Deletion-Compliance in the Absence of Privacy*

Jonathan Godin[1]** and Philippe Lamontagne[2]

[1] Université de Montréal, Montréal, Canada
[2] National Research Council Canada, Ottawa, Canada

Abstract. Garg, Goldwasser and Vasudevan (Eurocrypt 2020) invented the notion of deletion-compliance to formally model the "right to be forgotten", a concept that confers individuals more control over their digital data. A requirement of deletion-compliance is strong privacy for the deletion requesters since no outside observer must be able to tell if deleted data was ever present in the first place. Naturally, many real world systems where information can flow across users are automatically ruled out.
The main thesis of this paper is that deletion-compliance is a standalone notion, distinct from privacy. We present an alternative definition that meaningfully captures deletion-compliance without any privacy implications. This allows broader class of data collectors to demonstrate compliance to deletion requests and to be paired with various notions of privacy. Our new definition has several appealing properties:

- It is implied by the stronger definition of Garg, Goldwasser, and Vasudevan under natural conditions, and is equivalent when we add a strong privacy requirement.
- It is naturally composable with minimal assumptions.
- Its requirements are met by data structure implementations that do not reveal the order of operations, a concept known as history-independence.

Along the way, we discuss the many challenges that remain in providing a universal definition of compliance to the "right to be forgotten."

1 Introduction

Our increasingly online lives generate troves of personal data from our digital interactions. The widespread collection of this user-generated data for marketing and other purposes has lead in response to the concept of "right to be forgotten". This *right* stipulates that service providers do not hold onto data beyond its useful lifetime, and that users should retain ownership and control over their data and may, for example, ask for all copies and derivatives of their data to be deleted. This right has been codified in a handful of legislatures: the European General data Protection Regulation [GDPR] enshrines into law the right to erasure, Argentina's courts have made decisions that support the right to be forgotten [Car13], and California enacted a law requiring businesses to comply with data deletion requests [CCPA]. However, laws are by their very nature vague and open to interpretation. In response, researchers began to frame privacy laws in the formal language of cryptography [Nis+17; CN20] to facilitate compliance.

The present work continues this vein of research in the context of the "right to be forgotten", that was recently pioneered by Garg, Goldwasser, and Vasudevan [GGV20]. They have defined a notion of *deletion-compliance* that sets formal guidelines for service providers – that we will hereby refer to as *data-collectors* – to satisfy in order to comply

* This is the full version of [GL21].
** This work was done while visiting the National Research Council Canada.

with data deletion requests in accordance with the law. Deletion-compliance is not in itself a formal representation of a law, but rather a sufficient criteria to comply with the spirit of the law. In the formalism of [GGV20], a data-collector $\mathcal{X}$ interacts with two other entities: the *deletion-requester* $\mathcal{Y}$ that represents the set of users that request deletion of their data, and the *environment* $\mathcal{Z}$ that represents other users and any other entity. Interactions with the data-collector are described via a *protocol* π to which is associated a corresponding deletion protocol π_D. Garg, Goldwasser, and Vasudevan define deletion-compliance using the real vs ideal paradigm. In the *real* execution, the parties $\mathcal{X}$, $\mathcal{Y}$ and $\mathcal{Z}$ interact as prescribed by π and π_D and by their own programs. The restrictions are that every protocol π between $\mathcal{Y}$ and $\mathcal{X}$ is eventually followed by the associated deletion protocol π_D, and that $\mathcal{Y}$ is not allowed to communicate to $\mathcal{Z}$. Compared to this real execution is an *ideal* execution that differs by replacing $\mathcal{Y}$ with a *silent* variant $\mathcal{Y}_0$ that does not communicate with $\mathcal{X}$ nor $\mathcal{Z}$. The triple $(\mathcal{X}, \pi, \pi_D)$ is said to be deletion-compliant if for any $\mathcal{Y}$ and $\mathcal{Z}$, the state of $\mathcal{X}$ and the view of $\mathcal{Z}$ resulting from a real execution cannot be distinguished from those resulting from an ideal execution.

One consequence of the above framework is that for a data-collector to be deletion-compliant, it must *perfectly* protect the privacy of its users. The view of the environment, an entity that captures all third parties (other than the deletion-requester and the data-collector), is included in the indistinguishability requirement. This forces strict *privacy* of the data entrusted to the collector. This property was endorsed by the authors of [GGV20]: they argue that a data-collector that discloses information from its users loses control over that information and thus the ability to delete every copy. However, one can see from the definition that leakage of a single bit of information to the environment – whether $\mathcal{Y}$ is silent or not – is sufficient to distinguish the real and ideal executions. This criteria sets the bar very high in terms of privacy requirements. For example, it automatically rules out deletion-compliance of data-collectors that reveal some user information *as part of their core functionality*. This includes any service that involves the sharing of user-created content such as social networks, messaging apps, etc. Moreover, even well intended data-collectors can leak information in subtle ways. For example, machine learning models[3] reveal information at inference time about the training dataset [Sho+17; Wan+19], unless protective measures are taken [Dwo+06; AC19].

Motivated by these observations, we propose an alternative *weaker*, more realistic, definition that builds on top of their framework yet abstracts away any notion of privacy, focusing instead on capturing what it means to honestly comply with deletion requests. It enables a broader class of data-collectors to demonstrate compliance to the right-to-be-forgotten and to be paired with any suitable notion of privacy. Our definition is *simulation-based* and inspired by the concept of *zero-knowledge* for interactive proof systems [GMR89]. Intuitively, just as the existence of a simulator for the verifier in zero-knowledge proofs shows that the verifier *gains* no knowledge from the proof, the existence of a simulator for the data-collector shows that the data-collector *retains* no knowledge after deletion. A caveat to this analogy is that since we do not enforce privacy for the deletion-requester, the data-collector may retain some information on $\mathcal{Y}$ that it has learned through other means – e.g. by interacting with the environment $\mathcal{Z}$. To model this possibility, we give the view of $\mathcal{Z}$ to the simulator that we task with producing the state of $\mathcal{X}$. We say that $(\mathcal{X}, \pi, \pi_D)$ is *weakly deletion-compliant*

[3] The full version of [GGV20] gives an example of deletion-compliant data-collector that can train and delete from a machine learning model, but it does not have a *public* interface for making predictions with the machine learning model since it could leak data from $\mathcal{Y}$ to $\mathcal{Z}$.

if the simulator, given the view of $\mathcal{Z}$ as input, produces an output indistinguishable from the state of $\mathcal{X}$.

Limits of Cryptography for Modeling the "Right to be Forgotten". There is a fundamental limit to the formal treatment of the concept of "right to be forgotten". Formal definitions can only capture the deletion of data for which the deletion requester can demonstrate *ownership* over the data. One of the main motivators for such laws is to prevent forms of online abuse such as defamatory content or non-consensual sharing of adult content. The notion of *dereferencing* – the removal of data about an individual that was uploaded without consent – falls outside the scope of this work. See Section 1.2 below for references to work that propose solutions to this other problem.

Our work and that of [GGV20] assume honest behaviour of the data collector. Unless the data collector has a quantum memory [BI20], this assumption is necessary when modelling compliance to data deletion request. We stress that our notion of deletion-compliance is independent of any notion of privacy, and in particular does not excuse the data collector from respecting the user's privacy.

1.1 Contributions & Comparison with [GGV20]

Our work is in continuity with that of Garg, Goldwasser, and Vasudevan and builds on the same framework. They argue that for a data collector to retain the ability to delete data, it must not share data with any other party that does not also provide guarantees for the deletion of this data. Our definition demonstrates that this is more a design choice than a necessary condition, and that a meaningful notion of deletion-compliance exists in the absence of total privacy. Our weaker definition more closely matches the behavior that we expect from real-world data-collectors that comply with "right to be forgotten" laws.

Our approach has several advantages over the stronger definition of [GGV20], which from now on we refer to as *strong deletion-compliance*.

Composability. A consequence of the strong privacy requirement is that delegating data storage or computation becomes impossible. The solution of [GGV20] to this problem is to require that any third party $\mathcal{X}'$, which receives information on $\mathcal{Y}$ and is modeled as part of the environment $\mathcal{Z}$, also respects some form of deletion-compliance. The system as a whole then satisfies a notion called *conditional deletion-compliance*. Intuitively, a data-collector $\mathcal{X}$ that shares some data with $\mathcal{X}'$ is conditionally deletion-compliant if $\mathcal{X}'$ is deletion-compliant for the protocol describing the interaction of $\mathcal{X}$ and $\mathcal{X}'$. This solution is unsatisfactory for the main reason that it adds an assumption on the environment $\mathcal{Z}$ (containing $\mathcal{X}'$), which is modeled as an adversary whose goal it is to learn about $\mathcal{Y}$. In constast, our definition composes sequentially with no assumption on the environment or deletion-requester and only mild assumptions on the honest data-collector.

Parallel composition of strong deletion-compliance also suffers from the stark privacy requirement: it composes in parallel only for data-collectors and deletion-requesters *that do not communicate with each other*. Our weaker definition is easily shown to compose in parallel (for either deletion-requester or data-collector) without additional assumptions.

Ease of Compliance. We argue that the definition of strong deletion-compliance is too restrictive for most real-world applications. We give examples of data-collectors, some natural and some designed to prove our point, that satisfy the *spirit* of the "right to be forgotten" (and our definition), but that are not strongly deletion-compliant.

Garg, Goldwasser, and Vasudevan give as example a specific deletion-compliant data-collector built from a *history independent* dictionary. An implementation of an abstract data structure (ADS) is history independent if any two sequences of operations that lead to the same state for the ADS also lead to the same memory representation of the implementation. In contrast, we show that this is a general property of our definition: if a data-collector is implemented using exclusively history independent data structures, then it is weakly deletion-compliant.

Our simulation-based definition also paves the way for a *constructive* way to prove compliance to data-deletion requests. For example, the source code for a data-collector may be published together with the code for the corresponding simulator. An independent auditor could then run experiments to heuristically verify the claimed weak deletion-compliance.

Observations on Defining Deletion-Compliance. We elaborate and add to the discussions initiated by [GGV20] on issues that arise when attempting to define deletion-compliance. We highlight several subtleties that practitioners may face when trying to comply with (strong or weak) deletion-compliance. More precisely, we identify problems that occur when randomness is used in certain ways by the data-collector, and we point to subtle ways in which information on $\mathcal{Y}$ may remain in the data-collector post-deletion. We offer partial solutions to those issues and leave a definitive resolution to future work.

1.2 Related Work

The formal treatment of data privacy laws using ideas inspired by cryptography has been investigated outside of this work and [GGV20]. To the best of our knowledge, Nissim, Bembenek, Wood, Bun, Gaboardi, Gasser, O'Brien, Steinke, and Vadhan [Nis+17] is the first work using the formal language of cryptography to model privacy laws written in an ambiguous legal language. They formalize the FERPA privacy legislation from the United States as a cryptographic game-based definition and prove that the notion of differential privacy [Dwo+06] satisfies this definition. Cohen and Nissim [CN20] tackle this task of bridging legal and formal notions of privacy for the GDPR legislation [GDPR]. They provide a formal definition of the GDPR's concept "singling out" and show that differential privacy implies this notion, but that another privacy notion, k–anonymity [SS98], does not.

The notion that programs and databases (unintentionally) retain information beyond its intended lifetime is not new. The following papers focus on the setting of users interacting with programs on their own machines and ensuring that data cannot be recovered if the machine is compromised. Stahlberg, Miklau, and Levine [SML07] propose desired properties for forensically transparent systems where all data should be accessible through legitimate interfaces and not through forensic inspection of the machine state. They investigate common database implementations and find that they are vulnerable to forensic recoverability, then propose measures to reduce unintended data retention. Kannan, Altekar, Maniatis, and Chun [Kan+11] propose a technique to ensure that sensitive data does not linger in a program's memory state after its useful lifespan. They take snapshots of the application state and log events so that the program can be rewinded to a previous state and actions replayed to arrive at an "equivalent" state where the sensitive information is gone. A survey paper by Reardon, Basin, and Capkun [RBC13] reviews methods for securely deleting data from physical mediums. As such they are less interested in removing from a system all traces of data from a particular user, but to ensure that data marked for deletion cannot be recovered with forensics. Arkema and Sherr [AS21] introduce the concept of residue-free

computing that provides any application with an *incognito mode* preventing any trace data from being recorded on disk.

