, i)$ at interface S and outputs the result. $\mathbf{C}_b$ also keeps a copy of the result in its own internal memory.
- On $\mathcal{O}.\mathsf{Next}()$: $\mathbf{C}_b$ sends `askUpdate` at interface C and `update` at interface S.
- On $\mathcal{O}.\mathsf{Upd}(c)$: $\mathbf{C}_b$ checks if c is the challenge or an updated version of it, in which case it returns $\perp$, and if c was present in memory in the previous epoch. $\mathbf{C}_b$ can do this by keeping track of the entries it leaks and, at the end of each epoch, $\mathbf{C}_b$ sends $(\texttt{leak}, i)$ requests for every position i that has not been leaked in the current epoch yet and writes the results in its own internal memory. If c was not present, it returns $\perp$. If it was, let i be the index where c was written, $\mathbf{C}_b$ sends $(\texttt{leak}, i)$ at interface S and outputs the result (and writes it in its own internal memory).
- On $\mathcal{O}.\mathsf{Corr}(\mathsf{elt}, \hat{\mathsf{e}})$ where $\mathsf{elt} \in \{\mathsf{Key}, \mathsf{Token}\}$: $\mathbf{C}_b$ triggers the event $\mathcal{E}^{\texttt{leaked}}_{\mathsf{elt},\hat{\mathsf{e}}}$, sends $(\texttt{fetchelt}, \hat{\mathsf{e}})$ at interface S and outputs the result.
- On $\mathcal{O}.\mathsf{Chall}(m, c)$: Both reductions check that c was present in memory in the previous epoch and that its updated version has not been leaked yet in the current epoch. They also check that $m' := \mathsf{Dec}_k(c)$ and m have the same length, this can be done with a `read` request at interface C. They return $\perp$ if it is not the case. Otherwise, let i be the position where c was written. The reductions then proceed like this:
  - $\mathbf{C}_0$ sends $(\texttt{write}, n+1, m)$ at interface C then it sends $(\texttt{leak}, n+1)$ at interface S and outputs the result.
  - $\mathbf{C}_1$ sends $(\texttt{leak}, i)$ at interface S and outputs the result.
- On $\mathcal{O}.\mathsf{Upd}\tilde{\mathsf{C}}()$: Let i_b be the position where the challenge is written for $\mathbf{C}_b$. If a challenge has already been issued, $\mathbf{C}_b$ sends $(\texttt{leak}, i_b)$ at interface S and returns the result. If not, it returns $\perp$.

Moreover, the reductions have all the information to check for trivial wins. When connected to the real system $\mathbf{R}$, the reductions $\mathbf{C}_b$ exactly implements the games $\mathbf{G}^{\mathsf{IND\text{-}UE\text{-}CPA}}_{xxx}$ since the system leaks the same encryptions and updates as the ones produce by the game and correctly implements all of its oracles. Their observable behaviors being the same, we have $\mathbf{C}_0\mathbf{R} \equiv \mathbf{G}^{\mathsf{IND\text{-}UE\text{-}CPA}}_{\mathsf{ENC}}$ and $\mathbf{C}_1\mathbf{R} \equiv \mathbf{G}^{\mathsf{IND\text{-}UE\text{-}CPA}}_{\mathsf{UPD}}$.

Now, if we connect the reductions to the ideal system $\mathbf{I}$, where an arbitrary simulator σ is used, their behaviors can only differ on the challenge call. When asked for the challenge on the pair (m, c), reduction $\mathbf{C}_0$ writes m at position $n+1$ then it outputs the result of $(\texttt{leak}, n+1)$ meanwhile $\mathbf{C}_1$ outputs the result of $(\texttt{leak}, i)$ where i is the position where the ciphertext c was already written. In the ideal system $\mathbf{I}$, the result of a `leak` request is always ℓ, the length of the message stored. Since, letting m' be the underlying plaintext of c, the reductions checked that the two messages m and m' had the same length, the inputs of the simulator σ are the same and the behaviors are thus identical. In the following epochs, this remains true when the adversary calls the oracle $\mathcal{O}.\mathsf{Upd}\tilde{\mathsf{C}}$ to get an updated version of the challenge ciphertext. We can check that we never go beyond the leakage condition, it is never the case that a reduction leaks the content of a position j at more than one occasion in a given epoch. We conclude that the systems $\mathbf{C}_b\mathbf{I}$ are indistinguishable from one another. Written differently, we have $\mathbf{C}_0\mathbf{I} \equiv \mathbf{C}_1\mathbf{I}$ and the theorem is proven in the IND-UE-CPA case.

In the context of restricted leakage, we proved that the IND-UE-CPA security notion for UE schemes was the correct one to consider if clients want to protect the confidentiality and the age of the data they store on a remote server in the presence of an adversary only able to leak the contents of the server at most once per epoch. This use case is realistic since it includes the common scenario of data breaches: when attackers are able to dump the content of a whole database and run away with the data. In the context of such an intrusion we showed, through the use of CC, how to assess the situation, *i.e.* we are able to describe precisely the remaining security guarantees for the confidentiality of the content and the age of the data depending on what keys and tokens are available to the adversaries.

Any number of leaks: the CPA case.

This time, we are proving our theorem 4.1 in the context of unrestricted leakage. Just like before, let n be the size of the $\mathbf{USMR}^+$ and take $k \in \{1, \ldots, n\}$. We place ourselves in a time interval where we do not consider the confidentiality of messages other than the k-th one. In this case the simulator $\sigma_{k,\mathsf{CPA}^+}$ works as follows. It simulates the epoch keys and tokens and, on a $(\texttt{leakKey}, i)$ or $(\texttt{leakToken}, i)$ request, it checks if the event $\mathcal{E}^{\texttt{leaked}}_{\mathsf{Key},i}$ (respectively $\mathcal{E}^{\texttt{leaked}}_{\mathsf{Token},i}$) exists in the Global Event History and leaks the corresponding epoch key (respectively token) to the adversary if it is the case and $\perp$ otherwise. The simulator uses the ideal history to know which entries of the database correspond to fresh encryptions or updated encryptions. Together with its simulated epoch keys and tokens, this allows the simulator to maintain a simulated memory (and a simulated history) where fresh encryptions of $\mathbb{M}[k]$ (in the real world) are replaced with fresh encryptions of random plaintexts of length $|\mathbb{M}[k]|$ and updated ciphertexts encrypting $\mathbb{M}[k]$ (in the real world) are replaced with updates of ciphertexts of random plaintexts of length $|\mathbb{M}[k]|$. When $i \neq k$, the simulated memory and history perfectly match their real-world counterparts. Finally, on a $(\texttt{leak}, i)$ request, the simulator returns the i-th entry of its simulated memory.

IND-ENC-CPA+IND-UPD-CPA security is sufficient for a secure construction of $\mathbf{cUSMR}^+$

The fact that IND-ENC-CPA+IND-UPD-CPA is sufficient to construct $\mathbf{cUSMR}^+$ from $\mathbf{USMR}^+$ and **UpdKey** is expressed in Theorem 4.1 through an intersection of specifications. The following theorem shows how we construct each of those specifications.

Theorem 4.4. *Let Σ be a finite alphabet, $n \in \mathbb{N}$ and $k \in \{1, \ldots, n\}$. The protocol $\mathsf{ue} := (\mathsf{ue}_{\mathsf{cli}}, \mathsf{ue}_{\mathsf{ser}})$ described in Fig. 4.6 and Fig. 4.7 based on a UE scheme constructs the $\mathbf{cUSMR}^+_{\Sigma,n}$ from the basic $\mathbf{USMR}^+_{\Sigma,n}$ and **UpdKey** inside the interval $[P_{\texttt{only},k}(\mathcal{E}), P_{\texttt{compromised},k}(\mathcal{E})]$, with*

respect to the simulator $\sigma_{k,\mathsf{CPA}^+}$ described in Section 4.3.6. More specifically, we construct reductions $\mathbf{C}'$ and $\mathbf{C}''$ such that, for all distinguishers $\mathbf{D}$,

$$\Delta^{\mathbf{D}^{\mathcal{E}}}(\mathsf{ue}^{\mathsf{C}}_{\mathsf{cli}}\mathsf{ue}^{\mathsf{S}}_{\mathsf{ser}}[\mathbf{USMR}^{+}_{\Sigma,n}, \mathbf{UpdKey}], \sigma^{\mathsf{S}}_{k,\mathsf{CPA}^+}\,\mathbf{cUSMR}^{+}_{\Sigma,n}) \leq$$
$$q \cdot \Delta^{\mathbf{DC}'^{\mathcal{E}}}(\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}CPA}}, \mathbf{G}_1^{\mathsf{IND\text{-}ENC\text{-}CPA}}) + r \cdot \Delta^{\mathbf{DC}''^{\mathcal{E}}}(\mathbf{G}_0^{\mathsf{IND\text{-}UPD\text{-}CPA}}, \mathbf{G}_1^{\mathsf{IND\text{-}UPD\text{-}CPA}})$$

where q (resp. r) is an upper bound on the number of writes (resp. updates) made by the distinguisher to the memory location k.

The proof is very similar to the one detailed in the IND-UE-CPA case. We include it for completeness.

Proof. Let $\mathbf{R} := \mathsf{ue}^{\mathsf{C}}_{\mathsf{cli}}\mathsf{ue}^{\mathsf{S}}_{\mathsf{ser}}[\mathbf{USMR}^{+}, \mathbf{UpdKey}]$ be the the real system and $\mathbf{I} := \sigma^{\mathsf{S}}_{k,\mathsf{CPA}^+}\,\mathbf{cUSMR}^{+}$ be the ideal system. In order to determine the advantage of a distinguisher in distinguishing $\mathbf{R}$ from $\mathbf{I}$, denoted by $\Delta^{\mathbf{D}}(\mathbf{R}, \mathbf{I})$, we proceed with a sequence of systems. We introduce a hybrid system $\mathbf{S}$, then we determine the distinguishing advantages $\Delta^{\mathbf{D}}(\mathbf{R}, \mathbf{S})$ and $\Delta^{\mathbf{D}}(\mathbf{S}, \mathbf{I})$, the triangular inequality allows us to bound $\Delta^{\mathbf{D}}(\mathbf{R}, \mathbf{I})$ by the sum of those two advantages.

• Let $\mathbf{S}$ be a resource that behaves just like $\mathbf{R}$ when leaking updated ciphertexts and just like $\mathbf{I}$ when leaking fresh encryptions. Concretely, $\mathbf{S}$ maintains the same database as $\mathbf{R}$ using the UE scheme and, when $\mathbf{S}$ is asked to leak an updated ciphertext it returns it as is but when $\mathbf{S}$ is asked to leak a fresh ciphertext encrypting $\mathbb{M}[k]$, it returns an encryption of a random $\bar{x}$ of length $|\mathbb{M}[k]|$.

Let q be an upper bound on the number of $(\mathtt{write}, k, .)$ queries issued to the systems. We define a hybrid resource $\mathbf{H}_i$ that behaves just like $\mathbf{R}$ on the first i $(\mathtt{write}, k, .)$ queries and like $\mathbf{S}$ afterwards. Then we define a reduction $\mathbf{C}_i$ that behaves like $\mathbf{H}_i$ except it uses the game $\mathbf{G}_b^{\mathsf{IND\text{-}ENC\text{-}CPA}}$ oracles instead of doing the UE operations by itself and on the i-th $(\mathtt{write}, k, .)$ request (of the form $(\mathtt{write}, k, x)$) it challenges the game with input $(x, \bar{x})$ to receive the ciphertext. We have

$$\mathbf{R} \equiv \mathbf{H}_q \text{ and } \mathbf{S} \equiv \mathbf{H}_0$$

and

$$\mathbf{H}_i \equiv \mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}CPA}}\mathbf{C}_i \equiv \mathbf{G}_1^{\mathsf{IND\text{-}ENC\text{-}CPA}}\mathbf{C}_{i+1}$$

Indeed, this can be seen on the following timeline (Table 4.3)

j-th $(\mathtt{write}, k, .)$ query	$j < i$	$j = i$	$j = i + 1$	$j > i + 1$
$\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}CPA}}\mathbf{C}_i$	$\mathsf{Enc}(x)$	$\mathsf{Enc}(x)$	$\mathsf{Enc}(\bar{x})$	$\mathsf{Enc}(\bar{x})$
$\mathbf{G}_1^{\mathsf{IND\text{-}ENC\text{-}CPA}}\mathbf{C}_{i+1}$	$\mathsf{Enc}(x)$	$\mathsf{Enc}(x)$	$\mathsf{Enc}(\bar{x})$	$\mathsf{Enc}(\bar{x})$

Table 4.3: Leakage behavior of both systems for each $(\mathtt{write}, k, .)$ request.

Let $\mathbf{C}_I$ be a reduction that samples $i \in [1, q]$ at random and behaves like $\mathbf{C}_i$ and define $\mathbf{D}' := \mathbf{DC}_I$. We have,

$$\Pr[\mathbf{D}'(\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}CPA}}) = 1] = \frac{1}{q}\sum_{i=1}^{q} \Pr[\mathbf{D}(\mathbf{C}_i\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}CPA}}) = 1]$$

and

$$\Pr[\mathbf{D}'(\mathbf{G}_1^{\mathsf{IND\text{-}ENC\text{-}CPA}}) = 1] = \frac{1}{q}\sum_{i=1}^{q}\Pr[\mathbf{D}(\mathbf{C}_i\mathbf{G}_1^{\mathsf{IND\text{-}ENC\text{-}CPA}}) = 1]$$
$$= \frac{1}{q}\sum_{i=0}^{q-1}\Pr[\mathbf{D}(\mathbf{C}_i\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}CPA}}) = 1]$$

Finally, the advantage of the distinguisher in distinguishing system $\mathbf{R}$ from $\mathbf{S}$ is

$$\begin{aligned}
\Delta^{\mathbf{D}}(\mathbf{R},\mathbf{S}) &= \Delta^{\mathbf{D}}(\mathbf{H}_q,\mathbf{H}_0)\\
&= \Delta^{\mathbf{D}}(\mathbf{C}_q\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}CPA}},\mathbf{C}_0\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}CPA}})\\
&= |\Pr[\mathbf{D}(\mathbf{C}_q\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}CPA}}) = 1] - \Pr[\mathbf{D}(\mathbf{C}_0\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}CPA}}) = 1]|\\
&= |\sum_{i=1}^{q}\Pr[\mathbf{D}(\mathbf{C}_i\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}CPA}}) = 1] - \sum_{i=0}^{q-1}\Pr[\mathbf{D}(\mathbf{C}_i\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}CPA}}) = 1]|\\
&= q\cdot|\Pr[\mathbf{D}'(\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}CPA}}) = 1] - \Pr[\mathbf{D}'(\mathbf{G}_1^{\mathsf{IND\text{-}ENC\text{-}CPA}}) = 1]|\\
&= q\cdot\Delta^{\mathbf{D}'}(\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}CPA}},\mathbf{G}_1^{\mathsf{IND\text{-}ENC\text{-}CPA}})
\end{aligned}$$

• Let us consider the systems $\mathbf{S}$ and $\mathbf{I}$. By definition, $\mathbf{S}$ behaves just like $\mathbf{I}$ when leaking fresh encryptions but, when asked to leak what should be an updated ciphertext encrypting $\mathbb{M}[k]$, $\mathbf{S}$ returns this updated ciphertext while $\mathbf{I}$ simply returns an update of an encryption of a random $\bar{x}$ of length $|\mathbb{M}[k]|$.

