A preliminary version of this paper appears in the proceedings of Pubic-Key Cryptography 2015. This is the full version.

Interactive message-locked encryption and secure deduplication

Mihir Bellare[1]   Sriram Keelveedhi[2]

January 21, 2015

Abstract

This paper considers the problem of secure storage of outsourced data in a way that permits deduplication. We are for the first time able to provide privacy for messages that are both correlated and dependent on the public system parameters. The new ingredient that makes this possible is interaction. We extend the message-locked encryption (MLE) primitive of prior work to interactive message-locked encryption (iMLE) where upload and download are protocols. Our scheme, providing security for messages that are not only correlated but allowed to depend on the public system parameters, is in the standard model. We explain that interaction is not an extra assumption in practice because full, existing deduplication systems are already interactive.

Keywords: deduplication, cloud storage, message-locked encryption.

[1] Department of Computer Science & Engineering, University of California San Diego, 9500 Gilman Drive, La Jolla, California 92093, USA. Email: `mihir@eng.ucsd.edu`. URL: `http://cseweb.ucsd.edu/~mihir/`. Supported in part by NSF grants CNS-1228890 and CNS-1116800.

[2] Work done while author was at UCSD, supported in part by NSF grants CNS-1228890 and CNS-1116800. Email: `sriramkr@cs.ucsd.edu`. URL: `http://www.cs.ucsd.edu/users/skeelvee/`.

Contents

1 Introduction

The secure deduplication problem. Cloud storage providers such as Google, Dropbox and NetApp [31, 41, 51] derive significant cost savings from what is called *deduplication*. This means that if Alice and Bob upload the same data m, the service provider stores only one copy that is returned to Alice and Bob upon download.

Enter security, namely the desire of clients to keep their data private from the server. Certainly, Alice and Bob can conventionally encrypt their data under their passwords and upload the ciphertext rather than the plaintext. But then, even if they start from the same data m, they will end up with different ciphertexts C_A, C_B, foiling deduplication. The corresponding cost increase for the server would ultimately be passed to the clients in higher storage fees. It is thus in the interest of the parties to cooperate towards storage that is secure but deduplicatable.

Douceur et al. [30] provided the first solution, called convergent encryption (CE). The client encrypts its plaintext m with a *deterministic* symmetric encryption scheme under a k that is itself derived as a deterministic hash of the plaintext m. If Alice and Bob start with the same m, they will arrive at the same ciphertext, and thus deduplication is possible. Despite lacking an analysis until recently [11], CE has long been used in research and commercial systems [2, 4, 5, 17, 25, 26, 28, 35, 39, 46, 47, 52, 54], an indication of practitioners' interest in secure deduplication.

MLE. Bellare, Keelveedhi and Ristenpart (BKR) [11] initiated a theoretical treatment of secure deduplication aimed in particular at answering questions like, what security does CE provide and what can one prove about it? To this end they defined a primitive they called message-locked encryption (MLE). An MLE scheme specifies algorithms $\mathsf{K}, \mathsf{E}, \mathsf{D}, \mathsf{T}$. To encrypt m, let $k \leftarrow\!\$\ \mathsf{K}(p, m)$, where p is a system-wide public parameter, and return ciphertext $c \leftarrow\!\$\ \mathsf{E}(k, m)$. Decryption $m \leftarrow \mathsf{D}(k', c)$ recovers m as long as $k' \leftarrow\!\$\ \mathsf{K}(p, m)$ is any key derived from m. Tags, produced via $t \leftarrow \mathsf{T}(c)$, are a way to test whether the plaintexts underlying two ciphertexts are the same or not, all encryptions of m having the same tag but it being hard to find differing plaintexts with matching tags.

Any MLE scheme enables deduplication. Alice, having m_A, computes and retains a key $k_A \leftarrow\!\$\ \mathsf{K}(p, m_A)$ and uploads $c_A \leftarrow\!\$\ \mathsf{E}(k, m_A)$. The server stores c_A. Now Bob, having m_B, computes and retains a key $k_B \leftarrow\!\$\ \mathsf{K}(p, m_B)$ and uploads $c_B \leftarrow\!\$\ \mathsf{E}(k, m_B)$. If the tags of c_A and c_B match, which means $m_A = m_B$, then the server deduplicates, storing only c_A and returning it to both Alice and Bob upon a download request. Both can decrypt to recover the common plaintext. CE is a particular MLE scheme in which key generation is done by hashing the plaintext.

MLE security. BKR [11] noted that MLE can only provide security for unpredictable data. (In particular, it cannot provide semantic security.) Within this range, two data dimensions emerge:

1. Correlation: Security holds even when messages being encrypted, although individually unpredictable, are related to each other.
2. Parameter-dependence: Security holds for messages that depend on the public parameters.