As previously mentioned, our paper is only concerned with the case where the data being requested for deletion originated from the deletion-requester. Simeonovski, Bendun, Asghar, Backes, Marnau, and Druschel [Sim+15] propose a framework for deletion requests of an individual's data that has another source (e.g. news or social media). It automates the process of detecting personal information publicly available online, requesting deletion and proving eligibility of the deletion request. Derler, Ramacher, Slamanig, and Striecks [Der+19] propose a cryptographic solution to data deletion in a distributed setting based on a new primitive called identity-based puncturable encryption.

2 Preliminaries

Throughout the paper, we use $\lambda \in \mathbb{N}$ as a security parameter. We write PPT as shorthand for "probabilistic polynomial time" and we let $\mathsf{negl}(\lambda)$ denote an arbitrary negligible function, i.e. such that for every $k \in \mathbb{N}$, there exists $\Lambda > 0$ such for any $\lambda \geq \Lambda$, $\mathsf{negl}(\lambda) < \frac{1}{\lambda^k}$.

For two families of random variables $\mathcal{A} = \{A_\lambda\}_{\lambda\in\mathbb{N}}$ and $\mathcal{B} = \{B_\lambda\}_{\lambda\in\mathbb{N}}$. We say that $\mathcal{A}$ and $\mathcal{B}$ are ε–*indistinguishable* (resp. computationally ε–indistinguishable) and write $\mathcal{A} \approx_\varepsilon^S \mathcal{B}$ (resp. $\mathcal{A} \approx_\varepsilon^C \mathcal{B}$) to mean that for every unbounded (resp. PPT) distinguishers $\mathcal{D}$, for all $\lambda \in \mathbb{N}$

$$\left|\Pr\left[\mathcal{D}(A_\lambda) = 1\right] - \Pr\left[\mathcal{D}(B_\lambda) = 1\right]\right| \leq \varepsilon(\lambda). \tag{1}$$

We simply write $\mathcal{A} \approx^S \mathcal{B}$ (resp. $\mathcal{A} \approx^C \mathcal{B}$) and say $\mathcal{A}$ and $\mathcal{B}$ are (computationally) indistinguishable when ε is a negligible function of λ and even omit the superscript when clear from the context. We let $\Delta^S(\mathcal{A}, \mathcal{B})$ (resp. $\Delta^C(\cdot,\cdot)$) denote the supremum of (1) over the choice of unbounded (resp. PPT) distinguisher $\mathcal{D}$. This notion satisfies the triangle inequality: $\Delta(\mathcal{A}, \mathcal{B}) \leq \Delta(\mathcal{A}, \mathcal{C}) + \Delta(\mathcal{C} + \mathcal{B})$ for $\Delta \in \{\Delta^S, \Delta^C\}$.

2.1 The Execution Model

We use essentially the same formalism as in Garg, Goldwasser, and Vasudevan. We review it here and modify it slightly to suit our needs. Parties involved in an execution are modeled as *Interactive Turing Machines* (ITM) with several tapes.

Definition 1 (Copied from [GGV20]). *An interactive Turing Machine (ITM) is a Turing Machine M with the following tapes (i) a read-only* identifier *tape; (ii) a read-only* input *tape; (iii) a write-only* output *tape; (iv) a read-write* work *tape; (v) a single-read-only* incoming *tape; (vi) a single-write-only* outgoing *tape; (vii) a read-only* randomness *tape; and (viii) a read-only* control *tape.*

The state *of an ITM at any given point in its execution, denoted state_M, consists of the content of its work tape at that point. Its* view*, denoted by view_M, consists of the contents of its input, output, incoming, outgoing, randomness and control tapes at that point.*

ITMs are static objects that are instantiated by *instances of ITMs* (ITIs). Throughout this paper, we may refer to ITIs simply as "machines". Each ITI is equipped with a unique identifier and may be under the control of another ITI, in which case the *controlling* ITI can write on the *controlled* ITI's control and input tape and read its output tape. ITIs can be engaged in a *protocol* by writing on each other's incoming tapes in a way prescribed by that protocol. Any message an ITI writes on the incoming tape of another ITI is also copied

on its own outgoing tape. A protocol describes the actions of each participating ITI and what they write on each other's incoming tapes. An instantiation of a protocol is called a *session* uniquely identified by a unique session identifier sID. Each ITI engaged in a session of a protocol is attributed a unique party identifier pID for that particular session. To each session of π is associated a *deletion token* that is a function of sID and of the content of the incoming and outgoing tapes of the machines engaged in that session. The deletion token will be used as input to π_D to ask for the deletion of the information stored during session sID of π.

Note that the protocol π can in reality represent many possible protocols $(\pi_1, \ldots, \pi_n)$ by having the machine that initiates π begin its first message with the index i of which π_i it wants to run. We can assume without loss of generality that π is a two-round protocol, i.e. the machine that initiates π first sends one message (writes to the other machine's incoming tape) and the other machine responds. To implement longer interactions, the machines can use the session identifier to resume the interaction later in an asynchronous manner.

Special ITIs. We consider three special ITIs: the data-collector $\mathcal{X}$, the deletion-requester $\mathcal{Y}$ and the environment $\mathcal{Z}$. $\mathcal{Y}$ represents the clients that will request deletion and $\mathcal{Z}$ represents the rest of the world and any other machines interacting with $\mathcal{X}$ that may or may not request deletion. Instead of directly engaging in protocol sessions with each other, these special ITIs create new ITIs under their control to interact in this session. This way, $\mathcal{X}$ has no way of knowing if it is interacting with $\mathcal{Y}$ or $\mathcal{Z}$. By abuse of terminology, we say that $\mathcal{Y}$ (or $\mathcal{Z}$) initiates a protocol with $\mathcal{X}$ if both machines instantiate ITIs that engage in a protocol session.

The state or view of these special ITIs is the concatenated states or views of every ITI under its control. We assume w.l.o.g. that every ITI $\mathcal{Z}$ creates only engages in a single session of π since $\mathcal{Z}$ can task these ITIs to write the content of all of their tapes on their output tape and $\mathcal{Z}$ can provide these tapes to newly instantiated ITIs.

Execution Phases. The execution happens in two phases. In the *alive* phase, the special ITMs $\mathcal{X}$, $\mathcal{Y}$ and $\mathcal{Z}$ are instantiated with the security parameter $\lambda \in \mathbb{N}$ written on their input tape in unary and their randomness tapes initialized with a stream of random bits. The execution consists of a sequence of activations of $\mathcal{X}$, $\mathcal{Y}$ and $\mathcal{Z}$ and their controlled ITIs. An activated ITI runs until:

- It writes on the incoming tape of another machine; that machine then becomes activated.
- It writes on its own output tape and halts; the machine that created that ITI becomes activated.
- It creates a new ITI; that new ITI becomes activated.

Only one machine is activated at any single moment, the others are "paused", i.e. they do not read/write from/to any of their tapes. The order in which the ITIs are activated, with the exception of the above special cases, is under the control of the environment.

In the *terminate* phase, each ITI created by $\mathcal{Y}$ that initiated a session of π or π_D with $\mathcal{X}$ is activated until it halts. For every session of π for which the corresponding π_D has not been initiated, an ITI is created by $\mathcal{Y}$ to initiate a session of π_D with the corresponding deletion token and is executed until it halts.

Differences with [GGV20]. Garg, Goldwasser, and Vasudevan consider two distinct executions: the real execution described above and an *ideal* execution where $\mathcal{Y}$ is replaced with a

silent deletion-requester $\mathcal{Y}_0$ that does not initiate any protocol with $\mathcal{X}$. Since our definition is concerned with simulation and not with a real vs ideal scenario, we do not use this ideal execution, except when dealing directly with the definition of strong deletion-compliance.

Moreover, since the original definition of strong deletion-compliance requires the indistinguishability of the view of the environment in the real and ideal executions, $\mathcal{Y}$ is not allowed to write on the incoming tape of $\mathcal{Z}$'s controlled ITIs. Our definition does not imply nor require privacy from the environment, therefore we lift that restriction.

We introduce the following new elements of notation. For two ITIs M and M', we denote by (M, M') a new ITI that controls both M and M' (assuming they are not already under the control of another machine). Furthermore, we denote by $view_M^{M'}$ the subset of $view_M$ restricted to the interactions between M and M' (or of ITIs under their respective control).

2.2 Strong Deletion-Compliance

For completeness, we present the definition of strong deletion-compliance of [GGV20].

Definition 2 (Strong Deletion-Compliance). *Given a data-collector $(\mathcal{X}, \pi, \pi_D)$, an environment $\mathcal{Z}$ and a deletion-requester $\mathcal{Y}$, let $(state_{\mathcal{X}}^R, view_{\mathcal{Z}}^R)$ denote the state of $\mathcal{X}$ and view of $\mathcal{Z}$ in the real execution, and let $(state_{\mathcal{X}}^I, view_{\mathcal{Z}}^I)$ the corresponding variables in the ideal execution where $\mathcal{Y}$ is replaced with a special machine $\mathcal{Y}_0$ that does nothing – simply halts when it is activated. We say that $(\mathcal{X}, \pi, \pi_D)$ is* strongly statistically (resp. computationally) deletion-compliant *if, for all* PPT *environment $\mathcal{Z}$, all* PPT *deletion-requester $\mathcal{Y}$, and for all unbounded (resp.* PPT*) distinguishers $\mathcal{D}$, there is a negligible function ε such that for all $\lambda \in \mathbb{N}$:*

$$\left|\Pr[\mathcal{D}(state_{\mathcal{X}}^R, view_{\mathcal{Z}}^R) = 1] - \Pr[\mathcal{D}(state_{\mathcal{X}}^I, view_{\mathcal{Z}}^I) = 1]\right| \leq \varepsilon(\lambda) \ . \tag{2}$$

3 A More Realistic Notion of Deletion-Compliance

We now present our proposed definition of deletion-compliance. The definition is *simulation*-based, a data-collector $\mathcal{X}$ is weakly deletion-compliant if there is a simulator $\mathcal{S}$ that can produce the state of $\mathcal{X}$ from the view of the environment.

Definition 3 (Weak Deletion-Compliance). *Let $(\mathcal{X}, \pi, \pi_D)$ be a data-collector, $\mathcal{Y}$ be a deletion-requester and $\mathcal{Z}$ be an environment. Let $state_{\mathcal{X}}$ and $view_{\mathcal{Z}}^{\mathcal{X}}$ respectively denote the state of $\mathcal{X}$ and the view of the interaction of $\mathcal{Z}$ and $\mathcal{X}$ after the terminate phase. We say that $(\mathcal{X}, \pi, \pi_D)$ is* computationally (resp. statistically) ε–weakly deletion-compliant *if there exists a simulator $\mathcal{S}$ such that for all* PPT *(resp. unbounded) environment $\mathcal{Z}$, all* PPT *deletion-requester $\mathcal{Y}$ and all* PPT *(resp. unbounded) distinguishers D, there exists a negligible function ε such that for all $\lambda \in \mathbb{N}$*

$$\left|\Pr\left[\mathcal{D}(state_{\mathcal{X}}, view_{\mathcal{Z}}^{\mathcal{X}}) = 1\right] - \Pr\left[\mathcal{D}\big(\mathcal{S}(view_{\mathcal{Z}}^{\mathcal{X}}), view_{\mathcal{Z}}^{\mathcal{X}}\big) = 1\right]\right| \leq \varepsilon(\lambda) \tag{3}$$

where the probability is over the random tapes of $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$ and $\mathcal{D}$. We say that $(\mathcal{X}, \pi, \pi_D)$ is computationally (resp. statistically) weakly deletion-compliant if it is computationally (resp. statistically) $\mathsf{negl}(\lambda)$*–weakly deletion-compliant.*

We define the weak deletion-compliance error *naturally as smallest $\varepsilon \colon \mathbb{N} \to [0, 1]$ such that $(\mathcal{X}, \pi, \pi_D)$ is ε–weakly deletion-compliant.*

Intuitively, our definition asserts that the state of $\mathcal{X}$ after deletion can be completely described by its interaction with the environment. The key distinction with Definition 2 is that we no longer require that the environment contains no information on $\mathcal{Y}$ by distinguishing between a real execution and an ideal execution. In essence, everything that $\mathcal{X}$ may retain about $\mathcal{Y}$ after deletion is something that is already known to the environment. By giving $view_{\mathcal{Z}}^{\mathcal{X}}$ as input to the distinguisher, we ensure that the simulator outputs a state for $\mathcal{X}$ that is consistent with the view of the environment.

