Control, Confidentiality, and the Right to be Forgotten

Aloni Cohen[*]   Adam Smith[†]   Marika Swanberg[‡]   Prashant Nalini Vasudevan[§]

December 5, 2023

Abstract

Recent digital rights frameworks give users the right to *delete* their data from systems that store and process their personal information (e.g., the "right to be forgotten" in the GDPR).

How should deletion be formalized in complex systems that interact with many users and store derivative information? We argue that prior approaches fall short. Definitions of *machine unlearning* Cao and Yang [2015] are too narrowly scoped and do not apply to general interactive settings. The natural approach of *deletion-as-confidentiality* Garg et al. [2020] is too restrictive: by requiring secrecy of deleted data, it rules out social functionalities.

We propose a new formalism: *deletion-as-control*. It allows users' data to be freely used before deletion, while also imposing a meaningful requirement after deletion—thereby giving users more control.

Deletion-as-control provides new ways of achieving deletion in diverse settings. We apply it to social functionalities, and give a new unified view of various machine unlearning definitions from the literature. This is done by way of a new adaptive generalization of history independence.

Deletion-as-control also provides a new approach to the goal of *machine unlearning*, that is, to maintaining a model while honoring users' deletion requests. We show that publishing a sequence of updated models that are differentially private under continual release satisfies deletion-as-control. The accuracy of such an algorithm does not depend on the number of deleted points, in contrast to the machine unlearning literature.

---

[*]Department of Computer and Data Science, University of Chicago. `aloni@uchicago.edu`

[†]Department of Computer Science, Boston University. `ads22@bu.edu`

[‡]Department of Computer Science, Boston University. `marikas@bu.edu`

[§]Department of Computer Science, National University of Singapore. `prashant@comp.nus.edu.sg`

Contents

> Someday, this baby and other babies from her cohort will be 30 and there will be an absolutely bananas cache of data about what form and hue their poops took. Maybe someone will hack it and it will derail a presidential campaign news cycle.

<div align="right">*Alexandra Petri, from "What I've been up to the last four months"*</div>

1   Introduction

The long-term storage of modern data collection carries serious risks, including often-surprising disclosures [Fowler, 2022, Carlini et al., 2021, JASON, 2022, Cohen and Nissim, 2020, Ng, 2022], manipulation, and epistemic bubbles. The permanence of our digital footprints can also chill expression, with every word weighed against the risk of out-of-context blowback in the future.

Data protection laws around the world have begun to challenge this permanence. The EU's General Data Protection Regulation provides an individual *data subject* the right to request "the erasure of personal data concerning him or her" and delineates when a *data controller* must oblige. California followed suit in 2020, and similar rights take effect in Virginia, Colorado, Connecticut, and Utah in 2023 National Conference of State Legislatures [2022].

In the modern data ecosystem, however, it is not easy to articulate what constitutes the "erasure" of personal data. Data is not merely stored in databases—it is used to train machine learning models, compute and publish statistics, and drive decisions. Such complexity and nuance challenges simplistic thinking about erasure, and the sheer number of ways data are used precludes case-by-case reasoning about erasure compliance.

Giving users more control over data is today a central policy goal. Decades of cryptography has given us good definitional tools for reasoning about non-disclosure of data—enabling the development of technical solutions, informing policy decisions, and influencing practice. But we lack similar tools for reasoning about control over data and, in particular, deletion. While there has been a flurry of recent work on so-called *machine unlearning* [Cao and Yang, 2015, Ginart et al., 2019, Ullah et al., 2021, Bourtoule et al., 2021, Sekhari et al., 2021, Gupta et al., 2021, ...], often directly motivated by legal compliance, there remain basic gaps in our understanding.

This paper sheds light on what data deletion means in complex data processing scenarios, and how to achieve it. We provide a formulation that unifies and generalizes previous technical approaches that only partially answer this question. Though we make no attempt to strictly adhere to any specific legal right to erasure, we aim to incorporate more of its contours than prior technical work on erasure.

Our new formulation, called *deletion-as-control*, requires that after an individual Alice requests erasure, the data controller's future behavior and internal state should not depend on Alice's data *except insofar as that data has already affected other parties in the world*. In this way, Alice's autonomy need not require secrecy; she has a say about how her data is used regardless of past or future disclosure.

Our definition meaningfully captures a variety of data controllers, including ones facilitating social interactions and maintaining accurate predictive models. In contrast, prior approaches yield a patchwork of, at times, contradictory and counterintuitive interpretations of erasure.

1.1 Touchstone Examples

In order to understand our and prior approaches to data deletion, it is helpful to have in mind a few concrete examples of functionalities that separate our approaches from prior ones. We describe four touchstone functionalities, then briefly discuss how they relate to prior approaches and our new notion.

Private Cloud Storage. Users can upload files for cloud storage and future download. Only the originating user may download a file and the files are never used in any other way. The existence of files is only ever made known to the originating user, and the controller publishes no other information.

Public Bulletin Board. Users can submit posts to the public bulletin board. The bulletin board simply displays all user posts currently in the controller's internal storage, with no other functionalities (e.g. responding, messaging, etc.).

Batch Machine Learning. Users contribute data during some collection period. At the end of the period, the data controller trains a predictive model on the resulting dataset. The data controller then publishes the model.

Public Directory + Usage Statistics. Users upload their name and phone number to be listed in a directory. The data controller allows anybody to search for a listing in the directory, and each week reports a count of the number of distinct users that have looked up a phone number so far. (Other statistics are possible too—the weekly count of new users, say.)

The touchstone examples and prior approaches Figure 1 summarizes how the touchstone functionalities fare under deletion-as-control and under three prior approaches to defining deletion: *deletion-as-confidentiality*, *machine unlearning*, and *simulatable deletion* (Section 1.4).[1] Together, the touchstone functionalities illustrate that prior approaches constitute an inconsistent patchwork, each falling short on at least two of the examples.

Deletion-as-confidentiality [Garg et al., 2020] is over-restrictive. Briefly, it requires that third parties cannot distinguish whether a data subject Alice requested erasure from the controller or simply never interacted with the controller in the first place. This implies, among other things, that Alice's data is kept confidential from all other parties even if Alice never requests its erasure. This confidentiality-style approach is well-suited for Private Cloud Storage, but deletion-as-confidentiality precludes inherently social functionalities, like the Bulletin Board and Directory. No controller implementing these functionalities could ever satisfy deletion-as-confidentiality. Why would Alice post messages if they could never be made public?

On the other hand, machine unlearning—even in its strongest incarnation, due to Gupta et al. [2021]—is too narrowly scoped. It is specialized to the setting of machine learning and does not consider general interactive functionalities. The definition is not applicable to the Cloud Storage, Bulletin Board, and Directory functionalities. Definitions from the machine unlearning literature

---

[1]We introduce the terms *deletion-as-confidentiality* and *simulatable deletion* for the definitions of Garg et al. [2020] and Godin and Lamontagne [2021], respectively, to more clearly distinguish them from each other and from deletion-as-control. Garg et al. [2020] use the term *deletion-compliance*; Godin and Lamontagne [2021] use *strong deletion-compliance* and *weak deletion-compliance*, respectively.

are meaningful for the Batch Machine Learning functionality, where they correspond to versions of *history independence.*

Even where multiple definitions are meaningful, they may impose different requirements. For example, both deletion-as-confidentiality and deletion-as-control admit implementations of Batch Machine Learning that are *persistent* in that the published models never need to be updated. On the other hand, history independence requires that any useful model be updated after enough deletion requests.

The touchstone examples and deletion-as-control   We show that each of the touchstones can be implemented in a manner that satisfies our new notion, deletion-as-control.

Private Cloud Storage. To remove a user, the controller deletes all the user's files from its internal storage. Such a controller satisfies deletion-as-control if its data structures are *history independent* (Corollary 3.10).

Public Bulletin Board. To remove a user, the controller deletes all of the user's posts from its internal storage and, as a result, from the public-facing bulletin board. As with cloud storage, such a controller satisfies deletion-as-control if its data structures are history independent (Corollary 3.10).

Batch Machine Learning. Deletion-as-control is achieved if the dataset is deleted after training and training is done with differential privacy, e.g., using DP-SGD Bassily et al. [2014] (Corollary 5.2). To remove a user, the controller does nothing—it simply ignores deletion requests. The resulting deletion guarantee is parameterized by the privacy parameters ϵ and δ.

Public Directory + Usage Statistics. Deletion-as-control can be achieved by combining differential privacy and history independence (Corollary 6.5). The statistics are computed using a mechanism that satisfies a stringent form of DP—pan-privacy under continual release—while the public directly is implemented using a history independent data structure. To remove a user, the controller deletes their listing from the public directory and its associated data structures, but leaves the data structures for the DP statistics unaltered.

1.2   Contributions

Defining deletion-as-control   Our primary contribution is a formalization of *deletion-as-control*, an important step towards providing individuals greater control over the use of personal data. The new notion applies to general data controllers and interaction patterns among parties, building on the modeling of [Garg et al., 2020]. As described below, it unifies existing approaches within a coherent framework and captures all the touchstone examples of Section 1.1.

The goal embodied by deletion-as-control is not so much to hide data from others as to exercise control over how the data is used. Until Alice requests erasure, deletion-as-control should not limit the controller's usage of her data. But after erasure, the data controller's future behavior and internal state should not depend on Alice's data *except insofar as that data has already affected other parties in the world.* In this way, Alice's autonomy need not imply secrecy; she has a say about how her data is used regardless of past or future disclosure.

Our approach provides new ways of achieving meaningful deletion in diverse settings.

5

	Private Cloud Storage	Public Bulletin Board	Batch Machine Learning	Public Directory + Usage Stats
Deletion-as-Confidentiality [Garg et al., 2020]	✓	⊤	✓	⊤
Machine unlearning [Gupta et al., 2021, …]	✗	✗	✓	✗
Simulatable deletion [Godin and Lamontagne, 2021]	✓	⊥	⊥	⊥
Deletion-as-control (this work)	✓ (Corollary 3.10)	✓ (Corollary 3.10)	✓ (Corollary 5.2)	✓ (Corollary 6.5)

✓: Definition is satisfied by implementations with meaningful deletion guarantees.
⊥: Definition is under-restrictive: allows *vacuous* implementations with no meaningful deletion of any kind.
⊤: Definition is over-restrictive: no implementation of the functionality satisfies the definition.
✗: Definition does not apply to the functionality.

Figure 1: Application of deletion definitions to touchstone functionalities.

Capturing social functionalities via history independence Deletion-as-control applies to a wide range of controllers that provide "social" functionalities where prior approaches fall flat (e.g., the Public Bulletin Board touchstone). Along the way, we give a new unified view of the various machine unlearning definitions from the literature.

Both flow from a theorem roughly stating that deletion-as-control is implied by *adaptive history independence*, a generalization of the cryptographic notion of *history independence* [Micciancio, 1997, Naor and Teague, 2001] that we introduce. An implementation of a data structure is history independent if its memory representation reveals nothing more than the logical state of the data structure. That history independence is related to deletion is intuitive, and appears in [Garg et al., 2020, Godin and Lamontagne, 2021]. Machine unlearning imposes a similar requirement in the specific context of machine learning. Oversimplifying, a learned model (akin to the memory representation) must reveal nothing more than a model retrained from scratch (akin to the logical state). We make these connections precise.

New algorithms for machine learning via differential privacy Deletion-as-control provides a new approach for machine learning in the face of modern data rights. Very roughly, differential privacy (DP) provides deletion-as-control for free. Intuitively, if a person has (approximately) no impact on a trained model, mitigating that impact is trivial. In particular, if using an adaptive pan-private algorithm to maintain the model, it does not need to be updated in response to deletion requests, unlike machine unlearning algorithms. For the first time, this approach enables a meaningful deletion guarantee while bounding the worst-case loss compared to deletion-free learning.

Specifically, we describe two ways of compiling DP mechanisms into controllers satisfying deletion-as-control. The first applies to DP mechanisms that are run in a batch setting on a single, centralized dataset (e.g., the Batch Machine Learning touchstone). The second applies to mechanisms satisfying an adaptive variant of *pan-privacy under continual release* [Chan et al., 2011, Dwork et al., 2010b, Jain et al., 2021], including controllers that periodically update a model on an ongoing basis. The compilation from differential privacy is by way of deletion-as-confidentiality Garg et al. [2020], which we prove implies deletion-as-control.

Figure 2: Overview of the relationships between definitions and the roles of specific example controllers. Each oval represents a definition, and each icon represents an implementation of a touchstone functionality satisfying the corresponding definition. Oval containment corresponds to an implication between the definitions, though only roughly and subject to technicalities that we do not attempt to capture in this figure.

We combine this result with existing algorithms for private learning under continual release [Kairouz et al., 2021] to obtain new controllers that maintain a model with accuracy essentially identical to that of a model trained on the entire set of added records.

Capturing complex mechanisms via composition We show that deletion-as-control captures more functionalities than those collectively captured by history independence, differential privacy, and deletion-as-confidentiality. Specifically, we show how to implement the Public Directory + Usage Statistics touchstone, a functionality that cannot satisfy any of the above three properties. To do so, we prove that deletion-as-control enjoys a limited form of *parallel composition*.

1.3 Defining deletion-as-control

We define deletion-as-control in a way that allows arbitrary use of a person's data before deletion, but not after. The challenge is to provide meaningful privacy guarantees despite this feature, even in a general setting with an adaptive and randomized data controller, data subject (Alice), and an environment (representing all parties other than the data controller and Alice).

Our proposal is that after deletion, the data controller should be able to produce a plausible alternate explanation for its current state without appealing to Alice's participation. More specifically, *the state of the controller after deletion can be plausibly attributed to the interaction between the controller and the environment alone.* By this we mean that the state is about as likely—with probability taken over the controller's random coins—in the real world as in the hypothetical 'ideal' world where the environment's messages to the controller are unchanged but where Alice does not interact with the controller whatsoever. The result is that, after Alice's deletion, the controller's subsequent states depend on the interaction with Alice only insofar as that interaction affected other parties' interactions with the controller.

Importantly, we only require that the controller's state is plausible in the ideal world *given* the messages sent by the environment. *We do not require the environment's messages themselves to be plausible in the ideal world.* For example, suppose that on a public bulletin board, Bob simply copies and reposts Alice's posts. Bob's messages in the ideal world would still contain the content of Alice's posts, even though Alice is completely absent in the hypothetical ideal world. This is unavoidable—we want to allow the controller and the environment to use the subject's data arbitrarily before deletion. So the environment's queries may depend on the data subject's inputs, directly or indirectly.

In a bit more detail, our definition compares real and ideal worlds defined in a non-standard way.[2] The real world execution, denoted $\langle \mathsf{C}, \mathsf{E}, \mathsf{Y} \rangle$, involves three parties: a data controller C, a special data subject Y, and an environment E representing all other parties. (The notation $\langle \cdots \rangle$ denotes the transcript of an execution between the parties listed.) The execution ends after Y requests deletion and C processes the request. The real execution specifies (i) C's state $\mathsf{state}_\mathsf{C}$, (ii) the *queries* $\vec{\mathsf{q}}_\mathsf{E}$ sent by E to C, and (iii) the randomness R_C used by C. The ideal world execution, denoted $\langle \mathsf{C}(R'_\mathsf{C}), \mathsf{D}(\vec{\mathsf{q}}_\mathsf{E}) \rangle$, involves the same controller C and a dummy environment D that simply replays the queries $\vec{\mathsf{q}}_\mathsf{E}$. The controller's ideal world randomness R'_C is sampled by a simulator $\mathsf{Sim}(\vec{\mathsf{q}}_\mathsf{E}, R_\mathsf{C}, \mathsf{state}_\mathsf{C})$. The ideal execution specifies (i) C's ideal state $\mathsf{state}'_\mathsf{C}$, and (ii) the simulated randomness R'_C used by C.

Definition 1.1 $((\epsilon, \delta)$-deletion-as-Control (simplified))**.** *Given* C, E, Y, *and* Sim, *consider the following experiment. Sample* $R_\mathsf{C} \leftarrow \{0,1\}^{\mathbb{N}}$; *run the real execution* $(\vec{\mathsf{q}}_\mathsf{E}, \mathsf{state}_\mathsf{C}) \leftarrow \langle \mathsf{C}(R_\mathsf{C}), E, \mathsf{Y} \rangle$; *sample* $R'_\mathsf{C} \leftarrow \mathsf{Sim}(\vec{\mathsf{q}}_\mathsf{E}, R_\mathsf{C}, \mathsf{state}_\mathsf{C})$; *and run the ideal execution* $(\vec{\mathsf{q}}'_\mathsf{E}, \mathsf{state}'_\mathsf{C}) \leftarrow \langle \mathsf{C}(R'_\mathsf{C}), \mathsf{D}(\vec{\mathsf{q}}_\mathsf{E}) \rangle$.

We say a controller C *is* (ϵ, δ)*-deletion-as-control compliant if there exists* Sim *such that for all* E *and* Y: *(i)* $R'_\mathsf{C} \overset{\epsilon,\delta}{\approx} R_\mathsf{C}$; *(ii)* $\mathsf{state}'_\mathsf{C} = \mathsf{state}_\mathsf{C}$ *with probability at least* $1 - \delta$.

The notation $\overset{\epsilon,\delta}{\approx}$ denotes approximate indistinguishability parameterized by ϵ and δ (as in the definition of differential privacy). We give a complete version of Definition 1.1 in Section 2.

1.4 Prior Work

We give a brief discussion of prior definitions of deletion. Machine unlearning and deletion-as-confidentiality are discussed in detail in Sections 3.4 and 4, respectively.

Deletion-as-confidentiality Garg et al. [2020] define *deletion-as-confidentiality*.[3] It requires that the deleted data subject Alice leaves (approximately) no trace: the whole view of the environment along with the state of the controller after deletion should be as if Alice never existed. As a result, no third party may ever learn of Alice's presence — even if she never requests deletion. The strength of this definition is its strong, intuitive, interpretable guarantee.

But deletion-as-confidentiality is too restrictive (Figure 1). The stringent indistinguishability requirement precludes any functionality where users learn about each other. Implementations of the Private Cloud Storage and Batch Machine Learning functionalities can satisfy deletion-as-confidentiality, using history independence (cf. Section 3). But the Bulletin Board and Directory are ruled out. If Bob ever looks up Alice's messages, the confidentiality required by the definition is impossible.

Simulatable deletion Godin and Lamontagne [2021] introduce *simulatable deletion* as a relaxation of deletion-as-confidentiality, motivated by the observation that deletion-as-confidentiality rules out social functionalities.[4] Roughly, simulatable deletion requires that after a data subject Alice is deleted, the resulting state of the controller is simulatable given the environment's view.

---

[2]The non-real world is more 'hypothetical' or 'counter-factual' than 'ideal.' Regardless, we use the term 'ideal world' for continuity with [Garg et al., 2020] and decades of cryptography.

[3]Deletion-as-confidentiality is called *deletion-compliance* in Garg et al. [2020] and *strong deletion-compliance* in Godin and Lamontagne [2021].

[4]Simulatable deletion is called *weak deletion-compliance* in Godin and Lamontagne [2021].

This means that any information about Alice that is present in the controller's state is already present in the view of other parties.

Simulatable deletion is too permissive. A controller may indefinitely retain any information that has ever been shared with any third party. For example, the Public Bulletin Board need not delete Alice's posts if they have ever been read![5] As a result, simulatable deletion is essentially vacuous for functionalities where the controller's state need not be kept secret. The controller can simply publish its state, making simulation trivial. Turning to our touchstone examples (Section 1.1), while simulatable deletion imposes a meaningful requirement for the Private Cloud Storage functionality, it allows implementations with no meaningful deletion of any kind for the other three functionalities.

Machine unlearning This recent line of work specializes the question of deletion to the setting of machine learning [Cao and Yang, 2015, Ginart et al., 2019, Ullah et al., 2021, Bourtoule et al., 2021, Sekhari et al., 2021, Gupta et al., 2021, . . .]. Given a model $\vec{\theta} \leftarrow \mathsf{Learn}(\vec{x})$ and a data point $x^* \in \vec{x}$, that literature requires sampling a new model $\theta' \leftarrow \mathsf{Unlearn}(\vec{\theta}, x^*, \vec{x})$ approximately from the distribution $\mathsf{Learn}(\vec{x} \setminus \{x^*\})$ (i.e., approximating retraining from scratch). As we explain in Section 3.4, these definitions correspond to versions of *history independence*.

A drawback of machine unlearning is that it specialized to the setting of machine learning. It does not apply to general data controllers and interaction patterns among parties, including the Cloud Storage, Bulletin Board, and Directory functionalities (Figure 1).

History independence-style definitions of machine unlearning do impose a meaningful requirement for the Batch Machine Learning functionality. In fact history independence is a conceptually stricter requirement than deletion-as-control, setting aside many technicalities (Section 3.4). Differentially private (DP) learning illustrates the difference. Roughly, DP learning provides deletion-as-control for free; the resulting model can be published once and never updated. In contrast, history independence *require* updating the model when there are many deletions. To see why, consider $\vec{x}$ of size n and suppose all n people request deletion. These definitions would require the final model θ^* be essentially trivial—it should perform about as well as a the model θ_0 trained on an empty dataset. With the DP learner described above, θ^* performs just as well as the initial model θ.

1.5   Paper Structure

In Section 2 we define deletion-as-control. In Section 3 we define Adaptive History Independence and prove that it implies deletion-as-control (Theorem 3.8). We also show that the machine unlearning definition of Gupta et al. [2021] is a special case of Adaptive History Independence (Proposition 3.12). In Section 4 we show that deletion-as-confidentiality is a strengthening of our definition (Theorem 4.3). In Section 5, we relate differential privacy to our definition (Theorem 5.9) and in the process we define adaptive pan-private in the continual release model. Lastly, in Section 6 we prove a narrow composition result for our definition.

---

[5]Godin and Lamontagne [2021] actually consider a version of the Bulletin Board functionality for which simulatable deletion is meaningful. Crucially, their functionality does not track whether a post has been read. Hence their controller must actually delete Alice's posts. But if the bulletin board keeps read receipts or actively pushes new messages out to users, say, it would not have to delete these posts.