These dimensions are orthogonal, and the best would be security for correlated, parameter-dependent messages. This has not been achieved. What we have is schemes for correlated but parameter-independent messages [10, 11] and for non-correlated but parameter-dependent messages [1]. This past work is summarized in Figure 1 and we now discuss it in a little more detail.

Prior schemes. The definition of BKR [11], following [6], was security for correlated but parameter-independent messages. For this notion they proved security of CE in the ROM, gave new, secure ROM schemes, and made partial progress towards the challenging task of security without ROs. An efficient scheme in the standard model, also for correlated but parameter-independent messages, was provided in [10] assuming UCE-secure hash functions. (Specifically, against statistically unpredictable sources.)

Abadi, Boneh, Mironov, Raghunathan and Segev (ABMRS) [1] initiated treatment of security for parameter dependent messages, which they termed lock-dependent security. Achieving this is

Scheme(s)	**Type**	**Messages**		**STD/ROM**
		Correlated	Parameter-dependent	
CE, HCE1, HCE2, RCE [11]	MLE	Yes	No	ROM
XtDPKE, XtESPKE, ... [11]	MLE	Yes	No	STD
BHK [10]	MLE	Yes	No	STD
ABMRS [1]	MLE	No	Yes	RO
FCHECK	iMLE	**Yes**	**Yes**	**STD**

Figure 1: **Features of prior schemes (first four rows) and our scheme (last row).** We achieve security for the first time for messages that are *both* correlated *and* parameter dependent. Our scheme is in the standard model. The advance is made possible by exploiting interaction.

challenging. They gave a ROM solution that uses NIZK proofs to provide proofs of consistency. But to achieve security for parameter-dependent messages they were forced to sacrifice security for correlated messages. Their result assumes messages being encrypted are independently distributed.

Questions and goals. The question we pose and address in this paper is, is it possible to achieve the best of both worlds, meaning security for messages that are both correlated *and* parameter dependent? This is important in practice. As indicated above, schemes for secure deduplication are currently deployed and in use in many systems [2, 4, 5, 17, 25, 26, 28, 35, 39, 46, 47, 52, 54]. In usage, messages are very likely to be correlated. For example, suppose Alice has uploaded a ciphertext c encrypting a paper m she is writing. She edits m to m', and uploads the new version. The two plaintexts m, m' could be closely related, differing only in a few places. Also, even if messages of honest users are unlikely to depend on system parameters, attackers are not so constrained. Lack of security for parameter-dependent messages could lead to breaches. This is reflected for example in the BEAST attack on CBC in SSL/TLS [32]. We note that the question of achieving security for messages that are both correlated and parameter dependent is open both in the ROM and in the standard model.

Contributions in brief. We answer the above questions by providing a deduplication scheme secure for messages that are both correlated and parameter dependent. Additionally, our scheme is standard-model, not ROM. The key new ingredient is interaction. In our solutions, upload and download are interactive protocols between the client and server. To specify and analyze these protocols, we define a new primitive, interactive MLE or iMLE. We provide a syntax and definitions of security, then specify and prove correct our protocols.

iMLE turns out to be interesting in its own right and yields some other benefits. We are able to provide the first secure deduplication scheme that permits incremental updates. This means that if a client's message changes only a little, for example due to an edit to a file, then, rather than create and upload an entirely new ciphertext, she can update the existing one with communication cost proportional only to the distance between the new and old plaintexts. This is beneficial because communication is a significant fraction of the operating expenditure in outsourced storage services. For example, transferring one gigabyte to the server costs as much storing one gigabyte for a month or longer in popular storage services [3, 40, 49]. In particular, backup systems, an important use case for deduplication, are likely to benefit, as the operations here are incremental by nature. Incremental cryptography was introduced in [8, 9] and further studied in [13, 21, 34, 50].

Interaction? One might question the introduction of interaction. Isn't a non-interactive solution preferable? Our answer is that we don't "introduce" interaction. It is already present. Upload and download in real systems is inherently and currently interactive, even in the absence of security. MLE is a cryptographic core, not a full deduplication system. If MLE is used for secure deduplication, the uploads and downloads will be interactive, even though MLE is not, due to extra flows that the full system requires. Interaction being already present, it is natural to exploit it for security. In doing so, we are taking advantage of an existing resource rather than introducing an entirely new one.

MLE considered a single client. But in a full deduplication system, there are multiple clients concurrently executing uploads and downloads. Our iMLE model captures this. iMLE is thus going further towards providing security of the full system rather than just a cryptographic core. We know from experience that systems can fail in practice even when a "proven-secure" scheme is used if the security model does not encompass the full range of attacker capabilities or security goals of the implementation. Modeling that penetrates deeper into the system, as with iMLE, increases assurance in practice.