For the purpose of transparency, we note that there are trivial ways to modify any data-collector $\mathcal{X}$ to satisfy our definition. For example, consider a data-collector that, in each session of protocol π, sends the content of its work tape to the other machine. Obviously this data collector is weakly deletion-compliant since a simulator has direct access to the state of $\mathcal{X}$ in the view of $\mathcal{Z}$. However, since our definition is concerned with *honest* data collectors and is intended to model real-world systems, we do not concern ourselves with this sort of worst case behavior of $\mathcal{X}$. Similarly, we do not worry about the case where $\mathcal{Y}$ sends all its data to $\mathcal{Z}$. Our definition quantifies over all $\mathcal{Y}$, so the simulator must work for any $\mathcal{Y}$ and in particular for honest $\mathcal{Y}$ (i.e. that does not conspire with the environment).

Observe that something similar can be said of the definition of strong deletion-compliance. A data collector that makes no effort to comply to deletion requests can be turned into one that is strongly deletion-compliant by keeping snapshots of its state after each protocol execution, rewinding to a previous state when a deletion request occurs and replaying the protocols that did not originate from $\mathcal{Y}$. This is similar to what [Kan+11] propose for purging sensitive data from programs that were not designed with that goal in mind.

Deletion in Practice. Modern computer architectures are far from Turing machines. Deleting a file on modern operating systems simply removes any reference to that file's disk location from the file system. This is unsatisfactory from the point of view of deletion-compliance on many fronts. The file or fragments thereof can still be recovered until its location on disk is overwritten with new data. The common solution is to overwrite the disk location of the file with 0's or with random data upon deletion. However, there remains information in the disk on the *size* of this file and on *where* the data was stored, which can leak metadata on the nature and order of operations. There are also technological challenges to applying such an approach to single files with the increasing popularity of SSD storage [Wei+11]. An alternative solution known as *crypto-shredding* encrypts files with unique keys and overwrites the keys to delete the file. The resulting system would be computationally deletion-compliant, as opposed to statistically. However, we are left with a similar problem: the non-zero disk space of the data collector (i.e. the size of $state_{\mathcal{X}}$) scales with the number of protocols it ran. This leaks information on the deletion requester and thus is not (even computationally) deletion-compliant. It is an issue that affects both weak and strong deletion-compliance.

A potential remedy is to give the simulator a *masked* version of the deletion-requester's view. The masked view would consist of the same elements as $view_{\mathcal{Y}}^{\mathcal{X}}$, but instead of having the contents of the incoming and outgoing tapes, it has only the length of this content in unary. A similar solution was already hinted at by Garg, Goldwasser, and Vasudevan, i.e. have a deletion-requester that sends the "same kinds of messages to $\mathcal{X}$, but with different contents". With such a masked view of $\mathcal{Y}$, the simulator could know the timing of operations done by $\mathcal{X}$ and recreate any *gaps* in memory that resulted from deletion. We did not adopt this approach with our definition since doing so would make it incomparable to strong deletion-compliance.

Simulation and Randomness. How the data-collector uses randomness can have an influence on whether or not it satisfies Definition 3. Consider for example a data collector that has the following protocols:

- `query`(1^λ): Read λ bits of the randomness tape. Let x be the value. Store x in memory and return $y = f(x)$ where f is a cryptographic one-way function.
- `delete`(y): Search through memory to find x such that $f(x) = y$. If found, delete x from memory, otherwise do nothing.

Obviously such a data collector meets the intuition of deletion-compliance, and indeed it does appear to satisfy the strong variant of deletion-compliance (see discussion below). However, in our simulation-based definition the simulator has no way, given the view of $\mathcal{Z}$ that consists of elements y from the image of f, to efficiently recreate the state of $\mathcal{X}$ that consists of the preimages of the y's without the random tape of $\mathcal{X}$ that was used to sample the x's. Since the simulator must produce a state for $\mathcal{X}$ that is consistent with the view of $\mathcal{Z}$, the existence of an efficient simulator would contradict the one-wayness of f.

There are two apparent solutions to the above problem. The first is to give $\mathcal{S}$ access to the random tape of $\mathcal{X}$ in addition to the view of $\mathcal{Z}$ to recreate the working tape of $\mathcal{X}$. The other is to *not* give $\mathcal{S}$ access to the random tape of $\mathcal{X}$ and ask that the state of $\mathcal{X}$ can be efficiently computed only from the view of $\mathcal{Z}$. This limits the ways in which the data collector uses its "private" randomness beyond its initialization phase. We chose the second approach for the following reasons. First, it makes our definition closer, and thus more easily comparable, to strong deletion-compliance. Second, going with the first approach would give more power to the simulator, weakening further the definition. Third, most examples where this poses a problem like for the above data-collector can be remedied by employing *public randomness*, e.g. the above data-collector would receive the string x from the other machine in protocol `query`.

We remark that a similar issue appears in [GGV20]: in the real and ideal executions, different parts of the random tape of $\mathcal{X}$ may be used to run the corresponding protocols with $\mathcal{Z}$. For example, suppose that $\mathcal{X}$ is initialized with the same randomness tape in the real and ideal worlds. If in each execution of π, $\mathcal{X}$ reads n bits of its randomness tape (i.e. moves the head n positions to the right), then the position of the randomness tape head will differ for protocols π initiated by $\mathcal{Z}$ in the real and ideal worlds. In the real world, $\mathcal{Y}$ initiated π with $\mathcal{X}$ thus moving the head on the randomness tape and in the ideal world, $\mathcal{Y}_0$ does not initiate π thus not moving the randomness tape head. This will result in different views for the environment and, in the above data collector, a different state for $\mathcal{X}$. Their solution[4] to this issue is to condition on the views being the same in the real and ideal case. If the values for the view of $\mathcal{Z}$ have the same domain in the real and ideal worlds, it is then sufficient to bound the distance between the states of $\mathcal{X}$ in the real and ideal cases that led to the same view for $\mathcal{Z}$.

In general, this approach only works if $\mathcal{X}$ employs "public" randomness, i.e. if the views of $\mathcal{Y}$ and $\mathcal{Z}$ uniquely determine the random tape of $\mathcal{X}$ (even inefficiently so, as in the above data-collector). Otherwise, the real and ideal data-collector may use different private randomness leading to different states for $\mathcal{X}$, even if the view of $\mathcal{Z}$ is the same in both executions. For example, considering the above data collector, if the function f is not one-to-one (i.e. not a bijection), then conditioning on the real and ideal views of $\mathcal{Z}$ being the same no longer works since the real and ideal states for $\mathcal{X}$ may contain different x, x' such

[4] See Appendix A of their full version (https://arxiv.org/abs/2002.10635).

that $f(x) = f(x') = y$ is in the view of $\mathcal{Z}$. Along with this public randomness, the data-collector can use "private" randomness which is roughly defined as randomness that only affects the internal state of $\mathcal{X}$ and not the views of the machine it interacts with.

3.1 The Relationship Between Strong and Weak Deletion-Compliance

To substantiate our claim that the restrictions imposed by strong deletion-compliance (Definition 2) are too severe, we present a simple data-collector that demonstrates a separation between strong and weak deletion-compliance. This data-collector, albeit contrived, showcases the fact that leakage of a single bit of information on $\mathcal{Y}$ rules out the strong flavor of deletion-compliance, but not our weaker variant. The data-collector is described in Fig. 1.

Initialization:

1. Initialize an integer $q = 0$ and a set $X = \emptyset$ implemented as a sorted list.

Protocol π:

1. Pick $x \in \{0,1\}^\lambda \setminus X$ uniformly at random.
2. Increment $q \leftarrow q + 1$.
3. Return x and $(q \mod 2)$.

Protocol $\pi_D(x)$:

1. If $x \in X$, set $X \leftarrow X \setminus \{x\}$, decrement $q \leftarrow q - 1$ and return $ok = 1$
2. If $x \notin X$, return $ok = 0$.

Fig. 1: The data-collector $\mathcal{X}$ of Theorem 1.

Theorem 1. *The data-collector $(\mathcal{X}, \pi, \pi_D)$ described in Fig. 1 is not strongly deletion-compliant, but it is weakly deletion-compliant.*

Proof. We first show that $\mathcal{X}$ is not strongly deletion-compliant. To do so, we construct an environment $\mathcal{Z}$ and a deletion-requester $\mathcal{Y}$ such that the view of $\mathcal{Z}$ in the real and ideal executions are perfectly distinguishable.

The behavior of $\mathcal{Y}$ and $\mathcal{Z}$ in the alive phase is as follows:

- $\mathcal{Y}$: on first activation, initiate protocol π with $\mathcal{X}$. On subsequent activations, do nothing and halt.
- $\mathcal{Z}$: activate $\mathcal{Y}$ once. After $\mathcal{Y}$ halts, initiate protocol π with $\mathcal{X}$. When the instance of π concludes (i.e. the ad-hoc client-side ITI halts), declare the end of the alive phase.

In the terminate phase, $\mathcal{Y}$ invokes π_D with the deletion token it received from $\mathcal{X}$ during π as is required from the execution model.

Let $state^R_{\mathcal{X}}$ and $view^R_{\mathcal{Z}}$ be the random variables describing the state of $\mathcal{X}$ and view of $\mathcal{Z}$ at the end of the above real execution (with $\mathcal{Y}$ acting as described). Let $(x, p) \in \{0,1\}^\lambda \times \{0,1\}$ be the message sent from $\mathcal{X}$ to $\mathcal{Z}$ in the instance of π. Then $state^R_{\mathcal{X}} = (X, q) = (\{x\}, 1)$ and $view^R_{\mathcal{Z}} = (x, p) = (x, 0)$ since when $\mathcal{Z}$ initiates π, $\mathcal{Y}$ has already done its session of π and has not yet done π_D, so $q = 2$ during $\mathcal{Z}$'s instance of π, and $q = 1$ after $\mathcal{Y}$ initiates π_D. Let

$state^I_{\mathcal{X}}$ and $view^I_{\mathcal{Z}}$ be the same random variables in the ideal world where $\mathcal{Y}$ is replaced with $\mathcal{Y}_0$ that on every activation, does nothing and halts. Then $state^I_{\mathcal{X}} = (\{x\}, 1)$ is the same as in the real world, but $view^I_{\mathcal{Z}} = (x, 1)$ since $\mathcal{Y}_0$ does not initiate π and $q = 1$ in $\mathcal{Z}$'s session of π. The views of $\mathcal{Z}$ in the real and ideal executions can thus perfectly be distinguished.

To show that $\mathcal{X}$ is weakly deletion-compliant, we must give a simulator that recreates the state of $\mathcal{X}$ from the view of $\mathcal{Z}$ (in the real execution). Let $\mathcal{Y}$ and $\mathcal{Z}$ be an arbitrary deletion-requester and environment where $\mathcal{Z}$ initiates $t = \mathsf{poly}(\lambda)$ sessions of π or π_D with $\mathcal{X}$. The view of $\mathcal{Z}$ contains of the list of protocols initiated with $\mathcal{X}$ and the contents of the incoming and outgoing tapes of the client-side ITIs for those protocols. Then we can parse the view of $\mathcal{Z}$ as $view_{\mathcal{Z}} = \{(\tau_i, x_i, p_i, ok_i)\}_{i\in[t]}$ where we set $p_i = \perp$ when $\tau_i = \pi_D$ and $ok_i = \perp$ when $\tau_i = \pi$.

We define the simulator $\mathcal{S}$ as follows. Initialize a set $S = \emptyset$ and integer $q = 0$. For every $(\tau_i, x_i, p_i, ok_i) \in view_{\mathcal{Z}}$, if $\tau_i = \pi$, insert x_i in S and increment q, and if $\tau_i = \pi_D$, if $x_i \in S$, remove x_i from S and decrement q, else do nothing. The output of $\mathcal{S}$ is $(\mathtt{sort}(S), q)$. To prove indistinguishability, consider a deletion-requester $\mathcal{Y}$ that initiates $r = \mathsf{poly}(\lambda)$ sessions of π (and an equal amount of π_D) with $\mathcal{X}$. Let $\hat{X}$ be the r deletion tokens that $\mathcal{Y}$ received from $\mathcal{X}$ in its session of π. We assume that $x_i \notin \hat{X}$ for $i \in [t]$ and will add the probability of this event to the distinguishing advantage. Under this assumption, $state_{\mathcal{X}}$ is equal to $\mathcal{S}(view_{\mathcal{Z}})$ since the set X in the state of $\mathcal{X}$ contains exactly $S = \{x_i\}_{i\in[t]}$ (in sorted order) and $q = |S|$. Then the statistical distance between $state_{\mathcal{X}}$ and $\mathcal{S}(view_{\mathcal{Z}})$ is the probability that $\hat{X} \cap \{x_i\}_{i\in[t]} \neq \emptyset$ which is at most $\frac{\mathsf{poly}(\lambda)}{2^\lambda}$ since $\mathcal{Z}$ is PPT and can make at most $\mathsf{poly}(\lambda)$ queries to π and π_D. □

A desirable property of our definition of weak deletion-compliance is that it is implied by the stronger definition. There are some technical hurdles in proving that fact directly, however, mainly caused by the issue of randomness discussed earlier and the fact that strong deletion-compliance deals with two distinct real and ideal executions. The data collector does not start off with the same random tape in each executions, and even if it did, then it could result in observable differences in the views of the environment since randomness is consumed for the protocols initiated by $\mathcal{Y}$. We introduce an assumption that is used in this section only to prove that strong deletion-compliance implies weak deletion-compliance for the data collectors that satisfy this assumption.