2 Deletion-as-control

We define deletion-as-control in a way that allows arbitrary use of a person's data before deletion, but not after. Under such a definition, an adversary might completely learn the data before it is deleted, and even make it available after it is deleted! The challenge is to provide a meaningful guarantee despite this limitation, even in a general setting with adaptive and randomized data controllers, data subjects, and environments (representing all parties other than the data controller and distinguished data subject).

2.1 (ϵ, δ)-indistinguishability

We consider a notion of similarity of distributions closely related to differential privacy (Appendix B).

   We present some further technical tools in the appendices: a novel coupling lemma for this notion of indistinguishability (Appendix A) as well as background on differential privacy (Appendix B).

Definition 2.1. *Given parameter $\epsilon \geq 0$ and $\delta \in [0, 1)$, we say two probability distributions P and Q on the same set $\mathcal{X}$ (with the same σ-algebra of events $\Sigma_{\mathcal{X}}$) are (ϵ, δ)-indistinguishable and write $P \overset{\epsilon,\delta}{\approx} Q$ if, for every event $E \in \Sigma_{\mathcal{X}}$,*

$$P(E) \leq e^{\epsilon} Q(E) + \delta \ and \ Q(E) \leq e^{\epsilon} P(E) + \delta \, .$$

Slightly overloading this notation, we say two random variables X and Y taking values in the same measurable space $(\mathcal{X}, \Sigma_{\mathcal{X}})$ are (ϵ, δ)-indistinguishable, denoted $X \overset{\epsilon,\delta}{\approx} Y$ if, for every event $E \in \Sigma_{\mathcal{X}}$,

$$\Pr[X \in E] \leq e^{\epsilon} \Pr[Y \in E] + \delta \ and,$$
$$\Pr[Y \in E] \leq e^{\epsilon} \Pr[X \in E] + \delta \, .$$

   Because algorithms in our model run in unbounded time, their (countably infinite) random tapes belong to an uncountably infinite set. This means that not all sets of random tapes have well-defined probability. The σ-algebra $\Sigma_{\mathcal{X}}$ captures the set of events E for which $P(E)$ and $Q(E)$ are defined. This issue does not affect most proofs and definitions; we only make the σ-algebra of events explicit when necessary.

   In our case, the set $\mathcal{X}$ of random tapes for a single machine is $\{0, 1\}^{\mathbb{N}}$. Any execution that terminates reads only a finite prefix of the tape, and so the natural σ-algebra $\Sigma_{\mathcal{X}}$ is the smallest one containing the sets $E_w = \{w \| x : x \in \{0, 1\}^{\mathbb{N}}\}$ for all $w \in \{0, 1\}^*$, where $\|$ denotes string concatenation (that is, E_w is the set of infinite tapes with a particular finite prefix w). $\Sigma_{\mathcal{X}}$ contains every event E that depends on only finitely many bits of the tape. This is the standard σ-algebra for an infinite set of fair coin flips (see, e.g., Polyanskiy [2018]).

2.2 Parties and simplified execution model

Our definition uses the real/ideal cryptography, though not with a typical indistinguishability criterion. See Section 2.5 for complete details.

   The real world execution, denoted $\langle \mathsf{C}, \mathsf{E}, \mathsf{Y} \rangle$, involves three (possibly randomized) parties: a data controller C, an environment E, and a special data subject Y. The real interaction is arbitrary, limited only by the execution model described below. Parties have authenticated channels over

which they may interact freely. While Y has only a single channel to C, the environment has an unbounded number of channels (representing unbounded additional parties). While these channels are authenticated, the controller cannot distinguish the single channel to Y from those to E. The interaction continues until the data subject Y requests deletion, ending after the data controller C processes the request. We denote by state_C the final state of the controller.

The ideal world execution, denoted $\langle C, D \rangle$, involves the interaction of the same controller C as well as a dummy environment D. D takes as input the transcript from the real execution, denoted τ, and simply replays only E's queries, denoted $\vec{q}_E$. If $\vec{q}_E$ is empty, then D terminates without sending messages. Observe that C's responses and state in the ideal world are not fixed. They depend on C's ideal-world randomness, denoted R'_C. Moreover, the ideal interaction is defined relative to a particular instantiation of the real world interaction. In particular, the queries $\vec{q}_E$ may depend on C's real-world randomness, denoted R_C.

The controller's real-world randomness R_C consists of infinitely-many random bits sampled uniformly at random from $\{0,1\}^{\mathbb{N}}$. We denote this distribution $\mathcal{U}$. In the ideal world, a simulator Sim takes as input $(\vec{q}_E, R_C, \mathsf{state}_C)$ and generates C's ideal-world randomness R'_C. When we wish to emphasize the controller's randomness in the execution, we write $\langle C(R_C), E, Y \rangle$ and $\langle C(R'_C), D \rangle$.

An execution involves parties sending messages to each other until some termination condition is reached. Starting with E (real) or D (ideal), parties get activated when they receive a message, and deactivated when they send a message. Only a single party is active at a time. Parties communicate over authenticated channels. Because E represents all users besides the distinguished data subject Y, E has many distinct channels to C. Importantly, authentication allows parties to know on which channel a message was received, but not which party (i.e., E or Y) is on the other end of that channel. Each party is initialized with a uniform random tape which may only be read *once* over the course of the whole execution. If a party wishes to re-use bits from its randomness tape, it must store them in its internal state.

The real execution $\langle C, E, Y \rangle$ ends when Y requests deletion from C. The data subject's delete message activates the controller, who can then remove Y's data. The ideal execution $\langle C(R'_C), D(\vec{q}_E) \rangle$ ends after D sends its last query from $\vec{q}_E$ to C. In both cases, the execution ends after the final activation of C. We consider the controller's state at the end of the execution: $\mathsf{state}_C = C(\vec{q}_E; R_C)$ in the real world, and $\mathsf{state}'_C = C(\vec{q}_E; R'_C)$ in the ideal world. If an execution never ends, the state is defined to be $\bot$ and the transcript τ and its subset $\vec{q}_E$ are defined to be empty. For example, the real execution ends if and only if Y requests deletion.

Remark 2.2 (Keeping time). *In Section 5, we need a global clock. Balancing modelling simplicity with generality, we allow the environment to control time. Specifically, we introduce a special query* tick *that E can send to C thereby incrementing the clock. We do not allow Y to query* tick.

2.3 Defining Deletion-as-Control

We require that the internal state of the controller is about as likely in the real world and the ideal world, where probability is taken over C's random coins. Let state_C and state'_C be the internal states in the real and ideal executions. Consider a random variable R'_C which is sampled uniformly conditioned on $\mathsf{state}'_C = \mathsf{state}_C$ in the ideal execution where C uses randomness R'_C. Informally, our definition requires that the distributions of R'_C and R_C are close: $R'_C \overset{\epsilon,\delta}{\approx} R_C$. We do not require that the real and ideal executions are themselves (ϵ, δ)-indistinguishable. Instead, we require that the

Figure 3: The real (left) and ideal (right) executions. The communications with the controller C are meant to be "public API" calls (e.g. accessing a public website, possibly using login credentials). Notice that the dummy environment in the ideal world sends message $\vec{\mathsf{q}}_{\mathsf{E},i}$ regardless of what the previous responses were.

"explanations" in the real and ideal executions are (ϵ, δ)-indistinguishable—viewing the controller's randomness as the explanation of its state (relative to the environment's queries $\vec{\mathsf{q}}_{\mathsf{E}}$).

We extend this idea by considering ways of sampling R'_{C} other than the conditional distribution described above (which may not always be defined). In general, we allow a simulator Sim to sample R'_{C} as a function of the queries $\vec{\mathsf{q}}_{\mathsf{E}}$ from E to C, the real-world randomness R_{C}, and the real-world state $\mathsf{state}_{\mathsf{C}}$. (Although we view the simulator's output as an infinite-length bit sequence, it actually only needs to output a finite prefix.) We require that $R'_{\mathsf{C}} \stackrel{\epsilon,\delta}{\approx} R_{\mathsf{C}}$ and that $\mathsf{state}'_{\mathsf{C}} = \mathsf{state}_{\mathsf{C}}$ (or $\mathsf{state}_{\mathsf{C}} = \bot$) except with probability δ. Sampling R'_{C} conditioned on $\mathsf{state}'_{\mathsf{C}} = \mathsf{state}_{\mathsf{C}}$ is a useful default simulation strategy that we use throughout the paper, but there are sometimes much simpler ways to sample R'_{C}.

Definition 2.3 (Deletion as control). *Given a controller C, an environment E, a data subject Y and a simulator Sim, we consider the following experiment:*

- *$R_{\mathsf{C}} \leftarrow \mathcal{U}$ (R_{C} is a uniform random tape)*
- *$(\tau, \mathsf{state}_{\mathsf{C}}) \leftarrow \langle \mathsf{C}(R_{\mathsf{C}}), E, \mathsf{Y} \rangle$, where $\vec{\mathsf{q}}_{\mathsf{E}} \subseteq \tau$ are the messages from E to C.*
- *$R'_{\mathsf{C}} \leftarrow \mathsf{Sim}(\vec{\mathsf{q}}_{\mathsf{E}}, R_{\mathsf{C}}, \mathsf{state}_{\mathsf{C}})$*
- *$(\tau', \mathsf{state}'_{\mathsf{C}}) \leftarrow \langle \mathsf{C}(R'_{\mathsf{C}}), \mathsf{D}(\vec{\mathsf{q}}_{\mathsf{E}}) \rangle$.*

We say a controller C is (ϵ, δ)-deletion-as-control compliant if there exists Sim such that for all E and Y and for R'_{C}, R_{C} sampled as above:

1. *$R'_{\mathsf{C}} \stackrel{\epsilon,\delta}{\approx} R_{\mathsf{C}}$ (i.e., R_{C} and R'_{C} are similarly distributed), and*

2. *With probability at least $1 - \delta$, either $\mathsf{state}'_{\mathsf{C}} = \mathsf{state}_{\mathsf{C}}$ or $\mathsf{state}_{\mathsf{C}} = \bot$.*

For a particular class of data subjects $\mathcal{Y}$, we say a controller C is (ϵ, δ)-deletion-as-control compliant for $\mathcal{Y}$ if there exists Sim such that the above holds for all E and for all $\mathsf{Y} \in \mathcal{Y}$.

Example 2.4 (XOR Controller). *Consider a controller $\mathsf{C}_{\oplus}$ which maintains a k-bit state $\mathsf{state} \in \{0,1\}^k$ which is initialized uniformly at random. Upon receiving a message $x \in \{0,1\}^k$, $\mathsf{C}_{\oplus}$ updates*

its state state $\leftarrow$ state $\oplus x$, *sending nothing in return. If it receives any other message, including* delete, *it does nothing.*

$C_\oplus$ *satisfies* $(0, 0)$*-deletion-as-control. To see why, consider an execution of the real world, which ends when the data subject sends* delete. *At this point,* state $= R \oplus x_Y \oplus x_E$, *where* $R \in \{0, 1\}^k$ *is the random initialization,* x_Y *is the XOR of all messages* x *sent by* Y, *and* x_E *is the XOR of all messages* x *sent by* E. *Let* Sim *compute* x_E *from the queries* $\vec{q}_E$, *and output* $R' = $ state $\oplus x_E = R \oplus x_Y$. *This satisfies the definition: (1)* R' *is uniformly distributed because* x_Y *is independent of* R; *(2)* state$' = R' \oplus x_E = $ state.

Definition 2.5 (Default simulator). *The default simulator* Sim* *samples* R'_C *as follows:*

$$\text{Sim}(\vec{q}_E, R_C, \text{state}_C):$$
$$\text{Return } R'_C \sim \mathcal{U}|_{C(\vec{q}_E; R') = \text{state}_C} \text{ if such an } R' \text{ exists;}$$
$$\text{Otherwise, return } R'_C = R_C.$$

As we show next, E and Y can be assumed to be deterministic without loss of generality.

Lemma 2.6 (Deterministic environments and subjects). *Consider a controller* C. *Suppose that for some* $\epsilon, \delta \geq 0$, C *satisfies Definition 2.3 for all deterministic environments* E *and data subjects* Y *with simulator* Sim. *Then* C *satisfies Definition 2.3 (even for randomized* E *and* Y*) with the same simulator* Sim.

Proof. Fix a controller C, and let Sim be the simulator which show that C satisfies Definition 2.3 for all deterministic environments and data subjects. Now consider a pair of randomized ITMs E and Y with random coins denoted by R_E and R_Y.

For a fixed string r, let E_r denote the deterministic environment in which E's random tape is fixed to $R_E = r$; similarly for Y_r. Given strings r_E and r_Y, we can consider the execution of the deletion game in Definition 2.3 with the deterministic machines E_{r_E} and Y_{r_Y}. Let R'_{C, r_E, r_Y} denote the random coins output by the simulator in that game, and let P_{r_E, r_Y} denote the distribution of R'_{C, r_E, r_Y}. By hypothesis, $R'_{C, r_E, r_Y} \approx_{\epsilon, \delta} R_C$ for every r_E and r_Y.

Now consider the deletion game with the randomized machines E, Y. Let R'_C denote the output of the simulator in that game. Since the simulator Sim does not depend on the environment or data subject, the distribution of R'_C conditioned on the event that $R_E = r_E$ and $R_Y = r_Y$ is exactly P_{r_E, r_Y}. Thus, we have

$$(R_E, R_Y, R'_C) \equiv (R_E, R_Y, R'_{C, R_E, R_Y}) \approx_{\epsilon, \delta} (R_E, R_Y, R_C),$$

where $\equiv$ denotes that two random variables have identical distributions and we have used the convexity of the $\approx_{\epsilon, \delta}$ relation. Dropping the first two components, we get that $R'_C \approx_{\epsilon, \delta} R_C$, as desired.

Furthermore, by hypothesis, we have that state $= $ state$'$ with probability at least $1 - \delta$ for each setting of r_E and r_Y (since state depends only on R_E and R_Y via the queries $\vec{q}_E$). Averaging over r_E and r_Y shows that the overall probability that state $= $ state$'$ is also at least $1 - \delta$. $\qquad\square$

2.4 Discussion of the definition

On constraining C's state Our definition imposes a condition on the internal state of the controller at a moment in time. Namely that, immediately after the data subject Y is deleted,

the actual state of the controller state$_C$ can be plausibly attributed to the interaction between the controller and the environment alone. This in turn provides a guarantee for anything the controller may do in the future. Namely, if the real controller was replaced by the ideal controller at the moment of Y's deletion, the environment would never know.

As an alternative, one might consider restricting future behavior directly. We prefer to restrict the state as well. It is simpler to describe—for instance, because there is a natural termination condition. It is also future-proof: Any controller that satisfied the future-behavior version but not the state version could choose to, in the future, violate the guarantee by publishing the state at the time of Y's erasure. Imposing the condition on the state directly makes it impossible for the future behavior of the real and ideal controllers to deviate, rather than merely possible for them not to.

Differential privacy and deletion As we will see later in this paper, differential privacy (DP) can in some cases provide deletion-as-control almost automatically, with no additional action required of the data controller (Prop. 5.1 and Thm. 5.9). We believe that this makes sense both from the point of view of what DP means and the spirit of data protection regulations. When greater protection is warranted, deletion-as-control should not serve as the sole basis of analysis—nor should, perhaps, a right to erasure.

We're guided by a simple intuition: If a single individual has almost no influence on the result of data processing—the condition guaranteed by differential privacy—then nothing needs to be done to remove that individual's influence. This intuition closely tracks some prior approaches to deletion. For instance, to show that differentially private controllers satisfy deletion-as-*control*, we actually show that they meet the much stricter requirements of (approximate) deletion-as-*confidentiality* Garg et al. [2020]; see Lem. 5.10. Some existing machine unlearning algorithms embody the same intuition, leaving the trained model unaltered as long as the deleted data points had no effect on the resulting model Ginart et al. [2019].[6]

The fact that DP can provide deletion-as-control fits well with the data protection regulations, like GDPR and CCPA, that inspire our work. Generally, these laws give individuals rights regarding the processing of *personal data* relating to them.But these rights, including the right to erasure, do not extend to data that have been sufficiently anonymized.[7] If one believes that in some cases, DP anonymizes data for the purposes of GDPR, say, then in such cases the data controller need not take any further action when a data subject requests deletion. Whether DP releases constitute personal data is explored in recent work bridging computer science formalisms with legal analysis Nissim et al. [2017], Altman et al. [2021]. Though the general question remains unresolved, DP has been used to argue compliance with privacy laws for several high-profile data releases, including by the U.S. Census Bureau US Census Bureau [2023], Facebook King and Persily [2020], and Google Google [2020].

Of course, DP is not always the answer. For example, if a model was trained using data collected without proper consent, one might require that no benefit derived from the ill-gotten data remains. The Federal Trade Commission first adopted this type of *algorithmic disgorgement* in a

---

[6]In contrast, Thudi et al. [2022] argue that any definition where a data controller "do[es] not need to do anything and can claim the unlearning is done"—including machine unlearning definitions based on approximate history independence (Section 3.4)—is "not well-defined". We disagree.

[7]Recital 26 states this explicitly: "The principles of data protection should therefore not apply to anonymous information, namely information which does not relate to an identified or identifiable natural person or to personal data rendered anonymous in such a manner that the data subject is not or no longer identifiable."

2021 settlement with photo sharing app Everalbum [Slaughter et al., 2020]. Differential privacy should not shield against such algorithmic disgorgement.[8]

Deleting groups Our definition provides a guarantee for an individual data subject. What about groups? If many people request to be deleted, then each individual person enjoys the individual-level guarantee provided by deletion-as-control. But the group does not necessarily enjoy an analogous group-level guarantee. For example, the group-level deletion guarantee for the DP-based controllers in Section 5 decays linearly with the group size. For large groups, the group *as a group* doesn't enjoy meaningful protection.

This seems unavoidable in contexts where (useful) statistics are published once and not subsequently updated. It reflects a fundamental difference between deletion-as-control and the history independence-style definitions in the machine unlearning literature, discussed in Section 3.4. Suppose, for example, that a controller trains a model θ using data from n people. Then all n people request deletion, leaving the controller with a model θ^*. History independence would require that θ^* be essentially trivial: θ^* should perform about as well as a the model θ_0 trained on an empty dataset (Section 3.4). On the other hand, the DP-based controllers in Section 5 allow θ^* to perform very well on the learning task.

An individual-level guarantee is in line with data privacy laws. To whit, the GDPR grants a right to erasure to "the data subject" who is "[a] natural person" (Art. 4, 17). Even so, group-level deletion may be more appropriate in some settings (e.g., algorithmic disgorgement discussed above). Exploring group deletion is an important direction for future work.

Composition Composition is an important property of good cryptographic definitions. We do not yet have a complete picture of how deletion-as-control composes. Theorem 6.4 states a limited composition theorem that applies to parallel composition of two controllers at least one of which satisfies a very strong guarantee (specifically, it must implement a deterministic functionality with perfect, as opposed to approximate, deletion-as-control). By induction, this extends to the parallel composition of k controllers if all but one satisfy the strong guarantee. This can be used to reason about complex interactive functionalities built from multiple strongly history-independent data structures.

Proving more general composition for deletion-as-control is an important question for future work. Addressing it seems challenging since it is closely related to still-open questions about composition for differential privacy. For example, it was shown only very recently that differential privacy composes when mechanisms are run concurrently with adaptively interleaved queries [Vadhan and Zhang, 2022]. While that result allows adaptive query ordering, the dataset itself is fixed in advance. Deletion-as-control allows both queries and data to be specified adaptively. Proving composition of deletion-as-control seems only harder than the analogous question for differential privacy.

Other limitations of our approach We touch on two limitations of our approach. First, there is no quantification of "effort." The EU's right to be forgotten stems from *Google v Costeja*, where the Court of Justice for the European Union ruled that a search engine may be required to remove certain links from search results Court of Justice of the European Union [2013]. But there are

---

[8]Achille et al. [2023] seem to disagree, writing: "In many ways, differential privacy (DP) can be considered the 'gold standard' of model disgorgement".

limits. Today, Google will only remove the result from search queries related to the name of the person requesting deletion, but not from other search queries Google. This suggests a definition in which results are hidden from a low-resource adversary who only makes general searches, but not from an adversary with more side information or time, carrying out more targeted or exhaustive searches respectively. Modeling that sort of subtlety appears to require fundamental changes from all existing approaches, ours included.

Second, a failure of deletion as we formulate it doesn't map to an explicit attack on a system. It corresponds instead to a disconnect between the real execution and a counterfactual one in which Alice's data never existed but her effect on others' data remains. In this sense the definition is quite different from standard cryptographic ones, and it doesn't obviously correspond to an adversarial model nor combine well with other cryptographic definitions. This is also true of the history-independence approach, including the definitions in prior work on machine unlearning. Deletion-as-confidentiality [Garg et al., 2020] does have a more straightforward cryptographic flavor but, as we argue, its strict requirement is ill suited for many application.

2.5   Real and ideal executions in detail

Our definition involves the interaction of three parties in the real execution (C, E, and Y) and two parties in the ideal execution (C and D). Formally, the parties in our definition are interactive Turing machines (ITM) with behavior specified by code. An execution involves sequentially activating and deactivating the ITMs until a termination condition is reached. Activations are tied to message-passing as described below. At any time, at most one ITM is active. We denote the executions using $\langle C, E, Y \rangle$ or $\langle C, D \rangle$, respectively.

We require that each ITM eventually terminates for every setting of its tapes. (Note that this does not imply termination of an execution involving multiple such ITMs, who may for example pass messages back and forth forever.) One way to enforce termination is to ask that every party come with a time bound limiting the number of steps it executes when activated. We do not model this particular detail explicitly.

Each ITM has five tapes: work, input, output, randomness, and channel ID. An ITM's state at a given time includes only the contents of its work tape at the time. An ITM may freely read from and write to the work tape. The input tape is read-only and is reset when an ITM is deactivated. (At the start of the execution, input tapes are initialized with channel IDs as described below.) The output tape is write-only and is reset when an ITM is activated.

The randomness tape is initialized with uniform random bits.[9] To avoid a priori bounds on running time or randomness complexity, the randomness tape is countably infinite. It may only be read *once* over the course of the whole execution. That is, reading a bit from the randomness tape automatically advances that tape head one position, and there is no other way to move that tape head; if an ITM wishes to re-use bits from its randomness tape, it must copy them to its work tape.[10]

---

[9]One could instead view the controller's randomness as consisting of i.i.d. samples from an efficiently-sampleable distribution that may depend on C (e.g., Laplace noise or Gaussian noise). Uniform bits is without loss of generality, as the simulator Sim defined below may be inefficient.