We view iMLE as a natural extension of MLE. The latter abstracted out an elegant primitive at the heart of the secure deduplication problem that could be studied in isolation. We study the full deduplication system, leveraging MLE towards full solutions with added security features.

Duplicate faking. In a duplicate faking attack, the adversary concocts and uploads a perverse ciphertext c^* with the following property. When honest Alice uploads an encryption c of her message m, the server's test (wrongly) indicates that the plaintexts underling c^*, c are the same, so it discards c, returning c^* to Alice upon a download request. But when Alice decrypts c^*, she does not get back her original plaintext.

Beyond privacy, BKR [11] defined an integrity requirement for MLE called tag consistency whose presence provides security against duplicate faking attacks. The important tag consistency property is possessed by the prior MLE schemes of Figure 1 and also by our new iMLE schemes.

Deterministic schemes provide tag consistency quite easily and naturally. But ABMRS [1] indicate that security for parameter-dependent messages requires randomization. Tag consistency now becomes challenging to achieve. Indeed, providing it accounts for the use of NIZKs and the corresponding cost and complexity of the ABMRS scheme [1].

In the interactive setting, we capture the requirement underlying tag consistency by a recovery condition that is part of our soundness definition and requirement. Soundness in particular precludes duplicate faking attacks in the interactive setting. Our scheme provides soundness, in addition to privacy for messages that are both correlated and parameter dependent. Our FCHECK solution uses composable point function obfuscation [16] and FHE [18–20, 27, 36, 38, 55].

Closer look. We look in a little more detail at the main definitional and scheme contributions of our work.

Public parameters for an iMLE scheme are created by an Init algorithm. Subsequently, a client can register (Reg), upload (Put) and download (Get). Incremental schemes have an additional update (Upd). All these are interactive protocols between client and server. For soundness, we ask that deduplication happens as expected and that clients can recover their uploaded files even in the presence of an attacker which knows all the files being uploaded and also read the server's storage at any moment. The latter condition protects against duplicate-faking attacks. Our security condition is modeled on that of BKR [11] and requires privacy for correlated but individually unpredictable messages that may depend on the public parameters.

Our FCHECK construction, described and analyzed in Section 4, achieves soundness as well as privacy for messages that are both correlated and parameter dependent, all in the standard model, meaning without recourse to random oracles. The construction builds on a new primitive we call MLE-Without-Comparison (MLEWC). As the name indicates, MLEWC schemes are similar to MLE schemes in syntax and functionality, except that they do not support comparison between ciphertexts. We show that MLEWC can be realized in the standard model, starting from point function obfuscation [16] or, alternatively, UCE-secure hash function families [10]. However, comparison is essential to enable deduplication. To enable comparison, FCHECK employs an interactive protocol using a fully homomorphic encryption (FHE) scheme [18–20, 27, 36, 38, 55], transforming the MLEWC scheme into an iMLE scheme.

We then move on to the problem of incremental updates. Supporting incremental updates over

```
$\underline{\mathsf{Run}(1^\lambda, \mathsf{P}, \mathsf{inp})}$
$n \leftarrow 1$; $i \leftarrow 1$; $\mathtt{M} \leftarrow \epsilon$
$\mathbf{a}[1,1] \leftarrow \mathsf{inp}[1]$; $\mathbf{a}[2,1] \leftarrow \mathsf{inp}[2]$
While $\mathtt{T}[n] = \mathsf{False}$
  $(\mathbf{a}[n, i+1], \mathtt{M}, \mathtt{T}[n]) \leftarrow\!\$\ \mathsf{P}[n,i](1^\lambda, \mathbf{a}[n,i], \mathtt{M})$
  If $n = 2$ then $n \leftarrow 1$; $i \leftarrow i+1$ Else $n \leftarrow 2$
Ret $\mathtt{last}(\mathbf{a}[1]), \mathtt{last}(\mathbf{a}[2])$
```

```
$\underline{\mathsf{Msgs}(1^\lambda, \mathsf{P}, \mathsf{inp}, r)}$
$n \leftarrow 1$; $i \leftarrow 1$; $j \leftarrow 1$; $\mathbf{a}[1,1] \leftarrow \mathsf{inp}[1]$
$\mathbf{a}[2,1] \leftarrow \mathsf{inp}[2]$; $\mathtt{M} \leftarrow \epsilon$
While $\mathtt{T}[n] = \mathsf{False}$
  $(\mathbf{a}[n, i+1], \mathtt{M}, \mathtt{T}[n]) \leftarrow\!\$\ \mathsf{P}[n,i](1^\lambda, \mathbf{a}[n,i], \mathtt{M}; r[n,i])$
  If $n = 2$ then $n \leftarrow 1$; $i \leftarrow i+1$ Else $n \leftarrow 2$
  $\mathbf{M}[j] \leftarrow \mathtt{M}$; $j \leftarrow j+1$
Ret $\mathbf{M}$
```

Figure 2: **Left**: Running a two player protocol P. **Right**: The Msgs procedure returns the messages exchanged during the protocol when invoked with specified inputs and coins.