The assumption is as follows. We distinguish between two forms of randomness used by $\mathcal{X}$. First, $\mathcal{X}$ is allowed to use *private* randomness at initialization phase and during any protocol. Private randomness is defined such that changing the contents of the private random tape of $\mathcal{X}$ does not produce an observable change in the view of $\mathcal{Z}$ or $\mathcal{Y}$. The data collector $\mathcal{X}$ is also allowed a *public* random tape with the restriction that the content of that tape can be efficiently reconstructed using $view_{\mathcal{Z}}$ and $view_{\mathcal{Y}}$.

Definition 4. *We say that the machine $\mathcal{X}$ has* private/public–separable *random tape if it can be divided into two random tapes $priv_{\mathcal{X}}$ and $pub_{\mathcal{X}}$ that satisfy the following.*

- Private*: Let M be an arbitrary ITI interacting with $\mathcal{X}$ and let $view_M(R)$ be the view of M when interacting with $\mathcal{X}$ conditioned on $priv_{\mathcal{X}} = R$. Then for any values $R', R \in_R \{0,1\}^{\mathsf{poly}(\lambda)}$ for the private random tape $priv_{\mathcal{X}}$, $view_M(R) = view_M(R')$ with probability at least $1 - \mathsf{negl}(\lambda)$.*
- Public*: There exists an efficient function $f_{\mathcal{X}}(\cdot)$ such that $f_{\mathcal{X}}(view_{\mathcal{Z}}, view_{\mathcal{Y}})$ returns the content of public random tape $pub_{\mathcal{X}}$.*

Note that for every data collector given as example in (the full version of) [GGV20], its random tape can be divided in such a way. Intuitively, we ask that the data collector $\mathcal{X}$ be deterministic up to some random initialization that only affects the internal state of $\mathcal{X}$, and that all further randomness can be efficiently computed from the views of machines interacting with $\mathcal{X}$. We show that under the above assumption, if $\mathcal{X}$ is strongly deletion-compliant, then it is weakly deletion-compliant.

Theorem 2. *Let $(\mathcal{X}, \pi, \pi_D)$ be a strongly deletion-compliant data collector with public/private–separable random tape. Then $(\mathcal{X}, \pi, \pi_D)$ is weakly deletion-compliant.*

Proof. To compare the definition of weak deletion-compliance with that of strong deletion-compliance, we need to clarify the execution context. In [GGV20], there are two parallel executions: the real execution where $\mathcal{X}$ interacts with $\mathcal{Y}$ and $\mathcal{Z}$ through protocols π and π_D, and the ideal execution where $\mathcal{Y}$ is silent (i.e. it does not initiate any protocol). Moreover, in the formalism of [GGV20], $\mathcal{Y}$ is not allowed to send messages to $\mathcal{Z}$. For our definition, we are only interested in the real execution, where we introduce a simulator that reconstructs the state of $\mathcal{X}$ from only the view of $\mathcal{Z}$. We will use the real/ideal indistinguishability to show that this simulator works.

We consider the following simulator $\mathcal{S}$. On input $view_{\mathcal{Z}}$, it runs a new instance of $\mathcal{X}$ with a freshly initialized private random tape $priv_{\mathcal{X}}$ and, given the view of the environment $view_{\mathcal{Z}}$ it computes a public random tape $pub_{\mathcal{X}}$ of $\mathcal{X}$ consistent with that view (where the view of $\mathcal{Y}$ is assumed to be empty). It then simulates the execution of every protocol π and π_D in the same order they appear in $view_{\mathcal{Z}}$ by using the values of the outgoing messages of $\mathcal{Z}$ as incoming messages for $\mathcal{X}$ for every such protocol and using the random tapes $priv_{\mathcal{X}}$ and $pub_{\mathcal{X}}$ as the sources of randomness. After simulating every protocol in $view_{\mathcal{Z}}$, it outputs the state of its instance of $\mathcal{X}$.

Intuitively, the simulator can always simulate $\mathcal{X}$ in this way because the state of $\mathcal{X}$ is a deterministic function of its random tapes and of the views of $\mathcal{Y}$ and $\mathcal{Z}$. Since we may assume that $\mathcal{Y}$ is silent, by virtue of the indistinguishability of the real and ideal executions, the state of $\mathcal{X}$ is function only of $priv_{\mathcal{X}}$, $pub_{\mathcal{X}}$ and $view^I_{\mathcal{Z}}$ to which $\mathcal{S}$ has access to. By Definition 4, since $priv_{\mathcal{X}}$ has no impact on the view of $\mathcal{Z}$ and since $pub_{\mathcal{X}}$ is computed from this view, the above simulation will also produce responses from $\mathcal{X}$ that are consistent with the view of $\mathcal{Z}$. By the above observations, this simulation exactly replicates the state of $\mathcal{X}$ in the ideal execution where $\mathcal{Y}$ really is silent: $\mathcal{S}(view^I_{\mathcal{Z}}) = state^I_{\mathcal{X}}$.

The rest of the proof is the same whether we consider statistical or computational indistinguishability, so we use the notation "$\approx$" to represent either. We now compare $\mathcal{S}(view^I_{\mathcal{Z}})$ and $\mathcal{S}(view^R_{\mathcal{Z}})$. By assumption, we have that

$$(state^R_{\mathcal{X}}, view^R_{\mathcal{Z}}) \approx (state^I_{\mathcal{X}}, view^I_{\mathcal{Z}}) \ .$$

Seeking a contradiction, we suppose that $\mathcal{S}(view^I_{\mathcal{Z}}) \not\approx \mathcal{S}(view^R_{\mathcal{Z}})$. That means there exists a distinguisher $\mathcal{D}$ such that

$$\left|\Pr\left[\mathcal{D}(\mathcal{S}(view^I_{\mathcal{Z}})) = 1\right] - \Pr\left[\mathcal{D}(\mathcal{S}(view^R_{\mathcal{Z}})) = 1\right]\right| > \frac{1}{\mathsf{poly}(\lambda)} \ .$$

This allows us to define a distinguisher $\mathcal{D}'$ by $\mathcal{D}'(view_{\mathcal{Z}}) = \mathcal{D}\big(\mathcal{S}(view_{\mathcal{Z}})\big)$ to distinguish the view of $\mathcal{Z}$ in the real and ideal executions, which contradicts the fact that $view^I_{\mathcal{Z}} \approx view^R_{\mathcal{Z}}$. Therefore, we must have that $\mathcal{S}(view^I_{\mathcal{Z}}) \approx \mathcal{S}(view^R_{\mathcal{Z}})$. Finally, since $state^I_{\mathcal{X}} \approx state^R_{\mathcal{X}}$, we have that

$$\mathcal{S}(view^R_{\mathcal{Z}}) \approx \mathcal{S}(view^I_{\mathcal{Z}}) = state^I_{\mathcal{X}} \approx state^R_{\mathcal{X}} \ .$$

The time complexity of $\mathcal{S}$ is at most that of $\mathcal{X}$ in the real execution and of the function $f_{\mathcal{X}}$ that maps the view of $\mathcal{Z}$ to the public random tape of $\mathcal{X}$ which is assumed to be poly-time computable, and thus $\mathcal{S}$ is a PPT machine. □

Our definition of weak deletion-compliance also satisfies some form of converse of the above Theorem. We show that if a data-collector is weakly deletion-compliant and also protects $\mathcal{Y}$'s privacy in a precise manner, then it is strongly deletion-compliant. That is, we show that if we define privacy as what is implied by strong deletion-compliance – that the view of the environment is indistinguishable in the real and ideal execution where the deletion-requester is silent – then a private and weakly deletion-compliant data collector is strongly deletion-compliant.

Definition 5. *We say that a data-collector $(\mathcal{X}, \pi, \pi_D)$ is (computationally) ε–*privacy-preserving *if the view of $\mathcal{Z}$ in the real execution $view^R_{\mathcal{Z}}$ and the view of $\mathcal{Z}$ in an ideal execution $view^I_{\mathcal{Z}}$ where $\mathcal{Y}$ is replaced with a* silent *$\mathcal{Y}_0$ satisfy the following. For all (*PPT*) distinguisher $\mathcal{D}$,*

$$\left|\Pr[\mathcal{D}(view^R_{\mathcal{Z}}) = 1] - \Pr[\mathcal{D}(view^I_{\mathcal{Z}}) = 1]\right| \leq \varepsilon \ .$$

Theorem 3. *If $(\mathcal{X}, \pi, \pi_D)$ is both ε_1–weakly deletion-compliant and ε_2–privacy-preserving, then it is $2(\varepsilon_1 + \varepsilon_2)$–strongly deletion-compliant.*

Proof. By assumption of ε_1–weak deletion-compliance, there exists $\mathcal{S}$ such that for any $\mathcal{D}$,

$$\left|\Pr[\mathcal{D}(state^R_{\mathcal{X}}, view^R_{\mathcal{Z}}) = 1] - \Pr[\mathcal{D}(\mathcal{S}(view^R_{\mathcal{Z}}), view^R_{\mathcal{Z}}) = 1]\right| \leq \varepsilon_1 \ .$$

Furthermore, by the assumption of ε_2–privacy-preserving, $view^R_{\mathcal{Z}} \approx_{\varepsilon_2} view^I_{\mathcal{Z}}$. It directly follows that $\mathcal{S}(view^R_{\mathcal{Z}}) \approx_{\varepsilon_2} \mathcal{S}(view^I_{\mathcal{Z}})$. We now argue that $\mathcal{S}(view^I_{\mathcal{Z}})$ outputs a valid state for $\mathcal{X}$ (or something indistinguishable from a valid state). Indeed, if it wasn't the case, then one could either distinguish between $state^R_{\mathcal{X}}$ and $\mathcal{S}(view^R_{\mathcal{Z}})$ or between $\mathcal{S}(view^R_{\mathcal{Z}})$ and $\mathcal{S}(view^I_{\mathcal{Z}})$ which is not possible except with probability at most $\varepsilon_1 + \varepsilon_2$. The state of $\mathcal{X}$ produced as output of $\mathcal{S}(view^I_{\mathcal{Z}})$ must be the product of an ideal execution, since $view^I_{\mathcal{Z}}$ is the view of $\mathcal{Z}$ in an ideal execution where $\mathcal{Y}$ is replaced with the silent $\mathcal{Y}_0$. If we let $state^I_{\mathcal{X}}$ denote the true state of $\mathcal{X}$ in the ideal execution, we have that

$$\begin{aligned}
\Delta((state^R_{\mathcal{X}}, view^R_{\mathcal{Z}}), (state^I_{\mathcal{X}}, view^I_{\mathcal{Z}})) &\leq \Delta((state^R_{\mathcal{X}}, view^R_{\mathcal{Z}}), (\mathcal{S}(view^R_{\mathcal{Z}}), view^R_{\mathcal{Z}})) \\
&\quad + \Delta((\mathcal{S}(view^R_{\mathcal{Z}}), view^R_{\mathcal{Z}}), (state^I_{\mathcal{X}}, view^I_{\mathcal{Z}})) \\
&\leq \Delta((state^R_{\mathcal{X}}, view^R_{\mathcal{Z}}), (\mathcal{S}(view^R_{\mathcal{Z}}), view^R_{\mathcal{Z}})) \\
&\quad + \Delta((\mathcal{S}(view^R_{\mathcal{Z}}), view^R_{\mathcal{Z}}), (\mathcal{S}(view^I_{\mathcal{Z}}), view^I_{\mathcal{Z}})) \\
&\quad + \Delta((\mathcal{S}(view^I_{\mathcal{Z}}), view^I_{\mathcal{Z}}), (state^I_{\mathcal{X}}, view^I_{\mathcal{Z}})) \\
&\leq \varepsilon_1 + \varepsilon_2 + (\varepsilon_1 + \varepsilon_2)
\end{aligned}$$

□

4 Example: a Deletion-Compliant Message Board

We present a construction of data-collector that implements a public message board where users may post messages and fetch new messages. The data-collector presented in Fig. 2 is obviously not strongly deletion-compliant since the adversary can see the deletion-requester's

messages. The data collector allows machines to post messages to the message board through a protocol $\pi_{\texttt{post}}$. To post a message $m \in \{0,1\}^*$, the client machine sends it to $\mathcal{X}$ along with a *key* k used for deletion[5]. This key is used for uniquely identifying message and for authentication when deleting from the message board and is stored alongside the message in `list`. Letting the client maching choose k instead of $\mathcal{X}$ lets us use the results of Section 5.2 to show weak deletion-compliance of $\mathcal{X}$. To fetch the list of posted messages, a machine invokes $\pi_{\texttt{fetch}}$, $\mathcal{X}$ then sends the list of messages contained in the list (without the associated keys) in order of insertion. To delete using π_D, $\mathcal{X}$ searches the list for the key k and removes from the list the entry that contains this key.