[10]This is a departure from the model of Garg et al. [2020], wherein the randomness tape may be read multiple times without counting as part of the state. That would allow a controller to avoid deleting anything by, for instance, encrypting its state using its randomness as a secret key—a detail that was overlooked in Garg et al. [2020]. Our fix is to make the randomness read-once. As a result, a party must store in the work tape any randomness that it will later reuse.

Messages between parties are passed over authenticated channels, represented by a channel ID cID associated with two parties A and B. A single pair of parties may have many associated channels. Party A sends a message by writing (cID, msg) to its output tape, where msg $\in \{0,1\}^* \cup \{$delete$\}$, where delete is a special message (i.e., not in $\{0,1\}^*$). When A finishes writing its output, A is immediately deactivated. If cID corresponds to a channel ID between A and another party B, then (cID, msg) is written to B's input tape and B is activated. Otherwise, the special message fail is written to A's input tape and A is activated. Importantly, B does not learn the *party* that sent msg, only the *channel* cID over which it was sent.

Each ITM's channel ID tape is initialized with the cID's for channels over which the ITM can communicate. The channel ID tape is read-once, just like the randomness tape. In the real execution, E's tape is initialized with cIDs (countably infinite, as with the randomness tape). In the ideal execution, D's tape is similarly initialized with cIDs. The first is for communication with Y and the remainder are for communication with C. Y's input tape is initialized with only one cID for communication with C and one cID for communication with E. C's input is initially empty. It must learn the cIDs from messages it receives, and cannot distinguish its channel with Y from its channels with E or D.

The executions begin with the activation of E or D. Parties are deactivated when they write a message to their output tape and the recipient is activated. If a party halts without writing to its output tape, E or D is activated.

The real execution The real execution involves three (possibly randomized) parties: C, E, and Y using randomness R_C, R_E, and R_Y in $\{0,1\}^{\mathbb{N}}$, respectively. The execution ends after Y requests deletion from C. More precisely, after Y sends delete to C, C is activated one final time. The execution then ends when C halts or writes to its output tape. Note that the real execution may never terminate if Y never sends delete. We denote the real execution as $\langle C(R_C), E(R_E), Y(R_Y) \rangle$. When appropriate we omit the randomness, writing $\langle C, E, Y \rangle$.

The execution generates a (possibly empty) *transcript* τ of all messages sent to and received by C: $\tau = ((t, \text{sender}_t, \text{receiver}_t, \text{cID}_t, \text{msg}_t))_t$. For each t: one of sender_t or receiver_t is always C; cID_t is an id of a channel between sender_t and receiver_t; and $\text{msg}_t \in \{0,1\}^* \cup \{$delete$\}$. Though they may freely communicate, we omit messages between E and Y from the transcript. The transcript can be further divided into *queries* and *answers*. The queries are messages sent to C (by either E or Y). We denote by $\vec{q}$ the sub-transcript containing all queries, and by $\vec{q}_E$ only those messages sent by E. The answers $\vec{a}$ are messages to someone *from* C. We denote by $\vec{a}_E \subseteq \tau$ the ordered messages from C to E.

The real execution $\langle C(R_C), E(R_E), Y(R_Y) \rangle$ defines a transcript τ, a state state_C, and randomness R_C. The transcript τ is defined above. The state state_C consists of the contents of C's work tape at the end of the execution. If the execution does not terminate, we define $\text{state}_C = \bot$ and the transcript τ and its subset $\vec{q}_E$ are defined to be empty.

The ideal execution The ideal execution involves two parties: C and D. The dummy party D's input tape is initialized with the same channel IDs as E's tape in the real execution. The dummy simply replays the queries in $\vec{q}_E$ if any. At every activation, it sends the next query in the sequence to C using the same cID as in the real execution. Note that the *answers* D receives from C may be different from the real-world answers; however, D sends the same queries $\vec{q}_E$ regardless.

The controller C is exactly as in the real execution except that its randomness tape is initialized

using *simulated* randomness $R'_\mathsf{C} \leftarrow \mathsf{Sim}(\vec{\mathsf{q}}_\mathsf{E}, R_\mathsf{C}, \mathsf{state}_\mathsf{C})$ instead of *uniform* randomness $R_\mathsf{C} \sim \mathcal{U}$. The simulator is an (inefficient) algorithm Sim that takes as input $\vec{\mathsf{q}}_\mathsf{E}$, $\mathsf{state}_\mathsf{C}$, and R_C and produces output R'_C. Note that R_C and R'_C consist of countably infinitely-many bits; hence R_C and R'_C cannot be treated as conventional inputs and outputs to Sim. Instead, we give Sim access to a special tape on which R_C is written. Sim may overwrite finitely-many bits of this tape before it halts. We denote by R'_C the final contents of this tape.

We denote the ideal execution as $\langle \mathsf{C}(R'_\mathsf{C}), \mathsf{D}(\vec{\mathsf{q}}_\mathsf{E}) \rangle$. The ideal execution ends after D sends its last query from $\vec{\mathsf{q}}_\mathsf{E}$ to C. When this happens, C is activated one final time. The execution ends when C halts or writes to its output tape. state' is defined as the final state of C. Note that the ideal-world execution terminates if the real-world execution terminates. If the ideal execution does not terminate, we define $\mathsf{state}' = \bot$. In this case, $\vec{\mathsf{q}}_\mathsf{E}$ must be infinitely long and hence the corresponding real execution did not terminate.

3   History Independence and Deletion-as-control

History independence (HI) is concerned with the problem that the memory representation of a data structure may reveal information about the history of operations that were performed on it Micciancio [1997], Naor and Teague [2001], Hartline et al. [2005]. HI requires that the memory representation reveals nothing more than the current logical state of the data structure. Setting aside a number of technical subtleties, the conceptual connection to machine unlearning is immediate: if we consider a machine learning model as a representation of a dictionary data structure with insert and remove (i.e., unlearn) operations, a model is HI if and only if it satisfies machine unlearning.

In this section, we state the definition of (non-adaptive) history independence (Section 3.1). We then define a more general notion that allows for implementations that satisfy the conditions of HI *approximately* and *adaptively* (Section 3.2). The generalization is complex since we must explicitly model adaptivity in the interactions between a data structure and those issuing queries to it. Briefly, an adaptive adversary A interacting with the data structure produces two equivalent query sequences. Adaptive history independence (AHI) requires that the joint distribution of A's view and the data structure's state is the same under both sequences. Approximate AHI requires these distributions to be (ϵ, δ)-close.

We show that data controllers that satisfy approximate AHI also satisfy our notion of deletion-as-control (Section 3.3) with the same parameters. Finally, we show how existing definitions of *machine unlearning* and the corresponding constructions are all (weakenings of) our general notion of history independence (Section 3.4).

3.1   History independence

An *abstract data type (ADT)* is defined by a universe of operations $\{\mathsf{op}\}$ and a mapping ADT : $(\mathsf{op}, s_\mathsf{adt}) \mapsto (s'_\mathsf{adt}, \mathsf{out}_\mathsf{adt})$. We call $s_\mathsf{adt}, s'_\mathsf{adt} \in \{0, 1\}^*$ the *logical states* before and after operation, where s'_adt is a deterministic function of s_adt and op. We call $\mathsf{out}_\mathsf{adt} \in \{0, 1\}^* \cup \{\bot\}$ the *logical output*, which may be randomized. In subsequent sections we will assume without loss of generality that operations $\mathsf{op}(id)$ are tagged by $id \in \{0, 1\}^*$. We omit the tags where possible to reduce clutter.

Given an initial logical state s^0_adt and a sequence of operations $\sigma = (\mathsf{op}^1, \mathsf{op}^2, \dots)$, the ADT defines a sequence of logical states $(s^1_\mathsf{adt}, s^2_\mathsf{adt}, \dots)$ and a sequence of outputs $\overrightarrow{\mathsf{out}} = (\mathsf{out}^1_\mathsf{adt}, \mathsf{out}^2_\mathsf{adt}, \dots)$ by iterated application of ADT. We denote by $\mathsf{ADT}(\sigma).\mathsf{state}$ the final logical state that results from

this iterated application. When no initial state is specified, it is assumed to be the empty state.

Definition 3.1. *We say two sequences of operations σ and σ' are* logically equivalent*, denoted $\sigma \stackrel{L}{\equiv} \sigma'$, if* $\mathsf{ADT}(\sigma).\mathsf{state} = \mathsf{ADT}(\sigma').\mathsf{state}$. *Logical equivalence is an equivalence relation, and we denote by $[\sigma]$ a canonical sequence in the equivalence class of sequences $\{\sigma' : \sigma' \stackrel{L}{\equiv} \sigma\}$.*

An *implementation* (e.g., a computer program for a particular architecture) is a possibly randomized mapping $\mathsf{Impl} : (\mathsf{op}, s) \mapsto (s', \mathsf{out})$. We call s the *physical state* and out the *physical output*. Both may be randomized. Given an initial state and sequence of operations, Impl defines a sequence of physical states and outputs by iterated application. When no initial state is specified, it is assumed to be the empty state.

Definition 3.2 (History independence [Naor and Teague, 2001])**.** Impl *is a* weakly history independent *implementation of* ADT (WHI-implements ADT) *if*

$$\sigma \stackrel{L}{\equiv} \sigma' \implies \mathsf{Impl}(\sigma).\mathsf{state} \equiv \mathsf{Impl}(\sigma').\mathsf{state}, \tag{1}$$

where $\equiv$ denotes equality of distributions. Impl *is a* strongly history independent *implementation of* ADT *(SHI-implements ADT) if for all initial states s*

$$\sigma \stackrel{L}{\equiv} \sigma' \implies \mathsf{Impl}(\sigma, s).\mathsf{state} \equiv \mathsf{Impl}(\sigma', s).\mathsf{state}. \tag{2}$$

One can obtain approximate, *nonadaptive* versions of history independence by replacing $\equiv$ with $\stackrel{\epsilon,\delta}{\approx}$ in Definition 3.2. However, because the sequence of queries is specified ahead of time, such a definition's guarantees are not meaningful in interactive settings—see Example 3.3.

We do not define correctness of an implementation of an ADT. Thus every ADT trivially admits a SHI implementation (e.g., Impl always outputs $\bot$). Omitting correctness simplifies the specification of the ADT while allowing flexibility—for example, if approximate correctness suffices for an application. The usefulness of an implementation requires separate analysis. However, history independence simplifies this step: it suffices to analyze the utility for the canonical sequences of operations $[\sigma]$ instead of arbitrary sequences σ.

3.1.1 Strongly History Independent Dictionaries

To illustrate history independence, consider the dictionary ADT, which models a simple key-value store. Looking ahead, we will use history independent dictionaries to build deletion-compliant controllers from differential privacy. For our purposes, the keys will be party IDs *id*; the values can be arbitrary. The ADT supports operations $\mathsf{insert}(id)$, $\mathsf{delete}(id)$, $\mathsf{get}(id)$, and $\mathsf{set}(id, value)$, where set associates the key corresponding to *id* with *value*, and get returns the most recently set value. We assume that $\mathsf{insert}(id)$ is equivalent to $\mathsf{set}(id, \top)$ where $\top$ is a special default value.

Dictionaries are typically implemented as hash tables, but such data structures are generally *not* history independent.

For example, in hashing with open addressing, deletions are typically done lazily (by marking the cell for a deleted key with a special, "tombstone" value). So one can tell whether an item was deleted but also learn the deleted item's hash value. Storing a dictionary as a sorted list is inefficient—updates generally take time $\Omega(n)$, where n is the current number of keys—but it enjoys strong history independence. The sorted representation depends *only* on the logical contents of the

dictionary, not on the order of insertions nor which items were inserted then deleted. Since we do not focus on efficiency here, the reader may think of the sorted list as our default implementation of a dictionary. [11]

3.2 Adaptive History Independence (AHI)

The history independence literature gives no guarantees against adaptively-chosen sequences of queries, because the two sequences σ and σ' are fixed before the implementation's randomness is sampled.

Example 3.3. *Consider an implementation* Impl *of a dictionary with two operations:* insert(id) *and* delete(id). *Upon initialization,* Impl *outputs the first n bits of its uniform randomness tape, denoted r, and stores it in state $s^1 = r$. If the first operation is* insert(r), Impl *will store the subsequent sequence of operations it its state. Otherwise,* Impl *ignores all subsequent operations, setting $s = r$.* Impl *produces no outputs other than the initial* $\mathsf{out}^1 = r$. Impl *satisfies $(0, 2^{-n})$-approximate nonadaptive strong history independence: for any sequences σ and σ' fixed in advance, and for any initial s,* Impl(σ, s).state $= r =$ Impl(σ', s), *except with probability $2 \cdot 2^{-n}$. However, an adaptive adversary that sees* $\mathsf{out}^1 = r$ *can easily produce distinct $\sigma \stackrel{L}{\equiv} \sigma'$ such that* Impl(σ).state $= \sigma \neq \sigma' =$ Impl(σ').state.

Inspired by [Gupta et al., 2021], we extend the well-studied notion of history independence to the *adaptive* setting, where the sequence σ of operations is chosen adaptively by an algorithm interacting with an implementation of an ADT.

We consider an interaction $\langle$Impl$(R), \mathsf{A}\rangle$ between an algorithm A and the implementation Impl with random tape $R \sim \mathcal{U}$ (Figure 4).

In the interaction, A adaptively outputs an operation $\mathsf{op}^i \leftarrow \mathsf{A}(\mathsf{op}^1, \mathsf{out}^1, \ldots, \mathsf{op}^{i-1}, \mathsf{out}^{i-1})$, and receives the output out^i in return. The interaction defines a sequence of operations σ and corresponding outputs $\overrightarrow{\mathsf{out}}$. Eventually, A outputs a sequence σ^* that is logically equivalent to the sequence σ of operations performed so far. Impl is executed on σ^* and alternate randomness R^*, resulting in $s^* =$ Impl$(\sigma^*; R^*)$.state. We consider two variants: $R^* = R$, or $R^* \sim \mathcal{U}$ independent of R.

We consider the adversary's ability to distinguish the real state $s =$ Impl$(\sigma; R)$.state and the logically equivalent state $s^* =$ Impl$(\sigma^*; R^*)$.state, given its view $V_{\mathsf{Impl},\mathsf{A}} = (\sigma, \sigma^*, \overrightarrow{\mathsf{out}})$. Our definition of adaptive history independence requires that the joint distributions of $(V_{\mathsf{Impl},\mathsf{A}}, s)$ and $(V_{\mathsf{Impl},\mathsf{A}}, s^*)$ be (ϵ, δ)-close. We restrict ourselves to adversaries A such that $\langle$Impl$, \mathsf{A}\rangle$ always terminates, which we call *valid* adversaries.

Definition 3.4 (Adaptive (weak) history independence)**.** *An implementation* Impl *of an* ADT *is (ϵ, δ)-history independent (AHI) if for all valid adversaries* A*:*

$$(V_{\mathsf{Impl},\mathsf{A}}, s) \stackrel{\epsilon,\delta}{\approx} (V_{\mathsf{Impl},\mathsf{A}}, s^*)$$

where the distributions are given by the following probability experiment. The experiment has two versions (identical or independently drawn randomness):

---

[11]In fact, a strongly history independent hash table implementation with constant expected-time operations was described by Blelloch and Golovin [2007].

Figure 4: Adaptive history independence game, from Definition 3.4

1: $R \sim \mathcal{U}$
2: Either $R^* = R$ (identical randomness), or $R^* \sim \mathcal{U}$ (independent randomness)
3: $(\sigma, \overrightarrow{\mathsf{out}}, s) \leftarrow \langle \mathsf{Impl}(R), \mathsf{A} \rangle$
4: A outputs σ^*
5: $s^* \leftarrow \mathsf{Impl}(\sigma^*; R^*).\mathsf{state}$
6: If $\sigma^* \overset{L}{\not\equiv} \sigma$, then $s^* \leftarrow s$
7: $V_{\mathsf{Impl},\mathsf{A}} \leftarrow (\sigma, \sigma^*, \overrightarrow{\mathsf{out}})$

We say an implementation Impl is (ϵ, δ)-AHI if it satisfies either version of the definition (identical or independently drawn randomness).

Note that for both versions of Definition 3.4, R and R^* are identically distributed. One can generalize the definition to allow for any joint distribution over (R, R^*) such that the two marginal distributions are identical (all of our proofs would still go through for this general notion). However, we avoid this generality since the two versions that we present are sufficient for our needs.

The AHI definiton here corresponds to the approximate, adaptive version of *weak* history independence (Definition 3.2), which considers only the state state at a single point in time. This suffices for our purposes. One could generalize Definition 3.4 to allow an adversary to see the full internal state in the real execution at multiple points in time; this would correspond to strong history independence. We leave such an extension for future work.

From Strong Nonadaptive HI to Adaptive History Independence   To illustrate the definition, consider the case of strongly history independent dictionaries. The hash table construction of Blelloch and Golovin [2007] is a randomized data structure with the following property: for every setting of the random string $R = r$, and for every two logically equivalent sequences σ, σ^*, the data structure stores the same table $s(\sigma, r) = s(\sigma^*, r)$. (In fact, essentially all $(0,0)$-strongly HI data structures can be modified to satisfy this strong guarantee [Hartline et al., 2005]).

The data structure's answers will thus depend only on the logical data set. However, they might also depend on the randomness r—for instance, the answers might leak the length of the probe sequence needed to find a given element or other internal information. In an adaptive setting, queries might depend on r via the previous answers. To emphasize this dependence, we write the realized query as $\sigma(r)$.

This construction satisfies $(0,0)$-adaptive HI with identical randomness (where $R' = R$ in Step 2 of Definition 3.4) but *not* the version with fresh randomness ($R' \perp R$). To see why it satisfies the definition with identical randomness, consider the tuples $(\sigma(R), \sigma(R)^*, \overrightarrow{\mathsf{out}}, s(\sigma, R))$ and $(\sigma(R), \sigma(R)^*, \overrightarrow{\mathsf{out}}, s(\sigma^*, R))$ resulting from the game in Def. 3.4. By logical equivalence, the tuples are the same, and hence identically distributed. On the other hand, if we select R' independently of R, then the adversary will, in general, see a difference between $\sigma(R)$ and $s(\sigma^*, R')$ (for instance, the hash functions corresponding to R and R' will be different with high probability).

3.3 AHI and Deletion-as-Control

We prove two theorems showing that data controllers that implement history independent ADTs satisfy deletion-as-control. Before that, two technicalities remain. First, the statement only makes sense if the ADT itself supports some notion of deletion. Second, there is a mismatch between the syntax of ADTs/implementations and the deletion-as-control execution. Next we define *ADTs supporting logical deletion* and *controllers relative to an implementation* to handle the two issues, then state and prove the main theorems of this section.

Consider an ADT where operations $\{\mathsf{op}(id)\}$ are tagged with an identifier $id \in \{0,1\}^*$ (e.g., the channel IDs). For a sequence σ of operations and an id^*, let σ_{-id^*} be the sequence of operations with every operation with identifier id^* removed (that is, $\mathsf{op}(id^*)$ for all values of op).

Definition 3.5 (Logical Deletion). *ADT supports* logical deletion *if there exists an operation* delete *such that for all sequences of operations σ and for all IDs id^*: $(\sigma \| \mathsf{delete}(id^*)) \overset{L}{\equiv} \sigma_{-id^*}$.*

Logical deletion is important for ensuring the correctness property of deletion compliance (that the states in the two worlds are identical). Conversely, if an ADT does not support logical deletion, then one would expect any controller that faithfully implements the ADT to violate the deletion requirement.

The notion of logical deletion applies to a wide variety of abstract data types, not only dictionaries. For example, consider a public bulletin board, where users can post messages visible to everyone. The natural ADT would allow creation of new users, insertion and maybe deletion of specific posts, and deletion of an entire user account. This latter operation would undo all previous actions involving that account. We consider a family of data controllers C that are essentially just implementations of ADTs that can interface with the execution.

Definition 3.6 (Controller relative to an implementation). *Let* Impl *be an implementation of an* ADT. *We define the controller* $\mathsf{C}_{\mathsf{Impl}}$ *relative to* Impl *as the controller that maintains state* state *and works as follows:*

- *On input* $(\mathsf{cID}, \mathsf{msg})$:
  - $(\mathsf{state}', \mathsf{out}) \leftarrow \mathsf{Impl}(\mathsf{op}(\mathsf{cID}), \mathsf{state})$, *where* $\mathsf{op} \leftarrow \mathsf{msg}$.
  - *Write* $(\mathsf{cID}, \mathsf{out})$ *to the output tape.*
- *On input* fail: *Halt.*

We say C *is history independent (adaptively/strongly/weakly) if* Impl *is history independent (adaptively/strongly/weakly).*

Theorem 3.7. *For any ADT that supports logical deletion and any SHI implementation* Impl, *the controller* $\mathsf{C} = \mathsf{C}_{\mathsf{Impl}}$ *satisfies* $(0,0)$-deletion-as-control *with the simulator that outputs* $R'_{\mathsf{C}} = R_{\mathsf{C}}$.

Without the condition on the simulator, this is a corollary of the theorem below. The proof in Appendix C uses a foundational result in the study of history independence. Roughly, that SHI implies canonical representations for each logical state of the ADT [Hartline et al., 2005].

Theorem 3.8. *For any ADT that supports logical deletion and any* Impl *of the ADT satisfying* (ϵ, δ)-*AHI (with either variant of Definition 3.4), the controller* $\mathsf{C} = \mathsf{C}_{\mathsf{Impl}}$ *is* (ϵ, δ)-*deletion-as-control compliant.*

The proof uses a simple, novel result on indistinguishability, dubbed the Coupling Lemma, which we present and prove in Appendix A.

Proof of Theorem 3.8. Fix any deterministic E and Y. By Lemma 2.6 this is without loss of generality. Throughout this proof, we drop the subscript Impl from the controller $\mathsf{C}_{\mathsf{Impl}}$, and drop the subscript $\mathsf{C}_{\mathsf{Impl}}$ from the controller's state $\mathsf{state}_{\mathsf{C}_{\mathsf{Impl}}}$ and randomness $R_{\mathsf{C}_{\mathsf{Impl}}}$.