MLE schemes turns out to be challenging: deterministic MLE schemes cannot support incremental updates, as we show in Appendix B, while randomized MLE schemes seem to need complex machinery such as NIZK proofs of consistency [1] to support incremental updates while retaining the same level of security as deterministic schemes, which makes them unfit for practical usage. We show how interaction can be exploited to solve this problem. We describe an efficient ROM scheme IRCE that supports incremental updates. The scheme, in its simplest form, works like the randomized convergent encryption (RCE) scheme [11], where the message is encrypted with a random key using a blockcipher in counter (CTR) mode, and the random key is encrypted with a key derived by hashing the message. We show that this indirection enables incremental updates. However, RCE does not support strong tag consistency and hence cannot offer strong security against duplicate faking attacks. We overcome this in IRCE by including a simple response from the server as part of the upload process. We remark that IRCE is based off a core MLE (non-interactive) scheme permitting incremental updates, interaction being used only for tag consistency.

2 Preliminaries

We let $\lambda \in \mathbb{N}$ and 1^λ denote the security parameter and its unary representation. The empty string is denoted by ϵ. We let $|S|$ denote the size of a finite set S and let $s \leftarrow\!\$\ S$ denote sampling an element from S at random and assigning it to s. If $a, b \in \mathbb{N}$ and $a < b$, then $[a]$ denotes the set $\{1, \ldots, a\}$ and $[a, b]$ denotes the set $\{a, \ldots, b\}$. For a tuple $\mathbf{x}$, we let $|\mathbf{x}|$ denote the number of components in $\mathbf{x}$, and $\mathbf{x}[i]$ denote the i-th component, and $\mathtt{last}(\mathbf{x}) = \mathbf{x}[|\mathbf{x}|]$, and $\mathbf{x}[i,j] = \mathbf{x}[i] \ldots \mathbf{x}[j]$ for $1 \leq i \leq j \leq |\mathbf{x}|$. A binary string s is identified with a tuple over $\{0,1\}$. The guessing probability of a random variable X, denoted by $\mathbf{GP}(X)$, is defined as $\mathbf{GP}(X) = \max_x \Pr[X = x]$. The conditional guessing probability $\mathbf{GP}(X \mid Y)$ of a random variable X given a random variable Y are defined via $\mathbf{GP}(X \mid Y) = \sum_y \Pr[Y = y] \cdot \max_x \Pr[X = x | Y = y]$.

The Hamming distance between $s_1, s_2 \in \{0,1\}^\ell$ is given by $\mathsf{HAMM}(s_1, s_2) = \sum_{i=1}^{\ell} (s_1[i] \oplus s_2[i])$. We let $\mathsf{diff}_{\mathsf{HAMM}}(s_1, s_2) = \{i : s_1[i] \neq s_2[i]\}$ and $\mathsf{patch}_{\mathsf{HAMM}}(s_1, \delta)$ be the string s such that $s[i] = s_1[i]$ if $i \notin \delta$ and $s[i] = \neg s_1[i]$ if $i \in \delta$.

Algorithms are randomized and run in polynomial time (denoted by PT) unless otherwise indicated. We let $y \leftarrow A(a_1, \ldots; r)$ denote running algorithm A on $a_1, \ldots$ with coins r and assigning the output to y, and let $y \leftarrow\!\$\ A(a_1, \ldots)$ denote the same operation with random coins. We let $[A(a_1, \ldots)]$ denote the set of all y that have non-zero probability of being output by A on inputs $a_1, \ldots$. Adversaries are either algorithms or tuples of algorithms. A negligible function f approaches zero faster than the polynomial reciprocal; for every polynomial p, there exists $n_p \in \mathbb{N}$ such that $f(n) \leq 1/p(n)$ for all $n \geq n_p$.

We use the code-based game playing framework of [15] along with extensions of [53] and [11] when specifying security notions and proofs.

A two player q-round protocol P is represented through a $2 \times q$-tuple $(\mathsf{P}[i,j])_{i\in[2],j\in[q]}$ of algorithms where $\mathsf{P}[i,j]$ represents the action of the i-th player invoked for the j-th time. We let $\mathsf{P}[1]$ denote the player who initiates the protocol, and $\mathsf{P}[2]$ denote the other player. Each algorithm is invoked with 1^λ, an input $\mathbf{a}$, and a message $\mathtt{M} \in \{0,1\}^*$, and returns a 3-tuple consisting of an output $\mathbf{a}'$, an outgoing message $\mathtt{M}' \in \{0,1\}^*$, and a boolean $\mathtt{T}$ to indicate termination. The Run algorithm (Figure 2) captures the execution of P, and Msgs (Figure 2) returns the messages exchanged in an instance of P, when invoked with specified inputs and coins.