Initialization:

1. Initialize an empty `list` with `insert` and `remove` operations.

Protocol $\pi_{\texttt{post}}(k, m)$:

1. Execute `list.insert`(k, m).

Protocol $\pi_{\texttt{fetch}}()$:

1. For each pair (k, m) in `list`,
2. Send m to the other machine

Protocol $\pi_D(k)$:

1. Search `list` for an entry of the form $(k, \cdot)$.
2. If found, execute `list.remove` on that entry and return 1.
3. If not found, return 0.

Fig. 2: The "message board" data-collector $\mathcal{X}$.

A seemingly trivial, yet unfruitful approach to constructing the simulator $\mathcal{S}$ of Definition 3 is to argue that $\mathcal{Z}$ receives complete information on the state of $\mathcal{X}$ when protocol $\pi_{\texttt{fetch}}$ is executed. However, the simulator must depend only on $\mathcal{X}$ and work for any environment, and in particular for $\mathcal{Z}$ that never invokes $\pi_{\texttt{fetch}}$.

Theorem 4. *If* `list` *is a history-independent implementation of a list data structure, then the data-collector of Fig. 2 is weakly deletion-compliant.*

We can use the techniques of Section 5.2 to prove that this data collector satisfies weak deletion-compliance (Definition 3), so we defer the proof to that section. Nevertheless, it is instructive to explicitly describe a simulator for the data-collector $\mathcal{X}$ and sketch why it satisfies Definition 3. We construct $\mathcal{S}$ as follows:

1. Initalize an empty `list`.
2. For each protocol τ in $view_{\mathcal{Z}}$,
   - if $\tau = \pi_{\texttt{fetch}}$, do nothing.
   - if $\tau = \pi_{\texttt{post}}$, let (k, m) be the message sent by $\mathcal{Z}$ to $\mathcal{X}$; append (k, m) to `list`.

[5] Note that this key is not strictly necessary for deletion-compliance. $\mathcal{X}$ would still be deletion-compliant if anyone could delete every message.

– if $\tau = \pi_D$, let k be the key sent by $\mathcal{Z}$; if `list` has an entry of the form $(k, \cdot)$, execute `list.remove` to remove that entry.

3. The output of $\mathcal{S}$ is `list`.

The above simulator works because if `list` is instantiated with a history-independent implementation, then there is no distinction in the state of $\mathcal{X}$ between the case where each invocation of $\pi_{\texttt{post}}(k, m)$ is *ultimately* followed by the corresponding $\pi_D(k)$ and the case where $\pi_{\texttt{post}}(k, m)$ is not invoked at all.

5 Properties of Weak Deletion-Compliance

Our weaker definition of deletion-compliance has several appealing properties. In this section, we show the many ways in which weak deletion-compliance is composable.

5.1 Composability of Deletion-Requesters and Data-Collectors

We first show that the weak deletion-compliance error scales linearly with the number of instances of π initiated by $\mathcal{Y}$. As in [GGV20], we call an execution *k–representative* if the deletion-requester $\mathcal{Y}$ initiates k instances of π with $\mathcal{X}$.

Definition 6. *A deletion-requester is* k-representative *for $k \in \mathbb{N}$ if, when interacting with a data-collector with protocols (π, π_D), it initiates at most k instances of π.*

Theorem 5. *Let $(\mathcal{X}, \pi, \pi_D)$ be a data-collector with 1-representative weak deletion-compliance error ε_1. Then for all $k \in \mathbb{N}$, the k-representative weak deletion-compliance error ε_k of $\mathcal{X}$ is at most $2k \cdot \varepsilon_1$.*

Proof. Fix $\lambda \in \mathbb{N}$. We prove by induction on k that $\varepsilon_k(\lambda) \leq 2k\varepsilon_1(\lambda)$. We omit λ for the rest of the proof. Let $k \geq 2$ and assume that $\varepsilon_{k-1} \leq (k-1)\varepsilon_1$. Without loss of generality, for any k-representative deletion-requester $\mathcal{Y}$, we can assume $\mathcal{Y}$ is composed of k ITIs $\mathcal{Y}_1, \ldots, \mathcal{Y}_k$ allowed to read and write on each other's incoming and outgoing tapes where each $\mathcal{Y}_i$ initiates π with $\mathcal{X}$ at most once.

We can represent the execution with k–representative $\mathcal{Y}$ and environment $\mathcal{Z}$ as an execution with 1–representative $\mathcal{Y}_k$ and a new environment $\mathcal{Z}'$ so that in the new execution, $\mathcal{Y}_k$ interacts with $\mathcal{X}$ and $\mathcal{Y}_1, \ldots, \mathcal{Y}_{k-1}$ are part of the environment. Let EXEC_1 denote this 1–representative execution. In particular, the view of $\mathcal{Z}'$ contains the view of $\mathcal{Z}$ and the views of $\mathcal{Y}_1, \ldots, \mathcal{Y}_{k-1}$.

Consider a simulator $\mathcal{S}''$ such that

$$\max_{\mathcal{Z},\mathcal{Y},\mathcal{D}} \min_{\mathcal{S}'} \left| \Pr\left[\mathcal{D}(state_{\mathcal{X}}, view_{\mathcal{Z}'}) = 1\right] - \Pr\left[\mathcal{D}(\mathcal{S}'(view_{\mathcal{Z}'}), view_{\mathcal{Z}'}) = 1\right] \right| \tag{4}$$

attains the minimum. By assumption, the value of (4) is the 1–representative error of $\mathcal{X}$, which is exactly ε_1.

Now, consider the k–representative error

$$\varepsilon_k = \max_{\mathcal{Z},\mathcal{Y},\mathcal{D}} \min_{\mathcal{S}} \left| \Pr\left[\mathcal{D}(state_{\mathcal{X}}, view_{\mathcal{Z}}) = 1\right] - \Pr\left[\mathcal{D}(\mathcal{S}(view_{\mathcal{Z}}), view_{\mathcal{Z}}) = 1\right] \right| . \tag{5}$$

By the triangle inequality, (5) is upper-bounded by

$$\begin{aligned}\varepsilon_k &\leq \max_{\mathcal{Z},\mathcal{Y},\mathcal{D}} \Big|\Pr\big[\mathcal{D}(state_{\mathcal{X}}, view_{\mathcal{Z}}) = 1\big] - \Pr\big[\mathcal{D}(\mathcal{S}'(view_{\mathcal{Z}'}), view_{\mathcal{Z}}) = 1\big]\Big| \\ &\quad + \max_{\mathcal{Z},\mathcal{Y},\mathcal{D}} \min_{\mathcal{S}} \Big|\Pr\big[\mathcal{D}(\mathcal{S}'(view_{\mathcal{Z}'}), view_{\mathcal{Z}}) = 1\big] - \Pr\big[\mathcal{D}(\mathcal{S}(view_{\mathcal{Z}}), view_{\mathcal{Z}}) = 1\big]\Big| \\ &\leq \varepsilon_1 + \max_{\mathcal{Z},\mathcal{Y},\mathcal{D}} \min_{\mathcal{S}} \Big|\Pr\big[\mathcal{D}(\mathcal{S}'(view_{\mathcal{Z}'}), view_{\mathcal{Z}}) = 1\big] - \Pr\big[\mathcal{D}(\mathcal{S}(view_{\mathcal{Z}}), view_{\mathcal{Z}}) = 1\big]\Big|\end{aligned} \tag{6}$$

where the last inequality follows from (4) and the fact that giving the distinguisher less information in (6), i.e. the view of $\mathcal{Z}$ instead of $\mathcal{Z}'$, can only decrease the distinguishing probability.

We now bound the remaining part of the previous equation. Let $\mathcal{O}$ be the output of $\mathcal{S}'(view^{\mathcal{X}}_{\mathcal{Z}'})$. We observe the two following facts about $\mathcal{O}$:

1. By construction of $\mathcal{S}'$, $\mathcal{O}$ is indistinguishable from the state of some data collector resulting from a valid execution.
2. $view^{\mathcal{X}}_{\mathcal{Z}'}$ contains exactly the interactions between $\mathcal{Y}_1, \ldots, \mathcal{Y}_{k-1}, \mathcal{Z}$ and $\mathcal{X}$. In particular, it has access to the views of the 1-representative deletion-requesters $\mathcal{Y}_1, \ldots, \mathcal{Y}_{k-1}$, so it has the complete view of a valid $(k-1)$–representative execution.

From the two above observations, we deduce that the output $\mathcal{O}$ of $\mathcal{S}'$ is indistinguishable from the state of the state of the data collector obtained as the result of a $(k-1)$–representative execution where $\mathcal{Y}_1, \ldots, \mathcal{Y}_{k-1}$ and the environment $\mathcal{Z}$ interact with $\mathcal{X}$. Note that we do not require this execution to be exactly as the 1–representative execution EXEC_1 defined above, but only that it is a valid $(k-1)$–representative execution. Let $state'_{\mathcal{X}}$ be the state of $\mathcal{X}$ in this execution. As we have argued, $\mathcal{O} \approx_{\varepsilon_1} state'_{\mathcal{X}}$ corresponds to the state of $\mathcal{X}$ in a $(k-1)$-representative execution. It follows from the inductive hypothesis that there exists a simulator $\mathcal{S}$ for any $(k-1)$–representative execution, and so by the triangle inequality

$$\max_{\mathcal{Z},\mathcal{Y},\mathcal{D}} \min_{\mathcal{S}} \Big|\Pr\big[\mathcal{D}(\mathcal{S}'(view_{\mathcal{Z}'}), view_{\mathcal{Z}}) = 1\big] - \Pr\big[\mathcal{D}(\mathcal{S}(view_{\mathcal{Z}}), view_{\mathcal{Z}}) = 1\big]\Big| \leq \varepsilon_1 + \varepsilon_{k-1} \ . \tag{7}$$

Combining the two bounds for (6) and (7), we obtain

$$\Big|\Pr\big[\mathcal{D}(state_{\mathcal{X}}, view_{\mathcal{Z}}) = 1\big] - \Pr\big[\mathcal{D}(\mathcal{S}(view_{\mathcal{Z}}), view_{\mathcal{Z}}) = 1\big]\Big| \leq 2\varepsilon_1 + \varepsilon_{k-1} \leq 2k\varepsilon_1,$$

where we used the induction hypothesis $\varepsilon_{k-1} \leq 2(k-1)\varepsilon_1$. □

For data-collectors, there are two natural ways of combining them: in parallel where $\mathcal{Y}$ may interact with either $\mathcal{X}_1$ or $\mathcal{X}_2$, and sequentially where $\mathcal{Y}$ interacts with $\mathcal{X}_1$ and $\mathcal{X}_1$ interacts with another data-collector $\mathcal{X}_2$ as would be the case when delegating parts of its storage or computation.