Definition 3.4 considers two variants of AHI, depending on how the randomness R used in the initial Impl evaluation (with sequence σ) relates to the randomness R^* used in the logically equivalent evaluation (with sequence σ^*). (Either R and R^* are sampled i.i.d. or $R^* = R$.) Let $\mathfrak{D}$ be the joint distribution over (R, R^*) for which Impl enjoys the (ϵ, δ)-AHI guarantee. Observe that the marginal distribution of R^* is $\mathcal{U}$.

To prove that the controller C is deletion-as-control compliant, we will use the following simulator Sim: $\mathsf{Sim}(\vec{\mathsf{q}}_{\mathsf{E}}, R, \mathsf{state})$ samples $(\widetilde{R}, \widetilde{R}^*)$ from $\mathfrak{D}$ conditioned on the following event:

$$\vec{\mathsf{q}}_{\mathsf{E}}(\widetilde{R}) = \vec{\mathsf{q}}_{\mathsf{E}} \quad \wedge \quad \mathsf{C}(\vec{\mathsf{q}}_{\mathsf{E}}; \widetilde{R}^*) = \mathsf{state},$$

where $\vec{\mathsf{q}}_{\mathsf{E}}(\widetilde{R})$ are the queries from E to C in the execution $\langle \mathsf{C}(\widetilde{R}), \mathsf{E}, \mathsf{Y} \rangle$. Sim outputs $R' = \widetilde{R}^*$. The real execution's state is $\mathsf{state} = \mathsf{C}(\vec{\mathsf{q}}(R); R)$. The ideal execution's state is $\mathsf{state}' = \mathsf{C}(\vec{\mathsf{q}}_{\mathsf{E}}(R); R')$.

Claim 3.9. *Define f and g as follows:*

$$f(R) = \left(\vec{\mathsf{q}}_{\mathsf{E}}(R), \ \mathsf{C}(\vec{\mathsf{q}}(R); R) \right)$$
$$g(R, R^*) = \left(\vec{\mathsf{q}}_{\mathsf{E}}(R), \ \mathsf{C}(\vec{\mathsf{q}}_{\mathsf{E}}(R); R^*) \right),$$

Then $(R, R^*) \sim \mathfrak{D} \implies f(R) \overset{\epsilon, \delta}{\approx} g(R, R^*)$.

The proof of the claim is below. We use the claim to complete the proof of the theorem. Consider a procedure that samples $R \sim \mathcal{U}$, and then samples $(\widetilde{R}, \widetilde{R}^*) \sim \mathfrak{D}\big|_{f(R) = g(\widetilde{R}, \widetilde{R}^*)}$ (if possible, otherwise uniformly). Observe that this is exactly the distribution of $R \sim \mathcal{U}$ and the intermediate vales $(\widetilde{R}, \widetilde{R}^*)$ sampled by Sim defined above. Applying the Coupling Lemma (Lemma A.1) to the preceding claim, we get that the joint distribution over $(R, \widetilde{R}, \widetilde{R}^*)$ satisfies $(\widetilde{R}, \widetilde{R}^*) \overset{\epsilon, \delta}{\approx} \mathfrak{D}$, and $f(R) = g(\widetilde{R}, \widetilde{R}^*)$ with probability at least $1 - \delta$.

Sim outputs $R' = \widetilde{R}^*$. Because the marginal distribution of $\widetilde{R}^*$ is $\mathcal{U}$, we have that $R' \overset{\epsilon, \delta}{\approx} \mathcal{U}$. The equality of f and g implies that $\mathsf{state}' = \mathsf{C}(\vec{\mathsf{q}}_{\mathsf{E}}(R); R') = \mathsf{C}(\vec{\mathsf{q}}(R); R) = \mathsf{state}$. Hence, $\mathsf{C}_{\mathsf{Impl}}$ satisfies (ϵ, δ)-deletion-as-control with the simulator Sim. $\qquad \square$

Proof of Claim 3.9. We will use the adaptive history independence of Impl. We define an AHI adversary $\mathsf{A} = \mathsf{A}_{\mathsf{E}, \mathsf{Y}}$ (see Figure 5)

which gets query access to the implementation Impl in the AHI game. A emulates the execution $\langle \mathsf{C}, \mathsf{E}, \mathsf{Y} \rangle$ internally, querying Impl as needed to implement C. When the execution terminates, A

outputs $\sigma^* = \vec{q}_E$, the queries sent from E to C. The actual sequence of queries received by $Impl$ in the real execution is $\sigma = \vec{q}$, including all queries from both E and Y.

We claim that $\sigma^* \stackrel{L}{\equiv} \sigma$. To see why, observe that $\sigma^* = \sigma_{-Y}$ is the query sequence with all queries from Y removed. Moreover, σ ends in $\mathsf{delete}(Y)$. By the hypothesis that ADT supports logical deletion $\sigma \stackrel{L}{\equiv} \sigma_{-Y} = \sigma^*$.

Figure 5: Reduction from (ϵ, δ)-AHI to (ϵ, δ)-deletion-as-control.

By the (ϵ, δ)-AHI of $Impl$, $(V_{Impl,A}, s) \stackrel{\epsilon,\delta}{\approx} (V_{Impl,A}, s^*)$. By construction, the state $s = Impl(\sigma; R).\mathsf{state}$ is exactly the controller's state $s = C(\vec{q}; R)$ in the real execution $\langle C(R), E, Y \rangle$. Likewise, the state $s^* = Impl(\sigma^*; R^*)$ is exactly the the controller's state $s' = C(\vec{q}_E; R^*)$ in the ideal execution $\langle C(R^*), D(\vec{q}_E) \rangle$. A's view $V_{Impl,A}$ consists of $(\sigma, \sigma^*, \overrightarrow{out}) = (\vec{q}(R), \vec{q}_E(R), \overrightarrow{out})$. Hence,

$$f(R) = (\vec{q}(R), s) \stackrel{\epsilon,\delta}{\approx} (\vec{q}(R), s') = g(R, R^*). \tag{3}$$

$\square$

As a corollary to Theorem 3.8, any history independent implementation of the Private Cloud Storage touchstone or the Public Bulletin Board examples (from Section 1.1) satisfy deletion-as-control.

The Private Cloud Storage functionality works like a key-value store: each user has their own dictionary D_{id}. To upload a file, a user sends $Upload(id, filename, file)$ to the controller, which internally adds $(filename, file)$ to D_{id} (if $filename$ already exists in D_{id} then nothing happens). To download one of their own files, the user sends $Download(id, filename)$ to the controller and receives the corresponding $file$ from D_{id}, if one exists. To delete their account, the user sends $Delete(id)$, in which the controller removes dictionary D_{id}. Two sequences of operations are logically equivalent if, for every user id, the dictionary D_{id} has the same logical content after applying the two sequences.

We define the Public Bulletin Board functionality as follows: users can post a message by sending $Post(id, msg)$ to the controller; they can receive all messages currently on the board by sending $Read()$; and they can delete all of their messages by sending $Delete(id)$ to the controller. Internally, the Public Bulletin Board stores an ordered list of (id, msg) pairs (ordered by insertion time). Two sequences of operations are logically equivalent if they yield the same ordered list of (id, msg) pairs.

Corollary 3.10. *The Private Cloud Storage and Public Bulletin Board touchstone controllers (Section 1.1) implemented with (ϵ, δ)-adaptive history independence each satisfy (ϵ, δ)-deletion-as-control.*

3.4 From Prior Definitions of Machine Unlearning to History Independence

We claim that AHI captures the essence of existing definitions of "machine unlearning" (that is, protocols that update a machine learning model to reflect deletions from the training data). Each definition in the literature corresponds to a special case of history independence, though each weakens the definition in one or more ways (even when their constructions satisfy the stronger, general notion). For illustration, we discuss the approach of Gupta et al. [2021] in detail.

The basic correspondence comes via considering an abstract data type, which we dub the *Updatable ML*, that extends a dictionary: it maintains a multiset $\vec{x}$ of labeled examples from some universe $\mathcal{Z}$. In addition to allowing insert() and delete() operations, it accepts a possibly randomized operation predict which outputs a predictor (or other trained model) ψ trained on $\vec{x}$. The accuracy requirement for ψ is generally not fully specified, not least because many current machine learning methods don't come with worst-case guarantees. The literature on machine unlearning generally requires that the distribution of the final predictor ψ (e.g., the model parameters) is approximately the same as it would be for a minimal sequence of operations that leads to the same training data set. In particular, *deleting an individual Y should mean that ψ looks roughly the same as if Y had never appeared in $\vec{x}$.* In principle one could satisfy the requirement by simply retraining ψ from scratch every time the data set changes, though this may be practically infeasible. The literature therefore focuses on methods that allow for faster updates.

Specializing history independence to updatable ML First, we spell out how AHI (Definition 3.4) specializes to the Updatable ML ADT. Given a sequence of updates and prediction queries $\sigma = (u_1, ..., u_t)$, let $[\sigma]$ denote a canonical equivalent sequence of updates. For example, $[\sigma]$ may (a) discard any insertion/deletion pairs that operate on the same element, with the insertion appearing first, and (b) list the remaining operations in lexicographic order. In an interaction with an implementation Impl that is initialized with an empty data set, an adversary makes a sequence of queries σ (of which only the updates actually affect the state) and, eventually, terminates at some time step t. Let s the resulting state of the controller based on randomness R, and let $\vec{x}_t$ be the final logical data set. After the interaction ends, the adversary outputs some logically equivalent sequence σ^* (for example, it could choose the canonical sequence $\sigma^* = [\sigma]$). Finally, run Impl from scratch with randomness R' and queries σ^* to get state s^*. The logical data set $\vec{x}_t$ is the same for σ and σ^* by construction. Definition 3.4 requires that

$$(\sigma, s) \overset{\epsilon, \delta}{\approx} (\sigma, s^*). \tag{4}$$

Definitions from the literature. The exact definition of deletion varies from paper to paper (and some papers do not define terms precisely). All formulate the problem in terms of what we call the Updatable ML ADT. We claim their definitions are variations on adaptive history independence (Eq. 5), each with one or more of the following weakenings:

Restricted queries Some papers consider only a subset of allowed operations—e.g., in Ginart et al. [2019], Ullah et al. [2021], Sekhari et al. [2021] data is inserted as a batch, and then only deletions occur.

One output versus future behavior Some consider only the output of the system at one time (e.g., Neel et al. [2020], Sekhari et al. [2021], Gupta et al. [2021]), while others also consider the internal state (e.g., Ginart et al. [2019]). The narrower approach *does not constrain the future behavior of the system.* Among those definitions that consider the full state, none discuss issues of internal representations; in this section, we also elide this distinction, assuming that datasets can be represented internally using strongly history-independent data structures.

Nonadaptive queries Except for Gupta et al. [2021], the literature considers only adversaries that specify the set of queries to be issued in advance. For constructions that are $(0,0)$-HI (that is, in which the real and ideal distributions are identical), this comes at no loss of generality. However, the nonadaptive and adaptive versions of (ϵ, δ)-HI for $\delta > 0$ are very different [Gupta et al., 2021]. Even in Gupta et al. [2021], the length t of the query sequence is chosen nonadaptively.

Symmetric vs asymmetric indistinguishability Ginart et al. [2019] and Gupta et al. [2021] consider a one-sided weakening of (ϵ, δ)-indistinguishability (discussed in Remark 3.11).

The definitions of Ginart et al. [2019], Ullah et al. [2021], Bourtoule et al. [2021] consider *exact* variants of unlearning (with $\epsilon = \delta = 0$), while others Guo et al. [2020], Sekhari et al. [2021], Neel et al. [2020], Gupta et al. [2021], Golatkar et al. [2020a,b] consider approximate variants.

Despite generally formulating weaker definitions, the algorithms in the literature often satisfy history independence. In particular, because the constructions in Ginart et al. [2019], Cao and Yang [2015] are fully deterministic, it is easy to see that they satisfy adaptive, $(0,0)$-*strong history independence* (by Theorem 1 in Hartline et al. [2005]). Some constructions satisfy additional properties, such as storing much less information than the full data set $\vec{x}_t$ (e.g., Sekhari et al. [2021], Cao and Yang [2015]).

Adaptive Machine Unlearning (Gupta et al. [2021]) We discuss the relationship to one previous definition—that of Gupta et al. [2021]—in detail, since the definition is subtle and the relationship is technically nontrivial. In our notation, the data structure maintains several quantities: a data set $\vec{x}$, a collection of models $\vec{\theta}$, some supplementary data supp used for updating, and the most recently output predictor ψ. We say that an implementation of the Updatable ML ADT meets the syntax of Gupta et al. [2021] if its state can be written $s = (\vec{x}_t, \vec{\theta}_t, \text{supp}_t, \psi_t)$. Given an initial data set $\vec{x}_0$, the execution proceeds for t steps. At each step i, the adversary (called the "update requester") sends an update u_i, which is either an insertion or deletion. The data structure then updates its state to obtain $(\vec{x}_i, \vec{\theta}_i, \text{supp}_i, \psi_i)$ and returns ψ_i to the adversary.

The condition required by Gupta et al. is described by three parameters $\alpha, \beta, \gamma > 0$. For every time t and initial dataset $\vec{x}_0$, with probability $1 - \gamma$ over the updates $\vec{u} = u_1, ..., u_t$ that arise in an interaction between the data structure (initialized with $\vec{x}_0$ and randomness R) and the adversary, the models $\vec{\theta}_t$ should be distributed similarly to the models $\vec{\theta}_0^*$ one would get in a counterfactual execution in which the data structure was initialized with the data set $\vec{x}_0^* = \vec{x}_t$ (that is, $\vec{u}$ applied to $\vec{x}_0$) resulting from the real execution in canonical form (say, in sorted order), and no updates were made. They require:

$$\Pr_{\vec{u}=(u_1,...,u_t)} \left(\vec{\theta}_t \big|_{\vec{x}_0, u_1,...,u_t} \quad \approx_{\alpha,\beta} \quad \vec{\theta}_0^* \big|_{\vec{x}_0^*} \right) \geq 1 - \gamma \tag{5}$$

where $\vec{x}_0^* = (\vec{u} \text{ applied to } \vec{x}_0)$.

Remark 3.11. *Gupta et al. actually require only a one-sided version of the condition on distributions enforced by* (α, β)-*closeness. Specifically, no event should be much more likely in the models resulting from a real interaction than in ones generated from scratch with the resulting data set. This condition is inherited from a definition proposed by Ginart et al. [2019].*

As far as we know, all the actual algorithms designed for machine unlearning satisfy the stronger, symmetric version of the definition in Equation (5)*.*

Proposition 3.12 (AHI implies Adaptive Machine Unlearning). *Let* Impl *be a data structure that implements the Updatable ML ADT and satisfies* (ϵ, δ)-*approximate history independence with fresh randomness (Definition 3.4). If* Impl *fits the syntax of machine unlearning, then for all* $\beta > 0$*, it also satisfies* (α, β, γ)-*unlearning (Equation* (5)*) with* $\alpha = 3\epsilon$ *and* $\gamma = \frac{2\delta}{\beta} + \frac{2\delta}{1-\exp(-\epsilon)}$*.*

Proof. Consider an Updatable ML data structure Impl that satisfies approximate HI and fits the syntax of Gupta et al. [2021], that is, its state s can be written $(\vec{x}_t, \vec{\theta}_t, \text{supp}_t, \psi_t)$.

Let UpReq be an update requester for the machine unlearning definition. UpReq first chooses a time step t at which it will terminate and an initial data set $\vec{x}_0$, and then executes an interaction with Impl during which it makes update queries $\vec{u} = (u_1, ..., u_t)$ and receives predictors $\psi_1, ...\psi_t$. Let $\vec{x}_t$ be the final logical data set. We define a corresponding adversary A for history independence as follows:

1: **procedure** A(given black-box access to UpReq)
2:     Run UpReq to obtain t and $\vec{x}_0$.
3:     Compute a sequence of updates $\vec{u}$ such that $\vec{u}_0 \circ \emptyset = \vec{x}_0$
4:     Output queries $\vec{u}_0$ followed by the operation predict
5:     Receive ψ_0
6:     **for** $i = 1$ to t **do**
7:         Run UpReq on input ψ_{i-1} to get next update u_i
8:         Output query u_i followed by predict
9:         Receive ψ_i
10:     Output $\sigma^* = [\sigma] = [\vec{u}]$ where $\vec{u} = \vec{u}_0 \| u_1, ..., u_t$, and σ is the full realized sequence (namely, $\vec{u}$ with interleaved predict queries).

To apply HI, we consider an interaction between Impl and A, followed by a new execution of Impl with queries σ^* (which equals $[\vec{u}]$ for this adversary) and fresh randomness. Let $\vec{x}_t$ be the final data set (in both executions). Equation (4) implies that

$$(\sigma, \underbrace{\vec{x}_t, \vec{\theta}_t, \text{supp}_t, \psi_t}_{s}) \overset{\epsilon,\delta}{\approx} (\sigma, \underbrace{\vec{x}_t, \vec{\theta}_0^*, \text{supp}_0^*, \psi_0^*}_{s^*}).$$

In particular, since the realized updates $\vec{u}$ form a subsequence of σ, we have

$$(\vec{u}, \vec{\theta}_t) \overset{\epsilon,\delta}{\approx} (\vec{u}, \vec{\theta}_0^*). \tag{6}$$

To draw the connection with Adaptive Machine Learning, we must define what it means to initialize Impl with a nonempty data set (since such initialization occurs in the definition of Adaptive Machine Unlearning). On input a nonempty data set $\vec{x}_0$, we can compute the canonical sequence of updates $\vec{u}_0$ that would lead to $\vec{x}_0$ (as in Step 3 in A above) and run Impl with $\vec{u}$ as its first queries, before real interactions occur. With this definition, the triple $(\vec{u}, \vec{\theta}_t, \vec{\theta}_0^*)$ is distributed exactly as in (5).

By Lemma A.3, with probability at least $1 - \gamma$ over $\vec{u}$, we have $\vec{\theta}_t|_{\vec{u}} \overset{\epsilon', \delta'}{\approx} \vec{\theta}_0^*$, where $\epsilon' = 3\epsilon$ and $\gamma = \frac{2\delta}{\delta'} + \frac{2\delta}{1 - \exp(-\epsilon)}$, as desired. $\qquad \square$

The converse implication does not hold, but we claim that it would after some modifications of the definition of [Gupta et al., 2021]:

- The indistinguishability condition of [Gupta et al., 2021] (Equation (5)) applies only to the current collection of models $\vec{\theta}_t$; as such, *it does not constrain the system's entire future behavior but only the distribution of the output at a particular time* (see Section 2.4). History independence requires indistinguishability of the entire state $(\vec{x}_t, \vec{\theta}_t, \mathrm{supp}_t)$.

- In [Gupta et al., 2021], the stopping time t is determined before the execution begins. In contrast, the History Independence adversary may choose the stopping point adaptively.

- As discussed in Remark 3.11, the indistinguishability condition of [Gupta et al., 2021] is one-sided. Our definition of history independence requires the stronger, symmetric guarantee.

We do not include a formal statement of the equivalence, since by the time one spells out the strengthened model, the equivalence is nearly a tautology. The only difference not listed above is whether one requires indistinguishability of the states s and s^* *conditioned* on the updates $\vec{u}$—as in Eq. (5)—or rather indistinguishability of the *pairs* $(\vec{u}, s)$ and $(\vec{u}, s^*)$—as in Equation 4. These turn out to be equivalent up to changes of parameters, as shown by Lemma A.3.

We conjecture that the protocols of Gupta et al. [2021] satisfy adaptive history independence. A formal proof of this statement would require an appropriately symmetric strengthening of the max-information bound of [Rogers et al., 2016] (Thm 3.1 in ver. 2 on *arXiv*).

4   Deletion-as-confidentiality and Deletion-as-control

We study the relationship between the notion of deletion as confidentiality from Garg et al. [2020] (hereafter "the GGV definition") and our notion of deletion-as-control (Definition 2.3). Similar to Definition 2.3, deletion-as-confidentiality also considers two executions, real and ideal. At the end of the two executions, they compare the *view* of the environment V_E and the state of the controller $\mathsf{state}_\mathsf{C}$. Simplifying away (important but technical) details, the real GGV execution is roughly the same as in deletion-as-control. The ideal GGV execution simply drops all messages between Y and C. Informally, deletion-as-confidentiality requires that E's view and C's final state are indistinguishable in the real and ideal worlds.

In Section 4.1, we define *deletion-as-confidentiality*—the definition closest in spirit to the GGV definition but in our execution model. (We cannot directly compare our definition to GGV as they are defined in different models of interaction, stemming from a need in deletion-as-control to 'sync' the real and ideal executions.) In Section 4.2, we show that for many data subjects Y, deletion-as-confidentiality for Y implies deletion-as-control for Y (Theorem 4.3). Finally, we show that the implication cannot hold for all data subjects due to the differences in the execution models (Theorems 4.4 and 4.5).

4.1 Definition

Below, we define deletion-as-confidentiality by describing how it differs from deletion-as-control. After the definition, we briefly explain how our adaptation of deletion-as-confidentiality differs from that in Garg et al. [2020].

Similar to Definition 2.3, deletion-as-confidentiality also considers two executions—real and ideal. The real execution is almost the same as before, involving the three parties C, E, and Y. We will be interested in the *view* of the environment E, denoted V_E^{real} and the state of the controller C, denoted $\mathsf{state}_\mathsf{C}^{real}$, at the end of the execution. V_E^{real} consists of E's randomness R_E and the transcript of its messages $\tau_\mathsf{E}^{real} = (\vec{\mathsf{q}}_\mathsf{E}^{real}, \vec{\mathsf{a}}_\mathsf{E}^{real})$. (Both the view and state are $\perp$ if the execution does not terminate.) The GGV definition requires that controller's state and the environment's view are indistinguishable in the real world and in the ideal world wherein the data subject never communicates with anybody. This definition inherently requires that C can never reveal any information about one user's data or participation to another data or participation to another. Any information about user Y that C reveals to the E becomes part of E's view in the real world. If Y later requests deletion, this information would not be part of E's view in the ideal world.

The first major difference between GGV and deletion-as-control is the ideal execution. The ideal GGV execution also involves the same three parties C, E, and Y—no dummy party D as in deletion-as-control. The difference between the real and ideal GGV executions is that ideal GGV execution drops all messages between Y and C. This execution results in some view V_E^{ideal} and some state $\mathsf{state}_\mathsf{C}^{ideal}$.