Let r be an upper bound on the number of update queries issued to the systems. We define a hybrid resource $\mathbf{H}'_i$ that behaves just like $\mathbf{S}$ on the first i update queries to location k and like $\mathbf{I}$ afterwards. Then we define a reduction $\mathbf{C}'_i$ that behaves like $\mathbf{H}'_i$ except it uses the game $\mathbf{G}_b^{\mathsf{IND\text{-}UPD\text{-}CPA}}$ oracles instead of doing the UE operations by itself and on the i-th update computation for the encryption c of $\mathbb{M}[k]$, it challenges the game with input $(c,\bar{c})$ to receive either an updated version of c or $\bar{c}$, the encryption of a random $\bar{x}$ of length $|\mathbb{M}[k]|$. We have

$$\mathbf{S} \equiv \mathbf{H}'_r \text{ and } \mathbf{I} \equiv \mathbf{H}'_0$$

and

$$\mathbf{H}'_i \equiv \mathbf{G}_0^{\mathsf{IND\text{-}UPD\text{-}CPA}}\mathbf{C}'_i \equiv \mathbf{G}_1^{\mathsf{IND\text{-}UPD\text{-}CPA}}\mathbf{C}'_{i+1}$$

Indeed, this can be seen on the following timeline (Table 4.4)

j-th update query for $\mathbb{M}[k]$	$j<i$	$j=i$	$j=i+1$	$j>i+1$
$\mathbf{G}_0^{\mathsf{IND\text{-}UPD\text{-}CPA}}\mathbf{C}'_i$	$\mathsf{Upd}(c)$	$\mathsf{Upd}(c)$	$\mathsf{Upd}(\bar{c})$	$\mathsf{Upd}(\bar{c})$
$\mathbf{G}_1^{\mathsf{IND\text{-}UPD\text{-}CPA}}\mathbf{C}'_{i+1}$	$\mathsf{Upd}(c)$	$\mathsf{Upd}(c)$	$\mathsf{Upd}(\bar{c})$	$\mathsf{Upd}(\bar{c})$

Table 4.4: Leakage behavior of both systems for each update request ($\bar{c}$ is always the encryption of a random plaintext of length $|\mathbb{M}[k]|$).

Let $\mathbf{C}'_I$ be a reduction that samples $i \in [1,r]$ at random and behaves like $\mathbf{C}'_i$ and define $\mathbf{D}'' := \mathbf{D}\mathbf{C}'_I$. We have,

$$\Pr[\mathbf{D}''(\mathbf{G}_0^{\mathsf{IND\text{-}UPD\text{-}CPA}}) = 1] = \frac{1}{r}\sum_{i=1}^{r}\Pr[\mathbf{D}(\mathbf{C}'_i\mathbf{G}_0^{\mathsf{IND\text{-}UPD\text{-}CPA}}) = 1]$$

and

$$\begin{aligned}\Pr[\mathbf{D}''(\mathbf{G}_1^{\mathsf{IND\text{-}UPD\text{-}CPA}}) = 1] &= \frac{1}{r}\sum_{i=1}^{r}\Pr[\mathbf{D}(\mathbf{C}'_i\mathbf{G}_1^{\mathsf{IND\text{-}UPD\text{-}CPA}}) = 1] \\ &= \frac{1}{r}\sum_{i=0}^{r-1}\Pr[\mathbf{D}(\mathbf{C}'_i\mathbf{G}_0^{\mathsf{IND\text{-}UPD\text{-}CPA}}) = 1]\end{aligned}$$

Finally, the advantage of the distinguisher in distinguishing system $\mathbf{S}$ from $\mathbf{I}$ is

$$\begin{aligned}\Delta^{\mathbf{D}}(\mathbf{S},\mathbf{I}) &= \Delta^{\mathbf{D}}(\mathbf{H}'_r,\mathbf{H}'_0) \\ &= \Delta^{\mathbf{D}}(\mathbf{C}'_r\mathbf{G}_0^{\mathsf{IND\text{-}UPD\text{-}CPA}}, \mathbf{C}'_0\mathbf{G}_0^{\mathsf{IND\text{-}UPD\text{-}CPA}}) \\ &= |\Pr[\mathbf{D}(\mathbf{C}'_r\mathbf{G}_0^{\mathsf{IND\text{-}UPD\text{-}CPA}}) = 1] - \Pr[\mathbf{D}(\mathbf{C}'_0\mathbf{G}_0^{\mathsf{IND\text{-}UPD\text{-}CPA}}) = 1]| \\ &= |\sum_{i=1}^{r}\Pr[\mathbf{D}(\mathbf{C}'_i\mathbf{G}_0^{\mathsf{IND\text{-}UPD\text{-}CPA}}) = 1] - \sum_{i=0}^{r-1}\Pr[\mathbf{D}(\mathbf{C}'_i\mathbf{G}_0^{\mathsf{IND\text{-}UPD\text{-}CPA}}) = 1]| \\ &= r\cdot|\Pr[\mathbf{D}''(\mathbf{G}_0^{\mathsf{IND\text{-}UPD\text{-}CPA}}) = 1] - \Pr[\mathbf{D}''(\mathbf{G}_1^{\mathsf{IND\text{-}UPD\text{-}CPA}}) = 1]| \\ &= r\cdot\Delta^{\mathbf{D}''}(\mathbf{G}_0^{\mathsf{IND\text{-}UPD\text{-}CPA}}, \mathbf{G}_1^{\mathsf{IND\text{-}UPD\text{-}CPA}})\end{aligned}$$

• We use the triangular inequality to conclude. Let q be our upper bound on the number of writes and r be our upper bound on the number of updates. The advantage of the distinguisher in distinguishing the real system $\mathbf{R}$ from the ideal one $\mathbf{I}$ is

$$\begin{aligned}\Delta^{\mathbf{D}}(\mathbf{R},\mathbf{I}) &\leq \Delta^{\mathbf{D}}(\mathbf{R},\mathbf{S}) + \Delta^{\mathbf{D}}(\mathbf{S},\mathbf{I}) \\ &= q\cdot\Delta^{\mathbf{D}'}(\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}CPA}}, \mathbf{G}_1^{\mathsf{IND\text{-}ENC\text{-}CPA}}) + r\cdot\Delta^{\mathbf{D}''}(\mathbf{G}_0^{\mathsf{IND\text{-}UPD\text{-}CPA}}, \mathbf{G}_1^{\mathsf{IND\text{-}UPD\text{-}CPA}})\end{aligned}$$

So IND-ENC-CPA + IND-UPD-CPA is sufficient to securely construct the $\mathbf{cUSMR}^+$ in the unrestricted leakage model, where the age of each database entry is not hidden. □

In [BDGJ20] the authors showed that IND-UE-CPA security implied IND-ENC-CPA and IND-UPD-CPA security. Since we showed that IND-ENC-CPA + IND-UPD-CPA security is sufficient to securely construct the $\mathbf{cUSMR}^+$ from the $\mathbf{USMR}^+$ equipped with the $(\mathsf{ue}_{\mathsf{cli}}, \mathsf{ue}_{\mathsf{ser}})$ converters, we conclude that IND-UE-CPA security cannot be necessary for this secure construction in the unrestricted leakage model. This notion is thus unnecessarily strong in this setting.

IND-ENC-CPA and IND-UPD-CPA security are necessary for a secure construction of $\mathbf{cUSMR}^+$

We use the same proof technique as in the IND-UE-CPA case. We include the proof for completeness. We show that the IND-ENC-CPA and IND-UPD-CPA security notions are both necessary to securely construct the $\mathbf{cUSMR}^+$ from $\mathbf{USMR}^+$ and $\mathbf{UpdKey}$ using a UE scheme. Keeping our notations, we will start from the real system $\mathbf{R}$ and the ideal one $\mathbf{I}$, where this time we use an arbitrary simulator σ, and use reductions to construct the resources $\mathbf{G}_b^{\mathsf{IND\text{-}ENC\text{-}CPA}}$ (respectively $\mathbf{G}_b^{\mathsf{IND\text{-}UPD\text{-}CPA}}$) implementing the IND-ENC-CPA games (respectively the IND-UPD-CPA games). Formally, we will describe two reductions $\mathbf{C}_0$ and $\mathbf{C}_1$ such that

$$\mathbf{C}_0\mathbf{R} \equiv \mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}CPA}},\ \mathbf{C}_1\mathbf{R} \equiv \mathbf{G}_1^{\mathsf{IND\text{-}ENC\text{-}CPA}} \text{ and } \mathbf{C}_0\mathbf{I} \equiv \mathbf{C}_1\mathbf{I}$$

Let $\mathbf{A}$ be an adversary against the IND-ENC-CPA security notion. Using the triangular inequality, we have

$$\begin{aligned}\Delta^{\mathbf{A}}(\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}CPA}}, \mathbf{G}_1^{\mathsf{IND\text{-}ENC\text{-}CPA}}) &= \Delta^{\mathbf{A}}(\mathbf{C}_0\mathbf{R}, \mathbf{C}_1\mathbf{R}) \\ &\leq \Delta^{\mathbf{A}}(\mathbf{C}_0\mathbf{R}, \mathbf{C}_0\mathbf{I}) + \underbrace{\Delta^{\mathbf{A}}(\mathbf{C}_0\mathbf{I}, \mathbf{C}_1\mathbf{I})}_{=0} + \Delta^{\mathbf{A}}(\mathbf{C}_1\mathbf{I}, \mathbf{C}_1\mathbf{R}) \\ &= \Delta^{\mathbf{A}\mathbf{C}_0}(\mathbf{R}, \mathbf{I}) + \Delta^{\mathbf{A}\mathbf{C}_1}(\mathbf{R}, \mathbf{I})\end{aligned}$$

Thus, if a UE scheme securely construct (through a protocol) $\mathbf{I}$ from $\mathbf{R}$, then for any distinguisher $\mathbf{D}$, the distinguishing advantage $\Delta^{\mathbf{D}}(\mathbf{R}, \mathbf{I})$ is "small" and using the above inequality, the advantage of any adversary $\mathbf{A}$ in distinguishing the IND-ENC-CPA games will also be "small". We conclude that, under the above hypothesis, the UE scheme will be IND-ENC-CPA secure. We do the same thing with different reductions for the IND-UPD-CPA notion to prove our claim.

• We start with the IND-ENC-CPA notion. We are going to describe the two above reductions $\mathbf{C}_0$ and $\mathbf{C}_1$. Let n be an upper bound on the number of encryption queries issued to the games by the adversary. In the following, we consider resources of memory size $n+1$. This is done to have enough space to store the returned ciphertext (and the challenge ciphertext) in case the challenger ask for an update later on. Here is how the reductions implement the game oracles:

- On the i-th $\mathcal{O}.\mathsf{Enc}(x)$ query: $\mathbf{C}_b$ sends $(\texttt{write}, i, x)$ at interface C, then sends $(\texttt{leak}, i)$ at interface S and outputs the result.
- On $\mathcal{O}.\mathsf{Next}()$: $\mathbf{C}_b$ sends `askUpdate` at interface C and `update` at interface S.
- On $\mathcal{O}.\mathsf{Upd}(c)$: $\mathbf{C}_b$ checks if c is the challenge or an updated version of it and if c was present in memory in the previous epoch. If not, it returns $\perp$. If it is, let i be the index where c was written, $\mathbf{C}_b$ sends $(\texttt{leak}, i)$ at interface S and outputs the result.
- On $\mathcal{O}.\mathsf{Corr}(\mathsf{elt}, \hat{\mathsf{e}})$ where $\mathsf{elt} \in \{\mathsf{Key}, \mathsf{Token}\}$: $\mathbf{C}_b$ triggers the event $\mathcal{E}^{\texttt{leaked}}_{\mathsf{elt},\hat{\mathsf{e}}}$, sends $(\texttt{fetchelt}, \hat{\mathsf{e}})$ at interface S and outputs the result.
- On $\mathcal{O}.\mathsf{Chall}(m_0, m_1)$: If $m_0 \neq m_1$ and $|m_0| = |m_1|$, the reduction $\mathbf{C}_b$ sends $(\texttt{write}, n+1, m_b)$ at interface C then it sends $(\texttt{leak}, n+1)$ at interface S and outputs the result.
- On $\mathcal{O}.\mathsf{Upd}\tilde{\mathsf{C}}()$: If a challenge has already been issued, $\mathbf{C}_b$ sends $(\texttt{leak}, n+1)$ at interface S and returns the result. If not, it returns $\perp$.

Moreover, all the oracles variables and lists can be emulated faithfully by the reductions via internal calculations and memory. When connected to the real system $\mathbf{R}$, the reduction $\mathbf{C}_b$ exactly implements the game $\mathbf{G}_b^{\mathsf{IND\text{-}ENC\text{-}CPA}}$ since the system leaks the same encryptions and updates as the ones produce by the game. Their observable behaviors being the same, we have $\mathbf{C}_b\mathbf{R} \equiv \mathbf{G}_b^{\mathsf{IND\text{-}ENC\text{-}CPA}}$.

Now, if we connect the reductions to the ideal system $\mathbf{I}$, where an arbitrary simulator σ is used, their behaviors can only differ on the challenge call. When asked for the challenge, reduction $\mathbf{C}_b$ writes m_b at position $n+1$ then it outputs the result of $(\texttt{leak}, n+1)$. In the ideal system $\mathbf{I}$, the result of a `leak` request on a freshly written entry (which is the case here since we do not change epoch yet) is always the length of the entry. Since the reductions checked that $|m_0| = |m_1|$, the inputs of the simulator σ are the same and the behaviors are thus identical. This remains true in the following epochs, when the adversary calls the oracle $\mathcal{O}.\mathsf{Upd}\tilde{\mathsf{C}}$ to get an updated version of the challenge ciphertext, since the challenge's plaintext is never rewritten.

We conclude that the systems $\mathbf{C}_b\mathbf{I}$ are indistinguishable from one another. Written differently, we have $\mathbf{C}_0\mathbf{I} \equiv \mathbf{C}_1\mathbf{I}$ and the theorem is proven in the IND-ENC-CPA case.

• Since all the oracles, except the $\mathcal{O}$.Chall one, of the IND-UPD-CPA games are the same as the ones present in the IND-ENC-CPA one, we only need to redefine the reductions for this oracle. Let $\mathbf{C}'_b$ be a reduction that behaves exactly like $\mathbf{C}_b$ except:

- On $\mathcal{O}.\mathsf{Chall}(c_0, c_1)$: If $c_0 \neq c_1$ and $|c_0| = |c_1|$, the reduction $\mathbf{C}'_b$ checks that both c_0 and c_1 were present in the previous epoch. If they were, let i_0 and i_1 be the respective positions where these ciphertexts were written, $\mathbf{C}'_b$ sends $(\texttt{leak}, i_b)$ at interface S, returns the result and remembers i_b by writing it in its own internal memory. If not, it returns $\perp$.

Since we do not need space to write a new ciphertext when the challenge is issued in these games, we can use a $\mathbf{USMR}^+$ and $\mathbf{cUSMR}^+$ resources of size n instead of $n+1$ like before (remember that n is an upper bound on the number of calls to the encryption oracle). Since the challenge is written at position i_b, we need to slightly modify the behavior of $\mathbf{C}'_b$ when emulating the oracle $\mathcal{O}.\mathsf{Upd}\tilde{\mathsf{C}}$:

- On $\mathcal{O}.\mathsf{Upd}\tilde{\mathsf{C}}()$: If a challenge has already been issued, $\mathbf{C}_b$ sends $(\texttt{leak}, i_b)$ at interface S and returns the result. If not, it returns $\perp$.

The arguments to show that $\mathbf{C}'_b\mathbf{R} \equiv \mathbf{G}_b^{\text{IND-UPD-CPA}}$ and $\mathbf{C}'_0\mathbf{I} \equiv \mathbf{C}'_1\mathbf{I}$ remain the same as the ones used in the IND-ENC-CPA case presented above. The conclusion is also the same, mainly that

$$\begin{aligned}\Delta^{\mathbf{A}}(\mathbf{G}_0^{\text{IND-UPD-CPA}}, \mathbf{G}_1^{\text{IND-UPD-CPA}}) &= \Delta^{\mathbf{A}}(\mathbf{C}'_0\mathbf{R}, \mathbf{C}'_1\mathbf{R}) \\ &\leq \Delta^{\mathbf{A}}(\mathbf{C}'_0\mathbf{R}, \mathbf{C}'_0\mathbf{I}) + \underbrace{\Delta^{\mathbf{A}}(\mathbf{C}'_0\mathbf{I}, \mathbf{C}'_1\mathbf{I})}_{=0} + \Delta^{\mathbf{A}}(\mathbf{C}'_1\mathbf{I}, \mathbf{C}'_1\mathbf{R}) \\ &= \Delta^{\mathbf{AC}'_0}(\mathbf{R}, \mathbf{I}) + \Delta^{\mathbf{AC}'_1}(\mathbf{R}, \mathbf{I})\end{aligned}$$

And the IND-UPD-CPA security notion is also necessary to make this construction secure.