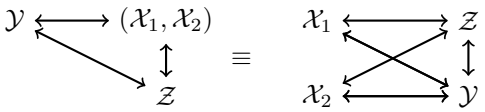
Definition 7. *Given two data-collectors $(\mathcal{X}_j, \pi_j, \pi_{j,D})$, $j = 1, 2$, we define the* parallel composition *$((\mathcal{X}_1, \mathcal{X}_2), (\pi_1, \pi_2), (\pi_{1,D}, \pi_{2,D}))$ of $\mathcal{X}_1$ and $\mathcal{X}_2$ as the data-collector with the following behavior:*

– *the data-collector $(\mathcal{X}_1, \mathcal{X}_2)$ runs an instance of each machine $\mathcal{X}_1$ and $\mathcal{X}_2$, each running independently of one another (i.e. having no access to each other's incoming/outgoing tapes);*

– *when $\mathcal{Y}$ or $\mathcal{Z}$ initiates an instance of π_i (resp. π_i^D), $i \in \{1,2\}$, with $(\mathcal{X}_1, \mathcal{X}_2)$, it runs protocol π_i (resp. π_i^D) with machine $\mathcal{X}_i$;*
– *the state of $(\mathcal{X}_1, \mathcal{X}_2)$ consists of the state of machines $\mathcal{X}_1$ and $\mathcal{X}_2$: $state_{(\mathcal{X}_1,\mathcal{X}_2)} = (state_{\mathcal{X}_1}, state_{\mathcal{X}_2})$.*

Theorem 6. *If $(\mathcal{X}_1, \pi_1, \pi_{1,D})$ and $(\mathcal{X}_2, \pi_2, \pi_{2,D})$ are statistical (resp. computational) weakly deletion-compliant with error ε_1 and ε_2, respectively, then their parallel composition is also statistical (resp. computational) weakly deletion-compliant with error $\varepsilon_1 + \varepsilon_2$.*

Proof. The proof is identical for statistical and computational flavors of weak deletion-compliance. The interactions of $\mathcal{Y}$ and $\mathcal{Z}$ with $(\mathcal{X}_1, \mathcal{X}_2)$ in the real execution can be represented by an equivalent execution with $\mathcal{X}_1$ and $\mathcal{X}_2$:



We can give two interpretation to this execution to invoke the weak deletion-compliance of $\mathcal{X}_i$: we can combine $\mathcal{X}_2$ to the environment and $\mathcal{X}_1$ becomes the data-collector, or we can combine $\mathcal{X}_1$ to the environment and $\mathcal{X}_2$ becomes the data-collector.

Let $\mathcal{Z}'$ denote the new environment in both executions. In the first case, we see that $view_{\mathcal{Z}'}^{\mathcal{X}_1} = view_{(\mathcal{X}_2,\mathcal{Z})}^{\mathcal{X}_1} = view_{\mathcal{Z}}^{\mathcal{X}_1}$, because $\mathcal{X}_1$ and $\mathcal{X}_2$ do not interact together. Similarly, in the second case, we have $view_{\mathcal{Z}'}^{\mathcal{X}_2} = view_{(\mathcal{X}_1,\mathcal{Z})}^{\mathcal{X}_2} = view_{\mathcal{Z}}^{\mathcal{X}_2}$. By the weak deletion-compliance property of $\mathcal{X}_1$ and $\mathcal{X}_2$, there exist two simulators $\mathcal{S}_1$ and $\mathcal{S}_2$ such that

$$\mathcal{S}_1(view_{(\mathcal{X}_2,\mathcal{Z})}^{\mathcal{X}_1}) \approx_{\varepsilon_1} state_{\mathcal{X}_1} \qquad \text{and} \qquad \mathcal{S}_2(view_{(\mathcal{X}_1,\mathcal{Z})}^{\mathcal{X}_2}) \approx_{\varepsilon_2} state_{\mathcal{X}_2} \ .$$

Since we have $view_{(\mathcal{X}_{3-j},\mathcal{Z})}^{\mathcal{X}_j} = view_{\mathcal{Z}}^{\mathcal{X}_j}$ for $j \in \{1,2\}$, and because $view_{\mathcal{Z}}^{\mathcal{X}_j}$ is contained in $view_{\mathcal{Z}}^{(\mathcal{X}_1,\mathcal{X}_2)}$, we can construct a simulator $\mathcal{S}$ that computes $view_{\mathcal{Z}}^{\mathcal{X}_j}$ from $view_{\mathcal{Z}}^{(\mathcal{X}_1,\mathcal{X}_2)}$, and outputs

$$\mathcal{S}(view_{\mathcal{Z}}^{(\mathcal{X}_1,\mathcal{X}_2)}) = \big(\mathcal{S}_1(view_{\mathcal{Z}}^{\mathcal{X}_1}), \mathcal{S}_1(view_{\mathcal{Z}}^{\mathcal{X}_2})\big) \ .$$

By the triangle inequality, we have

$$\begin{aligned}
&\Big|\Pr\big[\mathcal{D}(state_{\mathcal{X}}, view_{\mathcal{Z}}) = 1\big] - \Pr\big[\mathcal{D}(\mathcal{S}(view_{\mathcal{Z}}), view_{\mathcal{Z}}) = 1\big]\Big| \\
&= \Big|\Pr[\mathcal{D}((\mathcal{S}_1(view_{\mathcal{Z}}^{\mathcal{X}_1}), \mathcal{S}_1(view_{\mathcal{Z}}^{\mathcal{X}_2}), view_{\mathcal{Z}}) = 1] \\
&\qquad - \Pr[\mathcal{D}((state_{\mathcal{X}_1}, state_{\mathcal{X}_2}), view_{\mathcal{Z}}) = 1]\Big| \\
&\leq \Big|\Pr[\mathcal{D}((\mathcal{S}_1(view_{\mathcal{Z}}^{\mathcal{X}_1}), \mathcal{S}_1(view_{\mathcal{Z}}^{\mathcal{X}_2})), view_{\mathcal{Z}}) = 1] \\
&\qquad - \Pr[\mathcal{D}((state_{\mathcal{X}_1}, \mathcal{S}_1(view_{\mathcal{Z}}^{\mathcal{X}_2})), view_{\mathcal{Z}}) = 1]\Big| \\
&\quad + \Big|\Pr[\mathcal{D}((state_{\mathcal{X}_1}, \mathcal{S}_1(view_{\mathcal{Z}}^{\mathcal{X}_2})), view_{\mathcal{Z}}) = 1] \\
&\qquad - \Pr[\mathcal{D}((state_{\mathcal{X}_1}, state_{\mathcal{X}_2}), view_{\mathcal{Z}}) = 1]\Big| \\
&\leq \varepsilon_1 + \varepsilon_2
\end{aligned}$$

as required. □

Sequential composition of data-collectors is defined as follows.

Definition 8. *Given two data-collectors* $(\mathcal{X}_j, \pi_j, \pi_j^D)$, $j = 1, 2$, *we define the* sequential composition $(\mathcal{X}_2 \circ \mathcal{X}_1, \pi_2 \circ \pi_1, \pi_2^D \circ \pi_1^D)$ *of* $\mathcal{X}_1$ *and* $\mathcal{X}_2$ *as the data-collector with the following behavior:*

1. *the data-collector* $\mathcal{X}_2 \circ \mathcal{X}_1$ *runs an instance of each machine* $\mathcal{X}_1$ *and* $\mathcal{X}_2$*, each running independently of one another (i.e. having no access to each other's tapes);*
2. *whenever* $\mathcal{Y}$ *or* $\mathcal{Z}$ *initiates an instance of* $\pi_2 \circ \pi_1$ *with machine* $\mathcal{X}_2 \circ \mathcal{X}_1$*, run protocol* π_1 *between machine* $\mathcal{X}_1$ *and* $\mathcal{Y}$ *(or* $\mathcal{Z}$*) which in turn initiates an instance of* π_2 *between* $\mathcal{X}_1$ *and* $\mathcal{X}_2$*;*
3. *the state of* $\mathcal{X}_2 \circ \mathcal{X}_1$ *consists of the state of machines* $\mathcal{X}_1$ *and* $\mathcal{X}_2$*:* $state_{\mathcal{X}_2 \circ \mathcal{X}_1} = (state_{\mathcal{X}_1}, state_{\mathcal{X}_2})$.

Our goal is to show that if $\mathcal{X}_1$ and $\mathcal{X}_2$ are both weakly deletion-compliant, then their sequential composition is weakly deletion-compliant. To do so, we need to reconstruct the state of both $\mathcal{X}_1$ and $\mathcal{X}_2$ using their respective simulators. The difficulty is that in the case of $\mathcal{X}_2 \circ \mathcal{X}_1$, the simulator only has access to the view of $\mathcal{Z}$, and not the view of the interactions between $\mathcal{X}_1$ and $\mathcal{X}_2$. This view may contain elements that are necessary to reconstruct the individual states, but that are inaccessible to the simulator. To circumvent this issue, we add the restriction that the view between $\mathcal{X}_1$ and $\mathcal{X}_2$ for the protocol sessions that were first initiated by $\mathcal{Z}$ can be simulated by $\mathcal{S}$ having only access to the view of $\mathcal{Z}$. We call property ε*–independence* if this view can be reconstructed with error ε. It is satisfied for example when the messages between $\mathcal{X}_1$ and $\mathcal{X}_2$ in a session of π_2 that follows π_1 initiated by M is an efficient function of only the previous interactions of M and $\mathcal{X}_1$. In this scenario, seeing all of the interactions between M and $\mathcal{X}_1$ is sufficient to replicate the resulting interactions between $\mathcal{X}_1$ and $\mathcal{X}_2$.

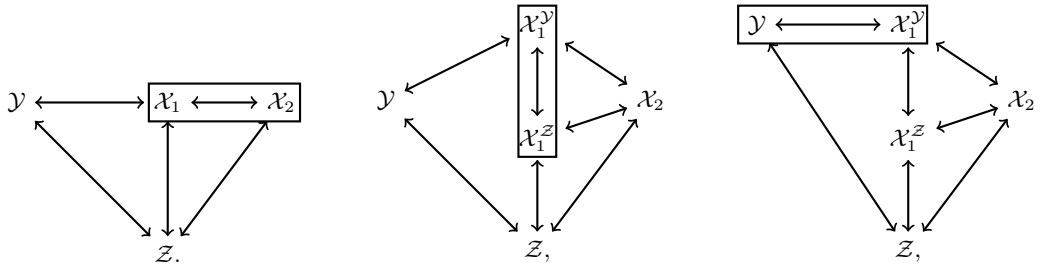
Definition 9. *Let* M *be an arbitrary ITI and let* $view^{\mathcal{X}_2}_{\mathcal{X}_1|M}$ *denote the view between* $\mathcal{X}_1$ *and* $\mathcal{X}_2$ *restricted to the instances of* π_2 *that follow an instance of* π_1 *initiated by* M*. We say that the sequential composition* $\mathcal{X}_2 \circ \mathcal{X}_1$ *of* $\mathcal{X}_1$ *and* $\mathcal{X}_2$ *is (computationally)* ε–independent *if there exists a* PPT *machine* $\mathcal{V}$ *such that for all ITI* M *interacting with* $\mathcal{X}_1$ *using protocol* π_1*, and for all (*PPT*) distinguisher* $\mathcal{D}$*,*

$$\left|\Pr[\mathcal{D}(\mathcal{V}(view^{\mathcal{X}_1}_M)) = 1] - \Pr[\mathcal{D}(view^{\mathcal{X}_2}_{\mathcal{X}_1|M}) = 1]\right| \leq \varepsilon(\lambda) \ .$$

Theorem 7. *If* $(\mathcal{X}_1, \pi_1, \pi_1^D)$ *and* $(\mathcal{X}_2, \pi_2, \pi_2^D)$ *are (computationally) weakly deletion-compliant data-collectors with respective error* ε_1 *and* ε_2*, then the (computationally)* ε*–independent sequential composition* $(\mathcal{X}_2 \circ \mathcal{X}_1, \pi_2 \circ \pi_1, \pi_2^D \circ \pi_1^D)$ *is (computationally) weakly deletion-compliant with error* $\varepsilon_1 + \varepsilon_2 + 2\varepsilon$*.*

Proof. We construct a simulator $\mathcal{S}$ such that given the view $view^{\mathcal{X}_2 \circ \mathcal{X}_1}_{\mathcal{Z}}$, produces an output indistinguishable from $(state_{\mathcal{X}_1}, state_{\mathcal{X}_2})$. To do this, we rely on the weak deletion-compliance property of $\mathcal{X}_j$ for $j = 1, 2$ and assume the existence of two simulators $\mathcal{S}_1$ and $\mathcal{S}_2$ such that $\mathcal{S}_j$ can recreate the state of $\mathcal{X}_j$ in an execution from the view of the environment. We now show how to simulate the state of $\mathcal{X}_2 \circ \mathcal{X}_1$ from the view of $\mathcal{Z}$ using $\mathcal{S}_1$ and $\mathcal{S}_2$.