Adversarial model. A secure deduplication system (built from an iMLE scheme) will operate in a setting with a server and several clients. Some clients will be controlled by an attacker, while others will be legitimate, belonging to honest users and following the protocol specifications. A resourceful attacker, apart from controlling clients, could gain access to server storage, and interfere with communications. Our adversarial model captures an iMLE scheme running in the presence of an attacker with such capabilities.

We now walk through an abstract game G, and explain how this is achieved. The games in the rest of the paper, for soundness, security, and other properties of iMLE largely follow this structure. The game G sets up and controls a server instance. The adversary A is invoked with access to a set of procedures. Usually, the objective of the game involves A violating some property guaranteed to legitimate clients like L, such as ability to recover stored files, or privacy of data.

The Msg procedure can send arbitrary messages to the server and can be used to create multiple clients, and run multiple instances of protocols, which could deviate from specifications.

The Init and Step procedures control a single legitimate client L. The Init procedure starts protocol instances on behalf of L, using inputs of A's choice. The Step procedure advances a protocol instance by running the next algorithm. Together, these procedures let A run several legitimate and corrupted protocol instances concurrently.

The State procedure returns the server's state, which includes stored ciphertexts, public parameters, etc.. In some games, it also returns the state and parameters of L. State provides only read access to the server's storage. This restriction is necessary. If A is allowed to modify the storage of the server, then it can always tamper with the data stored by the clients, making secure deduplication impossible.

We assume that A can read, delay and drop messages between the server and legitimate clients. However, A cannot tamper with message contents, reorder messages within a protocol, or redirect messages from one protocol instance to another. This assumption helps us simplify the protocol descriptions and proofs. Standard, efficient techniques can be used to transform the protocols from this setting to be secure in the presence of an attacker that can tamper and reorder messages [7].

3 Interactive message-locked encryption

Definition. An interactive message-locked encryption scheme iMLE consists of an initialization algorithm Init and three protocols $\mathsf{Reg}, \mathsf{Put}, \mathsf{Get}$. Initialization Init sets up server-side state: $\sigma_S \leftarrow_{\$} \mathsf{Init}(1^\lambda)$. Each protocol P consists of two players - a client $\mathsf{P}[1]$ (meaning that the client always initiates), and a server $\mathsf{P}[2]$. All server-side algorithms $\mathsf{P}[2,\cdot]$ take server-side state σ_S as input, and produce an updated state σ_S' as output. The Reg protocol registers new users; here, $\mathsf{Reg}[1]$ takes no input and returns client parameters $\sigma_C \in \{0,1\}^*$. The Put protocol stores files on the server; here, $\mathsf{Put}[1]$ takes plaintext $m \in \{0,1\}^*$ and σ_C as inputs, and outputs an identifier $f \in \{0,1\}^*$. The Get protocol retrieves files from the server; here, $\mathsf{Get}[1]$ takes identifier f and σ_C as inputs, and outputs plaintext $m \in \{0,1\}^*$.