Finally, we showed that, in the context of unrestricted leakage, the IND-ENC-CPA and IND-UPD-CPA security notions are together necessary and sufficient to construct a $\mathbf{cUSMR}^+$ from $\mathbf{USMR}^+$ and **UpdKey** using a UE scheme. This means that, for UE schemes, the combination of these two notions is the correct one to consider and aim for in this specific context. In particular, the use of a stronger security notion like IND-UE-CPA does not provide stronger security guarantees to clients in the setting of unrestricted leakage.

At most one leak per entry per epoch: the CCA case

We now consider updatable server-memory resources where the attacker can inject arbitrary messages into the database at its sub-interface S.2. Formally, S has access to the $(\texttt{inject}, i, x)$ request which assigns the value x to the i-th entry of the database $\mathbb{M}[i]$. This **USMR** with injections will be denoted by **iUSMR**, and its confidential variant **ciUSMR**. The resource **iUSMR** is described in Fig. 4.11. The description of resource **ciUSMR** is the same except for the return value of $(\texttt{leak}, i)$ requests which is replaced by $|\mathbb{M}[i]|$.

Injections enhance the power of the distinguisher (which is connected to every interface of the system) in the following manner. Just like before, the distinguisher can use the client interface C to write messages of its choice into the database and then leak a ciphertext associated

to this message by sending a leak request at interface S, essentially building an encryption oracle for itself, which motivated the use of a CPA security notion. In the present case, the distinguisher can also use injections at interface S to write ciphertexts of its choice directly into the database then use the read request at interface C to get the message associated to the injected ciphertext, essentially building a decryption oracle for itself. This new capability of the distinguisher motivates the use of a CCA security notion since it is tailored to deal with situations of the sort.

An important result, originally given by Canetti in [CKN03], is that the CCA security is unnecessarily strict for most applications. We show that it is the case for UE by showing that the CCA security notion is too strong for constructing the **ciUSMR** from the **iUSMR** using a UE scheme. To see this, suppose that we securely construct a **ciUSMR** from an **iUSMR** with an IND-UE-CCA secure UE scheme Π. We build a new UE scheme Π' that works just like Π except it appends a 0 bit at the end of every ciphertext, this bit being ignored when updating and decrypting ciphertexts. The scheme Π' is no longer IND-UE-CCA secure since the adversary can switch the last bit of the challenge to 1 and send it to the decryption oracle to recover the underlying plaintext. However, the scheme Π' can still be used for the secure construction described above. Indeed, we can use a simulator which does the following:

- On $(\texttt{inject}, i, c)$ the simulator replaces the last bit of c with a 0 and injects it at position i. If the last bit was a 1, the simulator can remember it in case it is asked to leak the ciphertext.

With this simulator, the observable behaviors of both the real and the ideal systems are the same as if the scheme Π was used. The construction is thus secure even though the scheme Π' was not IND-UE-CCA secure. This proves that IND-UE-CCA security is too strong for this construction. With this in mind, we focus on studying the weaker RCCA notion of [KLR19] instead.

Resource iUSMR

Sub-Interface S.2 of interface S

Input: $(\texttt{inject}, i, x) \in [n] \times \Sigma$

$\mathbb{M}[i] \leftarrow x$

Figure 4.11: The updatable server-memory resource with injections **iUSMR**. Only differences with **USMR** are shown.

Just like before, let n be the size of the **iUSMR** and take $k \in \{1, \ldots, n\}$. We place ourselves in a time interval where we do not consider the confidentiality of messages other than the k-th one. In this case the simulator $\sigma_{k,\mathsf{CCA}}$ works as follows. It simulates the epoch keys and tokens and, on a $(\texttt{leakKey}, i)$ or $(\texttt{leakToken}, i)$ request, it checks if the event $\mathcal{E}^{\texttt{leaked}}_{\mathsf{Key},i}$ (respectively $\mathcal{E}^{\texttt{leaked}}_{\mathsf{Token},i}$) exists in the Global Event History and leaks the corresponding epoch key (respectively token) to the adversary if it is the case and $\perp$ otherwise.

In the context of unrestricted leakage ((c) **iUSMR**$^+$), the simulator can use the ideal history to identify the database entries that correspond to fresh encryptions or updated ones. Together with its simulated epoch keys and tokens, this allows the simulator to maintain a simulated memory (and a simulated history) where fresh encryptions of $\mathbb{M}[k]$ (in the real world) are replaced with fresh encryptions of random plaintexts of length $|\mathbb{M}[k]|$ and updated ciphertexts encrypting $\mathbb{M}[k]$ (in the real world) are replaced with updates of ciphertexts of random plaintexts

of length $|\mathbb{M}[k]|$. When $i \neq k$, the simulated memory and history perfectly match their real-world counterparts since these messages are not confidential.

In the context of restricted leakage ($(c)\,\mathbf{iUSMR}^1$). On a $(\texttt{leak}, k)$ request, the simulator forwards it to the ideal resource to get a length ℓ and returns a fresh encryption of a random plaintext of length ℓ under the current epoch key. Also, on a $(\texttt{leak}, i)$ request, $i \neq k$, the simulator forwards it to the ideal resource to get a plaintext x and returns a fresh encryption of x under the current epoch key.

In all leakage contexts, on a $(\texttt{leak}, i)$ request, the simulator returns the i-th entry of its simulated memory. Finally, on an $(\texttt{inject}, i, c)$ request, $\sigma_{k,\mathsf{CCA}}$ tries to decrypt c with the current epoch key to get a message m or a decryption error $\perp$. If there is no decryption error, $\sigma_{k,\mathsf{CCA}}$ sends $(\texttt{inject}, i, m)$ to the resource, else it sends $(\texttt{inject}, i, \perp)$ instead. In both cases, the simulator sets c as the i-th entry of its simulated memory.

IND-UE-RCCA is sufficient for a secure construction of $\mathbf{ciUSMR}^1$

Formally, the fact that IND-UE-RCCA is sufficient to construct $\mathbf{ciUSMR}^1$ from $\mathbf{iUSMR}^1$ and **UpdKey** is expressed in Theorem 4.1 through an intersection of specifications. The following theorem shows how we construct each of those specifications when the message space $\mathcal{M}$ is large enough.

Theorem 4.5. *Let Σ be a finite alphabet, $n \in \mathbb{N}$ and $k \in \{1, \ldots, n\}$. The protocol $\mathsf{ue} := (\mathsf{ue}_{\mathsf{cli}}, \mathsf{ue}_{\mathsf{ser}})$ described in Fig. 4.6 and Fig. 4.7, based on a UE scheme with message space $\mathcal{M}$, constructs the $\mathbf{ciUSMR}^1_{\Sigma,n}$ from the basic $\mathbf{iUSMR}^1_{\Sigma,n}$ and **UpdKey** inside the interval $[P_{\texttt{only},k}(\mathcal{E}), P_{\texttt{compromised},k}(\mathcal{E})]$, with respect to the simulator $\sigma_{k,\mathsf{CCA}}$ described in Section 4.3.6. More specifically, we construct reductions such that, for all distinguishers $\mathbf{D}$,*

$$\Delta^{\mathbf{D}^{\mathcal{E}}}(\mathsf{ue}^{\mathsf{C}}_{\mathsf{cli}}\mathsf{ue}^{\mathsf{S}}_{\mathsf{ser}}[\mathbf{iUSMR}^1_{\Sigma,n}, \mathbf{UpdKey}], \sigma^{\mathsf{S}}_{k,\mathsf{CCA}}\,\mathbf{ciUSMR}^1_{\Sigma,n}) \leq$$

$$(2q + r)\Delta^{\mathbf{D}^{\mathcal{E}}}(\mathbf{G}^{\mathsf{IND\text{-}UE\text{-}RCCA}}_{\mathsf{UPD}}, \mathbf{G}^{\mathsf{IND\text{-}UE\text{-}RCCA}}_{\mathsf{ENC}}) + \frac{(q + r + 1)s}{|\mathcal{M}|}$$

where q (resp. r, s) is an upper bound on the number of writes (resp. updates, injections) made by the distinguisher.

Proof. Let $\mathbf{R} := \mathsf{ue}^{\mathsf{C}}_{\mathsf{cli}}\mathsf{ue}^{\mathsf{S}}_{\mathsf{ser}}[\mathbf{iUSMR}^1, \mathbf{UpdKey}]$ be the the real system and $\mathbf{I} := \sigma^{\mathsf{S}}_{k,\mathsf{CCA}}\,\mathbf{ciUSMR}^1$ be the ideal system, where σ_{CCA} is the simulator given in Section 4.3.6. The two systems behave differently when leaking the content of $\mathbb{M}[k]$: $\mathbf{R}$ leaks an encryption of $\mathbb{M}[k]$ while $\mathbf{I}$ leaks an encryption of a random plaintext of length $|\mathbb{M}[k]|$. In order to determine the advantage of a distinguisher in distinguishing $\mathbf{R}$ from $\mathbf{I}$, denoted by $\Delta^{\mathbf{D}}(\mathbf{R}, \mathbf{I})$, we proceed with a sequence of systems.

- Let $\mathbf{S}$ be a resource that behaves just like $\mathbf{R}$ except on query $(\texttt{leak}, k)$ where it leaks an encryption of a random plaintext of length $|\mathbb{M}[k]|$ instead of an encryption of $\mathbb{M}[k]$, if $\mathbb{M}[k]$ contains a fresh encryption and not an updated one. This happens if a query $(\texttt{write}, k, x)$ has been issued by the client in the current epoch. In the case when $\mathbb{M}[k]$ contains an updated version of a ciphertext, the two resources behave in the exact same way. In other words, $\mathbf{S}$ behaves just like $\mathbf{R}$ except when leaking fresh encryptions.

  Let q (resp. s) be an upper bound on the number of $(\texttt{write}, k, \cdot)$ (resp. injection) queries issued to the systems. We define a hybrid resource $\mathbf{H}_i$ that behaves just like $\mathbf{R}$ on the first i $(\texttt{write}, k, \cdot)$ queries and like $\mathbf{S}$ afterwards. Then we define a reduction $\mathbf{C}_i$ that behaves like $\mathbf{H}_i$ except it uses the game $\mathbf{G}^{\mathsf{IND\text{-}ENC\text{-}RCCA}}_b$ oracles instead of doing the UE operations by itself except on the i-th $(\texttt{write}, k, \cdot)$ request (of the form $(\texttt{write}, k, x)$) it challenges the game with

input $(x, \bar{x})$, where $\bar{x}$ is a random plaintext of length $|x|$, to receive the ciphertext $\tilde{c}$. On an $(\texttt{inject}, j, c)$ request, if $c \neq \tilde{c}$, $\mathbf{C}_i$ sends c to the game decryption oracle. If the answer is `invalid`, it assigns the value x to the j-th database entry $\mathbb{M}[j]$. We have

$$\mathbf{R} \equiv \mathbf{H}_q \text{ and } \mathbf{S} \equiv \mathbf{H}_0$$

and, if $\mathbf{D}$ is a distinguisher, we have

$$\Delta^{\mathbf{D}}(\mathbf{C}_i\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}RCCA}}, \mathbf{C}_{i+1}\mathbf{G}_1^{\mathsf{IND\text{-}ENC\text{-}RCCA}}) \leq \frac{s}{|\mathcal{M}|}$$

The two systems behave in the same way on write requests. However, a difference in behaviors comes up after the i-th $(\texttt{write}, k, \cdot)$ request and before the $i+1$-th one. Indeed, in this case, the system $\mathbf{C}_i\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}RCCA}}$ has issued a challenge on $(x, \bar{x})$ but $\mathbf{C}_{i+1}\mathbf{G}_1^{\mathsf{IND\text{-}ENC\text{-}RCCA}}$ has not. This means that, before the $i+1$-th $(\texttt{write}, k, \cdot)$ request, on an $(\texttt{inject}, j, c)$ where c is an encryption of $\bar{x}$, the decryption oracle of $\mathbf{C}_i\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}RCCA}}$ returns `invalid` and x is written at position j. Meanwhile, since there is no challenge yet, the decryption oracle of $\mathbf{C}_{i+1}\mathbf{G}_1^{\mathsf{IND\text{-}ENC\text{-}RCCA}}$ returns $\bar{x}$ and $\bar{x}$ is written at position j. A $(\texttt{read}, j)$ request then leads to a distinction of the two systems. Finding such a ciphertext c with s possible injections happens with probability $s/|\mathcal{M}|$. Moreover, we have

$$\Delta^{\mathbf{D}}(\mathbf{H}_q, \mathbf{C}_q\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}RCCA}}) = \Delta^{\mathbf{D}}(\mathbf{R}, \mathbf{C}_q\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}RCCA}}) \leq \frac{s}{|\mathcal{M}|}$$

This is due to the fact that, after the challenge is issued on the q-th $(\texttt{write}, k, \cdot)$ query, the behavior of the two systems differs after an $(\texttt{inject}, j, c)$ request when $c := \mathsf{Enc}(\bar{x})$, where $\bar{x}$ was the random plaintext used by the reduction during the challenge call. The system $\mathbf{R}$ returns $\bar{x}$ on a subsequent $(\texttt{read}, j)$ request while $\mathbf{C}_q\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}RCCA}}$ returns the challenge plaintext x. Finding such a ciphertext with s injections queries happens with probability $s/|\mathcal{M}|$. We also point out that

$$\Delta^{\mathbf{D}}(\mathbf{C}_0\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}RCCA}}, \mathbf{H}_0) = \Delta^{\mathbf{D}}(\mathbf{C}_0\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}RCCA}}, \mathbf{S}) = 0$$

This is because we are using $\mathbf{C}_0$, and not $\mathbf{C}_q$ like above, so no challenge is issued and the two systems behave in the exact same way.

Now let $\mathbf{C}_I$ be a reduction that samples $i \in [1, q]$ at random and behaves like $\mathbf{C}_i$ and define $\mathbf{D}' := \mathbf{D}\mathbf{C}_I$. We have,

$$\Pr[\mathbf{D}'(\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}RCCA}}) = 1] = \frac{1}{q}\sum_{i=1}^{q} \Pr[\mathbf{D}(\mathbf{C}_i\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}RCCA}}) = 1]$$

and

$$\begin{aligned}\Pr[\mathbf{D}'(\mathbf{G}_1^{\mathsf{IND\text{-}ENC\text{-}RCCA}}) = 1] &= \frac{1}{q}\sum_{i=1}^{q} \Pr[\mathbf{D}(\mathbf{C}_i\mathbf{G}_1^{\mathsf{IND\text{-}ENC\text{-}RCCA}}) = 1] \\ &\leq \frac{s}{|\mathcal{M}|} + \frac{1}{q}\sum_{i=0}^{q-1} \Pr[\mathbf{D}(\mathbf{C}_i\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}RCCA}}) = 1]\end{aligned}$$

Finally, the advantage of the distinguisher in distinguishing system $\mathbf{R}$ from $\mathbf{S}$ is

$$
\begin{aligned}
\Delta^{\mathbf{D}}(\mathbf{R},\mathbf{S}) &= \Delta^{\mathbf{D}}(\mathbf{H}_q,\mathbf{H}_0)\\
&\leq \Delta^{\mathbf{D}}(\mathbf{H}_q,\mathbf{C}_q\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}RCCA}}) + \Delta^{\mathbf{D}}(\mathbf{C}_q\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}RCCA}},\mathbf{C}_0\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}RCCA}})\\
&\quad + \Delta^{\mathbf{D}}(\mathbf{C}_0\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}RCCA}},\mathbf{H}_0)\\
&\leq \Delta^{\mathbf{D}}(\mathbf{C}_q\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}RCCA}},\mathbf{C}_0\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}RCCA}}) + \frac{s}{|\mathcal{M}|}\\
&\leq |\Pr[\mathbf{D}(\mathbf{C}_q\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}RCCA}})=1] - \Pr[\mathbf{D}(\mathbf{C}_0\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}RCCA}})=1]| + \frac{s}{|\mathcal{M}|}\\
&\leq |\sum_{i=1}^{q}\Pr[\mathbf{D}(\mathbf{C}_i\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}RCCA}})=1] - \sum_{i=0}^{q-1}\Pr[\mathbf{D}(\mathbf{C}_i\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}RCCA}})=1]| + \frac{s}{|\mathcal{M}|}\\
&\leq q\cdot|\Pr[\mathbf{D}'(\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}RCCA}})=1] - \Pr[\mathbf{D}'(\mathbf{G}_1^{\mathsf{IND\text{-}ENC\text{-}RCCA}})=1]| + \frac{s(q+1)}{|\mathcal{M}|}\\
&\leq q\cdot\Delta^{\mathbf{D}'}(\mathbf{G}_0^{\mathsf{IND\text{-}ENC\text{-}RCCA}},\mathbf{G}_1^{\mathsf{IND\text{-}ENC\text{-}RCCA}}) + \frac{s(q+1)}{|\mathcal{M}|}
\end{aligned}
$$

• Let us consider the systems $\mathbf{S}$ and $\mathbf{I}$. By definition, $\mathbf{S}$ behaves just like $\mathbf{I}$ except on query $(\texttt{leak},k)$ where it leaks an updated ciphertext of an encryption of $\mathbb{M}[k]$ (instead of a fresh encryption of a random $\bar{x}$ of length $|\mathbb{M}[k]|$ in the ideal system) if $\mathbb{M}[k]$ contains an updated encryption and not a fresh one. In the case when $\mathbb{M}[k]$ contains a fresh ciphertext, the two resources behave in the exact same way. Namely, they leak an encryption of a random $\bar{x}$ of length $|\mathbb{M}[k]|$.