The second major difference between the GGV execution and deletion-as-control concerns how the execution terminates. Recall that in the deletion-as-control game, the real execution ends as soon as C processes Y's first delete message (i.e., the next time C halts). And the ideal execution ends when all of E's queries from the real execution have been replayed by the dummy D. In contrast, the real and ideal GGV executions ends when E sends a special `finish` message to C. At that point, one of two things happens. If Y's last message to C was delete, then the execution immediately terminates. Otherwise, Y sends delete to C, and the execution ends as soon as C processes that message. This difference means that the execution's end time can depend on Y's view in deletion-as-control, but not in GGV. Theorem 4.5 leverages this gap to construct a controller that satisfies GGV but not deletion-as-control. This difference reflects our two fundamentally different approaches to defining the ideal execution. Y has no affect on the GGV ideal execution, and hence E must end the execution. Deletion-as-control introduces a mechanism to "sync" the real and ideal executions, thereby allowing Y to end both executions.

The last major difference between GGV and deletion-as-control is the indistinguishability requirement between the real and ideal executions. The GGV definition requires that the joint distribution of the variables $(V_\mathsf{E}^{real}, \mathsf{state}_\mathsf{C}^{real})$ is close to that of $(V_\mathsf{E}^{ideal}, \mathsf{state}_\mathsf{C}^{ideal})$. That is, the view of the environment E and the state of the controller C should be nearly the same irrespective of whether the subject Y sent its data to the controller and then deleted it, or did not send its data at all. In contrast, deletion-as-control imposes no requirement on V_E. Moreover, GGV only requires indistinguishability for Y's that do not send any messages to E. Without this restriction the definition would be far too limiting. For most useful controllers, Y would be able to tell whether its messages are being delivered to C or not (for example, by receiving an acknowledgement of data receipt), and if it conveys this information to E, E's view will become different in real and ideal executions.

Definition 4.1 ((ϵ, δ) Deletion-as-Confidentiality for Y). *For a data subject* Y, *a controller* C *is* (ϵ, δ)-deletion-as-confidentiality compliant for Y *if for all* E, *in the executions involving* C, E, *and* Y,

$$(V_{\mathsf{E}}^{real}, \mathsf{state}_{\mathsf{C}}^{real}) \stackrel{\epsilon, \delta}{\approx} (V_{\mathsf{E}}^{ideal}, \mathsf{state}_{\mathsf{C}}^{ideal}).$$

Definition 4.2 (Deletion-as-Confidentiality, adapted from Garg et al. [2020]). *Let* $\mathcal{Y}_{\mathsf{silent}}$ *be the set of data subjects* Y *that never send any messages to* E. C *is* (ϵ, δ)-deletion-as-confidentiality compliant *if it is* (ϵ, δ)-deletion-as-confidentiality compliant for all $\mathsf{Y} \in \mathcal{Y}_{\mathsf{silent}}$.

The version of deletion-as-confidentiality in Definition 2.3 differs from the original definition of Garg et al. [2020] in a few important ways. First, [Garg et al., 2020] allows users to request deletion of the information shared in specific interactions between Y and C. To do this, they define protocols that produce protocol-specific deletion tokens. Simplifying, we take deletion of a user's data to be all or nothing. Second, [Garg et al., 2020] allows users to delete many times, whereas we focus on a single deletion. Third, C's randomness tape is not read-once in [Garg et al., 2020]. As explained in Footnote 10, this allows a controller to evade deletion by encrypting its state with its randomness. Finally, the execution model of [Garg et al., 2020] does not have authenticated channels. As they show, authentication is necessary for non-trivial functionalities. In light of this, we chose to build authentication into our execution model (in the form of channel IDs).

4.2 Control vs. Confidentiality

In spirit, deletion-as-confidentiality imposes a stronger indistinguishability requirement than deletion-as-control. The former requires that no information about the deleted data is ever revealed, whereas the latter only requires that the effect of the deleted data is not present *after* the deletion happens. One might thus expect that any C that satisfies Definition 4.2 would also satisfy Definition 2.3. But deletion-as-control captures more general environments and data subjects than deletion-as-confidentiality. First, deletion-as-control allows Y to communicate freely with E, whereas deletion-as-confidentiality does not (i.e., $\mathsf{Y} \in \mathcal{Y}_{\mathsf{silent}}$). Second, deletion-as-control imposes a requirement as soon as Y deletes, whereas deletion-as-confidentiality only requires indistinguishability after E terminates the execution (which may be much later).

Let $\mathcal{Y}_{\mathsf{silent}}$ be as in Definition 4.2 and $\mathcal{Y}_{\mathsf{dummy}}$ be the set of Y that only delete when instructed by E. Let $\mathcal{Y}_{\mathsf{lift}} = \mathcal{Y}_{\mathsf{silent}} \cap \mathcal{Y}_{\mathsf{dummy}}$.

Theorem 4.3. *For any* C *and any* $\mathsf{Y} \in \mathcal{Y}_{\mathsf{lift}}$, *if* C *is* ($\epsilon, \delta$)-deletion-as-confidentiality compliant for Y, *then it is also* (ϵ, δ)-deletion-as-control compliant for Y.

Theorems 4.4 and 4.5 show that the restriction in Theorem 4.3 that $\mathsf{Y} \in \mathcal{Y}_{\mathsf{lift}} = \mathcal{Y}_{\mathsf{dummy}} \cap \mathcal{Y}_{\mathsf{silent}}$ is necessary.

Proof of Theorem 4.3. Fix any controller C that is (ϵ, δ)-deletion-as-confidentiality compliant for subset of Y's that call delete only when instructed to by E. Then, in particular, C is deletion-as-confidentiality compliant for *deterministic* environments E and data subjects Y in this class. By Lemma 2.6, it is sufficient to consider deterministic (E, Y) pairs to prove deletion-as-control compliance.

Fix a deterministic environment E and data subject Y that calls delete only when instructed to by E. Then, by Definition 4.1, we have

$$(V_{\mathsf{E}}^{real}, \mathsf{state}_{\mathsf{C}}^{real}) \stackrel{\epsilon, \delta}{\approx} (V_{\mathsf{E}}^{ideal}, \mathsf{state}_{\mathsf{C}}^{ideal}), \tag{7}$$

where the randomness in the two distributions is only over the controller's randomness R_C. Furthermore, since Y only deletes when E instructs it to, the GGV game and deletion-as-control game result in the same state_C upon termination.

We will construct a simulator Sim that satisfies Definition 2.3 for C, E, Y. Recall that Sim's task is as follows. For some randomly sampled value R_C and the resulting transcript $\tau = (\vec{q}, \vec{a})$ and controller state state_C, the simulator is given R_C, state_C, and $\vec{q}_E$ (the sequence of messages sent by E to C). The simulator is required to produce a R'_C such that when C interacts with the dummy environment $D(\vec{q}_E)$ using randomness R'_C, it results in the same state state_C with probability at least $1 - \delta$; further, $R'_C \overset{\epsilon,\delta}{\approx} R_C$ (with randomness coming from sampling the two).

In our case, we will use the "default simulator" (from Definition 2.5), which works as follows:

$$\mathsf{Sim}(\vec{q}_E, r_C, \text{state}_C) :$$
$$\text{Return } R'_C \sim \mathcal{U}|_{C(\vec{q}_E; R') = \text{state}_C} \text{ if such an } R'_C \text{ exists}$$
$$\text{Otherwise, return } R' \sim \mathcal{U}.$$

If such an R'_C exists, then the final state of C when interacting with $D(\vec{q}_E)$ using randomness R'_C will be equal to state_C. This is because, in C's view, this interaction is identical to the ideal execution of Definition 4.1 with C using randomness R'_C and E, Y. It remains to prove that R'_C exists with probability at least $1 - \delta$ and that the distribution of R'_C is (ϵ, δ)-close to that of R_C.

We can now apply Lemma A.1 to prove both of the statements. Define the following deterministic functions:

$$f(X) = (V_E^{real}(X), \text{state}_C^{real}(X)), \text{ and}$$
$$g(Y) = (V_E^{ideal}(Y), \text{state}_C^{ideal}(Y)),$$

where $V_E^{real}(X)$ denotes the view of E when X is used as the randomness for C in the real world. Let $X = R_C$, $Y = R_C$, and let $P \equiv Q$ be the distribution that both X and Y are sampled from. Then, by Equation (7), $f(X) \overset{\epsilon,\delta}{\approx} g(Y)$. Lastly, the simulator Sim defined samples R'_C from the conditional distribution described in Lemma A.1. Thus, by the lemma, we have that $R'_C \overset{\epsilon,\delta}{\approx} R_C$ and that $(V_E^{real}(X), \text{state}_C^{real}(X)) = (V_E^{ideal}(Y), \text{state}_C^{ideal}(Y))$ with probability at least $1 - \delta$. In particular, we have that $\text{state}_C^{real} = \text{state}_C^{ideal}$ with probability at least $1 - \delta$.

Thus, the simulator Sim for C satisfies both conditions of (ϵ, δ)-deletion-as-control for deterministic (E, Y) and by Lemma 2.6, C also satisfies (ϵ, δ)-deletion-as-control for randomized (E, Y), where Y calls delete only when instructed to by E. Lastly, since Sim does not rely on the code of Y or E, we can use the same simulator for *all* Y's in the class of data subjects that calls delete only when instructed to by E. □

Theorem 4.4. *There exists* C *and* $Y \in \mathcal{Y}_{\text{dummy}} \setminus \mathcal{Y}_{\text{silent}}$ *such that* C *satisfies* $(0,0)$*-deletion-as-confidentiality, but* C *does not satisfy* (ϵ', δ')*-deletion-as-control for* Y *any* $\epsilon' < \infty$, $\delta' < 1$.

Proof. Consider C that implements a write-only SHI dictionary modified as follows. Recall that communications in our execution model are in the form (cID, msg), where cID is a channel ID known only to the communicating parties. If C receives message $\text{msg} = \text{cID}$ with channel ID $\text{cID}' \neq \text{cID}$, then it ignores all subsequent delete messages of the form $(\text{cID}, \text{delete})$. In words, the user communicating on channel cID' can instruct C to ignore deletion requests from the user communicating on channel cID.

It is easy to see that C satisfies $(0,0)$-deletion-as-confidentiality. First, V_E is identical in the real and ideal executions because the dictionary is write-only and C sends no messages. Second state_C is identical in the real and ideal executions because the dictionary is SHI.

Recall $\mathcal{Y}_{\mathsf{dummy}} \setminus \mathcal{Y}_{\mathsf{silent}}$ is the set of data subjects Y that send messages to E and only delete when instructed by E. Consider Y that inserts itself into the dictionary, sends its cID_Y to E, and then calls delete. Consider E that sends cID_Y to C as soon as it receives it from Y. By construction, state_C contains Y's data at the end of the real execution, but state'_C in the dummy execution does not. Thus, if the implementation $\mathcal{D}$ is perfectly correct, there are no R_C, R'_C such that $\mathsf{state}_C = \mathsf{state}'_C$. This violates Definition 2.3 for any $\delta < 1$ and any $\epsilon < \infty$. $\qquad\square$

Theorem 4.5. *For any $\delta > 0$, there exists C and $Y \in \mathcal{Y}_{\mathsf{silent}}$ such that (i) C satisfies $(0,\delta)$-deletion-as-confidentiality for Y, and (ii) C does not satisfy (ϵ', δ')-deletion-as-control for Y for any $\epsilon < \infty$, $\delta' < 1$.*

Proof of Theorem 4.5. Let $T > 1/\delta$. Consider C implementing a write-only SHI dictionary $\mathcal{D}$ modified as follows. Upon receiving the first message from a channel cID, sample $t_{\mathsf{cID}} \in [1, T]$ uniformly at random, store it in the dictionary with key cID, and reply with t_{cID}. Upon receiving delete from cID, check the current size of the dictionary. If $|\mathcal{D}| = t_{\mathsf{cID}}$, remove cID the next time C is activated. Otherwise, remove cID immediately.

Consider Y that (i) sends $\mathsf{insert}()$ to C on its first activation, (ii) receives t_{cID_Y} in response, and (iii) sends delete to C after $t_{\mathsf{cID}_Y} - 1$ additional activations.

Claim 4.6. *C^* satisfies $(0,\delta)$-deletion-as-confidentiality for Y^*.*

Proof. Fix E and $Y \in \mathcal{Y}_{\mathsf{silent}}$. Because $\mathcal{D}$ is write-only, V_E is exactly the same in the real and ideal deletion-as-confidentiality executions. That view consists of k calls to $\mathsf{insert}()$ sent to C and the responses $t_1, \ldots, t_k$.

In contrast, state_C^{real} may differ at the end of the real and ideal executions. But only if $|\mathcal{D}| = t_{\mathsf{cID}_Y}$ when Y calls delete. In particular, $\mathsf{state}_C^{real} \neq \mathsf{state}_C^{ideal}$ requires E to make exactly $t_{\mathsf{cID}_Y} - 1$ calls to $C.\mathsf{insert}()$ over the course of the execution. But E has no information about t_{cID_Y} besides its prior distribution. From C it receives only the t_i, which are independent. From Y it receives nothing, as $Y \in \mathcal{Y}_{\mathsf{silent}}$. Therefore, the probability that $\mathsf{state}_C^{real} \neq \mathsf{state}_C^{ideal}$ is at most $1/T$.

$$d_{TV}\left((V_E^{real}, \mathsf{state}_C^{real}), (V_E^{ideal}, \mathsf{state}_C^{ideal}) \right)$$
$$= d_{TV}\left(\mathsf{state}_C^{real}, \mathsf{state}_C^{ideal} \right) < \frac{1}{T}$$

Hence $(V_E^{real}, \mathsf{state}_C^{real}) \overset{0, \frac{1}{T}}{\approx} (V_E^{ideal}, \mathsf{state}_C^{ideal})$. $\qquad\square$

Claim 4.7. *C^* does not satisfy (ϵ', δ')-deletion-as-control for Y^* for any $\epsilon < \infty$, $\delta' < 1$.*

Proof. Consider E that repeats two operations: (i) activate Y, and (ii) send $\mathsf{insert}()$ to C along a new channel. The execution $\langle C(R_C), Y, E \rangle$ always ends with Y calling delete when $|\mathcal{D}| = t_{\mathsf{cID}_Y}$. By construction, $\mathcal{D}$ does not remove Y from $\mathcal{D}$. So at the end of the real execution, state_C contains the dictionary $\mathcal{D}$ that includes Y.

In contrast, the dummy execution $\langle C(R'_C), D(\vec{q}_E) \rangle$ includes no calls to $\mathsf{insert}()$ from Y. So at the end of the dummy execution, state'_C contains the dictionary $\mathcal{D}'$ that does not include Y. Thus,

if the implementation $\mathcal{D}$ is perfectly correct, there are no R_C, R'_C such that $\mathsf{state}_\mathsf{C} = \mathsf{state}'_\mathsf{C}$. This violates Definition 2.3 for any $\delta < 1$ and any $\epsilon < \infty$. $\qquad\square$

This completes the proof of Theorem 4.5. $\qquad\square$

5 Differential Privacy and Deletion-as-control

This section describes two ways of compiling (ϵ, δ)-differentially private mechanisms $\mathcal{M}$ into controllers satisfying (ϵ, δ)-deletion-as-compliance. The first applies to DP mechanisms that are run in a batch setting on a single, centralized dataset: data summarization, query release, or DP-SGD, for example (Section 5.1). The second applies to mechanisms satisfying pan-privacy under continual release (Section 5.3). Along the way we define non-adaptive event-level pan privacy (one intrusion) with continual release, first defined by Chan et al. [2011], Dwork et al. [2010b], and an adaptive variant, building on the adaptive continual release definition of Jain et al. [2021] (Section 5.2). Both of our compilers make use of a SHI dictionary $\mathcal{D}$ (Section 3.1.1).

There is a strong intuitive connection between (approximate) history independence and deletion-as-control, as illustrated by the results of the previous section. This intuition is so strong that many prior works on machine unlearning *essentially equate* deletion with history independence (Section 3.4).

However, *the examples of this section show that our notion of deletion-as-control is much broader than history independence.* For instance, consider a controller as follows. At some time t_0 the controller computes a differentially-private approximation out_{t_0} to the current number of users in the data set, and the controller stores out_{t_0} and makes it available at all later times. Intuitively, this controller does not satisfy any version of history independence: even if every user request deletion at time $t_0 + 1$, the stored out_{t_0} is unchanged—making this history easy to distinguish from one where the all users deleted at time $t_0 - 1$. It could, however, still satisfy deletion-as-control.

Keeping time Throughout this section we consider systems with a global clock. This allows for controllers that publish the number of weekly active users, say. For simplicity and generality, we allow the environment to control time. Specifically, we introduce a special query tick that only E can send to C to increment the clock.

5.1 Batch differential privacy

Let $\mathcal{M} : \mathcal{D} \mapsto \mathcal{M}(\mathcal{D})$ be a non-interactive differentially private mechanism $\mathcal{M}$.

Algorithm 1 defines a simple controller $\mathsf{C}^{\mathsf{batch}}_\mathcal{M}$ that satisfies deletion-as-control.

It works in three phases: before tick, during tick, and after tick. At the beginning, $\mathsf{C}^{\mathsf{batch}}_\mathcal{M}$ populates a dataset $\mathcal{D}$ stored as a SHI dictionary from its input stream, returning $\bot$ in response to every query. When it receives the tick, it evaluates $\mathcal{M}(\mathcal{D})$, stores the result as out, and erases the dictionary $\mathcal{D}$. For all future queries, $\mathsf{C}^{\mathsf{batch}}_\mathcal{M}$ simply returns out. We assume for simplicity that the mechanism $\mathcal{M}$ is a function only of the logical contents of $\mathcal{D}$, and is independent of its memory representation.

Proposition 5.1.

    1. If $\mathcal{M}$ is (ϵ, δ)-DP, then $\mathsf{C}^{\mathsf{batch}}_\mathcal{M}$ satisfies (ϵ, δ)-deletion-as-control.

Algorithm 1 $C_{\mathcal{M}}^{\mathsf{batch}}$ satifying deletion-as-control from batch DP mechanism $\mathcal{M}$

 1: **procedure** INITIALIZE   ▷ Run on first activation
 2:      Initialize a SHI dictionary $\mathcal{D}$
 3:      Initialize $\mathsf{out} = \bot$ ▷ $\bot$ is never returned by $\mathcal{M}$

 4: **procedure** ACTIVATE($\mathsf{op}(id)$)
 5:     **if** ($\mathsf{op} \neq \mathsf{tick}$) $\wedge$ ($\mathsf{out} = \bot$) **then**
 6:         // before tick
 7:         $\mathcal{D}.\mathsf{op}(id)$ ▷ Insert/modify/delete item
 8:         Return $\bot$
 9:     **else if** ($\mathsf{op} = \mathsf{tick}$) **then**
10:         // during tick
11:         Set $\mathsf{out} \leftarrow \mathcal{M}(\mathcal{D})$
12:         Delete $\mathcal{D}$ ▷ eg: iteratively call $\mathcal{D}.\mathsf{delete}(id)$
13:         Return out
14:     **else**
15:         // after tick
16:         Return out

---

2. *For any $\epsilon > 0$, suppose $\mathcal{M}$ is the Laplace mechanism with parameter ϵ applied to a count of the number of record in its input. Then $C_{\mathcal{M}}^{\mathsf{batch}}$ satisfies (ϵ, δ)-deletion-as-control but is not (ϵ', δ')-HI for any $\epsilon' < \infty$ and $\delta' < 1$.*

The second part of the proposition really applies to any DP mechanism that releases useful information about its inputs—the argument relies just on the fact that $\mathcal{M}$ acts differently on the empty data set than it does on a data set with many records. In particular, the DP Machine learning example from Section 1.1, which trains a model using DP-SGD, satisfies deletion-as-control.

Corollary 5.2. *The (ϵ, δ)-DP Machine Learning touchstone controller (Section 1.1) satisfies (ϵ, δ)-deletion-as-control.*

Proof of Proposition 5.1. (1) To reduce clutter, let $\mathsf{C} = C_{\mathcal{M}}^{\mathsf{batch}}$ throughout this proof. The simulator is described in Algorithm 2. We must show that for all deterministic E and Y (i) $R_{\mathsf{C}}' \overset{\epsilon,\delta}{\approx} R_{\mathsf{C}}$, and (ii) $(\mathcal{D}', \mathsf{out}') = (\mathcal{D}, \mathsf{out})$ with probability at least $1 - \delta$.

Fix deterministic E and Y. Until a tick query, C's output is determinstically $\bot$. Hence, all queries that precede tick are fixed in advance—including whether the sequence contains a tick at all. If there is no tick, then the whole sequence is fixed.

We consider two cases depending on whether the real sequence of queries $\vec{\mathsf{q}}$ includes tick. Suppose there is no tick in $\vec{\mathsf{q}}$. By construction, C's state $\mathsf{out} = \bot$ and $\mathcal{D}$ may be non-empty. $\mathsf{Sim}(\vec{\mathsf{q}}_{\mathsf{E}}, R_{\mathsf{C}}, \mathsf{out})$ outputs $R_{\mathsf{C}}' = R_{\mathsf{C}}$. Consider $\mathcal{D}$ in the real execution resulting from queries $\vec{\mathsf{q}}$ and randomness R_{C}, and $\mathcal{D}'$ in the ideal execution resulting from $\vec{\mathsf{q}}_{\mathsf{E}}$ and R_{C}'. The only difference between $\vec{\mathsf{q}}$ and $\vec{\mathsf{q}}_{\mathsf{E}}$ is Y's queries. Because $\vec{\mathsf{q}}$ ends with delete from Y, and dictionaries support logical deletion, the logical contents of $\mathcal{D}$ and $\mathcal{D}'$ are identical. Applying Theorem 1 from Hartline et al. [2005][12], $\mathcal{D}' = \mathcal{D}$. In this case, the simulation is perfect.

---

[12]The theorem states that reversible $(0,0)$-SHI data structures have canonical representations, up to initial randomness.