Soundness. We require two conditions. First is deduplication, meaning that if a client puts a ciphertext of a file already on the server, then the storage should not grow by the size of the file.

```
$\underline{\textsc{Main}(1^\lambda)}$   // $\textsc{Rec}^{\mathsf{A}}_{\mathsf{iMLE}}(1^\lambda)$
$\mathsf{win} \leftarrow \mathsf{False}$; $\sigma_S \leftarrow\!\$\ \mathsf{Init}(1^\lambda)$
$\mathsf{A}^{\textsc{Reg},\textsc{Init},\textsc{Step},\textsc{Msg},\textsc{State}}(1^\lambda)$; Ret $\mathsf{win}$

$\underline{\textsc{Reg}}$   // Set up the legitimate client $L$.
$(\sigma_C, \sigma_S) \leftarrow\!\$\ \mathsf{Run}(\mathsf{Reg}, \epsilon, \sigma_S)$

$\underline{\textsc{Init}(\mathsf{P}, \mathsf{inp})}$   // Start a protocol with $L$.
If $\mathsf{P} \notin \{\mathsf{Put}, \mathsf{Get}\}$ then ret $\perp$
$\mathsf{p} \leftarrow \mathsf{p} + 1$; $j \leftarrow \mathsf{p}$; $\mathbf{PS}[j] = \mathsf{P}$
$\mathbf{a}[j, 1] \leftarrow \mathsf{inp}$; $\mathtt{N}[j] \leftarrow 1$; $\mathtt{M}[j] \leftarrow \epsilon$; Ret $j$

$\underline{\textsc{Msg}(\mathsf{P}, i, \mathtt{M})}$   // Send a message to the server.
If $\mathsf{P} \notin \{\mathsf{Reg}, \mathsf{Put}, \mathsf{Get}, \mathsf{Upd}\}$ then ret $\perp$
$(\sigma_S, \mathtt{M}, \mathtt{N}, \mathtt{T}) \leftarrow\!\$\ \mathsf{P}[2, i](1^\lambda, \sigma_S, \mathtt{M})$; Ret $\mathtt{M}$
```

```
$\underline{\textsc{Step}(j)}$   // Advance an instance by one step.
$\mathsf{P} \leftarrow \mathbf{PS}[j]$; $n \leftarrow \mathtt{N}[j]$; $i \leftarrow \mathtt{rd}[j]$
If $\mathtt{T}[j, n]$ then return $\perp$
If $n = 2$ then $\mathsf{inp} \leftarrow \sigma_S$ else $\mathsf{inp} \leftarrow \mathbf{a}[j, i]$
$(\mathsf{outp}, \mathtt{M}[j], \mathtt{T}[j, n]) \leftarrow\!\$\ \mathsf{P}[n, i](1^\lambda, \mathsf{inp}, \mathtt{M}[j])$
If $n = 2$ then
  $\sigma_S \leftarrow \mathsf{outp}$; $\mathtt{N}[j] \leftarrow 1$; $\mathtt{rd}[j] \leftarrow \mathtt{rd}[j] + 1$
Else $\mathbf{a}[j, i + 1] \leftarrow \mathsf{outp}$; $\mathtt{N}[j] \leftarrow 2$
If $\mathtt{T}[j, 1] \wedge \mathtt{T}[j, 2]$ then $\textsc{WinCheck}(j)$
Ret $\mathtt{M}[j]$

$\underline{\textsc{WinCheck}(j)}$   // Check if $\mathsf{A}$ has won.
If $\mathbf{PS}[j] = \mathsf{Put}$ then
  $(\sigma_C, m) \leftarrow \mathbf{a}[j, 1]$; $f \leftarrow \mathtt{last}(\mathbf{a}[j])$; $T[f] \leftarrow m$
If  $\mathbf{PS}[j] = \mathsf{Get}$ then
  $(\sigma_C, f) \leftarrow \mathbf{a}[j, 1]$; $m' \leftarrow \mathtt{last}(\mathbf{a}[j])$
  $\mathsf{win} \leftarrow \mathsf{win} \vee (m' \neq T[f])$
```

Figure 3: The Rec game. The State procedure returns σ_S, σ_C.

A small increase towards book-keeping information, that is independent of the size of the file, is permissible. More precisely, there exists a bound $\ell : \mathbb{N} \to \mathbb{N}$ such that for all server-side states $\sigma_S \in \{0,1\}^*$, for all valid client parameters (derived through Reg with fresh coins) σ_C, σ'_C, for all $m \in \{0,1\}^*$, the expected increase in size of σ''_S over σ'_S when $(f', \sigma'_S) \leftarrow\!\$\ \mathsf{Run}(\mathsf{Put}, (\sigma_C, m), \sigma_S)$ and $(f', \sigma''_S) \leftarrow\!\$\ \mathsf{Run}(\mathsf{Put}, (\sigma'_C, m), \sigma'_S)$ is bounded by $\ell(\lambda)$.

The second condition is correct recovery of files: if a legitimate client puts a file on the server, it should be able to get the file later. We formalize this requirement by the Rec game of Figure 3, played with an adversary A, which gets access to procedures $\textsc{Reg}, \textsc{Init}, \textsc{Step}, \textsc{Msg}, \textsc{State}$. We provide an overview of these procedures here.

The Reg procedure sets up a legitimate client L by running $\mathsf{Run}(1^\lambda, \mathsf{Reg}, (\epsilon, \sigma_S))$. The Init procedure lets A run protocols on behalf of L. It takes input inp and P, where P has to be one of $\mathsf{Put}, \mathsf{Get}$, and inp should be the a valid input for $\mathsf{P}[1, 1]$. A new instance of P is set up, and A is returned $j \in \mathbb{N}$, an index to the instance. The Step procedure takes input j, advances the instance by one algorithm unless the current instance has terminated. The outgoing message is returned to A. The inputs and outputs of the protocol steps are all stored in an array $\mathbf{a}$. The State procedure returns σ_S, σ_C.