Let r (resp. s) be an upper bound on the number of update (resp. injection) queries issued to the systems. We define a hybrid resource $\mathbf{H}'_i$ that behaves just like $\mathbf{S}$ on the first i update queries and like $\mathbf{I}$ afterwards. Then we define a reduction $\mathbf{C}'_i$ that behaves like $\mathbf{H}'_i$ except it uses the game $\mathbf{G}_{\mathsf{xxx}}^{\mathsf{IND\text{-}UE\text{-}RCCA}}$, where $\mathsf{xxx}\in\{\mathsf{ENC},\mathsf{UPD}\}$, oracles instead of doing the UE operations by itself and on the i-th update computation for the encryption c of $\mathbb{M}[k]$, it challenges the game with input $(\bar{x},c)$ to receive either a fresh encryption of a random $\bar{x}$ or the updated version of c. We have

$$\mathbf{S}\equiv\mathbf{H}'_r \text{ and } \mathbf{I}\equiv\mathbf{H}'_0$$

and, if $\mathbf{D}$ is a distinguisher, we have

$$\Delta^{\mathbf{D}}(\mathbf{C}'_i\mathbf{G}_{\mathsf{UPD}}^{\mathsf{IND\text{-}UE\text{-}RCCA}},\mathbf{C}'_{i+1}\mathbf{G}_{\mathsf{ENC}}^{\mathsf{IND\text{-}UE\text{-}RCCA}})\leq\frac{s}{|\mathcal{M}|}$$

The two systems behave in the same way on update requests. However, a difference in behaviors comes up after the i-th update request of $\mathbb{M}[k]$ and before the $i+1$-th one. Indeed, in this case, the system $\mathbf{C}'_i\mathbf{G}_{\mathsf{UPD}}^{\mathsf{IND\text{-}UE\text{-}RCCA}}$ has issued a challenge on $(\bar{x},c)$ but $\mathbf{C}'_{i+1}\mathbf{G}_{\mathsf{ENC}}^{\mathsf{IND\text{-}UE\text{-}RCCA}}$ has not. This means that, before the $i+1$-th update request of $\mathbb{M}[k]$, on an $(\texttt{inject},j,c')$ request where c' is an encryption of $\bar{x}$, the decryption oracle of $\mathbf{C}'_i\mathbf{G}_0^{\mathsf{IND\text{-}UPD\text{-}RCCA}}$ returns $\texttt{invalid}$ and x (the underlying plaintext of c) is written at position j. Meanwhile, since there is no challenge yet, the decryption oracle of $\mathbf{C}'_{i+1}\mathbf{G}_{\mathsf{ENC}}^{\mathsf{IND\text{-}UE\text{-}RCCA}}$ returns $\bar{x}$ and $\bar{x}$ is written at position j. A $(\texttt{read},j)$ request then leads to a distinction of the two systems. Finding such a ciphertext c' with s possible injections happens with probability $s/|\mathcal{M}|$.

Let $\mathbf{C}'_I$ be a reduction that samples $i\in[1,r]$ at random and behaves like $\mathbf{C}'_i$ and define

$\mathbf{D}'' := \mathbf{D}\mathbf{C}'_I$. We have,

$$\Pr[\mathbf{D}''(\mathbf{G}^{\mathsf{IND\text{-}UE\text{-}RCCA}}_{\mathsf{UPD}}) = 1] = \frac{1}{r}\sum_{i=1}^{r}\Pr[\mathbf{D}(\mathbf{C}'_i\mathbf{G}^{\mathsf{IND\text{-}UE\text{-}RCCA}}_{\mathsf{UPD}}) = 1]$$

and

$$\Pr[\mathbf{D}''(\mathbf{G}^{\mathsf{IND\text{-}UE\text{-}RCCA}}_{\mathsf{ENC}}) = 1] \leq \frac{s}{|\mathcal{M}|} + \frac{1}{r}\sum_{i=0}^{r-1}\Pr[\mathbf{D}(\mathbf{C}'_i\mathbf{G}^{\mathsf{IND\text{-}UE\text{-}RCCA}}_{\mathsf{UPD}}) = 1]$$

This time, since the injection behaviors of both systems are the same, we have

$$\mathbf{H}'_r \equiv \mathbf{C}'_r\mathbf{G}^{\mathsf{IND\text{-}UE\text{-}RCCA}}_{\mathsf{UPD}} \quad \text{and} \quad \mathbf{H}'_0 \equiv \mathbf{C}'_0\mathbf{G}^{\mathsf{IND\text{-}UE\text{-}RCCA}}_{\mathsf{UPD}}$$

Finally, the advantage of the distinguisher in distinguishing system $\mathbf{S}$ from $\mathbf{I}$ is

$$\begin{aligned}
\Delta^{\mathbf{D}}(\mathbf{S},\mathbf{I}) &= \Delta^{\mathbf{D}}(\mathbf{H}'_r,\mathbf{H}'_0)\\
&= \Delta^{\mathbf{D}}(\mathbf{C}'_r\mathbf{G}^{\mathsf{IND\text{-}UE\text{-}RCCA}}_{\mathsf{UPD}}, \mathbf{C}'_0\mathbf{G}^{\mathsf{IND\text{-}UE\text{-}RCCA}}_{\mathsf{UPD}})\\
&= |\Pr[\mathbf{D}(\mathbf{C}'_r\mathbf{G}^{\mathsf{IND\text{-}UE\text{-}RCCA}}_{\mathsf{UPD}}) = 1] - \Pr[\mathbf{D}(\mathbf{C}'_0\mathbf{G}^{\mathsf{IND\text{-}UE\text{-}RCCA}}_{\mathsf{UPD}}) = 1]|\\
&= |\sum_{i=1}^{r}\Pr[\mathbf{D}(\mathbf{C}'_i\mathbf{G}^{\mathsf{IND\text{-}UE\text{-}RCCA}}_{\mathsf{UPD}}) = 1] - \sum_{i=0}^{r-1}\Pr[\mathbf{D}(\mathbf{C}'_i\mathbf{G}^{\mathsf{IND\text{-}UE\text{-}RCCA}}_{\mathsf{UPD}}) = 1]|\\
&\leq r\cdot|\Pr[\mathbf{D}''(\mathbf{G}^{\mathsf{IND\text{-}UE\text{-}RCCA}}_{\mathsf{UPD}}) = 1] - \Pr[\mathbf{D}''(\mathbf{G}^{\mathsf{IND\text{-}UE\text{-}RCCA}}_{\mathsf{ENC}}) = 1]| + \frac{rs}{|\mathcal{M}|}\\
&\leq r\cdot\Delta^{\mathbf{D}''}(\mathbf{G}^{\mathsf{IND\text{-}UE\text{-}RCCA}}_{\mathsf{UPD}}, \mathbf{G}^{\mathsf{IND\text{-}UE\text{-}RCCA}}_{\mathsf{ENC}}) + \frac{rs}{|\mathcal{M}|}
\end{aligned}$$

• We use the triangular inequality to conclude. Let q, r and s be upper bounds on the number of writes, updates and injections, respectively. The advantage of the distinguisher in distinguishing the real system $\mathbf{R}$ from the ideal one $\mathbf{I}$ is

$$\begin{aligned}
\Delta^{\mathbf{D}}(\mathbf{R},\mathbf{I}) &\leq \Delta^{\mathbf{D}}(\mathbf{R},\mathbf{S}) + \Delta^{\mathbf{D}}(\mathbf{S},\mathbf{I})\\
&\leq q\cdot\Delta^{\mathbf{D}'}(\mathbf{G}^{\mathsf{IND\text{-}ENC\text{-}RCCA}}_{0}, \mathbf{G}^{\mathsf{IND\text{-}ENC\text{-}RCCA}}_{1}) +\\
&\quad r\cdot\Delta^{\mathbf{D}''}(\mathbf{G}^{\mathsf{IND\text{-}UE\text{-}RCCA}}_{\mathsf{UPD}}, \mathbf{G}^{\mathsf{IND\text{-}UE\text{-}RCCA}}_{\mathsf{ENC}}) + \frac{(q+r+1)s}{|\mathcal{M}|}\\
&\leq 2q\cdot\Delta^{\mathbf{D}'\mathbf{C}''}(\mathbf{G}^{\mathsf{IND\text{-}UE\text{-}RCCA}}_{\mathsf{UPD}}, \mathbf{G}^{\mathsf{IND\text{-}UE\text{-}RCCA}}_{\mathsf{ENC}}) +\\
&\quad r\cdot\Delta^{\mathbf{D}''}(\mathbf{G}^{\mathsf{IND\text{-}UE\text{-}RCCA}}_{\mathsf{UPD}}, \mathbf{G}^{\mathsf{IND\text{-}UE\text{-}RCCA}}_{\mathsf{ENC}}) + \frac{(q+r+1)s}{|\mathcal{M}|}\\
&\leq (2q+r)\cdot\Delta^{\mathcal{D}}(\mathbf{G}^{\mathsf{IND\text{-}UE\text{-}RCCA}}_{\mathsf{UPD}}, \mathbf{G}^{\mathsf{IND\text{-}UE\text{-}RCCA}}_{\mathsf{ENC}}) + \frac{(q+r+1)s}{|\mathcal{M}|}
\end{aligned}$$

Where we use the notation $\Delta^{\mathcal{D}}(\mathbf{X},\mathbf{Y}) := \sup_{\mathbf{D}\in\mathcal{D}}\Delta^{\mathbf{D}}(\mathbf{X},\mathbf{Y})$ and where the reduction $\mathbf{C}''$ is a straightforward adaptation, in the RCCA case, of the one given by Boyd *et al.* in [BDGJ20, Theorem 2.4] to prove the following in the CCA case:

Proposition 4.6. *Let* Π *be a UE scheme. For any* ENC-CCA *adversary* $\mathbf{A}$ *against* Π*, there exists a reduction* $\mathbf{C}''$ *such that*

$$\Delta^{\mathbf{A}}(\mathbf{G}^{\mathsf{ENC\text{-}CCA}}_{0}, \mathbf{G}^{\mathsf{ENC\text{-}CCA}}_{1}) \leq 2\cdot\Delta^{\mathbf{A}\mathbf{C}''}(\mathbf{G}^{\mathsf{IND\text{-}UE\text{-}CCA}}_{\mathsf{ENC}}, \mathbf{G}^{\mathsf{IND\text{-}UE\text{-}CCA}}_{\mathsf{UPD}})$$

□

IND-UE-RCCA is necessary for a secure construction of ciUSMR^1

Proof. We show that the IND-UE-RCCA security notion is necessary to securely construct the **ciUSMR**^{1} from a **iUSMR**^{1} using a UE scheme. Keeping our notations, we will start from the real system $\mathbf{R} := \mathsf{ue}^{\mathsf{C}}_{\mathsf{cli}}\mathsf{ue}^{\mathsf{S}}_{\mathsf{ser}}\,\mathbf{iUSMR}^1$ and the ideal one $\mathbf{I} := \sigma^{\mathsf{S}}\,\mathbf{ciUSMR}^1$, where this time an arbitrary simulator σ is used, and use reductions to construct the resources $\mathbf{G}^{\mathsf{IND\text{-}UE\text{-}RCCA}}_{\mathsf{xxx}}$ implementing the IND-UE-RCCA games. Formally, we will describe two reductions $\mathbf{C}_0$ and $\mathbf{C}_1$ such that

$$\mathbf{C}_0\mathbf{R} \equiv \mathbf{G}^{\mathsf{IND\text{-}UE\text{-}RCCA}}_{\mathsf{ENC}},\ \mathbf{C}_1\mathbf{R} \equiv \mathbf{G}^{\mathsf{IND\text{-}UE\text{-}RCCA}}_{\mathsf{UPD}} \text{ and } \mathbf{C}_0\mathbf{I} \equiv \mathbf{C}_1\mathbf{I}$$

Let $\mathbf{A}$ be an adversary against the IND-UE-RCCA security notion. Using the triangular inequality, we have

$$\begin{aligned}\Delta^{\mathbf{A}}(\mathbf{G}^{\mathsf{IND\text{-}UE\text{-}RCCA}}_{\mathsf{ENC}}, \mathbf{G}^{\mathsf{IND\text{-}UE\text{-}RCCA}}_{\mathsf{UPD}}) &= \Delta^{\mathbf{A}}(\mathbf{C}_0\mathbf{R}, \mathbf{C}_1\mathbf{R}) \\ &\leq \Delta^{\mathbf{A}}(\mathbf{C}_0\mathbf{R}, \mathbf{C}_0\mathbf{I}) + \Delta^{\mathbf{A}}(\mathbf{C}_0\mathbf{I}, \mathbf{C}_1\mathbf{I}) + \\ &\quad \Delta^{\mathbf{A}}(\mathbf{C}_1\mathbf{I}, \mathbf{C}_1\mathbf{R}) \\ &= \Delta^{\mathbf{AC}_0}(\mathbf{R}, \mathbf{I}) + \Delta^{\mathbf{AC}_1}(\mathbf{R}, \mathbf{I})\end{aligned}$$

• We are going to describe two reductions $\mathbf{C}_0$ and $\mathbf{C}_1$ that connect to a **iUSMR**^{1} (or **ciUSMR**^{1}) resource and implement the oracles of the $\mathbf{G}^{\mathsf{IND\text{-}UE\text{-}RCCA}}_{\mathsf{xxx}}$ games. Let n (resp. q) be an upper bound on the number of encryption (resp. decryption) queries issued to the games by the adversary. In the following, we consider resources of memory size $n + q + 1$. This is done to have enough space to store the ciphertexts returned by the encryption oracle, the ciphertexts sent to the decryption oracle, as well as the challenge plaintext in case the challenger ask for updates to the update oracle later on. We use the same reductions as in the CPA case, the only difference being the addition of the decryption oracle. Since its behavior depends on the challenge oracle, we also describe this oracle below.