Suppose there is a tick in $\vec{q}$. By construction, C's state $\mathcal{D}$ is empty and out $\neq \perp$. The simulator samples R'_C uniformly conditioned on out$'$ = out (if such R'_C exists, uniformly otherwise). Consider $\mathcal{D}$ in the real execution resulting from queries $\vec{q}$, and $\mathcal{D}'$ in the ideal execution resulting from $\vec{q}_E$. $\mathcal{D}$ and $\mathcal{D}'$ represent fixed neighboring datasets: differing only on Y. Take $f(R) = \mathcal{M}(\mathcal{D}; R)$ and $g(R) = \mathcal{M}(\mathcal{D}'; R)$. Because $\mathcal{M}$ is (ϵ, δ)-DP, $f(R) \stackrel{\epsilon, \delta}{\approx} g(R)$. The simulator samples R'_C as in the statement of Lemma A.1. Applying that lemma completes the proof.

(2) When $\mathcal{M}$ is the Laplace mechanism, $C^{\mathsf{batch}}_{\mathcal{M}}$ satisfies deletion as control by part (1). To see why it does not satisfy even weak, nonadaptive history independence, consider a (nonadaptive) sequence σ_n of n distinct insertions, followed by a tick, followed by n corresponding deletions. The minimal equivalent sequence $\sigma^* = [\sigma_n]$ consists only of the tick. The value out generated at the tick will follow $n + \mathsf{Lap}(1/\epsilon)$. If we consider the event $E_n = \{\text{out} < n/2\}$, then E_n occurs with probability at least $1 - \exp(\epsilon n/2)$ in the real world. However, if we were to run a dummy execution of the controller $C^{\mathsf{batch}}_{\mathcal{M}}$ with σ^*, the probability of E_n would be at most $\exp(\epsilon n)$, regardless of whether we use the same randomness R as in the real world or a fresh string R'. Suppose, for contradiction, that $C^{\mathsf{batch}}_{\mathcal{M}}$ satisfies (ϵ', δ')-HI. Then

$$1 - \exp(-\epsilon n) \leq \Pr_{\text{real}}(E_n) \leq \exp(\epsilon') \cdot \Pr_{\text{dummy}}(E_n)$$
$$\leq \exp(\epsilon' - \epsilon n) + \delta'.$$

For any $\epsilon' > 0$ and $\delta' < 1$, we can get a contradiction by choosing $n > \frac{1}{\epsilon} \ln\left(\frac{e^{\epsilon'} + 1}{1 - \delta'}\right)$. $\qquad \square$

Algorithm 2 Sim for $C = C^{\mathsf{batch}}_{\mathcal{M}}$

1: **input** R_C, $\vec{q}_E$, out
2: **if** tick $\notin \vec{q}_E$ **then**
3: $\quad R'_C \leftarrow R_C$
4: **else**
5: $\quad$ Run the default Sim (Def. 2.5)
   $\quad\quad$ Namely, sample R'_C uniformly conditioned on out$'$ = out, where out$'$ = $C(\vec{q}_E; R'_C)$ is the output of C in the ideal execution (with queries $\vec{q}_E$). If no such R'_C exists, sample R'_C uniformly at random.
6: Return R'_C

5.2 Adaptive pan-privacy with continual release

In this section, we formalize the definition of adaptive pan-privacy with continual release (against a single intrusion), extending the non-adaptive versions originally defined in Chan et al. [2011], Dwork et al. [2010b]. The continual release setting concerns an *online controller* that processes a stream of elements and produces outputs at regular time intervals. We say two streams are (event-level) *neighboring* if they differ in at most one stream item. We consider privacy against an adversary that can adaptively choose neighboring streams (one of which is processed by the online mechanism) and inspect the internal state of the online algorithm once. This adaptive game defines the adversary's view. Informally, adaptive pan-privacy under continual release requires that, for all adaptive adversaries, the adversary's view for the two streams is (ϵ, δ)-close.

Preliminaries A *stream* $\mathbf{x} \in (\{\mathsf{regular}, \mathsf{intrusion}, \mathsf{challenge}\} \times \mathcal{X})^*$ consists of elements of the form $(\mathsf{code}, \mathsf{op}(id))$. The *control code* $\mathsf{code} \in \{\mathsf{regular}, \mathsf{intrusion}, \mathsf{challenge}\}$ controls the execution of the pan-privacy continual release security game and is never seen by the mechanism itself (see Algorithm 3). The *data operation* $\mathsf{op}(id) \in \mathcal{X}$ is sent to the pan-private mechanism. The universe of data operations $\mathcal{X}$ includes a special tick operation (which requires no id tag) which indicates to the mechanism that one time step has passed.

Definition 5.3 (Online Algorithm). *An online algorithm $\mathcal{M}$ is initialized with a time horizon T and is defined by an internal algorithm I. Algorithm $\mathcal{M}$ processes a stream of elements through repeated application of $I : (\mathsf{op}(id), \mathsf{state}_{\mathcal{M}}) \mapsto (\mathsf{state}'_{\mathcal{M}}, \mathsf{out})$, which (with randomness) maps a stream element and the current internal state to a new internal state and an output (which may consist of $\perp$). For simplicity, we only consider online algorithms that produce outputs when a tick occurs. That is, $\mathsf{op}(id) \neq \mathsf{tick}$ implies that $\mathsf{out} = \perp$. The internal state of $\mathcal{M}$ includes a clock which counts the number of ticks that have previously occurred in the stream. When $\mathcal{M}$'s internal number of clock ticks reaches T, it stops processing new stream elements and always outputs $\perp$.*

Any stream $\mathbf{x}$ with a single instance of $\mathsf{challenge}$ naturally gives rise to two neighboring sequences of data operations $\mathbf{x}^{(\mathtt{in})}$ and $\mathbf{x}^{(\mathtt{out})}$ which include or exclude the challenge operation, respectively.

Definition 5.4 (Event-level neighboring sequences). *For stream $\mathbf{x}$ containing a single instance of $\mathsf{challenge}$, we define two event-level neighboring sequences $\mathbf{x}^{(\mathtt{in})}$ and $\mathbf{x}^{(\mathtt{out})}$ as follows:*

$$\mathbf{x}^{(\mathtt{in})} := \{\mathsf{op}(id) \mid (\mathsf{regular}, \mathsf{op}(id)) \in \mathbf{x} \text{ or}$$
$$(\mathsf{challenge}, \mathsf{op}(id)) \in \mathbf{x}\}$$
$$\mathbf{x}^{(\mathtt{out})} := \{\mathsf{op}(id) \mid (\mathsf{regular}, \mathsf{op}(id)) \in \mathbf{x}\}.$$

Event-level pan privacy We formalize the definition of non-adaptive pan-privacy with continual release (against a single intrusion), based on [Chan et al., 2011, Dwork et al., 2010b]. Although our version of the definition is identical in spirit to previous definitions, our setting has two complications which are notationally challenging but not conceptually so. First, the algorithm $\mathcal{M}$ may process any number of sequence items before producing an output. Second, we consider sequences as neighboring if they differ by insertion or deletion of items, which causes a discrepancy in sequence indices. We have chosen to write the non-adaptive definition so that it is clearly a special case of the adaptive version, defined later in the section.

Let $\mathbf{x}$ be a stream with a single $\mathsf{intrusion}$. For online algorithm $\mathcal{M}$ and $\mathsf{side} \in \{\mathsf{in}, \mathsf{out}\}$, we denote by $V_{\mathcal{M},\mathbf{x}}^{(\mathsf{side})} = (\mathsf{state}, \overrightarrow{\mathsf{out}})$ the view of an attacker who sees (i) the sequence of outputs $\overrightarrow{\mathsf{out}}$ produced by $\mathcal{M}(\mathbf{x}^{(\mathsf{side})})$, and (ii) a snapshot of $\mathsf{state}_{\mathcal{M}}$ of the internal state of $\mathcal{M}$ at the time indicated by $\mathsf{intrusion}$ in $\mathbf{x}$.

Definition 5.5 (Non-Adaptive Event-Level PP with CR [Chan et al., 2011, Dwork et al., 2010b]). *An online algorithm $\mathcal{M}$ satisfies (ϵ, δ)-non-adaptive event-level pan-privacy in the continual release model if for all streams $\mathbf{x}$ with at most one $\mathsf{challenge}$ and at most one $\mathsf{intrusion}$:*

$$V_{\mathcal{M},\mathbf{x}}^{(\mathtt{in})} \stackrel{\epsilon, \delta}{\approx} V_{\mathcal{M},\mathbf{x}}^{(\mathtt{out})}.$$

Adaptive pan privacy We define adaptive event-level pan privacy with continual release against a single intrusion, following the continual-release model of Jain et al. [2021] (which did not allow for intrusions). In this game, the adversary A decides when to increment timesteps (i.e., by querying (regular, tick)), when to issue a single challenge query, when to intrude, and when to terminate the game (i.e., using code = ⊥). Depending on whether side is in or out, $\mathcal{M}$ is either given the challenge query or not. Thus, the neighboring data streams we consider differ by the insertion or deletion of a single query. We require that for the challenge query, op ≠ tick. We assume for simplicity that $\mathcal{M}(\mathsf{op}(id))$ produces no output unless op = tick.

Algorithm 3 Privacy game $\Pi_{\mathcal{M},\mathsf{A}}^{(\mathtt{side})}$ for the adaptive event-level pan privacy with continual release model. The game halts if any assertion fails.

1: **procedure** RUNGAME(time horizon $T \in \mathbb{N}$)
2:      // initialize global variables
3:      $\mathcal{M}$.INITIALIZE(T)
4:      intruded ← false
5:      challenged ← false

6:      // interaction between A and $\mathcal{M}$
7:      out ← ⊥
8:      **repeat**
9:          $(\mathsf{code}, \mathsf{op}(id)) \leftarrow \mathsf{A}(out)$
10:          out ← ACTIVATE(code, op(id))
11:      **until** code = ⊥

12: **procedure** ACTIVATE(code, op(id))
13:      **if** code = intrusion **then**
14:          **Assert:** intruded = false
15:          intruded ← true
16:          **return** state$_{\mathcal{M}}$

17:      **if** code = regular **then**
18:          **return** $\mathcal{M}(\mathsf{op}(id))$ ▷ If op ≠ tick, then $\mathcal{M}(\mathsf{op}(id)) = \bot$

19:      **if** code = challenge **then**
20:          **Assert:** (challenged = false) and (op ≠ tick)
21:          challenged ← true
22:          **if** side = in **then**
23:              Run $\mathcal{M}(\mathsf{op}(id))$ ▷ By assumption op(id) ≠ tick and $\mathcal{M}(\mathsf{op}(id)) = \bot$
24:          **return** ⊥

25:      **else**
26:          **return** ⊥

Definition 5.6. *We denote by $V_{\mathcal{M},\mathsf{A}}^{(\mathtt{side})}$ the view of* A *in the game* $\Pi_{\mathcal{M},\mathsf{A}}^{(\mathtt{side})}$, *consisting of* A*'s internal*

randomness and the transcript of all messages A *sends and receives.*[13]

Definition 5.7 (Adaptive Event-Level PP with CR)**.** *A mechanism $\mathcal{M}$ is (ϵ, δ)-DP in the adaptive event-level pan-privacy with continual release model if for all adversaries* A,

$$V^{(\mathtt{in})}_{\mathcal{M},\mathsf{A}} \overset{\epsilon,\delta}{\approx} V^{(\mathtt{out})}_{\mathcal{M},\mathsf{A}}.$$

Remark 5.8. *The definition above makes sense in a setting where the state* $\mathsf{state}_{\mathcal{M}}$ *does not have information about the mechanism's future random coins. If the adversary could deduce future randomness, then security would be unachievable: an adversary could intrude at time 0, learn all the algorithm's randomness, and then easily tell whether the mechanism had received x^* or not.*

An alternative, which does allow for the adversary learning future randomness, is to require that the challenge is possible only before intrusion (specifically, Line 20 additionally assert that $\mathsf{intruded} = \mathsf{false}$*). This definition is satisfiable by interesting mechanisms—for example, a mechanism that initializes a counter with Laplace noise, then adds subsequent stream elements x in $[0,1]$ to the counter, and outputs the final value on the first clock tick, satisfies the weaker, alternative definition.*

5.3   From pan-privacy to deletion-as-control

Algorithm 4 describes a general transformation from an event-level pan private algorithm to a controller satisfying deletion-as-control. It uses as a building block a strongly history independent dictionary $\mathcal{D}$ (Section 3.1.1). For each *id*, the controller passes the first operations $\mathsf{op}(id)$ to the underlying pan private mechanism $\mathcal{M}$. It also passes all tick, which produce output. $\mathsf{C}^{\mathsf{pp}}_{\mathcal{M}}$ uses the dictionary $\mathcal{D}$ to check whether a given *id* has already issued a query to $\mathcal{M}$. To delete, *id* is removed from $\mathcal{D}$ but $\mathcal{M}$ is unaffected. (Note that a deleted user can then issue a new query to $\mathcal{M}$; we make no guarantees for such users.)

We assume that the randomness tape of $\mathsf{C}^{\mathsf{pp}}_{\mathcal{M}}$ defined in Algorithm 4 is partitioned into two independent strings: $R_{\mathcal{M}}$ to be used by the mechanism $\mathcal{M}$, and $R_{\mathcal{D}}$ to be used by $\mathcal{D}$. Additionally, we assume that the state of $\mathsf{C}^{\mathsf{pp}}_{\mathcal{M}}$ can be partitioned into one part containing $\mathsf{state}_{\mathcal{M}}$ and another part with $\mathsf{state}_{\mathcal{D}}$. Beyond from $\mathsf{state}_{\mathcal{M}}$ and $\mathsf{state}_{\mathcal{D}}$, the controller $\mathsf{C}^{\mathsf{pp}}_{\mathcal{M}}$ uses only ephemeral state and no additional randomness.

---

[13]One could instead define A's view as its internal state at the end of the game. Our version of the view contains enough information to compute that internal state and is simpler to work with.

Algorithm 4 $C_{\mathcal{M}}^{pp}$ Deletion Compliant Controller Based on Event-Level Pan-Private Algorithm

1: **procedure** INITIALIZE(Time horizon $T \in \mathbb{N}$, privacy parameters ϵ, δ, query access to event-level pan-private algorithm $\mathcal{M}$)
2:     Initialize $\mathcal{M}$ with parameters (ϵ, δ, T)
3:     Initialize an empty SHI dictionary $\mathcal{D}$

4: **procedure** ACTIVATE(op(id))
5:     **if** op = tick **then**   ▷ By assumption Y can never query tick
6:         **return** $\mathcal{M}$(tick)
7:     **if** op = delete **then**
8:         $\mathcal{D}$.delete(id)   ▷ If $id \notin \mathcal{D}$, nothing happens.
9:         **return** $\perp$
10:    **if** $id \in \mathcal{D}$ **then**
11:       **return** $\perp$
12:    **if** $id \notin \mathcal{D}$ **then**
13:       $\mathcal{D}$.insert(id)
14:       $\mathcal{M}$(op(id))
15:       **return** $\perp$ ▷ If op $\neq$ tick, then $\mathcal{M}$(op(id)) $= \perp$

Theorem 5.9. *If a controller $\mathcal{M}$ satisfies (ϵ, δ)-adaptive event-level pan-privacy with continual-release, then the composed controller $C_{\mathcal{M}}^{pp}$ as described in Algorithm 4 satisfies (ϵ, δ)-deletion-as-control.*

We will prove the statement by showing that $C_{\mathcal{M}}^{pp}$ satisfies (ϵ, δ)-deletion-as-confidentiality (Lemma 5.10) and then apply Theorem 4.3 to prove that $C_{\mathcal{M}}^{pp}$ satisfies (ϵ, δ)-deletion-as-control.

Lemma 5.10. *If a controller $\mathcal{M}$ satisfies (ϵ, δ)-adaptive event-level pan-privacy with continual release, then the composed controller $C_{\mathcal{M}}^{pp}$ as described in Algorithm 4 satisfies (ϵ, δ)-deletion-as-confidentiality (Definition 4.1).*

Proof. To reduce clutter, let $C = C_{\mathcal{M}}^{pp}$ throughout this proof. Fix a controller $\mathcal{M}$ that satisfies (ϵ, δ)-adaptive event-level pan-privacy, and fix E and Y in the deletion-as-confidentiality game. We must show that $(V_E^{real}, \text{state}_C^{real}) \overset{\epsilon, \delta}{\approx} (V_E^{ideal}, \text{state}_C)$ using the pan-privacy hypothesis. Recall that in the deletion-as-confidentiality game, Y does not send any messages to E.

Below, we construct an adversary $A = A_{E,Y}$ for the adaptive pan-privacy game of $\mathcal{M}$ (see Figure 6). After adaptively querying $\mathcal{M}$ (as in Algorithm 3), A produces output Z. We will argue two key properties: First, if side = in, then $Z = Z^{(in)}$ will be distributed as $(V_E^{real}, \text{state}_C^{real})$. The latter is the view of the environment and state of the controller in the real confidentiality execution (Definition 4.1). Second, if side = out, then $Z^{(out)}$ will be distributed as $(V_E^{ideal}, \text{state}_C^{ideal})$, the corresponding quantities in the ideal confidentiality execution. The adaptive pan-privacy of $\mathcal{M}$ implies that A's views with side $\in \{\text{in}, \text{out}\}$ are (ϵ, δ)-indistinguishable. Post-processing implies that $Z^{(in)} \overset{\epsilon, \delta}{\approx} Z^{(out)}$, completing the proof.

Adversary A emulates a real GGV execution among E, Y, and C, passing messages among them and performing each party's computations as defined in Section 4 and depicted in Figure 6. To emulate E and Y, A simply runs their code. To emulate C, A maintains an internal SHI dictionary

Figure 6: How A emulates a real GGV execution among E, Y, and $\mathsf{C}^{\mathsf{pp}}_{\mathcal{M}}$.

$\mathcal{D}$ as in Algorithm 4 (using state $\mathsf{state}_{\mathcal{D}}$) but does not emulate $\mathcal{M}$ directly. Instead, A makes a queries in the pan-privacy game by calling ACTIVATE in Algorithm 3 as follows. Let id_{Y} be Y's id. When the emulated parties E or Y sends message (id, op) intended for emulated C, A does the following:

1. If op is a special message from E to terminate the alive phase of execution: Run $\mathcal{D}.\mathsf{delete}(id_{\mathsf{Y}})$, call
   ACTIVATE$(\mathsf{intrusion}, \perp)$. Upon receiving $\mathsf{state}_{\mathcal{M}}$, end the GGV execution and output $Z = (V_{\mathsf{E}}, \mathsf{state}_{\mathsf{C}})$ where V_{E} is the view of the emulated E and $\mathsf{state}_{\mathsf{C}} = (\mathsf{state}_{\mathcal{M}} \| \mathsf{state}_{\mathcal{D}})$.

2. Else if $\mathsf{op} = \mathsf{delete}$: Run $\mathcal{D}.\mathsf{delete}(id)$ and send response $\perp$ to party id (on behalf of the emulated C).

3. Else if $id \in \mathcal{D}$: Send response $\perp$ to party id (on behalf of the emulated C).

4. Else if $id \neq id_{\mathsf{Y}}$: $\mathcal{D}.\mathsf{insert}(id)$, ACTIVATE$(\mathsf{regular}, \mathsf{op}(id))$, and forward the response from $\mathcal{M}$ to party id (on behalf of the emulated C).

5. Else: Call ACTIVATE$(\mathsf{challenge}, \mathsf{op}(id))$ and forward the response to Y (on behalf of the emulated C).

Let μ^{GGV} be the sequence of queries to $\mathcal{M}$ made by the emulated C (all calls to Line 14 of Algorithm 4). Each query in μ^{GGV} results in a single call to ACTIVATE. These in turn may result in a query to $\mathcal{M}$ on Lines 18 and 23 of Algorithm 3. But, due to the pan-privacy game some of these queries may not actually reach $\mathcal{M}$: if $\mathsf{side} = \mathsf{out}$ (Line 22), or if an assertion is violated (Lines 14 or 20). Let $\mu^{PP,\mathsf{side}}$ be the sequence of queries that actually reach $\mathcal{M}$ for $\mathsf{side} \in \{\mathsf{in}, \mathsf{out}\}$ (i.e., all calls to Lines 18 and 23).

Claim 5.11. *Consider an execution of the adversary* A *above in the pan-privacy game with mechanism* $\mathcal{M}$. *Then, with probability 1:*

1. $\mu^{PP,\mathsf{in}} = \mu^{GGV,real}$, *and*

2. $\mu^{PP,\mathsf{out}} = \mu^{GGV,ideal}$.

Proof. Part 1 (side = in)*:* As explained before the claim, every query to $\mathcal{M}$ made by the emulated C corresponds to a call to ACTIVATE (and vice-versa). Every call to ACTIVATE results in a query in $\mu^{PP,\text{in}}$ unless side = out or one of the assertions in Algorithm 3 (Lines 14 and 20) is violated. Because side = in, it suffices to show that the assertions are never violated. One assertion requires that intrusion is only called once (Line 14). This is by the construction of A above.

The second assertion requires that challenge is called only once and only with op $\neq$ tick (Line 20). This follows three properties of μ^{GGV}. First, challenge is only called when Y makes a non-delete query. Second, Y cannot make tick queries. Finally, μ^{GGV} contains at most one non-delete query from Y. This follows automatically from the restrictions on Y in the deletion-as-confidentiality game; specifically, after Y makes a delete query to C, it may not send any additional messages to C.

Part 2 (side = out)*:* By construction of A, Y's queries in μ^{GGV} causes A to send a challenge query in the pan-privacy game. The only difference from Part 1 is that these queries are never sent to $\mathcal{M}$ in the pan-privacy game (Line 22 of Algorithm 3). And, although the emulated Y receives $\perp$ responses from A (on behalf of C)–which it would not get in the ideal world–Y can not send messages to E and so can not influence the view of E at all. □

Next we show that $Z^{(\text{in})}$ is distributed as $(V_\mathsf{E}^{real}, \mathsf{state}_\mathsf{C}^{real})$. Consider a real-world GGV execution where all parties use the same randomness as in A's emulated execution. If side = in, the view of the real-world GGV environment $V_\mathsf{E}^{GGV,real}$ is identical to the view of the emulated environment $V_\mathsf{E}^{PP,\text{in}}$. From E's view, the only difference between the two executions is how $\mathcal{M}$ is implemented and queried. By Part 1 of Claim 5.11, the queries processed by $\mathcal{M}$ are identical in the two executions. Hence E's views of the two executions are identical. Likewise, the states $\mathsf{state}_\mathsf{C} = (\mathsf{state}_\mathcal{D} \| \mathsf{state}_\mathcal{M})$ are identical in the two executions.