If an instance j has terminated, then Step runs WinCheck, which maintains a table T. If j is an instance of Put, then m and identifier f are recovered from $\mathbf{a}[j]$ and $T[f]$ gets m. If j is an instance of Get, then WinCheck obtains f and the recovered plaintext m', and checks if $T[f] = m'$. If this fails, either because $T[f]$ is some value different from m', or is undefined, then WinCheck sets the win flag, which is the condition for A to win the game. We associate advantage $\mathsf{Adv}^{\mathsf{rec}}_{\mathsf{iMLE},\mathsf{A}}(\lambda) = \Pr[\textsc{Rec}^{\mathsf{A}}_{\mathsf{iMLE}}(1^\lambda)]$ with iMLE and A. For recovery correctness, we require that the advantage should be negligible for all PT A.

Security. The primary security requirement for iMLE schemes is privacy of unpredictable data. Unpredictability (plaintexts drawn from a distribution with negligible guessing probability) is a prerequisite for privacy in MLE schemes [11], as without unpredictability, a simple brute-force attack can recover the contents of a ciphertext by generating keys from all candidate plaintexts and checking if decrypting the ciphertext with the key leads back to the candidate plaintext. A similar argument extends unpredictability as a requirement to secure deduplication schemes as well. We formalize

```
$\textsc{Main}(1^\lambda)$   // $\textsc{Priv}^{\mathsf{S},\mathsf{A}}(1^\lambda)$
    $b \leftarrow\!\$\; \{0,1\}$; $\mathtt{p} \leftarrow 0$; $\sigma_S \leftarrow\!\$\; \mathsf{Init}(1^\lambda)$; $\mathbf{m}_0, \mathbf{m}_1 \leftarrow\!\$\; \mathsf{S}(1^\lambda, \epsilon)$
    $b' \leftarrow\!\$\; \mathsf{A}^{\textsc{Put},\textsc{Upd},\textsc{Step},\textsc{Msg},\textsc{Reg},\textsc{State}}(1^\lambda)$; Ret $(b = b')$

$\textsc{Reg}$
    $(\sigma_C, \sigma_S) \leftarrow\!\$\; \mathsf{Run}(\mathsf{Reg}, \epsilon, \sigma_S)$

$\textsc{Put}(i)$   // Start a Put instance
    $\mathtt{p} \leftarrow \mathtt{p} + 1$; $\mathbf{PS}[\mathtt{p}] = \mathsf{Put}$; $\mathbf{a}[\mathtt{p}, 1] \leftarrow \vec{m}_b[i]$
    $\mathtt{N}[\mathtt{p}] \leftarrow 1$; $\mathtt{M}[\mathtt{p}] \leftarrow \epsilon$; Ret $\mathtt{p}$

$\textsc{State}$
    If $\mathtt{cheat} = \mathsf{False}$ then $\mathsf{done} \leftarrow \mathsf{True}$; ret $\sigma_S$ else ret $\perp$
```

```
$\textsc{Main}(1^\lambda)$   // $\textsc{PDPriv}^{\mathsf{S},\mathsf{A}}(1^\lambda)$
    $b \leftarrow\!\$\; \{0,1\}$; $\mathtt{p} \leftarrow 0$; $\sigma_S \leftarrow\!\$\; \mathsf{Init}(1^\lambda)$
    $b' \leftarrow\!\$\; \mathsf{A}^{\textsc{Ptxt},\textsc{Put},\textsc{Upd},\textsc{Step},\textsc{Msg},\textsc{Reg},\textsc{State}}(1^\lambda)$
    Ret $(b = b')$

$\textsc{Ptxt}(d)$
    $\vec{m}_0, \vec{m}_1 \leftarrow\!\$\; \mathsf{S}(1^\lambda, d)$

$\textsc{Msg}(\mathsf{P}', \mathtt{M})$   // Send a message to the server
    If $\mathsf{P}' \notin \{\mathsf{Reg}, \mathsf{Put}, \mathsf{Get}, \mathsf{Upd}\}$ then ret $\perp$
    $(\sigma_S, \mathtt{M}, \mathtt{T}) \leftarrow\!\$\; \mathsf{P}'[2](1^\lambda, \sigma_S, \mathtt{M})$; Ret $(\sigma_S, \mathtt{M}, \mathtt{N}, \mathtt{T})$

$\textsc{Step}(j)$   // Advance an instance by one step.
    $\mathsf{P} \leftarrow \mathbf{PS}[j]$; $n \leftarrow \mathtt{N}[j]$; $i \leftarrow \mathtt{rd}[j]$
    If $\mathtt{T}[j, n]$ or $\mathsf{done}$ then return $\perp$
    If $n = 2$ then $\mathsf{inp} \leftarrow \sigma_S$ else $\mathsf{inp} \leftarrow \mathbf{a}[j, i]$
    $(\mathsf{outp}, \mathtt{M}[j], \mathtt{T}[j, n]) \leftarrow\!\$\; \mathsf{P}[n, i](1^\lambda, \mathsf{inp}, \mathtt{M}[j])$
    If $n = 2$ then $\sigma_S \leftarrow \mathsf{outp}$; $\mathtt{N}[j] \leftarrow 1$; $\mathtt{rd}[j] \leftarrow \mathtt{rd}[j] + 1$
    Else $\mathbf{a}[j, i+1] \leftarrow \mathsf{outp}$; $\mathtt{N}[j] \leftarrow 2$
    If $n = 1$ and $\mathtt{T}[j, n]$ then $T_f[\mathbf{a}[j, 1]] \leftarrow \mathtt{last}(\mathbf{a}[j])$
```

Figure 4: The PRIV and PDPRIV security games. Apart from MAIN, the games share the same code for all procedures. The PDPRIV game has an additional PTXT procedure.

unpredictability as follows.