- On $\mathcal{O}.\mathsf{Chall}(m_0, c_1)$: Both reductions check that c_1 was present in memory in the previous epoch and that its updated version has not been leaked yet in the current epoch. They also check that $m_1 := \mathsf{Dec}_k(c_1)$ and m_0 have the same length, this can be done with a `read` request at interface C. They return $\perp$ if it is not the case. Otherwise, the reductions keep m_0 and m_1 in memory and let i be the position where c_1 was written. The reductions then proceed like this:
  - $\mathbf{C}_0$ sends $(\texttt{write}, n + q + 1, m_0)$ at interface C then it sends $(\texttt{leak}, n + q + 1)$ at interface S and outputs the result.
  - $\mathbf{C}_1$ sends $(\texttt{leak}, i)$ at interface S and outputs the result.
- On the j-th $\mathcal{O}.\mathsf{Dec}(c)$:
  - Before the challenge call, $\mathbf{C}_b$ sends an $(\texttt{inject}, n+j, c)$ request at interface S. Then, it sends a $(\texttt{read}, n + j)$ request at interface C to get a result m and returns m.
  - After the challenge call, $\mathbf{C}_b$ sends the same requests as before to get the result m. Only this time, if $m \in \{m_0, m_1\}$ they return `invalid` and if not they return m.

Moreover, the reductions have all the information to check for trivial wins. When connected to $\mathbf{R}$, the reductions $\mathbf{C}_b$ exactly implements the games $\mathbf{G}^{\mathsf{IND\text{-}UE\text{-}RCCA}}_{\mathsf{xxx}}$. Their observable behaviors being the same, we have $\mathbf{C}_0\mathbf{R} \equiv \mathbf{G}^{\mathsf{IND\text{-}UE\text{-}RCCA}}_{\mathsf{ENC}}$ and $\mathbf{C}_1\mathbf{R} \equiv \mathbf{G}^{\mathsf{IND\text{-}UE\text{-}RCCA}}_{\mathsf{UPD}}$.

Now, if we connect the reductions to the ideal system $\mathbf{I}$, where an arbitrary simulator σ is used, their behaviors can only differ on the challenge call. When asked for the challenge on the pair (m_0, c_1), reduction $\mathbf{C}_0$ writes m_0 at position $n+q+1$ then it outputs the result of $(\texttt{leak}, n+q+1)$ meanwhile $\mathbf{C}_1$ outputs the result of $(\texttt{leak}, i)$ where i is the position where the ciphertext c_1 was already written. In the ideal system $\mathbf{I}$, the result of a `leak` request is always the length of the message stored. Since, letting m_1 be the underlying plaintext of c_1, the reductions checked that the two messages m_0 and m_1 had the same length, the inputs of the simulator σ are the same and the behaviors are thus identical. We can check that we never go beyond the leakage condition, it is never the case that a reduction leaks the content of a position j at more than one occasion in a given epoch. We conclude that the systems $\mathbf{C}_b\mathbf{I}$ are indistinguishable from one another. Written differently, we have $\mathbf{C}_0\mathbf{I} \equiv \mathbf{C}_1\mathbf{I}$ and the theorem is proven in the IND-UE-RCCA case. □

In the context of restricted leakage, we proved that the IND-UE-RCCA security notion for UE schemes is the correct one to consider if clients want to protect the confidentiality and the age of the data they store on a remote server in the presence of an adversary who can inject arbitrary data and is only able to leak the contents of the server at most once between each key update. Very similarly to the CPA case, we can extend these results to the $\mathbf{ciUSMR}^+$ construction in the unrestricted leakage context to prove that the IND-ENC-RCCA + IND-UPD-RCCA notion is necessary and sufficient for this construction.

4.4 A composable and unified treatment of Private Information Retrieval

4.4.1 Our modelization of Private Information Retrieval

In this section, the **ISMR** is renamed **DB** to model PIR. The resource **DB** is described in Fig. 4.12. It has the following interfaces:

- Interface C_0: The user (or client) can use this interface to initialize the state of the database. When the user is done with this interface, he can send the `initComplete` request which sets the ACTIVE boolean to `true`. This turns off the interface C_0 and unlocks all the others interfaces of the resource. This interface is useful to model PIR protocols working on encrypted or encoded data.
- Interface C: The user main interface. We rename `askInteraction` into $(\texttt{query}, q)$ which sends a query q to the resource. When the servers all respond, the user can also use `reconstruct` to recover their answers.
- Interface S: The interface of the server, who will adopt a semi-honest (also called honest-but-curious) adversarial behavior. This means that the server accepts to plug a converter in its interface to handle `answer` requests (we renamed `interact` to `answer`) according to the PIR protocol. But the server will also try to use the information it has access to, mainly the requests of the clients (through `getQuery` requests), the content of the database (through `read` requests) and a log file of all honest actions (through `getHist`), in order to break the privacy guarantees of the clients.

For simplicity we will present a database that can only process a single query and answer. To construct a database with multiples queries, one can compose multiple single-use databases in parallel or parameterize the database with the number of query treated. In order to give security

guarantees for an unbounded-query database, one has to show that the security guarantees hold for any number of queries. We also limit the number of clients to one for simplicity but, as most PIR protocols support an arbitrary number of clients, it is also possible to increase the number of clients by duplicating the interface C or by letting each client connect to the same interface. We will give a generalization of **DB** to multiple servers later on.

Resource $\mathrm{DB}_{\Sigma,n}$

Initialization

INIT, ACTIVE $\leftarrow$ `false`, $\mathbb{M} \leftarrow [\,]$, HIST $\leftarrow [\,]$, $q \leftarrow \perp$, $a \leftarrow \perp$

Interface C_0

Input: $(\texttt{init}, \mathbb{M}') \in \Sigma^n$
  if not INIT
    $\mathbb{M} \leftarrow \mathbb{M}'$
    HIST $\leftarrow$ HIST $\|$ $(0, \texttt{init})$
    INIT $\leftarrow$ `true`

Input: $(\texttt{read}, i) \in [1, n]$
  if INIT **and not** ACTIVE
    HIST $\leftarrow$ HIST $\|$ $(0, \texttt{R}, i)$
    return $\mathbb{M}[i]$

Input: $(\texttt{write}, i, x) \in [1, n] \times \Sigma$
  if INIT **and not** ACTIVE
    HIST $\leftarrow$ HIST $\|$ $(0, \texttt{W}, i, x)$
    $\mathbb{M}[i] \leftarrow x$

Input: `initComplete`
  if INIT **and not** ACTIVE
    ACTIVE $\leftarrow$ `true`

Interface C

Input: $(\texttt{query}, q')$
  if ACTIVE
    if $q = \perp$
      $q \leftarrow q'$
      HIST $\leftarrow$ HIST $\|$ $(\texttt{query}, q)$

Input: `reconstruct`
  if ACTIVE
    if $a \neq \perp$
      return a

Interface Interface S

Input: $(\texttt{answer}, a')$
  if $q \neq \perp$ **and** $a = \perp$
    $a \leftarrow a'$
    HIST $\leftarrow$ HIST $\|$ $(\texttt{answer}, a)$

Input: $(\texttt{read}, i) \in [1, n]$
  return $\mathbb{M}[i]$

Input: `getQuery`
  return q

Input: `getHist`
  return HIST

Figure 4.12: Our **ISMR** viewed as a basic interactive database **DB** with size n and alphabet Σ.

In order to construct a stronger database from the basic **DB**, one can use a PIR protocol $(\mathcal{Q}, \mathcal{A}, \mathcal{R})$ and use converters on interfaces C and S. The client converter is described in Fig. 4.13 and the server converter in Fig. 4.14. Note that the client converter $\mathsf{pir}_{\mathsf{cli}}$ modifies the request `query` by changing its parameter to be an index $i \in [1, n]$ corresponding to the database entry he wishes to retrieve. The server converter does the same by modifying the `answer` request to have no parameter at all, the converter taking care of computing an answer using the $\mathcal{A}$ algorithm of the PIR protocol. Moreover, the $\mathsf{pir}_{\mathsf{ser}}$ converter makes sure that the `answer` request produces well formed answers for the clients but this converter does not alter the behavior of `getQuery` and `read` requests at interface S. This means that those requests can still be used by the server to gather information and try to find out the index i of interest for the client.

A converter can also be used on interface C_0 if the database content needs to be modified in any way. For example, a converter on C_0 can be used to encode the database if the PIR protocol $(\mathcal{Q}, \mathcal{A}, \mathcal{R})$ relies on an error correcting code.

Converter $\mathsf{pir}_{\mathsf{cli}}$

Initialization

$ind \leftarrow \perp,\ \ s \leftarrow \perp$

Interface `out`

Input: $(\texttt{query}, i) \in [1, n]$

if $ind = \perp$

$s \xleftarrow{\$} \mathcal{S}$

$ind \leftarrow i$

$q \leftarrow \mathcal{Q}(ind, s)$

output $(\texttt{query}, q)$ at interface C of **DB**

Input: `reconstruct`

output `reconstruct` at interface C of **DB**

Let a be the result

if $a \neq \perp$

return $\mathcal{R}(a, ind, s)$

Figure 4.13: Description of the client converter $\mathsf{pir}_{\mathsf{cli}}$ for a PIR protocol $(\mathcal{Q}, \mathcal{A}, \mathcal{R})$.

Converter $\mathsf{pir}_{\mathsf{ser}}$

Interface `out`

Input: `answer`

output `getQuery` at interface S of **DB**

Let q be the result

if $q \neq \perp$

Retrieve $\mathbb{M}$ with `read` requests at interface S

$a \leftarrow \mathcal{A}(\mathbb{M}, q)$

output $(\texttt{answer}, a)$ at interface S of **DB**

Figure 4.14: Description of the server converter $\mathsf{pir}_{\mathsf{ser}}$ for a PIR protocol $(\mathcal{Q}, \mathcal{A}, \mathcal{R})$.

We introduce a new database resource **PrivDB** with stronger security guarantees that we hope to achieve with a PIR protocol. In **DB**, the adversary had access to the queries via the `getQuery` request and had also access to a log file which contained the query, the answer and the possible requests sent by C_0 during the initialization of the resource. In order to ensure privacy for the client's queries, the database **PrivDB** will hide those queries and answers from the adversary. Since the queries and answers are not available to the adversary in **PrivDB** they should be useless to him in the real world and in particular they should not help him get any information about the index i desired by the user.

Furthermore, we will also require the resource **PrivDB** to be "correct" in the sense that when sending the `reconstruction` request (after a query and an answer), the user gets back the database record he queried. The resource **PrivDB** is given in Fig. 4.15.

Resource PrivDB$_{\Sigma,n}$

Initialization

INIT, ACTIVE $\leftarrow$ `false`, $\mathbb{M} \leftarrow [\,]$, HIST $\leftarrow [\,]$, $q \leftarrow \perp, a \leftarrow \perp, ind \leftarrow \perp$

Interface C

Input: $(\texttt{query}, i) \in [1, n]$
  if ACTIVE
    if $q = \perp$
      $q \leftarrow \texttt{ok}$
      HIST $\leftarrow$ HIST $||$ (`query`)
      $ind \leftarrow i$

Input: `reconstruct`
  if ACTIVE
    if $a \neq \perp$
      return $\mathbb{M}[ind]$[a]

Interface S

Input: `answer`
  if ACTIVE
    if $q \neq \perp$ **and** $a = \perp$
      $a \leftarrow \texttt{ok}$
      HIST $\leftarrow$ HIST $||$ (`answer`)

Input: $(\texttt{read}, i) \in [1, n]$
  return $\mathbb{M}[i]$

Input: `getQuery`
  return q[b]

Input: `getHist`
  return HIST

[a] If the server has the capability to overwrite data, we have to return the *ind*-th record of the database as it was when the answer was computed.

[b] The two possible values for q are $\perp$ and `ok`.

Figure 4.15: The private database **PrivDB** where queries and answers are useless to an adversary (privacy) and where the client gets the item he queried (correctness). Interface C_0 remains unchanged.

The construction is valid if the resource **DB** with the PIR protocol is indistinguishable from the resource **PrivDB** with a simulator. The simulator is here to simulate the real world behavior from the adversary point of view. The simulator is thus plugged in the S interface, processing and choosing the return values of each request of the adversary at this interface. Here, the goal of the simulator is twofold.

First, it has to simulate a real-world history with an ideal one. Recall that the ideal history does not have any query or answer in it and that the simulator has no way to know which index i is wanted by the user. The simulation will then be as follows: the simulator will choose an index i' uniformly at random and compute a query q and an answer a for this index using the algorithms $\mathcal{Q}$ and $\mathcal{A}$. It will then replace the ideal query and answer of the ideal history with the simulated query and answer.

The second goal of the simulator is to maintain a simulated database if need be. Indeed,

since the adversary can read the database using $(\texttt{read}, i)$ requests at interface S and since there is no protocol on the ideal resource, the simulator has to apply any transformation that the protocol could have used on the database records. For example, the simulator has to encode the ideal database content if an error correcting code is used by the real world protocol or encrypt the database if the protocol has done so in the real world. Then on a $(\texttt{read}, i)$ request to the ideal resource, the simulator returns the i-th record of the simulated database. The simulator is given in full detail in Fig. 4.18.

We claim that telling apart the real view from the simulated one is as hard as attacking the privacy property of the PIR protocol. Indeed, the `query` observable behavior at interface C is the same in both worlds and the same holds for `answer`. For `reconstruction` the real world has to match the ideal one where it always returns the database record queried by the user. This is the case if the PIR protocol is correct. At interface S, the difference between the real and the simulated history is that the query and answer of the real one are computed using the index chosen by the client whereas in the simulated history they are computed using a random index. For a distinguisher to tell if a query and answer pair was computed using the index i he chose (since it has access to the client interface) or a different index, he would have to learn some information about the index used in this execution of the PIR protocol (mainly if it was computed using i or not) thus breaking the privacy property. For the `read` request available at the S interface, the simulator knows which protocol was used at the interface C_0 to setup the database (with error correcting codes, encryption, ...) and can thus perfectly simulate the real database by applying the same transformation to the ideal database (which he has access to thanks to `read` requests).

We can conclude that if a correct and private PIR protocol is used in the construction, the advantage of the distinguisher in distinguishing the real world from the ideal world is no greater than the advantage of an attacker in telling apart a pair (q, a) of the protocol's query and answer computed using an index i chosen by the attacker from a pair computed using a random index.

Theorem 4.7. *Let Σ be a finite alphabet and $n \in \mathbb{N}$. The protocol $\mathsf{pir} := (\mathsf{pir}_{\mathsf{cli}}, \mathsf{pir}_{\mathsf{ser}})$ described in Fig. 4.13 and Fig. 4.14 based on a PIR scheme $(\mathcal{Q}, \mathcal{A}, \mathcal{R})$ constructs the private database $\mathbf{PrivDB}_{\Sigma,n}$ from the basic database $\mathbf{DB}_{\Sigma,n}$, with respect to the simulator $\mathsf{sim}^{\mathcal{Q},\mathcal{A},\mathcal{R}}_{Priv}$ as defined in Fig. 4.18 and the dummy converter honDB (that disables any adversarial behavior). More specifically, we construct a reduction $\mathbf{C}$ such that, for all distinguishers $\mathbf{D}$,*

$$\Delta^{\mathbf{D}}(\mathsf{honDB}^{\mathsf{S}}\mathsf{pir}^{\mathsf{C}}_{\mathsf{cli}}\mathsf{pir}^{\mathsf{S}}_{\mathsf{ser}}\,\mathbf{DB}_{\Sigma,n}, \mathsf{honDB}^{\mathsf{S}}\,\mathbf{PrivDB}_{\Sigma,n}) = 0$$

$$\text{and } \Delta^{\mathbf{D}}(\mathsf{pir}^{\mathsf{C}}_{\mathsf{cli}}\mathsf{pir}^{\mathsf{S}}_{\mathsf{ser}}\,\mathbf{DB}_{\Sigma,n}, (\mathsf{sim}^{\mathcal{Q},\mathcal{A},\mathcal{R}}_{Priv})^{\mathsf{S}}\,\mathbf{PrivDB}_{\Sigma,n}) = \Delta^{\mathbf{DC}}(\mathbf{G}_0, \mathbf{G}_1)^2$$

where the privacy games $\mathbf{G}_b$ are described in Fig. 4.16.