We begin by constructing the state of $\mathcal{X}_2$ using $\mathcal{S}_2$. Observe that in the real execution (Fig. 3a), we can divide $\mathcal{X}_1$ into two parts $\mathcal{X}_1^{\mathcal{Y}}$ and $\mathcal{X}_1^{\mathcal{Z}}$ as follows (Fig. 3b). Machine $\mathcal{X}_1^{\mathcal{Y}}$ interacts only with $\mathcal{Y}$ and machine $\mathcal{X}_1^{\mathcal{Z}}$ only with $\mathcal{Z}$. We allow $\mathcal{X}_1^{\mathcal{Y}}$ and $\mathcal{X}_1^{\mathcal{Z}}$ to arbitrarily message each other to faithfully replicate the behavior of $\mathcal{X}_1$ from $\mathcal{Y}$'s and $\mathcal{Z}$'s points of view.



(a) Real execution with data collector $\mathcal{X}_2 \circ \mathcal{X}_1$.

(b) Simulate $\mathcal{X}_1$ with two new ITIs.

(c) Defining a new deletion requester and environment.

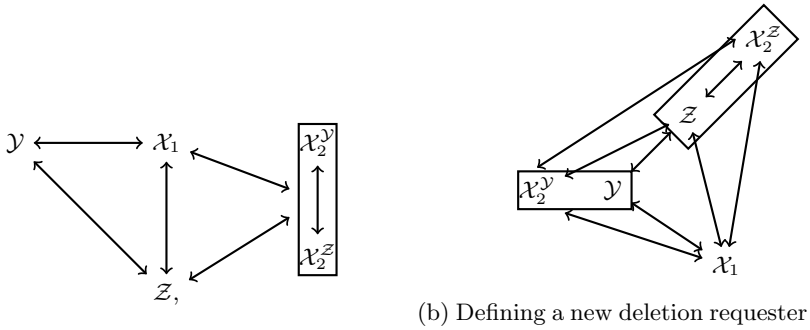
Fig. 3: Simulating the state of $\mathcal{X}_1$ for sequential composition.

Note that neither machine knows whether it is interacting with $\mathcal{Y}$ or $\mathcal{Z}$ since in reality, it is interacting with ITIs under their control, but this is not necessary for the proof; we only require that those machines are well-defined once we fix $\mathcal{Y}$ and $\mathcal{Z}$. We can combine $\mathcal{X}_1^{\mathcal{Y}}$ with $\mathcal{Y}$ to define a new deletion-requester $\mathcal{Y}'$, and $\mathcal{X}_1^{\mathcal{Z}}$ with $\mathcal{Z}$ to define a new environment $\mathcal{Z}'$. Note that this corresponds to a valid execution with data collector $\mathcal{X}_2$, deletion requester $\mathcal{Y}'$ and environment $\mathcal{Z}'$ because any protocol π_2 initiated by $\mathcal{Y}'$ will be followed by the corresponding π_2^D in the terminal phase since those are exactly the protocols π_2 that follow the π_1's initiated by $\mathcal{Y}$. By assumption there exists a simulator $\mathcal{S}_2$ that can simulate the state of $\mathcal{X}_2$ in any execution from the view of the environment $\mathcal{Z}'$. We thus have that $\mathcal{S}_2(view_{\mathcal{Z}'}^{\mathcal{X}_2}) \approx state_{\mathcal{X}_2}$. By the assumption that the composition is ε–independent, $view_{\mathcal{Z}'}^{\mathcal{X}_2}$, which consists of the view of any protocol between $\mathcal{X}_1^{\mathcal{Z}}$ and $\mathcal{X}_2$ initiated by $\mathcal{Z}$, corresponds to $view_{\mathcal{X}_1|\mathcal{Z}}^{\mathcal{X}_2}$ from Definition 9, and can thus be (approximately) computed efficiently from $view_{\mathcal{Z}}^{\mathcal{X}_1}$ using a function $\mathcal{V}_1$.

To simulate the state of $\mathcal{X}_1$ using $\mathcal{S}_1$, the idea is similar. Going back to the real execution (Fig. 3a), we can separate $\mathcal{X}_2$ in two parts: $\mathcal{X}_2^{\mathcal{Y}}$ receives the requests from $\mathcal{X}_1$ originating from $\mathcal{Y}$ and $\mathcal{X}_2^{\mathcal{Z}}$ receives those from $\mathcal{X}_1$ originating from $\mathcal{Z}$ (again, neither machine knows which party initiated the session of π_1 that resulted in its interaction with $\mathcal{X}_1$). The two components can communicate in the interest of simulating faithfully the execution of $\mathcal{X}_2$. This alternative execution is depicted in Fig. 4a. We now define a new environment $\mathcal{Z}' = (\mathcal{Z}, \mathcal{X}_2^{\mathcal{Z}})$ and a new deletion-requester $\mathcal{Y}' = (\mathcal{Y}, \mathcal{X}_2^{\mathcal{Y}})$ (Fig. 4b). The new machines $\mathcal{Y}'$ and $\mathcal{Z}'$ now interact with $\mathcal{X}_1$ through the protocols $\pi_2 \circ \pi_1$ and $\pi_2^D \circ \pi_1^D$, e.g. when $\mathcal{Y}'$ initiates π_1 with $\mathcal{X}_1$, $\mathcal{X}_1$ initiates a session of π_2 with $\mathcal{Y}'$, and after π_2 concludes, they finish the session of π_1. Since every protocol between $\mathcal{Y}'$ and $\mathcal{X}_1$ is initiated by $\mathcal{Y}$ (and thus followed by the corresponding deletion request), machine $\mathcal{Y}'$ still satisfies the definition of a deletion requester. Since $\mathcal{X}_1$ is weakly deletion-compliant, there exists a simulator $\mathcal{S}_1$ such that $\mathcal{S}_1(view_{\mathcal{Z}'}^{\mathcal{X}_1}) \approx state_{\mathcal{X}_1}$. Note that $view_{\mathcal{Z}'}^{\mathcal{X}_1}$ can be efficiently computed from $view_{\mathcal{Z}}^{\mathcal{X}_1}$ by the assumption of ε–independence: $view_{\mathcal{Z}'}^{\mathcal{X}_1}$ corresponds to $view_{\mathcal{X}_2|\mathcal{Z}}^{\mathcal{X}_1}$, so there is an efficient algorithm $\mathcal{V}_2$ such that $\mathcal{V}_2(view_{\mathcal{Z}}^{\mathcal{X}_1}) \approx_\varepsilon view_{\mathcal{Z}'}^{\mathcal{X}_1}$.

To conclude the proof, we define the simulator $\mathcal{S}$ that, given $view_{\mathcal{Z}}^{\mathcal{X}_2 \circ \mathcal{X}_1}$, it first computes $view_{(\mathcal{Z}, \mathcal{X}_2^{\mathcal{Z}})}^{\mathcal{X}_1}$ and $view_{(\mathcal{X}_1^{\mathcal{Z}}, \mathcal{Z})}^{\mathcal{X}_2}$ using $\mathcal{V}_1$ and $\mathcal{V}_2$, then applies the corresponding simulator: on input $view_{\mathcal{Z}}$, $\mathcal{S}$ outputs

$$\big(\mathcal{S}_1(\mathcal{V}_1(view_{\mathcal{Z}})), \mathcal{S}_2(\mathcal{V}_2(view_{\mathcal{Z}}))\big) \approx_{\varepsilon'} state_{\mathcal{X}_2 \circ \mathcal{X}_1}$$



(a) Simulating $\mathcal{X}_2$ using two new ITIs.

(b) Defining a new deletion requester and environment.

Fig. 4: Simulating the state of $\mathcal{X}_2$ for sequential composition.

where by the triangle inequality, $\varepsilon' \leq \varepsilon_1 + \varepsilon_2 + 2\varepsilon$ □

5.2 History Independence Implies Weak Deletion-Compliance

History independence is a concept introduced by Naor and Teague [NT01] to capture the notion that the memory representation of an abstract data structure (ADS) does not reveal any more information than what can be inferred from the content or state of the data structure. In particular, a history independent implementation does not reveal the history of operations that lead to its current state.

As a simple example of a history independent data structure, consider the abstract data structure consisting of a set with insertion and deletion. A history independent implementation of this ADS could consist of keeping the elements from the set in a sorted list. The memory representation (the list) is uniquely determined by the state of the ADS (the elements in the set). More generally, any implementation of an ADS that has a *canonical* representation for each ADS state is history independent [Har+05].

The original work on deletion-compliance [GGV20] uses the example of a history independent dictionary from [NT01] to build a strongly deletion-compliant data-collector that acts as a store of information. We show that when we drop the privacy requirements of strong deletion-compliance and instead adopt the weaker definition, history independence is a sufficient condition to obtain weak deletion-compliance for a broad class of data-collectors. We will use the more general definition of history independence for arbitrary ADS of [Har+05].

An abstract data structure can be represented as a graph where the nodes are the possible states of the ADS and the edges are operations that send the ADS from one state to the next. Let A and B be states of an ADS. A sequence of operations S (i.e. a path in the graph) that takes state A to state B is denoted by $A \xrightarrow{S} B$. The notation "$a \in A$" means that a is a *memory representation* of ADS state A for some implied implementation of the ADS. We let $\Pr[a \xrightarrow{S} b]$ denote the probability that, starting from memory representation a of state A, the sequence of operations S maps to the memory representation b of state B

Definition 10 (Strong History Independence [Har+05]). *An ADS implementation is strongly history independent if, for any two sequences of operation S and T that take the data structure from state A to state B, the distribution over memory representations after*

S is identical to the distribution after T. That is,

$$(A \xrightarrow{S} B) \text{ and } (A \xrightarrow{T} B) \implies \forall a \in A, \forall b \in B,\ \Pr[a \xrightarrow{S} b] = \Pr[a \xrightarrow{T} b].$$

We define deletion in the context of abstract data structure as follows.

Definition 11. *A* deletion operation *of an ADS operation p is an operation p_D such that for all sequences of operations R, S, T and every states A, B, we have that $A \xrightarrow{RpSp_DT} B$ and $A \xrightarrow{RST} B$.*

We now consider a data-collector $(\mathcal{X}, \pi, \pi_D)$ whose state transition function can be described by an abstract data structure. We assume that each session s of protocol π in an execution correspond to a sequence of operations Π^s in the abstract data structure. Naturally, we ask that π_D defines a sequence of operations Π^s_D that reverses the corresponding sequence Π^s in the ADS, as in Definition 11. We show that if $\mathcal{X}$ implements this ADS in a history independent way, then it is weakly deletion-compliant.

The simulation strategy presented in the proof of Theorem 8 requires the data collector to be deterministic. This is not so much a restriction since [Har+05] have shown that strongly history independent data structures must be deterministically determined by their content, up to some random initialization (e.g. the choice of a hash function).

Theorem 8. *Let $(\mathcal{X}, \pi, \pi_D)$ be a data-collector such that*

1. *$\mathcal{X}$ is an implementation of an abstract data structure whose states are the possible values for the work tape of $\mathcal{X}$,*
2. *$\mathcal{X}$ is deterministic up to some randomness used for initialization that does change the views of the other machines,*
3. *to each session of protocol π corresponds a sequence of state transitions T for the abstract data structure, and*
4. *for each session of π with state transition T, the corresponding session of π_D is a deletion operation T_D as defined in Definition 11.*

If $\mathcal{X}$ is history-independent, then $\mathcal{X}$ is weakly deletion-compliant.

Proof. During an execution between $\mathcal{X}$, $\mathcal{Y}$ and $\mathcal{Z}$, a list of protocol sessions are initiated with $\mathcal{X}$. Let $\Pi = (\tau_1, \ldots, \tau_N)$ be this list where each τ_i is the session ID of an instance of π or π_D. To each τ_i we associate the sequence of state transitions T_i that represent the change in the state of $\mathcal{X}$ resulting from the execution of τ_i, i.e. such that

$$state^i_{\mathcal{X}} \xrightarrow{T_i} state^{i+1}_{\mathcal{X}} \tag{8}$$

where $state^k_{\mathcal{X}}$ is the state of $\mathcal{X}$ after the kth protocol τ_k. Let $L \subset [N]$ be the indices such that $i \in L$ if and only if τ_i was initiated by $\mathcal{Y}$. Since $\mathcal{Y}$ is the deletion-requester, for every τ_i ($i \in L$) that corresponds to a session of π, there is a $j > i$ such that τ_j is the corresponding session of π_D. Let $R \subset [N] \times [N]$ be the set of pairs (i, j) with $i \in L$ such that τ_j is the session of π_D that corresponds to session τ_i of π as described above. For every $(i, j) \in R$, the state transitions T_i and T_j corresponding to τ_i and τ_j satisfy Def. 11, i.e. T_j undoes T_i in the ADS.