Finally we show that $Z^{(\text{out})}$ is distributed as $(V_\mathsf{E}^{ideal}, \mathsf{state}_\mathsf{C}^{ideal})$. Consider an ideal-world GGV execution where all parties use the same randomness as in A's emulated execution. In the ideal GGV execution, Y cannot send messages to C, whereas in the emulated execution it can; however, by part 2 of Claim 5.11, the queries processed by $\mathcal{M}$ are identical in the two executions. Hence E's views of the two executions are identical. Likewise, the states $\mathsf{state}_\mathsf{C} = (\mathsf{state}_\mathcal{D} \| \mathsf{state}_\mathcal{M})$ are identical in the two executions.

Lastly, $Z^{(\text{side})}$ is a post-processing of $V_{\mathsf{A},\mathcal{M}}^{(\text{side})}$ in the adaptive pan-privacy game. Hence, by event-level (ϵ, δ)-adaptive-pan-privacy with continual release, we have $Z^{(\text{in})} \overset{\epsilon,\delta}{\approx} Z^{(\text{out})}$. Finally, since the dictionary $\mathcal{D}$ is SHI, the state $\mathsf{state}_\mathcal{D}$ in the emulation is identically distributed to $\mathsf{state}_\mathcal{D}$ in the GGV real and ideal worlds (since in all cases Y's removal from $\mathcal{D}$ is undetectable). This implies that $(V_\mathsf{E}^{real}, \mathsf{state}_\mathsf{C}^{real}) \overset{\epsilon,\delta}{\approx} (V_\mathsf{E}^{ideal}, \mathsf{state}_\mathsf{C}^{ideal})$, completing the proof. □

Proof of Theorem 5.9. Our proof combines Lemma 5.10 and Theorem 4.3. To reduce clutter, let $\mathsf{C} = \mathsf{C}_\mathcal{M}^{\mathsf{PP}}$ throughout this proof. Fix an (ϵ, δ)-adaptive event-level pan-private controller $\mathcal{M}$ with continual release. Then, by Lemma 5.10, the controller C described in Algorithm 4 satisfies (ϵ, δ)-deletion-as-confidentiality. Recall that deletion-as-confidentiality only considers the subset of data subjects $\mathcal{Y}_{\mathsf{lift}}$ satisfing two requirements. First, Y does not send messages to E. Second, Y only delete's when instructed to by E. If we restrict to this class $\mathcal{Y}_{\mathsf{lift}}$ of data subjects, then by Theorem 4.3, C is (ϵ, δ)-deletion-as-control compliant for $\mathcal{Y}_{\mathsf{lift}}$.

What remains to argue is that for C specifically, we can restrict to $\mathcal{Y}_{\mathsf{lift}}$ without loss of generality (using the same simulator Sim as for Y $\in \mathcal{Y}_{\mathsf{lift}}$). It suffices to show that for any (E, Y) there exists

Figure 7: Construction of E' and Y' with internal emulated parties E, Y interacting with $\mathsf{C}^{\mathsf{pp}}_{\mathcal{M}}$..

$(\mathsf{E}', \mathsf{Y}')$ with $\mathsf{Y}' \in \mathcal{Y}_{\mathsf{lift}}$ such that the transcripts in the executions $\langle \mathsf{C}(R_{\mathsf{C}}), \mathsf{E}, \mathsf{Y} \rangle$ and $\langle \mathsf{C}(R_{\mathsf{C}}), \mathsf{E}', \mathsf{Y}' \rangle$ are identical. Intuitively, this holds because C only ever sends $\perp$ to Y. Recall that Y cannot make any tick queries, which is the only way for $\mathcal{M}(\mathsf{op}(id))$ (or C) to produce non-$\perp$ output. Hence, the view of Y can easily be simulated by E, and the restriction that Y does not communicate with E is without loss of generality. From there, we can restrict Y to be a dummy party without loss of generality, satisfying the requirements of $\mathcal{Y}_{\mathsf{lift}}$.

Fix any E, Y. Figure 7 depicts E' and Y'. The subject Y' forwards messages from E' to C, but sends no messages to E'. The environment E' emulates E and Y in its head and emulates their interactions with other parties as follows:

- E' passes messages between emulated parties E and Y freely.

- E' forwards queries from E intended for C; similarly, E' forwards responses from C to E.

- E' forwards queries from Y intended for C to the external Y' (who then forwards this to C). When reactivated, E' sends the response $\perp$ to Y.

By construction, Y' satisfies the condition that it only sends delete if E' tells it to, and it never sends any message to E'. Hence, $\mathsf{Y}' \in \mathcal{Y}_{\mathsf{lift}}$. By the construction in Algorithm 4, C always sends $\perp$ in response to Y's queries. Using this, it is easy to show that the emulation is perfect. Hence, the transcripts seen by C in the two executions $\langle \mathsf{C}(R_{\mathsf{C}}), \mathsf{E}, \mathsf{Y} \rangle$ and $\langle \mathsf{C}(R_{\mathsf{C}}), \mathsf{E}', \mathsf{Y}' \rangle$ are identical.

$\square$

5.4 Examples of pan-private controllers

5.4.1 Counters

Perhaps the most widely studied algorithm that is pan-private under continual release is the tree mechanism of Dwork et al. [2010a], Chan et al. [2011]. Although originally formulated without pan-privacy, it relies on the composition of summations over different time intervals. Each of these can be made pan-private against a single intrusion by adding noise when the counter is initialized and again at release [Dwork et al., 2010b]. The overall mechanism retains the same asymtptotic guarantees, since the noise at most doubles, and enjoys pan-privacy. We summarize its properties here:

42

Proposition 5.12 (Combining Dwork et al. [2010b] with Dwork et al. [2010a], Chan et al. [2011])**.** *There is an (ϵ, δ)-adaptively event-level pan-private under continual-release algorithm that takes a time horizon T and a sequence of inputs $x_1, x_2, ... \in [0, 1]$ and at the t-th* tick*, for $t \in [T]$, releases* out$_t$ *such that*

$$\mathsf{out}_t = \sum_{\substack{i \, : \, x_i \text{ received} \\ \text{by time } t}} x_i + Z_t,$$

where Z_t's are distributed (independently of x_i's but not each other) as $Z_t \sim N(0, \sigma_t^2)$ with $\sigma_t = O\left(\frac{\sqrt{\log^3 T \log(1/\delta)}}{\epsilon}\right)$.

In Section 6, we use the tree mechanism as a building block to construct a controller that satisfies deletion-as-control but not history independence, confidentiality, nor differential privacy. Moreover, one can use the tree mechanism to construct a controller that satisfies deletion-as-control but not history independence (cf. Proposition 5.1). The controller could, for example, use the tree mechanism to count the number of distinct users n it has seen. Applying Theorem 5.9, a controller could publish $n + N(0, \sigma_t^2)$ even after all users have requested deletion, all while satisfying deletion-as-control. *Such functionality is impossible under history independence.*

5.4.2 Learning under Pan-Private Continual Release (and with Deletion)

As an example, we show how the results of this section allow one to maintain a model trained via gradient descent with relatively little noise—the algorithm's error is similar to that of state of the art federated machine learning algorithms.

Consider the following optimization problem: given a loss function $\ell : \Theta \times \mathcal{X} \to \mathbb{R}$ and a data set $\vec{x} = (x_1, ..., x_n)$, we aim to find $h \in \Theta$ which approximately minimizes

$$\mathcal{L}_{\vec{x}}(h) = \sum_{i=1}^{n} \ell(h; x_i). \tag{8}$$

To analyze convergence, we assume a convex loss function. Nevertheless, the method we give applies more broadly—the privacy or deletion guarantees hold regardless of convexity, and the algorithms makes sense as long as the loss function is roughly convex near its minimum.

Algorithms for this problem that are differentially private under continual release were studied for online learning and efficient distributed learning [Kairouz et al., 2021]. The algorithm they consider is not panprivate. However, it only access the data via the tree-based mechanism for continual release of a sum Dwork et al. [2010a], Chan et al. [2011] (in this case, releasing the sum of the gradients of users' contributions to the overall loss function over the evaluation of a first-order optimization algorithm). We observed above that this can be made panprivate with only a constant factor increase in the added noise (Proposition 5.12)

To allow us to apply the convergence analysis of Kairouz et al. [2021] as a black box, we assume: 1. Θ is convex, and $\ell(\cdot; x)$ is convex for every choice of x; 2. ℓ is 1-Lipschitz in h (that is, suppose $\Theta \in \mathbb{R}^d$ and for all h, h' and x, we have $|\ell(h; x) - \ell(h'; x)| \leq \|h - h'\|_2$); 3. One new user arrives between adjacent ticks, though deletions may occur at any time.

Proposition 5.13 (Derived from Theorem 5.1 from *arXiv v3* of Kairouz et al. [2021])**.** *There is an (ϵ, δ)-adaptively pan-private, continual-release algorithm that takes takes a time horizon T, a*

learning rate $\lambda > 0$, and a sequence of inputs $x_1, x_2, \ldots$ and at the t-th tick *releases a model h_t such that, for every $t \in [T]$, data set $\vec{x}_t \in \mathcal{Z}^n$ (consisting of records received by tick t), $h^* \in \Theta$, and $\beta > 0$, with probability at least $1 - \beta$ over the coins of the algorithm,*

$$\mathcal{L}_{\vec{x}_t}(h_t) - \mathcal{L}_{\vec{x}_t}(h^*) = O\left(\|h^*\| \cdot \left(\frac{1}{\sqrt{n}} + d^{1/4}\sqrt{\frac{\ln^2(1/\delta)\ln(1/\beta)}{\epsilon n}}\right)\right)$$

This result immediately yields a controller that satisfies (ϵ, δ)-deletion as control and maintains a model whose accuracy tracks that of the best model trained on all the arrivals so far (where repeated arrivals of the same user are ignored). The version above uses a fixed learning rate λ which can be tuned to get error $\tilde{O}(d^{1/4}/\sqrt{n})$ for any particular n; however, one could also decrease the learning rate as $1/\sqrt{t}$ to get bounds that hold for all data set sizes.

In contrast to the HI-style algorithms proposed for machine unlearning Gupta et al. [2021], this controller need not update the model when a user is deleted.

6 Parallel Composition

In this section, we show that our definition does more than unify approaches to deletion based on history independence, confidentiality, and differential privacy. We describe a controller that satisfies deletion-as-control but none of the other three notions. To prove that the controller satisfies deletion-as-control, we prove that the definition enjoys a limited form of *parallel composition*: a controller that is built from two constituent sub-controllers, each of which is run independently (Figure 8), will be deletion compliant if the component controllers are compliant and satisfy additional conditions. We focus on a special case that suffices for our needs, and leave a general treatment of composition of deletion-as-control for future work.

To anchor the discussion, we consider the touchstone composed controller "Public Directory with DP Statistics," described in Algorithm 5. It provides a public directory (e.g., a phone book) which, in addition to answering directory queries, periodically reports the total number of users that have made queries to the directory. We build the touchstone controller C from two SHI dictionaries $\mathcal{D}$ and $\mathcal{U}$ (Section 3.1.1) and the pan-private tree mechanism $\mathcal{M}$ (Proposition 5.12). $\mathcal{D}$ implements the directory functionalities of reading and writing. $\mathcal{M}$ keeps track of an approximate count of users—current and former—that have looked up an entry in $\mathcal{D}$ for the first T epochs (e.g., the time between ticks). $\mathcal{U}$ is an auxiliary dictionary used to ensure that $\mathcal{M}$ counts distinct users.

To prove that C satisfies deletion-as-control, we prove that one can build deletion-as-control controllers from two constituent sub-controllers by parallel composition. We consider the special case where one sub-controller satisfies $(0, 0)$-deletion-as-control and implements a *deterministic functionality*, and where both sub-controllers are *query-response controllers*. We define these next. Note that the controllers given in Section 3.2 and Section 5 are all query-response controllers. While we restrict our attention to a single controller of each type, the result immediately extends by induction to many query-response controllers satisfying $(0, 0)$ deletion-as-control with deterministic functionalities.

Definition 6.1 (Deterministic functionality)**.** *A controller C implements a* deterministic functionality *if for all E, Y, the transcript of $\langle \mathsf{C}(R_\mathsf{C}), \mathsf{E}, \mathsf{Y} \rangle$ is independent of R_C.*

44

Algorithm 5 Public directory with DP statistics

1: **procedure** INITIALIZE(time horizon $T \in \mathbb{N}$, privacy parameters ϵ, δ) $\triangleright$ Run on first activation
2:     Initialize two SHI dictionaries $\mathcal{D}$ for the directory and $\mathcal{U}$ for users.
3:     Initialize the tree mechanism $\mathcal{M}$ with parameters (ϵ, δ, T)

4: **procedure** ACTIVATE($\mathsf{cID}, \mathsf{op}(arg)$)
5:     **if** $\mathsf{op} = \mathsf{delete}$ **then**
6:         $\mathcal{D}.\mathsf{delete}(\mathsf{cID})$
7:         $\mathcal{U}.\mathsf{delete}(\mathsf{cID})$
8:         **return** $\perp$
9:     **if** $\mathsf{op} = \mathsf{set}$ **then**
10:        $\mathcal{D}.\mathsf{set}(\mathsf{cID}, arg)$
11:        **return** $\perp$
12:     **if** $\mathsf{op} = \mathsf{get}$ **then**
13:        **if** $\mathsf{cID} \notin \mathcal{U}$ **then**
14:           $\mathcal{U}.\mathsf{set}(\mathsf{cID}, 1)$
15:           Insert the value 1 into $\mathcal{M}$'s data stream
16:        **return** $\mathcal{D}.\mathsf{get}(arg)$
17:     **if** $\mathsf{op} = \mathsf{getCount}$ **then**
18:        **return** $\mathcal{M}(\mathsf{tick})$

Figure 8: Parallel composition of controllers C_0 and C_1.

Definition 6.2 (Query-response controller). C *is a* query-response *controller if it always replies to the same party that activated it. Namely, when* C *is activated with* $(\mathsf{cID}, \mathsf{msg})$ *on its input tape, it always halts with* $(\mathsf{cID}, \mathsf{msg}')$ *on its output tape for some* msg'.

The example controllers we discuss are generally query-response. But a messaging server, for example, might not be: a request from Alice to send something Bob could result in a push from the server to Bob.

Definition 6.3 (Parallel composition). *Let* C_0 *and* C_1 *be controllers. We define their* parallel composition, *denoted* $\mathsf{C}_{0\|1}$, *to be the controller that emulates* C_0 *an* C_1 *internally. When activated with* op_{id}, $\mathsf{C}_{0\|1}$ *computes* $\mathsf{out}_0 \leftarrow \mathsf{C}_0(\mathsf{op}_{id})$ *and* $\mathsf{out}_1 \leftarrow \mathsf{C}_1(\mathsf{op}_{id})$ *and returns* $\mathsf{out}_{0\|1} = (\mathsf{out}_0, \mathsf{out}_1)$. *The controllers* C_0 *and* C_1 *are emulated with disjoint portions of* $\mathsf{C}_{0\|1}$*'s randomness tape* $R_{0\|1} = (R_0, R_1)$ *and work tape* $\mathsf{state}_{0\|1} = (\mathsf{state}_0, \mathsf{state}_1)$.

Theorem 6.4. *Let* C_0 *and* C_1 *be query-response controllers and let* $C_{0\|1}$ *be their parallel composition. If* C_0 *satisfies* (ϵ, δ)-*deletion-as-control, and if* C_1 *satisfies* $(0, 0)$-*deletion-as-control and has deterministic functionality, then* $\mathsf{C}_{0\|1}$ *satisfies* (ϵ, δ)-*deletion-as-control.*

Proof of Theorem 6.4. Define the parallel simulator
$\mathsf{Sim}_{0\|1}(\vec{\mathsf{q}}_\mathsf{E}, R_{0\|1}, \mathsf{state}_{0\|1})$ as outputting $R'_{0\|1} = (R'_0, R'_1)$ where each half is computed using the corresponding simulator. That is, for each $i \in \{0, 1\}$:

$$R'_i \sim \mathsf{Sim}_i(\vec{\mathsf{q}}_\mathsf{E}, R_i, \mathsf{state}_i).$$

By hypothesis, all of the following hold for all E, Y.

- $(R'_0 \overset{\epsilon,\delta}{\approx} R_0)$ AND $(R'_1 \equiv R_1)$,
- $(\mathsf{state}_1 = \mathsf{state}'_1)$ OR $(\mathsf{state}_1 = \bot)$
- With probability at least $1 - \delta$: $(\mathsf{state}_0 = \mathsf{state}'_0)$ OR $(\mathsf{state}_0 = \bot)$.

We must show that for all E, Y, both of the following hold in the real-ideal probability experiment given in Definition 2.3.

- $R'_{0\|1} \overset{\epsilon,\delta}{\approx} R_{0\|1}$
- With probability at least $1 - \delta$: $\mathsf{state}_{0\|1} = \mathsf{state}'_{0\|1} \neq \bot$ or $\mathsf{state}_{0\|1} = \bot$ (i.e., the real execution doesn't terminate).

We consider the state first. First, observe the following:

$$(\mathsf{state}_0 = \bot) \implies (\mathsf{state}_1 = \bot), \text{ and}$$
$$(\mathsf{state}_0 \neq \bot) \implies (\mathsf{state}_1 \neq \bot) \implies (\mathsf{state}_1 = \mathsf{state}'_1)$$

This is because both state_0 and state_1 result from a single real execution $\langle \mathsf{C}_{0\|1}, \mathsf{E}, \mathsf{Y} \rangle$ that either terminates ($\mathsf{state}_{0\|1} = \bot$) or does not ($\mathsf{state}_{0\|1} \neq \bot$). Hence,

$$\Pr\left[\mathsf{state}_{0\|1} \in \{\mathsf{state}'_{0\|1}, \bot\}\right]$$
$$= \Pr\left[\mathsf{state}_0 \in \{\mathsf{state}'_0, \bot\}\right] \geq 1 - \delta$$

.

It remains to show that $R'_{0\|1} \overset{\epsilon,\delta}{\approx} R_{0\|1}$. Below, we prove that R'_0 and R'_1 are independent random variables. This suffices because $(R'_0 \overset{\epsilon,\delta}{\approx} R_0)$ and $(R'_1 \equiv R_1)$.

Let $\vec{q}$ be the queries received by $C_{0\|1}$ in the real execution. By construction, these are also the queries received by the emulated C_0 and C_1. Observe that for $i \in \{0,1\}$ there exists randomized S_i such that $R'_i \sim S_i(R_i, \vec{q})$. Namely, R'_i only depends on R_i and $\vec{q}$. This is because $R'_i \leftarrow \mathsf{Sim}_i(R_i, \vec{q}_E, \mathsf{state}_i)$ where $\vec{q}_E$ and state_i are deterministic functions of R_i and $\vec{q}$ (and the code of $C_{0\|1}$, E, Y which are fixed).

The above implies two useful facts. First, R'_1 is independent of R'_0 conditioned on $\vec{q}$. This follows from the fact that $R'_i \sim S_i(R_i, \vec{q})$, combined with the fact that R_1 and R_0 are independent (by construction). Second, R'_1 is independent of the random variable $\vec{q}$. That is, for all r'_1 and q, $\Pr[R'_1 = r'_1 | \vec{q} = q] = \Pr[S_1(R_1, \vec{q}) = r'_1 | \vec{q} = q] = \Pr[S_1(R_1, q) = r'_1] = \Pr[R'_1 = r'_1]$. The middle equality uses the independence of $\vec{q}$ and R_1, which follows from the hypothesis that C_1 has a deterministic functionality.

Fix r'_0 and r'_1. We check independence of R'_0 and R'_1 as follows.

$$
\begin{aligned}
\Pr[R'_0 = r'_0 \wedge R'_1 = r'_1] &= \Pr[R'_0 = r'_0] \cdot \sum_q \Pr[\vec{q} = q | R'_0 = r'_0] \cdot \Pr[R'_1 = r'_1 | \vec{q} = q, R'_0 = r'_0] \\
&= \Pr[R'_0 = r'_0] \cdot \sum_q \Pr[\vec{q} = q | R'_0 = r'_0] \cdot \Pr[R'_1 = r'_1 | \vec{q} = q] \\
&= \Pr[R'_0 = r'_0] \cdot \Pr[R'_1 = r'_1] \cdot \sum_q \Pr[\vec{q} = q | R'_0 = r'_0] \\
&= \Pr[R'_0 = r'_0] \cdot \Pr[R'_1 = r'_1]
\end{aligned}
$$

The second equality uses the independence of R'_1 and R'_0 conditioned on $\vec{q}$. The third equality uses the independence of R'_1 and $\vec{q}$. $\square$

Corollary 6.5. *The Public Directory with DP Statistics (Algorithm 5) satisfies (ϵ, δ)-deletion-as-control.*

Proof. Let C_0 be the controller $C_{\mathcal{M}}^{\mathsf{pp}}$ defined relative to the pan-private tree mechanism $\mathcal{M}$ (Algorithm 4). Let C_1 be the controller relative to a SHI dictionary (Definition 3.6). By construction, C_0 and C_1 are query-response controllers and C_1 has a deterministic functionality. C_0 satisfies (ϵ, δ) deletion-as-control and C_1 satisfies (ϵ, δ) deletion-as-control (Theorems 5.9 and 3.8 respectively). Algorithm 5 is equivalent to the parallel composition $C_{0\|1}$ of C_0 and C_1. By parallel composition, Algorithm 5 satisfies (ϵ, δ) deletion-as-control. $\square$

7 Conclusion

Defining deletion-as-control in a way that is both expressive and meaningful is the central challenge of our work. We believe that our definition succeeds, providing a new perspective to the ongoing discussion on how to give users control over their data.

More work is needed to understand how deletion-as-control handles the complexity of real-life functionalities. For example, we are far from understanding the implications for something

like Twitter, though the Public Bulletin Board serves as a starting point. Analyzing complex functionalities may involve further studying the adaptive variants of history independence and pan-privacy defined in this work. Beyond specific functionalities, more work is needed to interpret the guarantees provided by our notion. In particular, we have a limited view of what can be said about groups of individuals and about composition.