Proof. We need to find a reduction $\mathbf{C}$ such that $\mathbf{CG}_0 \equiv \mathsf{pir}^{\mathsf{C}}_{\mathsf{cli}}\mathsf{pir}^{\mathsf{S}}_{\mathsf{ser}}\,\mathbf{DB}_{\Sigma,n}$ and $\mathbf{CG}_1 \equiv (\mathsf{sim}^{\mathcal{Q},\mathcal{A},\mathcal{R}}_{Priv})^{\mathsf{S}}\,\mathbf{PrivDB}_{\Sigma,n}$. Then, for all distinguishers $\mathbf{D}$, we will have that the advantage of $\mathbf{D}$ in distinguishing the real system from the ideal one is

$$\begin{aligned}\Delta^{\mathbf{D}}(\mathsf{pir}^{\mathsf{C}}_{\mathsf{cli}}\mathsf{pir}^{\mathsf{S}}_{\mathsf{ser}}\,\mathbf{DB}_{\Sigma,n}, (\mathsf{sim}^{\mathcal{Q},\mathcal{A},\mathcal{R}}_{Priv})^{\mathsf{S}}\,\mathbf{PrivDB}_{\Sigma,n}) &= \Delta^{\mathbf{D}}(\mathbf{CG}_0, \mathbf{CG}_1)\\ &= \Delta^{\mathbf{DC}}(\mathbf{G}_0, \mathbf{G}_1)\end{aligned}$$

which is the result we are looking for. The reduction $\mathbf{C}$ is given in Fig. 4.17.

This reduction, when connected to the game $\mathbf{G}_0$, exhibits the exact same behavior as the real system $\mathbf{DB}$ equipped with the protocol, and, when connected to the game $\mathbf{G}_1$, exhibits the exact same behavior as the ideal system $\mathbf{PrivDB}$ equipped with the simulator. □

[2] If the PIR protocol is IT-secure, then this advantage is 0 for all distinguishers.

Game G_b

Initialization

$\mathbb{M} \leftarrow [\,]$, $n \leftarrow \perp$

Interface out

Input: $(\texttt{init}, \mathbb{M}')$

$\mathbb{M} \leftarrow \mathbb{M}'$

$n \leftarrow |\mathbb{M}|$

Input: $(\texttt{chall}, i)$

$s \xleftarrow{\$} \mathcal{S}$

if $b = 0$

$j \leftarrow i$

else

$j \xleftarrow{\$} [n]$

$q \leftarrow \mathcal{Q}(j, s)$

$a \leftarrow \mathcal{A}(\mathbb{M}, q)$

return (q, a)

Figure 4.16: The privacy games for the PIR protocol $(\mathcal{Q}, \mathcal{A}, \mathcal{R})$.

Note that the proof is the same whether the PIR protocol is computationally secure or information-theoretically secure. In the first case, we consider efficient (*i.e.* polynomial time) distinguishers while in the other one we consider all possible distinguishers. In both cases, we construct the same private database **PrivDB**, with distance (*i.e.* best distinguishing advantage) 0 with respect to all possible distinguishers in the IT setting and with a non-zero distance, linked to the underlying computational assumption, with respect to only efficient distinguishers in the computational setting.

Remark 13. Since the random string s does not leak in any way to the distinguisher or the curious server (which is accurate since it is known only to the client), the distinguisher cannot use the reconstruction algorithm $\mathcal{R}$ to tell if the query and answer pair he observes really comes from the index i he chose or not.

The efficiency of the PIR protocol is given by: $|q| + |a|$ for its communication complexity, and its computational complexity is given by the sum of those of $\mathcal{Q}$, $\mathcal{A}$ and $\mathcal{R}$.

4.4.2 Generalization to multiple servers

We present **MultDB**, a generalization of **DB** in the multiple server case. For the sake of simplicity, we put the same database on all servers but it is possible to modify the behavior of interface C_0 to allow the client to modify each database independently. Similarly, we generalize the converters into $\mathsf{mult\_pir}_{\mathsf{cli}}$ and $\mathsf{mult\_pir}_{\mathsf{ser}_j}$ in the multiple server setting. Those converters are described in Fig. 4.20 and Fig. 4.21. The three algorithms $\mathcal{Q}, \mathcal{A}$ and $\mathcal{R}$ are still present and used in the same way as before.

In the multi-server case, we introduce a threshold t which allows the curious servers to form a coalition of size at most t in order to share the capabilities of their interfaces. This means that each server in the coalition can see the histories of the other members as well as read their memory. The database $\mathbf{MultDB}_{n,k,t}$ can be configured by choosing n the size of the database,

Reduction C

Initialization

INIT, ACTIVE, QUERIED, ANSWERED $\leftarrow$ `false`

$\mathbb{M} \leftarrow [\,], \text{HIST} \leftarrow [\,], q \leftarrow \perp, a \leftarrow \perp, ind \leftarrow \perp$

Interface Interface C_0

Input: $(\texttt{init}, \mathbb{M}') \in \Sigma^n$
  if not INIT
    $\mathbb{M} \leftarrow \mathbb{M}'$
    HIST $\leftarrow$ HIST $\|$ $(0, \texttt{init})$
    INIT $\leftarrow$ `true`

Input: $(\texttt{read}, i) \in [1, n]$
  if INIT **and not** ACTIVE
    HIST $\leftarrow$ HIST $\|$ $(0, \texttt{R}, i)$
    return $\mathbb{M}[i]$

Input: $(\texttt{write}, i, x) \in [1, n] \times \Sigma$
  if INIT **and not** ACTIVE
    HIST $\leftarrow$ HIST $\|$ $(0, \texttt{W}, i, x)$
    $\mathbb{M}[i] \leftarrow x$

Input: `initComplete`
  if INIT **and not** ACTIVE
    ACTIVE $\leftarrow$ `true`
    Send $(\texttt{init}, \mathbb{M})$ at interface `out` of $\mathbf{G}_b$

Interface C

Input: $(\texttt{query}, i)$
  if ACTIVE **and not** QUERIED
    QUERIED $\leftarrow$ `true`
    $(q, a) \leftarrow (\texttt{chall}, i)$ at interface `out` of $\mathbf{G}_b$
    $ind \leftarrow i$
    HIST $\leftarrow$ HIST $\|$ $(\texttt{query}, q)$

Input: `reconstruct`
  if ACTIVE **and** ANSWERED
    return $\mathbb{M}[ind]$

Interface S

Input: `answer`
  if QUERIED
    ANSWERED $\leftarrow$ `true`
    HIST $\leftarrow$ HIST $\|$ $(\texttt{answer}, a)$

Input: $(\texttt{read}, i) \in [1, n]$
  return $\mathbb{M}[i]$

Input: `getQuery`
  if QUERIED
    return q
  else
    return $\perp$

Input: `getHist`
  return HIST

Figure 4.17: The reduction $\mathbf{C}$ connected to a privacy game $\mathbf{G}_b$.

Simulator $\mathsf{sim}_{Priv}^{\mathcal{Q},\mathcal{A},\mathcal{R}}$

Initialization

$H \leftarrow [\,], \mathbb{M}_{sim} \leftarrow [\,], pos \leftarrow 1,\ s \leftarrow \perp, ind \leftarrow \perp, q \leftarrow \perp, a \leftarrow \perp$

Interface Interface S

Input: `answer`
  UPDATE()
  return a

Input: `getHist`
  UPDATE()
  return H

Input: `getQuery`
  UPDATE()
  return q

Input: $(\texttt{read}, i)$
  UPDATE()
  return $\mathbb{M}_{sim}[i]$

procedure UPDATE
  $\mathbb{M}' \leftarrow [(\texttt{read}, i)$ to **PrivDB for** $i = 1 \ldots n]$
  $\mathbb{M}_{sim} \leftarrow \mathcal{P}_{\mathsf{C}_0}(\mathbb{M}')$
  HIST $\leftarrow$ `getHist` to **PrivDB**
  for $j = pos \ldots |\text{HIST}|$ **do**
    if HIST$[j] = (\texttt{query})$
      $ind \xleftarrow{\$} [1, n]$
      $s \xleftarrow{\$} \mathcal{S}$
      $q \leftarrow \mathcal{Q}(ind, s)$
      $H \leftarrow H \,||\, (\texttt{query}, q)$
    else if HIST$[j] = (\texttt{answer})$
      $a \leftarrow \mathcal{A}(\mathbb{M}, q)$
      $H \leftarrow H \,||\, (\texttt{answer}, a)$
    else
      $H \leftarrow H \,||$ HIST$[j]$
    $pos \leftarrow pos + 1$

Figure 4.18: The simulator for the protocol composed of $(\mathcal{Q}, \mathcal{A}, \mathcal{R})$. $\mathcal{P}_{\mathsf{C}_0}$ denotes the transformation applied to the memory $\mathbb{M}$ by a possible converter plugged in interface C_0.

k the number of servers holding this database (or a modified/distributed version of it) and t the largest number of malicious servers allowed to cooperate. This resource is described in Fig. 4.19.

We can then define a new resource with better security guarantees than those of **MultDB**. Just like in the one server case, the new resource **PrivMultDB** requires that `reconstruct` returns the database record desired by the client when he issued its `query` request. **PrivMultDB** also requires that the curious servers S_j do not get to see the queries and answers when using `getHist`. This resource is described in Fig. 4.22. For the construction of $\mathbf{PrivMultDB}_{n,k,t}$ to hold, we require the PIR protocol used to be both correct and t-private.

4.4.3 Instantiations in the multi-server case

We give two known examples of PIR schemes that construct **PrivMultDB** from **MultDB**. The first scheme is based on Locally Decodable Codes (LDC). The second uses Shamir's secret sharing [Sha79]. The focus of this section is not to explain LDCs or secret sharing but rather to showcase the power and the flexibility of our construction.

Using Locally Decodable Codes

Given a message $\mathbf{m} := (m_1, \ldots, m_n)$ encoded into a codeword $\mathbf{c}$ and an index $i \in [1, n]$, an LDC permits to recover m_i while only reading a sublinear number of symbols of $\mathbf{m}$. Formally, LDCs can be defined by a straightforward adaption of Definition 1.11. We introduce the following property of LDCs.

Definition 4.2 (Smoothness - informal)**.** *We say that a LDC C with locality k is t-smooth if for every index i and every query $q := (q_1, \ldots, q_k)$ issued by the decoding algorithm, any restrictions of q to at most t of its coordinates are uniformly distributed.*

We can now describe the following PIR protocol. Let C be a t-smooth LDC with locality k, dimension n and length n'. We need a converter on interface C_0 that encodes the database $\mathbb{M}$ and distributes it on the k different servers of $\mathbf{MultDB}_{n',k,t}$. Let $\mathbb{M}'$ be the encoded database. Let s be a random string that will be used as a source of randomness in algorithm $\mathcal{Q}$. On input i and s, the algorithm $\mathcal{Q}$ chooses k queries $q_1, \ldots, q_k$ which corresponds to k indices to be read in the codeword. Each one is sent to the corresponding server. The answer algorithm returns $\mathcal{A}(j, \mathbb{M}', q_j) := \mathbb{M}'[q_j]$. The reconstruction algorithm $\mathcal{R}$ runs the local decoder on inputs $a_1, \ldots, a_k$ and i.

We claim that this protocol used on $\mathbf{MultDB}_{n',k,t}$ securely constructs, for all distinguishers, the private database $\mathbf{PrivMultDB}_{n,k,t}$. Indeed, since the code is t-smooth, the distributions of the restrictions of the random variables $\mathcal{Q}(i, \cdot)_j$ to at most t queries are identical for all $i \in [n]$ since they are in fact all uniform. This means that a coalition of at most t servers gain no information on i by sharing their queries together. It is thus impossible even for an unbounded distinguisher to distinguish between real queries and simulated ones if the coalition size is at most t.

Using Secret-Sharing

Let k be the number of servers, t the desired coalition size threshold, n the size of the database and i the index of the database record desired by the client. We present a t-private PIR scheme using the (t, k)-Shamir secret-sharing [Sha79]. The database $\mathbb{M}$ is viewed as a matrix with c columns where c is chosen to be the closest possible integer to $\sqrt{n}$.

The client starts by choosing k non zero evaluation points $\alpha_1, \ldots, \alpha_k$ in a suitable finite field. On inputs i and s, the algorithm $\mathcal{Q}$ determines the column c_i where the i-th record lies,

Resource MultDB$_{\Sigma,n,k,t}$

Initialization

INIT, ACTIVE, COALITION $\leftarrow$ `false`
HIST$_j \leftarrow [\,], q_j \leftarrow \perp, a_j \leftarrow \perp, b_j \leftarrow$ `false` **for** $j = 1 \ldots k$
$\mathbb{M} \leftarrow [\,]$

Interface C_0

Input: $(\texttt{init}, \mathbb{M}') \in \Sigma^n$
  if not INIT
    $\mathbb{M} \leftarrow \mathbb{M}'$
    for $j = 1 \ldots k$ **do**
      HIST$_j \leftarrow$ HIST$_j$ || $(0, \texttt{init})$
    INIT $\leftarrow$ `true`
Input: $(\texttt{read}, i) \in [1, n]$
  if INIT **and not**ACTIVE
    for $j = 1 \ldots k$ **do**
      HIST$_j \leftarrow$ HIST$_j$ || $(0, \texttt{R}, i)$
    return $\mathbb{M}[i]$
Input: $(\texttt{write}, i, x) \in [1, n] \times \Sigma$
  if INIT **and not** ACTIVE
    for $j = 1 \ldots k$ **do**
      HIST$_j \leftarrow$ HIST$_j$ || $(0, \texttt{W}, i, x)$
    $\mathbb{M}[i] \leftarrow x$
Input: `initComplete`
  if INIT **and not** ACTIVE
    ACTIVE $\leftarrow$ `true`

Interface C

Input: $(\texttt{query}, q'_1, \ldots, q'_k)$
  if ACTIVE
    if $(q_1, \ldots, q_k) = (\perp, \ldots, \perp)$
      for $j = 1 \ldots k$ **do**
        $q_j \leftarrow q'_j$
        HIST$_j \leftarrow$ HIST$_j$ || $(\texttt{query}, j, q_j)$
Input: `reconstruct`
  if ACTIVE
    if $a_1 \neq \perp$ **and** $\ldots$ **and** $a_k \neq \perp$
      return $(a_1, \ldots, a_k)$

Interface W

Input: $(\texttt{formCoalition}, b'_1, \ldots, b'_k) \in \{\texttt{true}, \texttt{false}\}^k$
  if $|\{1 \leq j \leq k \mid b'_j = \texttt{true}\}| \leq t$ **and not** COALITION
    $(b_1, \ldots, b_k) \leftarrow (b'_1, \ldots, b'_k)$
    COALITION $\leftarrow$ `true`

Interfaces $\mathsf{S}_j, j \in [1, k]$

Input: $(\texttt{answer}, a)$
  if ACTIVE
    if $q_j \neq \perp$ **and** $a_j = \perp$
      $a_j \leftarrow a$
      HIST$_j \leftarrow$ HIST$_j$ || $(\texttt{answer}, j, a_j)$
Input: $(\texttt{read}, i) \in [1, n]$
  return $\mathbb{M}[i]$
Input: `getHist`
  if COALITION **and** b_j
    return HIST$_j$
Input: `getQuery`
  if COALITION **and** b_j
    return q_j

Figure 4.19: The basic database **MultDB** for k servers where at most t of them can form a coalition, allowing them to share information together.

Converter $\mathsf{mult\_pir}_{\mathsf{cli}}$

Initialization

$ind \leftarrow \perp, s \leftarrow \perp$

Interface `out`

Input: $(\texttt{query}, i) \in [1, n]$

if $ind = \perp$

$s \xleftarrow{\$} \mathcal{S}$

$ind \leftarrow i$

$(q_1, \ldots, q_k) \leftarrow \mathcal{Q}(ind, s)$

output $(\texttt{query}, q_1, \ldots, q_k)$ at interface C of **MultDB**

Input: `reconstruct`

output `reconstruct` at interface C of **MultDB**

Let $(a_1, \ldots, a_k)$ be the result

if $a_1 \neq \perp$ **and** $a_k \neq \perp$

return $\mathcal{R}(a_1, \ldots, a_k, ind, s)$

Figure 4.20: Description of the client converter $\mathsf{mult\_pir}_{\mathsf{cli}}$ for a PIR protocol $(\mathcal{Q}, \mathcal{A}, \mathcal{R})$ in the multi-server setting.