The simulator $\mathcal{S}$ of Definition 3 first initializes $\mathcal{X}$ with a fresh random tape and simulates $\mathcal{X}$ by invoking protocols π and π_D initiated by $\mathcal{Z}$ using the contents of the incoming and

outgoing tapes from the view of $\mathcal{Z}$. Since $\mathcal{X}$ is deterministic after initialization, this simulation will lead to a state for $\mathcal{X}$ that is consistent with the view of $\mathcal{Z}$. Using the notation introduced above, $\mathcal{S}$ simulates the execution of protocol sessions τ_i for $i \in [N] \setminus L$. This sequence contains the same protocols as in the real execution, except for the protocols π and π_D initiated by $\mathcal{Y}$.

We compare the simulation of $\mathcal{S}$ using $view_{\mathcal{Z}}^{\mathcal{X}}$ to the real execution between $\mathcal{X}$, $\mathcal{Y}$ and $\mathcal{Z}$. In the real execution, protocol sessions $\tau_1, \ldots, \tau_N$ are executed sequentially, resulting in state transitions $T_1, \ldots, T_N$. Let X denote the state of the ADS implemented by $\mathcal{X}$ such that $\emptyset \xrightarrow{T_1 \ldots T_N} X$. By the strong history independence of $\mathcal{X}$ (Def. 10) and the definition of deletion for abstract data structures (Def. 11), we have that for every $(i, j) \in R$ the sequence of state transitions $T_1, \ldots, T_{i-1}, T_{i+1}, \ldots, T_{j-1}, T_{j+1}, \ldots, T_N$ also maps the initial state $\emptyset$ to state X. Therefore, if we let $T_{[N]\setminus L}$ denote the state transitions for every $i \notin L$ (i.e. only for the sessions τ_i initiated by $\mathcal{Z}$), we have that $\emptyset \xrightarrow{T_{[N]\setminus L}} X$. By the history independence of $\mathcal{X}$, the internal representation $state_{\mathcal{X}}$ of $\mathcal{X}$ for the ADS state X resulting from $T_1, \ldots, T_N$ in the real execution is identically distributed to the state $\mathcal{S}(view_{\mathcal{Z}})$ in the simulated execution resulting from $T_{[N]\setminus L}$. We have thus shown that for every distinguisher $\mathcal{D}$,

$$\left| \Pr\left[\mathcal{D}(state_{\mathcal{X}}, view_{\mathcal{Z}}) = 1\right] - \Pr\left[\mathcal{D}(\mathcal{S}(view_{\mathcal{Z}}^{\mathcal{X}}), view_{\mathcal{Z}}) = 1\right] \right| = 0 \ .$$

□

Proof of Theorem 4 Using Theorem 8, it is very easy to prove that the data-collector of Section 4 is weakly deletion-compliant. We only need to show that it satisfies all the requirements of Theorem 8. Let $\mathcal{X}$ be the data-collector of Fig. 2,

1. it implements a history-independent list: its state is the state of the list;
2. it is deterministic;
3. each protocol $\pi_{\texttt{post}}$, $\pi_{\texttt{fetch}}$ and π_D corresponds to a (possibly empty) transition in the list abstract data structure; and
4. for some k, each execution of $\pi_D(k)$ triggers the ADS operation `list.remove` of the corresponding ADS operation `list.insert` triggered by $\pi_{\texttt{post}}(k, \cdot)$.

6 Conclusion & Open Questions

We have shown that the concept of compliance to the "right to be forgotten" is compatible with formalisms that do not necessarily provide privacy from third parties. Under our new definition, a data collector is able to prove that it has forgotten the deletion requester's data by showing that its state is consistent with having only interacted with the other users of the system.

An interesting question is to further study the interplay of privacy and deletion-compliance. For example, if a protocol π is private even against a malicious data collector for some appropriate notion of privacy, then does that imply that any data collector is deletion-compliant when deletion requesters interact through π?

Despite our more permissive definition to allow a larger class of data collectors, there are still natural situations that fall outside of that class. For example, a data collector that encrypts the data it collects and merely throws away the key upon deletion should intuitively satisfy the notion of computational deletion-compliance. However, the difference

in size in the state of the data collector in the cases where $\mathcal{Y}$'s data is present or not allows to distinguish both cases. We have proposed a potential solution to this problem – giving the simulator a *masked* view of $\mathcal{Y}$ – that could form the basis of future work.

References

[AC19] Mohammad Al-Rubaie and J. Morris Chang. "Privacy-Preserving Machine Learning: Threats and Solutions". In: *IEEE Security Privacy* 17.2 (Mar. 2019). Conference Name: IEEE Security Privacy, pp. 49–58. ISSN: 1558-4046. DOI: 10.1109/MSEC.2018.2888775.

[AS21] Logan Arkema and Micah Sherr. "Residue-Free Computing". In: *Proceedings on Privacy Enhancing Technologies* 2021.4 (2021), pp. 389–405. DOI: doi:10.2478/popets-2021-0076. URL: https://doi.org/10.2478/popets-2021-0076.

[BI20] Anne Broadbent and Rabib Islam. "Quantum Encryption with Certified Deletion". In: *Theory of Cryptography*. Ed. by Rafael Pass and Krzysztof Pietrzak. Cham: Springer International Publishing, 2020, pp. 92–122. ISBN: 978-3-030-64381-2.

[Car13] Edward L. Carter. "Argentina's Right to be Forgotten". In: *Emory International Law Review* 27.1 (2013).

[CCPA] *California Consumer Privacy Act*. URL: https://oag.ca.gov/privacy/ccpa (visited on 02/11/2021).

[CN20] Aloni Cohen and Kobbi Nissim. "Towards formalizing the GDPR's notion of singling out". In: *Proceedings of the National Academy of Sciences* 117.15 (Apr. 14, 2020). Publisher: National Academy of Sciences Section: Physical Sciences, pp. 8344–8352. ISSN: 0027-8424, 1091-6490. DOI: 10.1073/pnas.1914598117. URL: https://www.pnas.org/content/117/15/8344 (visited on 02/09/2021).

[Der+19] David Derler, Sebastian Ramacher, Daniel Slamanig, and Christoph Striecks. *I Want to Forget: Fine-Grained Encryption with Full Forward Secrecy in the Distributed Setting*. 912. 2019. URL: https://eprint.iacr.org/2019/912 (visited on 02/15/2021).

[Dwo+06] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. "Calibrating Noise to Sensitivity in Private Data Analysis". In: *Theory of Cryptography*. Ed. by Shai Halevi and Tal Rabin. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2006, pp. 265–284. ISBN: 978-3-540-32732-5. DOI: 10.1007/11681878_14.

[GDPR] *Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation)*. URL: https://eur-lex.europa.eu/eli/reg/2016/679/oj (visited on 02/11/2021).

[GGV20] Sanjam Garg, Shafi Goldwasser, and Prashant Nalini Vasudevan. "Formalizing Data Deletion in the Context of the Right to Be Forgotten". In: *Advances in Cryptology EUROCRYPT 2020*. Ed. by Anne Canteaut and Yuval Ishai. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 373–402. ISBN: 978-3-030-45724-2. DOI: 10.1007/978-3-030-45724-2_13.

[GL21] Jonathan Godin and Philippe Lamontagne. "Deletion-Compliance in the Absence of Privacy". In: *2021 18th International Conference on Privacy, Security and Trust (PST)*. 2021, pp. 1–10. DOI: 10.1109/PST52912.2021.9647774.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. "The Knowledge Complexity of Interactive Proof Systems". In: *SIAM Journal on Computing* 18.1 (Feb. 1, 1989). Publisher: Society for Industrial and Applied Mathematics, pp. 186–208. ISSN: 0097-5397. DOI: 10.1137/0218012. URL: https://epubs.siam.org/doi/10.1137/0218012 (visited on 02/09/2021).

[Har+05] Jason D. Hartline, Edwin S. Hong, Alexander E. Mohr, William R. Pentney, and Emily C. Rocke. "Characterizing History Independent Data Structures". In: *Algorithmica* 42.1 (May 1, 2005), pp. 57–74. ISSN: 1432-0541. DOI: 10.1007/s00453-004-1140-z. URL: https://doi.org/10.1007/s00453-004-1140-z (visited on 09/24/2020).

[Kan+11] Jayanthkumar Kannan, Gautam Altekar, Petros Maniatis, and Byung-Gon Chun. "Making programs forget: enforcing lifetime for sensitive data". In: *Proceedings of the 13th USENIX conference on Hot topics in operating systems*. HotOS'13. USA: USENIX Association, May 9, 2011, p. 23. (Visited on 05/12/2021).

[Nis+17] Kobbi Nissim, Aaron Bembenek, Alexandra Wood, Mark Bun, Marco Gaboardi, Urs Gasser, David R. O'Brien, Thomas Steinke, and Salil Vadhan. "Bridging the Gap between Computer Science and Legal Approaches to Privacy". In: *Harvard Journal of Law & Technology (Harvard JOLT)* 31 (2017), p. 687. URL: https://heinonline.org/HOL/Page?handle=hein.journals/hjlt31&id=705&div=&collection=.

[NT01] Moni Naor and Vanessa Teague. "Anti-Persistence: History Independent Data Structures". In: *Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing*. STOC '01. Hersonissos, Greece: Association for Computing Machinery, 2001, pp. 492–501. ISBN: 1581133499. DOI: 10.1145/380752.380844. URL: https://doi.org/10.1145/380752.380844.

[RBC13] Joel Reardon, David Basin, and Srdjan Capkun. "SoK: Secure Data Deletion". In: *2013 IEEE Symposium on Security and Privacy*. 2013 IEEE Symposium on Security and Privacy. ISSN: 1081-6011. May 2013, pp. 301–315. DOI: 10.1109/SP.2013.28.

[Sho+17] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. "Membership inference attacks against machine learning models". In: *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 3–18. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7958568.

[Sim+15] Milivoj Simeonovski, Fabian Bendun, Muhammad Rizwan Asghar, Michael Backes, Ninja Marnau, and Peter Druschel. "Oblivion: Mitigating Privacy Leaks by Controlling the Discoverability of Online Information". In: *Applied Cryptography and Network Security*. Ed. by Tal Malkin, Vladimir Kolesnikov, Allison Bishop Lewko, and Michalis Polychronakis. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2015, pp. 431–453. ISBN: 978-3-319-28166-7. DOI: 10.1007/978-3-319-28166-7_21.

[SML07] Patrick Stahlberg, Gerome Miklau, and Brian Neil Levine. "Threats to privacy in the forensic analysis of database systems". In: *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*. SIGMOD '07. New York, NY, USA: Association for Computing Machinery, June 11, 2007, pp. 91–

102. ISBN: 978-1-59593-686-8. DOI: 10.1145/1247480.1247492. URL: https://doi.org/10.1145/1247480.1247492 (visited on 05/12/2021).

[SS98] Pierangela Samarati and Latanya Sweeney. *Protecting Privacy when Disclosing Information: k-Anonymity and Its Enforcement through Generalization and Suppression*. Tech. rep. 1998.

[Wan+19] Zhibo Wang, Mengkai Song, Zhifei Zhang, Yang Song, Qian Wang, and Hairong Qi. "Beyond Inferring Class Representatives: User-Level Privacy Leakage From Federated Learning". In: *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*. IEEE INFOCOM 2019 - IEEE Conference on Computer Communications. ISSN: 2641-9874. Apr. 2019, pp. 2512–2520. DOI: 10.1109/INFOCOM.2019.8737416.

[Wei+11] Michael Wei, Laura M. Grupp, Frederick E. Spada, and Steven Swanson. "Reliably erasing data from flash-based solid state drives". In: *Proceedings of the 9th USENIX conference on File and stroage technologies*. FAST'11. USA: USENIX Association, Feb. 15, 2011, p. 8. ISBN: 978-1-931971-82-9. (Visited on 02/08/2021).