These are directions for future technical work, but also for normative, legal, and policy considerations. Consider, for example, the goal of machine unlearning: maintaining a model while respecting requests to delete. The algorithms in the machine unlearning literature seek to approximate what one would get by retraining from scratch (that is, history independence). But using adaptive pan-privacy, one can satisfy deletion-as-control without updating the model in response to deletion requests. Each behavior might be appropriate for a different setting, depending on the most relevant measure of model accuracy. The difference may also relate to whether one adopts an individual- or group-based view of a right to erasure. We hope that our work inspires further exploration of these questions.

Acknowledgements

References

A. Achille, M. Kearns, C. Klingenberg, and S. Soatto. Ai model disgorgement: Methods and choices. *arXiv preprint arXiv:2304.03545*, 2023.

M. Altman, A. Cohen, K. Nissim, and A. Wood. What a hybrid legal-technical analysis teaches us about privacy regulation: The case of singling out. *BUJ Sci. & Tech. L.*, 27:1, 2021.

R. Bassily, A. Smith, and A. Thakurta. Private empirical risk minimization: Efficient algorithms and tight error bounds. In *2014 IEEE 55th annual symposium on foundations of computer science*, pages 464–473. IEEE, 2014.

G. E. Blelloch and D. Golovin. Strongly history-independent hashing with applications. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2007. doi: 10.1109/FOCS.2007.36.

L. Bourtoule, V. Chandrasekaran, C. A. Choquette-Choo, H. Jia, A. Travers, B. Zhang, D. Lie, and N. Papernot. Machine unlearning. In *42nd IEEE Symposium on Security and Privacy*, 2021. doi: 10.1109/SP40001.2021.00019.

Y. Cao and J. Yang. Towards making systems forget with machine unlearning. In *2015 IEEE Symposium on Security and Privacy*, pages 463–480. IEEE, 2015.

N. Carlini, F. Tramer, E. Wallace, M. Jagielski, A. Herbert-Voss, K. Lee, A. Roberts, T. Brown, D. Song, U. Erlingsson, et al. Extracting training data from large language models. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2633–2650, 2021.

T.-H. H. Chan, E. Shi, and D. Song. Private and continual release of statistics. *ACM Transactions on Information and System Security (TISSEC)*, 14(3):1–24, 2011.

A. Cohen and K. Nissim. Linear program reconstruction in practice. *Journal of Privacy and Confidentiality*, 10(1), Jan. 2020. doi: 10.29012/jpc.711. URL `https://journalprivacyconfidentiality.org/index.php/jpc/article/view/711`.

Court of Justice of the European Union. Press release no 70/14. `https://curia.europa.eu/jcms/upload/docs/application/pdf/2014-05/cp140070en.pdf`, May 2013.

C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor. Our data, ourselves: Privacy via distributed noise generation. In *International Conference on the Theory and Applications of Cryptographic Techniques*, EUROCRYPT '06, pages 486–503, St. Petersburg, Russia, 2006.

C. Dwork, M. Naor, T. Pitassi, and G. N. Rothblum. Differential privacy under continual observation. In L. J. Schulman, editor, *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010*, pages 715–724. ACM, 2010a. doi: 10.1145/1806689.1806787.

C. Dwork, M. Naor, T. Pitassi, G. N. Rothblum, and S. Yekhanin. Pan-private streaming algorithms. In *ICS*, pages 66–80, 2010b.

C. Dwork, F. McSherry, K. Nissim, and A. D. Smith. Calibrating noise to sensitivity in private data analysis. *J. Priv. Confidentiality*, 2016. doi: 10.29012/jpc.v7i3.405.

B. Fowler. Data breaches break record in 2021. *CNET*, 2022. URL `https://www.cnet.com/news/privacy/record-number-of-data-breaches-reported-in-2021-new-report-says/`.

S. Garg, S. Goldwasser, and P. N. Vasudevan. Formalizing data deletion in the context of the right to be forgotten. In *Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part II 30*, pages 373–402. Springer, 2020.

A. Ginart, M. Guan, G. Valiant, and J. Y. Zou. Making ai forget you: Data deletion in machine learning. In *Advances in Neural Information Processing Systems*, pages 3513–3526, 2019.

J. Godin and P. Lamontagne. Deletion-compliance in the absence of privacy. In *2021 18th International Conference on Privacy, Security and Trust (PST)*, pages 1–10. IEEE, 2021.

A. Golatkar, A. Achille, and S. Soatto. Eternal sunshine of the spotless net: Selective forgetting in deep networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9304–9312, 2020a.

A. Golatkar, A. Achille, and S. Soatto. Forgetting outside the box: Scrubbing deep networks of information accessible from input-output observations. *arXiv preprint arXiv:2003.02960*, 2020b.

Google. Right to be forgotten overview. `https://support.google.com/legal/answer/10769224`. Accessed: 2023-04-18.

Google. Helping public health officials combat covid-19. https://blog.google/technology/health/covid-19-community-mobility-reports/, 2020. Accessed: 2023-04-18.

C. Guo, T. Goldstein, A. Hannun, and L. van der Maaten. Certified data removal from machine learning models, 2020.

V. Gupta, C. Jung, S. Neel, A. Roth, S. Sharifi-Malvajerdi, and C. Waites. Adaptive machine unlearning. *Advances in Neural Information Processing Systems*, 34:16319–16330, 2021.

J. D. Hartline, E. S. Hong, A. E. Mohr, W. R. Pentney, and E. C. Rocke. Characterizing history independent data structures. *Algorithmica*, 42(1):57–74, 2005.

P. Jain, S. Raskhodnikova, S. Sivakumar, and A. D. Smith. The price of differential privacy under continual observation. *CoRR*, abs/2112.00828, 2021. URL https://arxiv.org/abs/2112.00828.

JASON. Consistency of data products and formal privacy methods for the 2020 Census. Panel Report JSR 21-02, JASON, The MITRE Corporation, 1 2022.

P. Kairouz, B. McMahan, S. Song, O. Thakkar, A. Thakurta, and Z. Xu. Practical and private (deep) learning without sampling or shuffling. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning (ICML)*, 2021.

S. P. Kasiviswanathan and A. D. Smith. On the 'semantics' of differential privacy: A bayesian formulation. *Journal of Privacy and Confidentiality*, 2014. Available as v3 of https://arxiv.org/abs/0803.3946.

G. King and N. Persily. Unprecedented facebook urls dataset now available for academic research through social science one. https://socialscience.one/blog/unprecedented-facebook-urls-dataset-now-available-research-through-social-science-one, 2020. Accessed: 2023-04-18.

D. Micciancio. Oblivious data structures: applications to cryptography. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 456–464, 1997.

M. Naor and V. Teague. Anti-persistence: History independent data structures. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 492–501, 2001.

National Conference of State Legislatures. State laws related to digital privacy. https://www.ncsl.org/technology-and-communication/state-laws-related-to-digital-privacy, June 2022. Accessed: 2023-04-18.

S. Neel, A. Roth, and S. Sharifi-Malvajerdi. Descent-to-delete: Gradient-based methods for machine unlearning, 2020.

A. Ng. Homeland Security records show 'shocking' use of phone data, ACLU says. *Politico*, 2022. URL https://www.politico.com/news/2022/07/18/dhs-location-data-aclu-00046208.

K. Nissim, A. Bembenek, A. Wood, M. Bun, M. Gaboardi, U. Gasser, D. R. O'Brien, T. Steinke, and S. Vadhan. Bridging the gap between computer science and legal approaches to privacy. *Harv. JL & Tech.*, 31:687, 2017.

Y. Polyanskiy. Two fundamental probabilistic models. `https://ocw.mit.edu/courses/6-436j-fundamentals-of-probability-fall-2018/resources/mit6_436jf18_lec02/`, 2018. Accessed: 2023-04-18.

R. Rogers, A. Roth, A. Smith, and O. Thakkar. Max-information, differential privacy, and post-selection hypothesis testing. In *IEEE Symposium on Foundations of Computer Science*, FOCS '16, pages 487–494, 2016. `https://arxiv.org/abs/1604.03924`.

A. Sekhari, J. Acharya, G. Kamath, and A. T. Suresh. Remember what you want to forget: Algorithms for machine unlearning. *arXiv preprint arXiv:2103.03279*, 2021.

R. K. Slaughter, J. Kopec, and M. Batal. Algorithms and economic justice: A taxonomy of harms and a path forward for the federal trade commission. *Yale JL & Tech.*, 23:1, 2020.

A. Thudi, H. Jia, I. Shumailov, and N. Papernot. On the necessity of auditable algorithmic definitions for machine unlearning. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 4007–4022, 2022.

E. Ullah, T. Mai, A. Rao, R. Rossi, and R. Arora. Machine unlearning via algorithmic stability. *arXiv preprint arXiv:2102.13179*, 2021.

US Census Bureau. Why the Census Bureau chose differential privacy. Technical Report C2020BR-03, US Census Bureau, March 2023.

S. Vadhan and W. Zhang. Concurrent composition theorems for all standard variants of differential privacy. *arXiv preprint arXiv:2207.08335*, 2022. To appear in the 55th Annual ACM Symposium on Theory of Computing (STOC), 2023.

A   Lemmas on Couplings and Marginal Distributions

Lemma A.1 (Coupling Lemma). *Let P, Q be probability distributions on sets $\mathcal{X}$ and $\mathcal{Y}$, respectively, and let $f : \mathcal{X} \to \mathcal{Z}$ and $g : \mathcal{Y} \to \mathcal{Z}$ be (deterministic) functions with the same codomain $\mathcal{Z}$. Suppose that $f(X) \approx_{\epsilon, \delta} g(Y)$ when $X \sim P$ and $Y \sim Q$.*

Consider the collection of distributions $\{Q_x \in \Delta(\mathcal{Y}) : x \in \mathcal{X}\}$ on the set $\mathcal{Y}$, where Q_x denotes the distribution on Y conditioned on the event that $g(Y) = f(x)$ (that is $Q_x = Q|_{\{y : g(y) = f(x)\}}$). In the case that $g^{-1}(f(x))$ is empty, Q_x assigns probability to a default value $\perp \in \mathcal{Y}$.

If we select $X \sim P$ and then sample $Y' \sim Q_X$ (so that $\Pr(Y' = y | X = x) = Q_x(y)$), then

1. $Y' \approx_{\epsilon, \delta} Y$, and

2. $f(X) = g(Y')$ with probability at least $1 - \delta$ over (X, Y).

For example, consider an (ϵ, δ)-DP mechanism $\mathcal{M}$ using randomness $R \sim \mathcal{U}$, and let $\mathbf{x}, \mathbf{x}'$ be a pair of neighboring inputs. Take $f(R) = \mathcal{M}(\mathbf{x}; R)$ and $g(R) = \mathcal{M}(\mathbf{x}'; R)$. Lemma A.1 gives a way to jointly sample randomness (R, R') such that $R \sim \mathcal{U}$, $R' \overset{\epsilon, \delta}{\approx} \mathcal{U}$, and with probability at least $1 - \delta$: $\mathcal{M}(\mathbf{x}; R) = \mathcal{M}(\mathbf{x}'; R')$. For simple DP mechanisms, it is easy to understand what this sampler does. For instance, let $\mathsf{Lap}(1/\epsilon; R)$ be an algorithm that samples from the Laplace distribution when $R \sim \mathcal{U}$ is uniform. Consider $\mathcal{M}(\mathbf{x}; R) = f(\mathbf{x}) + \mathsf{Lap}(1/\epsilon; R)$ for some f with

sensitivity 1. The sampler in Lemma A.1 samples $R \sim \mathcal{U}$, and samples R' uniformly conditioned on $\mathsf{Lap}(1/\epsilon; R') = f(\mathbf{x}') - f(\mathbf{x})$.

Proof. Consider the randomized map $G^* : \mathcal{Z} \to \mathcal{Y}$ that inverts g for inputs drawn from Q—namely, $G^*(z)$ returns a random element according to $Q|_{\{y:g(y)=z\}}$ and returns the default element $\perp$ with probability 1 if $g^{-1}(z)$ is empty. Running G^* on $g(Y)$ returns a sample distributed identically to Y conditioned on $g(Y)$. That is,

$$\Big(g(Y),\ G^*(g(Y))\Big) \equiv \Big(g(Y),\ Y\Big).$$

Since the $\approx_{\epsilon,\delta}$ relation is preserved by processing, and since $f(X) \approx_{\epsilon,\delta} g(Y)$, we have

$$\Big(g(Y),\ G^*(g(Y))\Big) \approx_{\epsilon,\delta} \Big(f(X),\ G^*(f(X))\Big).$$

Now for every $x \in \mathcal{X}$, the random variable $G^*(f(x))$ is distributed as Q_x. Therefore, $G^*(f(X))$ is distributed as Y' from the theorem statement and $Y' \approx_{\epsilon,\delta} Y$ (the first requirement of the theorem). Furthermore, $g(Y')$ equals $f(X)$ except when $f(X)$ lies outside the image of g. Since $f(X) \approx_{\epsilon,\delta} g(Y)$, the value $f(X)$ lies outside the image of g with probability at most δ. This establishes the theorem's second requirement. $\qquad\square$

A computable analogue of Lemma A.1 To demonstrate deletion-as-control compliance, we need a simulator to sample ideal-world randomness R'_C whose marginal distribution is close to the uniform distribution while also guaranteeing that $\mathsf{state}'_\mathsf{C} = \mathsf{state}_\mathsf{C}$. A natural strategy is to sample R'_C from distribution conditioned on the equality of the states. Applying Lemma A.1, the resulting distribution suffices for our purposes in this paper, where $X = R_\mathsf{C}$ and $Y = R'_\mathsf{C}$. But there is a technicality that needs to be dealt with. In full generality, the conditional distributions Q_x in that lemma may not be sampleable in finite time. This will not be a problem for us, as our applications of the lemma will be restricted to functions that depend only a finite prefix of R_C and R'_C. For completeness, we state a computable analogue of Lemma A.1. Elsewhere, we will elide this technicality and invoke the simpler Lemma A.1.

Lemma A.2 (Computable Coupling Lemma). *Let P, Q, $\mathcal{X}$, $\mathcal{Y}$, $\mathcal{Z}$, f, g, and Q_x as in Lemma A.1. Let $\mathcal{X} = \{0,1\}^* = \mathcal{Y}$. Suppose that $g(y)$ depends only on a finite prefix of $y \in \{0,1\}^*$. Suppose also that for all $x \in \{0,1\}^*$, either $f(x)$ depends only on a finite prefix of $x \in \{0,1\}^*$, or $g^{-1}(f(x)) = \emptyset$. Then for any x, $Y \sim Q_x$ can be written as $Y_1 \| Y_2$ where $|Y_1|$ is finite and Y_2 is uniform in $\{0,1\}^*$ and independent of Y_1.*

In our applications, X and Y will be R_C and R'_C respectively. f and g will be functions of the real and ideal executions, respectively. By construction, an ideal execution is always finite, and therefore $g(Y)$ will only depend on a finite prefix of R'_C. A real execution is either finite—in which case $f(X)$ only depends on a finite prefix of R_C—or it is infinite. If it is infinite the real execution (and f) outputs $\perp$, whereas the ideal execution (and g) can never output $\perp$.

Lemma A.3 (Conditioning Lemma [Kasiviswanathan and Smith, 2014]). *Let (A, B) and (A', B') be pairs of random variables over the same domain $\mathcal{A} \times \mathcal{B}$ such that $(A, B) \stackrel{\epsilon,\delta}{\approx} (A', B')$. Then for all $\delta' > 0$, with probability at least $1 - \gamma$ over $a \leftarrow A$, we have*

$$B|_{A=a} \stackrel{\epsilon',\delta'}{\approx} B'|_{A'=a},$$

where $\epsilon' = 3\epsilon$ and $\gamma = \frac{2\delta}{\delta'} + \frac{2\delta}{1-\exp(-\epsilon)}$.

B Differential Privacy Preliminaries

A dataset $\mathbf{x} = (x_1, \ldots, x_n) \in \mathcal{X}^n$ is a vector of elements from universe $\mathcal{X}$. Two datasets are *neighbors* if they differ in at most one coordinate. Informally, differential privacy requires that an algorithm's output distributions are similar on all pairs of neighboring datasets. We use two different variants of differential privacy. The first one (and the main one used in this paper) is the standard definition of differential privacy.

Definition B.1 (Differential Privacy [Dwork et al., 2016, 2006]). *A randomized algorithm* $\mathcal{M}$: $\mathcal{X}^n \to \mathcal{Y}$ *is* (ϵ, δ)*-differentially private if, for every pair of neighboring datasets* $\mathbf{x}, \mathbf{x}' \in \mathcal{X}^n$*, the distributions of* $\mathcal{M}(\mathbf{x})$ *and* $\mathcal{M}(\mathbf{x}')$ *are defined for the same* σ*-algebra* $\Sigma_\mathcal{Y}$ *and*

$$\mathcal{M}(\mathbf{x}) \overset{\epsilon,\delta}{\approx} \mathcal{M}(\mathbf{x}') .$$

Differential privacy protects the privacy of groups of individuals, and is closed under post-processing. Moreover, it is closed under adaptive composition. For a fixed dataset $\mathbf{x}$, *adaptive composition* states that the results of a sequence of computations satisfies differential privacy even when the chosen computation $\mathcal{M}_t(\cdot)$ at time t depends on the outcomes of previous computations $\mathcal{M}_1(\mathbf{x}), \ldots, \mathcal{M}_{t-1}(\mathbf{x})$. Under adaptive composition, the privacy parameters add up.

Lemma B.2 (Post-Processing). *If* $\mathcal{M} : \mathcal{X}^n \to \mathcal{Y}$ *is* (ϵ, δ)*-differentially private, and* $\mathcal{B} : \mathcal{Y} \to \mathcal{Z}$ *is any randomized function, then the algorithm* $\mathcal{B} \circ \mathcal{M}$ *is* (ϵ, δ)*-differentially private.*

Lemma B.3 (Group Privacy). *Every* (ϵ, δ)*-differentially private algorithm* $\mathcal{M}$ *is* $\left(k\epsilon, \delta\frac{e^{k\epsilon}-1}{e^\epsilon-1}\right)$*-differentially private for groups of size* k*. That is, for all datasets* $\mathbf{x}, \mathbf{x}'$ *such that* $\|\mathbf{x} - \mathbf{x}'\|_0 \leq k$ *and all subsets* $Y \subseteq \mathcal{Y}$*,*

$$\Pr[\mathcal{M}(\mathbf{x}) \in Y] \leq e^{k\epsilon} \cdot \Pr[\mathcal{M}(\mathbf{x}') \in Y] + \delta \cdot \frac{e^{k\epsilon}-1}{e^\epsilon-1}.$$

Definition B.4 (Composition of (ϵ, δ)-differential privacy). *Suppose* $\mathcal{M}$ *is an adaptive composition of differentially private algorithms* $\mathcal{M}_1, \ldots, \mathcal{M}_T$*. If for each* $t \in [T]$*, algorithm* $\mathcal{M}_t$ *is* (ϵ_t, δ_t)*-differentially private, then* $\mathcal{M}$ *is* $(\sum_t \epsilon_t, \sum_t \delta_t)$*-differentially private.*

C Strong History Independence and Deletion-as-control

We show that strong history independence implies deletion-as-control. Throughout, we use SHI to mean $(0, 0)$-SHI.

A foundational result in the study of history independent data structures characterizes the behavior of SHI implementations of a large class of ADTs called reversible ADTs.

Definition C.1 (Reversible ADT). *An ADT is* reversible *if, for all logical states* A *and* B *reachable from the initial state, there exists some finite sequence of operations* σ *such that* $\mathsf{ADT}(A, \sigma).\mathsf{state} = B$*. That is, all reachable states are* mutually reachable*.*

Theorem C.2 (Hartline et al. [2005]). *A reversible data structure is SHI if and only if each logical state has a single canonical memory representation for every setting of the implementation's randomness tape.*

Claim C.3. *If an ADT supports logical deletion, then it is reversible. The converse is not true.*

Proof. (Logical deletion $\implies$ reversible): To prove reversibility, it is sufficient to give a sequence of operations that takes the ADT back to its initial state. Fix a state A that is reachable from the initial state. Pick any sequence σ that takes the initial state to A. Delete all of the *id*'s in σ one by one. By logical deletion, the resulting state is the initial state.

(Reversible $\not\implies$ logical deletion): Consider the ADT where the set of (non-delete) operations is the space of IDs and the logical state consists of the most recent ID. Suppose also that $\mathsf{out_{adt}} = \bot$ always. This ADT is reversible: for any IDs A and B, $\mathsf{ADT}((A, B)).\mathsf{state} = B$. (We note that this ADT trivially admits a SHI implementation.)

Consider two sequences of IDs (A, C) and (B, C), where $A \neq B$. If this ADT supports logical deletion, then we derive a contradiction:

$$(A, \bot) = \mathsf{ADT}((A, C, \mathsf{delete}(C))) = \mathsf{ADT}((C, \mathsf{delete}_C))$$
$$= \mathsf{ADT}((B, C, \mathsf{delete}(C))) = (B, \bot),$$

where the first and last equalities follow from logical deletion and the second and third follow by construction. This counter example illustrates that deletion may require the data controller to keep sufficient records of all users to update the state. $\square$

We now prove that SHI implies deletion-as-control. We restate the theorem for convenience.

Theorem 3.7. *For any ADT that supports logical deletion and any SHI implementation* Impl*, the controller* $\mathsf{C} = \mathsf{C_{Impl}}$ *satisfies* $(0, 0)$*-deletion-as-control with the simulator that outputs* $R'_\mathsf{C} = R_\mathsf{C}$*.*

Proof of Theorem 3.7. Fix C and any E and Y in the deletion-as-control game. By the definition of logical deletion, the sequence $\vec{\mathsf{q}}$ in the real world and $\vec{\mathsf{q}}'$ in the ideal world are logically equivalent. Let Sim be the simulator that always outputs $R'_\mathsf{C} = R_\mathsf{C}$. By Theorem C.2, the real-world and ideal-world states are identical (since the randomness and the logical states are identical). So, we satisfy the first condition of deletion-as-control. Furthermore, since the value of R'_C equals the value of R_C, they are identically distributed. Thus, this controller satisfies $(0, 0)$-deletion-as-control. $\square$