Converter $\mathsf{mult\_pir}_{\mathsf{ser}_j}$

Interface `out`

Input: `answer`

output `getQuery` at interface S_j of **MultDB**

Let q_j be the result

if $q_j \neq \perp$

Retrieve $\mathbb{M}$ with `read` requests at interface S_j

$a_j \leftarrow \mathcal{A}(j, \mathbb{M}, q_j)$

output $(\texttt{answer}, a_j)$ at interface S_j of **MultDB**

Figure 4.21: Description of the server converter $\mathsf{mult\_pir}_{\mathsf{ser}_j}$ for a PIR protocol $(\mathcal{Q}, \mathcal{A}, \mathcal{R})$ in the multi-server setting.

Resource PrivMultDB$_{\Sigma,n,k,t}$

Initialization

INIT, ACTIVE, COALITION $\leftarrow$ `false`
HIST$_j \leftarrow [\,], q_j \leftarrow \perp, a_j \leftarrow \perp, b_j \leftarrow$ `false` **for** $j = 1 \ldots k$
$\mathbb{M} \leftarrow [\,], ind \leftarrow \perp$

Interface C

Input: $(\texttt{query}, i) \in [1, n]$
if ACTIVE
if $(q_1, \ldots, q_k) = (\perp, \ldots, \perp)$
$(q_1, \ldots, q_k) \leftarrow (\texttt{ok}, \ldots, \texttt{ok})$
for $j = 1 \ldots k$ **do**
HIST$_j \leftarrow$ HIST$_j$ || $(\texttt{query}, j)$
$ind \leftarrow i$

Input: `reconstruct`
if ACTIVE
if $a_1 \neq \perp$ **and** $\ldots$ **and** $a_k \neq \perp$
return $\mathbb{M}[ind]$

Interfaces $\mathsf{S}_j, j \in [1, k]$

Input: `answer`
if ACTIVE
if $q_j \neq \perp$ **and** $a_j = \perp$
$a_j \leftarrow \texttt{ok}$
HIST$_j \leftarrow$ HIST$_j$ || $(\texttt{answer}, j)$

Input: `getHist`
if COALITION **and** b_j
return HIST$_j$

Input: $(\texttt{read}, i) \in [1, n]$
if COALITION **and** b_j
return $\mathbb{M}[i]$

Interface W

Input: $(\texttt{formCoalition}, b'_1, \ldots, b'_k) \in \{\texttt{true}, \texttt{false}\}^k$
if $|\{1 \leq j \leq k \mid b'_j = \texttt{true}\}| \leq t$ **and not** COALITION
$(b_1, \ldots, b_k) \leftarrow (b'_1, \ldots, b'_k)$
COALITION $\leftarrow$ `true`

Figure 4.22: The private (and correct) database **PrivMultDB** with k servers where at most t of them can form a coalition. The interface C_0 remains unchanged.

and secret shares component-wise the vector e_{c_i} (with a 1 in position c_i and 0 everywhere else) into k shares $s_1, \ldots, s_k$ using Shamir secret-sharing. The shares $s_1, \ldots, s_k$ are the output of the algorithm $\mathcal{Q}$. The evaluation points and the source of randomness needed for these operations are given in the string s. On inputs $\mathbb{M}$, j, and s_j the algorithm $\mathcal{A}$ computes the product $a_j := \mathbb{M}s_j$ and outputs it. Finally, on inputs $\mathbb{M}, i, a_1, \ldots, a_k, s$ the algorithm $\mathcal{R}$ can compute the Lagrange interpolation for each component of the a_j to reconstruct the secret:

$$\begin{aligned}
&\sum_{j=1}^{k} a_j \prod_{i=1, i\neq j} \frac{\alpha_i}{\alpha_i - \alpha_j} \\
=&\sum_{j=1}^{k} \mathbb{M}s_j \prod_{i=1, i\neq j} \frac{\alpha_i}{\alpha_i - \alpha_j} \\
=&\mathbb{M}e_{c_i}
\end{aligned}$$

The algorithm thus recovers the c_i-th column of the database which contains the i-th record, the one of interest for the client. $\mathcal{R}$ returns this record.

Here, the correctness of the PIR protocol directly follows from the correctness of the secret sharing scheme. The same holds for t-privacy: if a coalition of at most t servers cannot learn anything about e_{c_i} from their combined shares then they cannot learn anything about the index i requested by the client. This protocol thus yields the same construction guarantees as the one described using a PIR protocol based on LDCs.

4.4.4 The case of Byzantine servers

We can further strengthen the capabilities of **PrivMultDB** by allowing the servers to send an ill-formed answer or even to not answer at all after receiving the client's query. Such a server is called a Byzantine server. In the following, we introduce a threshold u which is an upper-bound on the number of Byzantine servers. After receiving the client's query, a Byzantine server can choose to assign the special symbol ϵ to its answer. This symbol means that the answer is of no use to the client (corresponding to an absence of answer or an ill-formed answer during the execution of the PIR protocol). The Byzantine servers are designated by the environment (at interface W). We present the new $\mathbf{ByzPrivMultDB}_{\Sigma,n,k,t,u}$ resource in Fig. 4.23. We also need to allow at most u Byzantine servers in the real resource **MultDB**, which defines a new resource $\mathbf{ByzMultDB}_{\Sigma,n,k,u}$. The additions needed being the same as the ones made to **PrivMultDB**, we also refer to Fig. 4.23 for those. The converters $\mathsf{mult\_pir}_{\mathsf{cli}}$ and $\mathsf{mult\_pir}_{\mathsf{ser}_j}$ remain unchanged.

The simulator used in the proof of construction of the new resource **ByzPrivMultDB** needs to be slightly modified to account for the aforementioned additions. In particular, the simulator just forwards the requests `badAnswer` to interface S_j and, when it simulates the history of the j-th server, if an entry of the form $(\texttt{answer}, j, \epsilon)$ is present, the simulator does not need to simulate an answer and can just copy this entry in its simulated history for the j-th server.

Finally, we can relax the definition of correctness for PIR protocols using the aforementioned threshold u by saying that a PIR protocol is u-correct if the user can recover its desired record even if at most u servers deviate from the protocol by providing incorrect answers or no answers at all.

Remark 14. We mention a concurrent and parallel work of Chen-Da Liu-Zhang and Ueli Maurer [LZM20]. They proposed a new way of making security statements in CC about coalitions of dishonest parties in the context of multiparty computation. They introduce a new type of

Resource ByzPrivMultDB$_{\Sigma,n,k,t,u}$

Initialization

BYZANTINES $\leftarrow$ `false`

$c_j \leftarrow$ `false` **for** $j = 1 \dots k$

Interface Interface $\mathsf{S}_j, j \in [1, k]$

Input: `badAnswer`

if BYZANTINES **and** c_j **and** $a_j = \perp$

if $q_j \neq \perp$

$a_j \leftarrow \epsilon$

HIST$_j \leftarrow$ HIST$_j$ || (`answer`, j, ϵ)

Interface Interface W

Input: (`formByzantines`, $c'_1, \dots, c'_k$) $\in \{$`true`, `false`$\}^k$

if $|\{1 \leq j \leq k \mid c'_j =$ `true`$\}| \leq u$ **and not** BYZANTINES

$(c_1, \dots, c_k) \leftarrow (c'_1, \dots, c'_k)$

BYZANTINES $\leftarrow$ `true`

Figure 4.23: The private and Byzantine-resistant database **ByzPrivMultDB** with k servers where at most t of them can form a coalition to share information and u of them can endorse a Byzantine behavior. Only the additions to **PrivMultDB** appear. The same additions are made to **MultDB** to define **ByzMultDB**.

relaxation, parameterized by the set Z of potentially dishonest parties, allowing to capture the guarantees for every such dishonest set Z. When modeling multi-server PIR, we consider the special case in which nothing is guaranteed if Z contains too many parties. Since their security definitions rely on relaxations, they do not need to model the ability to form coalitions inside resources. Instead, the possibility and the impact of coalitions only appear in security statements (*e.g.* in construction theorems).

We would advise to use our treatment of interactivity to design resources and to use the Z-relaxation of [LZM20] to make more general and abstract security statements. This way, one can design interactive resources devoid of unnecessary and technical details while still making meaningful security statements. For more details on Z-relaxations, see [LZM20, Sections 2.4 & 2.5].

Conclusion

We conclude with some future prospects for the works presented in this thesis. In Chapter 2, we showed that our TOGA-UE scheme has a malleability property which prevents it from being CCA secure. Hence, it would be interesting to find a transformation that would make TOGA-UE CCA secure. Moreover, we would like to propose new post-quantum constructions for UE based on error-correcting codes. In Chapter 3, we presented a framework for deriving PoR schemes from LCCs. We instantiated our framework using graph codes but we were unable to give concrete parameters because of the difficulty of the MSMD (Minimum Subgraph of Minimum Degree) problem. Thus, we are trying to construct regular graphs with large minimum subgraphs of small minimum degree. However, even if we succeeded, it is unlikely that these graphs have good properties for coding. Thus, we would like to instantiate our framework with other families of high-rate LCCs such as the multiplicity codes of Kopparty *et al.* [KSY11]. Finally, the use of the CC model made all of our constructions composable. Thus, it would be interesting to find new applications or existing protocols that use UE, PoRs, LCCs or PIR as a building block.

Bibliography

[ACLS18] Sebastian Angel, Hao Chen, Kim Laine, and Srinath T. V. Setty. PIR with compressed queries and amortized query processing. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, pages 962–979. IEEE Computer Society, 2018.

[ADFMP20] Navid Alamati, Luca De Feo, Hart Montgomery, and Sikhar Patranabis. Cryptographic group actions and applications. In *Advances in Cryptology – ASIACRYPT 2020: 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7–11, 2020, Proceedings, Part II*, page 411–439, Berlin, Heidelberg, 2020. Springer-Verlag.

[AdVS14] Daniel Augot, Françoise Levy dit Vehel, and Abdullatif Shikfa. *A Storage-Efficient and Robust Private Information Retrieval Scheme Allowing Few Servers*, pages 222–239. Cryptology and Network Security. Springer International Publishing, 2014.

[Amb97] Andris Ambainis. Upper bound on communication complexity of private information retrieval. In Pierpaolo Degano, Roberto Gorrieri, and Alberto Marchetti-Spaccamela, editors, *Automata, Languages and Programming, 24th International Colloquium, ICALP'97, Bologna, Italy, 7-11 July 1997, Proceedings*, volume 1256 of *Lecture Notes in Computer Science*, pages 401–407. Springer, 1997.

[AS16] Sebastian Angel and Srinath T. V. Setty. Unobservable communication over fully untrusted infrastructure. In Kimberly Keeton and Timothy Roscoe, editors, *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016*, pages 551–569. USENIX Association, 2016.

[BBD+22] Jeremy Booher, Ross Bowden, Javad Doliskani, Tako Boris Fouotsa, Steven D Galbraith, Sabrina Kunzweiler, Simon-Philipp Merz, Christophe Petit, Benjamin Smith, Katherine E Stange, et al. Failing to hash into supersingular isogeny graphs. *arXiv preprint arXiv:2205.00135*, 2022.

[BDGJ20] Colin Boyd, Gareth T. Davies, Kristian Gjøsteen, and Yao Jiang. Fast and secure updatable encryption. In *Advances in Cryptology – CRYPTO 2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part I*, page 464–493. Springer-Verlag, 2020.

[Beu22] Ward Beullens. Graph-theorethic algorithms for the alternating trilinear form equivalence problem. Cryptology ePrint Archive, Paper 2022/1528, 2022. https://eprint.iacr.org/2022/1528.

[BHPJ13] Peter Beelen, Tom Høholdt, Fernando Piñero, and Jørn Justesen. On the dimension of graph codes with reed-solomon component codes. In *2013 IEEE International Symposium on Information Theory*, pages 1227–1231, 2013.

[BI01] Amos Beimel and Yuval Ishai. Information-theoretic private information retrieval: A unified construction. In Fernando Orejas, Paul G. Spirakis, and Jan van Leeuwen, editors, *Automata, Languages and Programming, 28th International Colloquium, ICALP 2001, Crete, Greece, July 8-12, 2001, Proceedings*, volume 2076 of *Lecture Notes in Computer Science*, pages 912–926. Springer, 2001.

[BIKR02] Amos Beimel, Yuval Ishai, Eyal Kushilevitz, and Jean-François Raymond. Breaking the $o(n^{1/(2k-1)})$ barrier for information-theoretic private information retrieval. In *43rd Symposium on Foundations of Computer Science (FOCS 2002), 16-19 November 2002, Vancouver, BC, Canada, Proceedings*, pages 261–270. IEEE Computer Society, 2002.

[BJO09] Kevin D. Bowers, Ari Juels, and Alina Oprea. Proofs of retrievability: Theory and implementation. In *Proceedings of the 2009 ACM Workshop on Cloud Computing Security*, CCSW '09, pages 43–54, New York, NY, USA, 2009. ACM.

[BLMR13] Dan Boneh, Kevin Lewi, Hart Montgomery, and Ananth Raghunathan. *Key Homomorphic PRFs and Their Applications*, pages 410–428. Advances in Cryptology - CRYPTO 2013. Springer Berlin Heidelberg, 2013.

[BM18] Christian Badertscher and Ueli Maurer. Composable and robust outsourced storage. In *Topics in Cryptology - CT-RSA 2018 - The Cryptographers' Track at the RSA Conference 2018, San Francisco, CA, USA, April 16-20, 2018, Proceedings*, pages 354–373, 2018.

[CG97] Benny Chor and Niv Gilboa. Computationally private information retrieval (extended abstract). In Frank Thomson Leighton and Peter W. Shor, editors, *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*, pages 304–313. ACM, 1997.

[CGKS95] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, USA, 23-25 October 1995*, pages 41–50. IEEE Computer Society, 1995.

[CKN03] Ran Canetti, Hugo Krawczyk, and Jesper B. Nielsen. *Relaxing Chosen-Ciphertext Security*, pages 565–582. Advances in Cryptology - CRYPTO 2003. Springer Berlin Heidelberg, 2003.

[CLM+18] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. Csidh: an efficient post-quantum commutative group action. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 395–427. Springer, 2018.

[CMS99] Christian Cachin, Silvio Micali, and Markus Stadler. Computationally private information retrieval with polylogarithmic communication. In Jacques Stern, editor, *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, volume 1592 of *Lecture Notes in Computer Science*, pages 402–414. Springer, 1999.

[CMT13] Sandro Coretti, Ueli Maurer, and Björn Tackmann. *Constructing Confidential Channels from Authenticated Channels-Public-Key Encryption Revisited*, pages 134–153. Advances in Cryptology - ASIACRYPT 2013. Springer Berlin Heidelberg, 2013.

[CPS08] Jean-Sébastien Coron, Jacques Patarin, and Yannick Seurin. The random oracle model and the ideal cipher model are equivalent. In David Wagner, editor, *Advances in Cryptology – CRYPTO 2008*, pages 1–20, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[DG16] Zeev Dvir and Sivakanth Gopi. 2-server PIR with subpolynomial communication. *J. ACM*, 63(4):39:1–39:15, 2016.

[DVW09] Yevgeniy Dodis, Salil Vadhan, and Daniel Wichs. Proofs of retrievability via hardness amplification. In *Proceedings of the 6th Theory of Cryptography Conference on Theory of Cryptography*, TCC '09, pages 109–127, Berlin, Heidelberg, 2009. Springer-Verlag.

[Efr12] Klim Efremenko. 3-query locally decodable codes of subexponential length. *SIAM J. Comput.*, 41(6):1694–1703, 2012.

[EFRS90] P. Erdös, R.J. Faudree, C.C. Rousseau, and R.H. Schelp. Subgraphs of minimal degree k. *Discrete Mathematics*, 85(1):53–58, 1990.

[FMM21] Andrés Fabrega, Ueli Maurer, and Marta Mularczyk. A fresh approach to updatable symmetric encryption. *IACR Cryptol. ePrint Arch.*, page 559, 2021.

[GIKM00] Yael Gertner, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin. Protecting data privacy in private information retrieval schemes. *J. Comput. Syst. Sci.*, 60(3):592–629, 2000.

[GKS13] Alan Guo, Swastik Kopparty, and Madhu Sudan. New affine-invariant codes from lifting. In *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science*, ITCS '13, pages 529–540, New York, NY, USA, 2013. ACM.

[GR05] Craig Gentry and Zulfikar Ramzan. Single-database private information retrieval with constant communication rate. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *Automata, Languages and Programming, 32nd International Colloquium, ICALP 2005, Lisbon, Portugal, July 11-15, 2005, Proceedings*, volume 3580 of *Lecture Notes in Computer Science*, pages 803–815. Springer, 2005.

[Hen16] Ryan Henry. Polynomial batch codes for efficient IT-PIR. *Proc. Priv. Enhancing Technol.*, 2016(4):202–218, 2016.

[HJ06] Tom Høholdt and Jørn Justesen. Graph codes with reed-solomon component codes. In *2006 IEEE International Symposium on Information Theory*, pages 2022–2026, 2006.

[HJ11] Tom Høholdt and Jørn Justesen. The minimum distance of graph codes. In *Proceedings of the Third International Conference on Coding and Cryptology*, IWCC'11, page 201–212, Berlin, Heidelberg, 2011. Springer-Verlag.

[HLW06] Shlomo Hoory, Nathan Linial, and Avi Widgerson. Expander graphs and their application. *Bulletin (New Series) of the American Mathematical Society*, 43, 10 2006.

[HOW13] Brett Hemenway, Rafail Ostrovsky, and Mary Wootters. Local correctability of expander codes. In Fedor V. Fomin, Rūsiņš Freivalds, Marta Kwiatkowska, and David Peleg, editors, *Automata, Languages, and Programming*, pages 540–551, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[IKOS04] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Batch codes and their applications. In László Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 262–271. ACM, 2004.

[Jia20] Yao Jiang. The direction of updatable encryption does not matter much. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020*, pages 529–558, Cham, 2020. Springer International Publishing.

[JK07] Ari Juels and Burton S. Kaliski, Jr. Pors: Proofs of retrievability for large files. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07, pages 584–597, New York, NY, USA, 2007. ACM.

[JM20] Daniel Jost and Ueli Maurer. *Overcoming Impossibility Results in Composable Security Using Interval-Wise Guarantees*, pages 33–62. Advances in Cryptology - CRYPTO 2020. Springer International Publishing, 2020.

[JMM19] Daniel Jost, Ueli Maurer, and Marta Mularczyk. *A Unified and Composable Take on Ratcheting*, pages 180–210. Theory of Cryptography. Springer International Publishing, 2019.

[JQSY19] Zhengfeng Ji, Youming Qiao, Fang Song, and Aaram Yun. General linear group action on tensors: A candidate for post-quantum cryptography. In Dennis Hofheinz and Alon Rosen, editors, *Theory of Cryptography*, pages 251–281, Cham, 2019. Springer International Publishing.

[Jue01] Ari Juels. Targeted advertising ... and privacy too. In David Naccache, editor, *Topics in Cryptology - CT-RSA 2001, The Cryptographer's Track at RSA Conference 2001, San Francisco, CA, USA, April 8-12, 2001, Proceedings*, volume 2020 of *Lecture Notes in Computer Science*, pages 408–424. Springer, 2001.

[KLDF16] Albert Kwon, David Lazar, Srinivas Devadas, and Bryan Ford. Riffle: An efficient communication system with strong anonymity. *Proc. Priv. Enhancing Technol.*, 2016(2):115–134, 2016.

[KLR19] Michael Klooß, Anja Lehmann, and Andy Rupp. (r)cca secure updatable encryption with integrity protection. In Y. Ishai and V. Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019*, pages 68–99, Cham, 2019. Springer International Publishing.

[KO97] Eyal Kushilevitz and Rafail Ostrovsky. Replication is NOT needed: SINGLE database, computationally-private information retrieval. In *38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22, 1997*, pages 364–373. IEEE Computer Society, 1997.

[KO00] Eyal Kushilevitz and Rafail Ostrovsky. One-way trapdoor permutations are sufficient for non-trivial single-server private information retrieval. In Bart Preneel, editor, *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*, volume 1807 of *Lecture Notes in Computer Science*, pages 104–121. Springer, 2000.

[KSY11] Swastik Kopparty, Shubhangi Saraf, and Sergey Yekhanin. High-rate codes with sublinear-time decoding. In *Proceedings of the Forty-third Annual ACM Symposium on Theory of Computing*, STOC '11, pages 167–176, New York, NY, USA, 2011. ACM.

[KT00] Jonathan Katz and Luca Trevisan. On the efficiency of local decoding procedures for error-correcting codes. In *Proceedings of the Thirty-second Annual ACM Symposium on Theory of Computing*, STOC '00, pages 80–86, New York, NY, USA, 2000. ACM.

[Lav18] Julien Lavauzelle. *Codes with locality: constructions and applications to cryptographic protocols*. PhD thesis, Université Paris-Saclay (ComUE), 2018. Thèse de doctorat dirigée par Françoise Levy-Dit-Vehel.

[LLDV16] Julien Lavauzelle and Françoise Levy-Dit-Vehel. New proofs of retrievability using locally decodable codes. In *International Symposium on Information Theory ISIT 2016*, pages 1809 – 1813, Barcelona, Spain, 2016.

[LR22] Antonin Leroux and Maxime Roméas. Updatable encryption from group actions. Cryptology ePrint Archive, Paper 2022/739, 2022. https://eprint.iacr.org/2022/739.

[LT18] Anja Lehmann and Björn Tackmann. Updatable encryption with post-compromise security. In J. B. Nielsen and V. Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018*, pages 685–716, Cham, 2018. Springer International Publishing.

[LZM20] Chen-Da Liu-Zhang and Ueli Maurer. Synchronous constructive cryptography. In Rafael Pass and Krzysztof Pietrzak, editors, *Theory of Cryptography*, pages 439–472, Cham, 2020. Springer International Publishing.

[Mau12] Ueli Maurer. Constructive cryptography – a new paradigm for security definitions and proofs. In Sebastian Mödersheim and Catuscia Palamidessi, editors, *Theory of Security and Applications*, pages 33–56, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[MOT+11] Prateek Mittal, Femi G. Olumofin, Carmela Troncoso, Nikita Borisov, and Ian Goldberg. Pir-tor: Scalable anonymous communication using private information retrieval. In *20th USENIX Security Symposium, San Francisco, CA, USA, August 8-12, 2011, Proceedings*. USENIX Association, 2011.

[MOT20] T. Moriya, H. Onuki, and T. Takagi. Sigamal: a supersingular isogeny-based pke and its application to a prf. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 551–580. Springer, 2020.

[Mul54] D. E. Muller. Application of boolean algebra to switching circuit design and to error detection. *Transactions of the I.R.E. Professional Group on Electronic Computers*, EC-3(3):6–12, 1954.

[Nis22] Ryo Nishimaki. The direction of updatable encryption does matter. In *Public-Key Cryptography – PKC 2022: 25th IACR International Conference on Practice and Theory of Public-Key Cryptography, Virtual Event, March 8–11, 2022, Proceedings, Part II*, page 194–224, Berlin, Heidelberg, 2022. Springer-Verlag.

[OI07] Rafail Ostrovsky and William E. Skeith III. A survey of single-database private information retrieval: Techniques and applications. In Tatsuaki Okamoto and Xiaoyun Wang, editors, *Public Key Cryptography - PKC 2007, 10th International Conference on Practice and Theory in Public-Key Cryptography, Beijing, China, April 16-20, 2007, Proceedings*, volume 4450 of *Lecture Notes in Computer Science*, pages 393–411. Springer, 2007.

[PSS11] David Peleg, Ignasi Sau, and Mordechai Shalom. On approximating the d-girth of a graph. In Ivana Černá, Tibor Gyimóthy, Juraj Hromkovič, Keith Jefferey, Rastislav Královič, Marko Vukolić, and Stefan Wolf, editors, *SOFSEM 2011: Theory and Practice of Computer Science*, pages 467–481, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[PSU13] Maura B. Paterson, Douglas R. Stinson, and Jalaj Upadhyay. A coding theory foundation for the analysis of general unconditionally secure proof-of-retrievability schemes for cloud storage. *Journal of Mathematical Cryptology*, 7(3):183–216, 2013.

[RBBK01] Phillip Rogaway, Mihir Bellare, John Black, and Ted Krovetz. Ocb: A block-cipher mode of operation for efficient authenticated encryption. In *Proceedings of the 8th ACM Conference on Computer and Communications Security*, CCS '01, page 196–205, New York, NY, USA, 2001. Association for Computing Machinery.

[Ree54] I. Reed. A class of multiple-error-correcting codes and the decoding scheme. *Transactions of the IRE Professional Group on Information Theory*, 4(4):38–49, 1954.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '05, page 84–93, New York, NY, USA, 2005. Association for Computing Machinery.

[RZWZ20] Noga Ron-Zewi, Mary Wootters, and Gilles Zémor. Linear-time erasure list-decoding of expander codes. In *2020 IEEE International Symposium on Information Theory (ISIT)*, pages 379–383, 2020.

[SCM05] Len Sassaman, Bram Cohen, and Nick Mathewson. The pynchon gate: a secure method of pseudonymous mail retrieval. In Vijay Atluri, Sabrina De Capitani di Vimercati, and Roger Dingledine, editors, *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society, WPES 2005, Alexandria, VA, USA, November 7, 2005*, pages 1–9. ACM, 2005.

[Sha49] C. E. Shannon. Communication theory of secrecy systems. *The Bell System Technical Journal*, 28(4):656–715, 1949.

[Sha79] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, nov 1979.

[SS96] M. Sipser and D.A. Spielman. Expander codes. *IEEE Transactions on Information Theory*, 42(6):1710–1722, 1996.

[SW08] Hovav Shacham and Brent Waters. Compact proofs of retrievability. In Josef Pieprzyk, editor, *Advances in Cryptology - ASIACRYPT 2008*, pages 90–107, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[Tan81] R. Tanner. A recursive approach to low complexity codes. *IEEE Transactions on Information Theory*, 27(5):533–547, 1981.

[Tan84] R. Michael Tanner. Explicit concentrators from generalized n-gons. *SIAM Journal on Algebraic Discrete Methods*, 5(3):287–293, 1984.

[TDJ+22] Gang Tang, Dung Hoang Duong, Antoine Joux, Thomas Plantard, Youming Qiao, and Willy Susilo. Practical post-quantum signature schemes from isomorphism problems of trilinear forms. Cryptology ePrint Archive, Report 2022/267, 2022.

[TL20] Nianqi Tang and Yun Lin. Fast encoding and decoding algorithms for arbitrary (n, k) Reed-Solomon codes over $\mathbb{F}_{2^m}$. *IEEE Communications Letters*, 24(4):716–719, 2020.

[Yek08] Sergey Yekhanin. Towards 3-query locally decodable codes of subexponential length. *J. ACM*, 55(1):1:1–1:16, 2008.

[Zém01] Gilles Zémor. On expander codes. *IEEE Transactions on Information Theory*, 47(2):835–837, 2001.

List of Figures

ECOLE DOCTORALE

Titre : Modélisation et construction de protocoles cryptographiques interactifs pour le stockage distant

Mots clés : Stockage distant, protocoles cryptographiques, codes correcteurs, chiffrement avec mise à jour, preuves de récupérabilité, sécurité composable

Résumé :
Cette thèse porte sur la sécurité du stockage, de l'accès et de la maintenance de données distantes. En effet, le stockage distant fait émerger de nouvelles menaces pour ses utilisateurs. Nous nous intéressons aux trois protocoles suivants. Tout d'abord, les Preuves de Récupérabilité (PoR) permettent à un utilisateur qui accède rarement à ses données de vérifier qu'elles sont bien toujours stockées et intactes sur le serveur. Ensuite, le Chiffrement avec Mise à Jour (UE) rend possible la rotation de clés potentiellement compromises en utilisant peu de bande passante. Enfin, la Récupération Privée d'Information (PIR) rend les requêtes d'un client confidentielles. Le but de cette thèse peut être résumé en trois étapes. En premier lieu, nous développons des notions de sécurité modulaires qui expriment les garanties de sécurité attendues par des applications concrètes. Ensuite, nous vérifions si les définitions de sécurité existantes sont suffisantes, voire nécessaires, pour apporter nos garanties. Finalement, si les schémas cryptographiques existants n'atteignent pas nos nouvelles définitions, nous les améliorons ou proposons de nouveaux schémas qui le font. Nos définitions de sécurité sont données dans le modèle Cryptographie Constructive. Voici les contributions de cette thèse. Nous montrons comment construire des schémas UE avec des actions de groupe. Tout d'abord, nous proposons le premier schéma UE post-quantique permettant un nombre illimité de mises à jour. Il s'agit du premier schéma UE post-quantique dont la sécurité ne repose pas sur des problèmes de réseaux. Enfin, nous montrons comment il pourrait être possible d'obtenir le premier schéma d'UE post-quantique résistant aux attaques à chiffrés choisis. Malheureusement, ce schéma ne dispose pas d'instanciations pratiques pour le moment. Pour les PoRs, nous montrons qu'un schéma de Lavauzelle et Levy-dit-Vehel n'est pas aussi sûr que l'on pouvait le croire. Nous proposons un cadre pour concevoir des PoRs efficaces et sûrs en utilisant des codes localement corrigibles. Avec notre cadre, nous donnons une généralisation sûre du schéma précédent. De plus, nous utilisons des codes expanseurs pour construire un PoR avec de meilleurs paramètres. Nous étendons également CC aux protocoles interactifs qui délèguent des calculs à un adversaire. Nous donnons la première modélisation composable d'UE en évitant un résultat d'impossibilité classique. Nous utilisons notre modèle pour comprendre quelles notions de sécurité d'UE sont adaptées aux applications concrètes. Concernant le PIR, nous en donnons un modèle composable et unifié.

Title : Modeling and construction of interactive cryptographic protocols for outsourced storage

Keywords : Outsourced storage, cryptographic protocols, error correcting codes, updatable encryption, proofs of retrievability, composable security

Abstract :
This thesis deals with the security of the storage, the access and the maintenance of outsourced data. Indeed, outsourced storage raises new threats for users. We focus on the three following protocols. First, Proofs of Retrievability (PoR) allows a user who rarely accesses its data to be sure that it is stored on the server and that it did not suffer any alterations. Second, Updatable Encryption (UE) permits the user of an encrypted database to rotate its cryptographic keys with low bandwith usage. Third, Private Information Retrieval (PIR) allows a user to make the way it accesses outsourced data confidential. The goal of this thesis can be summarized in three steps. In step one, we develop modular security notions and models that closely match the security expectations of real-world solutions for the three above problems. Then, in step two, we check if existing security definitions are sufficient, and sometimes also necessary, to provide the security guarantees identified in step one. Finally, we determine if existing cryptographic schemes reach our security definitions and, if not, we improve them or propose new constructions that do. Our security statements are phrased in the Constructive Cryptography (CC) model of Maurer. The contributions made in this thesis are the following. We study the problem of building UE in the group action framework. First, we propose the first post-quantum UE scheme that supports an unbounded number of key updates. Second, our new scheme is the first post-quantum UE scheme whose security does not rely on lattice-based problems. We also show how to obtain a post-quantum UE scheme secure against chosen ciphertexts attacks using group actions. Unfortunately, we do not have any practical instantiation for this last scheme currently. As for PoRs, we show that the security of a PoR of Lavauzelle and Levy-dit-Vehel was overestimated. We propose a framework for the design of secure and efficient PoR schemes based on Locally Correctable Codes. We use our framework to give a secure generalization of the previously mentioned PoR. Furthermore, we use expander codes to design another PoR scheme with better parameters. We also extend CC so as to handle interactive protocols which delegate computations to an adversary. We use it to model UE and PIR. As for UE, we give the first composable modeling of UE by circumventing an impossibility result known as the commitment problem. We use this modeling to understand which security notion for UE is best suited for different real-world applications. As for PIR, we give a composable and unified treatment of many PIR variants.

Institut Polytechnique de Paris
91120 Palaiseau, France