A source S is an algorithm that on input 1^λ and a string $d \in \{0,1\}^*$ returns a pair of tuples $(\mathbf{m}_0, \mathbf{m}_1)$. There exist $m : \mathbb{N} \to \mathbb{N}$ and $\ell : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that $|\mathbf{m}_0| = |\mathbf{m}_1| = m(\lambda)$, and $|\mathbf{m}_0[i]| = |\mathbf{m}_1[i]| = \ell(\lambda, i)$ for all $i \in [m(\lambda)]$. All components of $\mathbf{m}_0$ and $\mathbf{m}_1$ are unique. The guessing probability $\mathbf{GP}_{\mathsf{S}}(\lambda)$ of S is defined as $\max_{i,b,d}(\mathbf{GP}(\mathbf{m}_b[i]))$ when $(\mathbf{m}_0, \mathbf{m}_1) \leftarrow\!\$\; \mathsf{S}(1^\lambda, d)$. We say that S is unpredictable if $\mathbf{GP}_{\mathsf{S}}(\cdot)$ is negligible. We say that S is a single source if it only outputs one tuple, but satisfies the other conditions. We say that S is an auxiliary source if it outputs a string $z \in \{0,1\}^*$ along with $\mathbf{m}_0, \mathbf{m}_1$ and if it holds that guessing probability conditioned on z is negligible.

The PRIV game of Figure 4, associated with iMLE, a source S and an adversary A, captures privacy for unpredictable messages independent of the public parameters of the system. As with REC, the game starts by running $\sigma_S \leftarrow\!\$\; \mathsf{Init}(1^\lambda)$ to set up the server-side state. The game then runs S to get $(\mathbf{m}_0, \mathbf{m}_1)$, picks a random bit b, and uses $\mathbf{m}_b$ as messages to be put on the server. Then, A is invoked with access to REG, PUT, STEP, MSG and STATE. The REG, STATE, and MSG oracles behave in the same way as in REC. The STEP oracle here is similar to that of REC, except that it does not invoke WINCHECK. Adversary A can initialize an instance of Put with a plaintext $\mathbf{m}_b[i]$ by calling PUT(i).

We associate advantage $\mathsf{Adv}^{\mathsf{priv}}_{\mathsf{iMLE},\mathsf{S},\mathsf{A}}(\lambda) = 2\Pr[\textsc{Priv}^{\mathsf{S},\mathsf{A}}_{\mathsf{iMLE}}(1^\lambda)] - 1$ with a iMLE a source S and an adversary A. We require that the advantage should be negligible for all PT A for all unpredictable PT S.

The PDPRIV game of Figure 4 extends PRIV-security to messages depending on the public parameters of the system, a notion termed lock-dependent security in [1]. Here, we term this parameter-dependent security. In this game, the adversary A gets access to a PTXT procedure, which runs $\mathsf{S}(1^\lambda, \sigma_S)$ to get $\mathbf{m}_0, \mathbf{m}_1$. The other procedures follow PRIV. A simpler approach is to run S with σ_S when the game starts (i.e. in main) as in PRIV. However, this leads to trivial constructions where Init is a dummy procedure, and the system parameters are generated when the first client registers. This is avoided in PDPRIV by letting A decide, through PTXT, when S is to be run. We associate advantage $\mathsf{Adv}^{\mathsf{ldpriv}}_{\mathsf{iMLE},\mathsf{S},\mathsf{A}}(\lambda) = 2\Pr[\textsc{PDPriv}^{\mathsf{S},\mathsf{A}}_{\mathsf{iMLE}}(1^\lambda)] - 1$ with a scheme iMLE a source S and an adversary A. We

```
$\mathsf{Put}[1]((pk, sk), m)$                                                $\mathsf{Put}[2](\sigma_S, \epsilon)$
$c_f \leftarrow\!\$\ \mathbf{E}_{pk}(m)$
                                          $\xrightarrow{pk, c_f}$        $c_r \leftarrow\!\$\ \mathsf{E_f}(1^\lambda, pk, 0^{\kappa(\lambda)})$; $c_i \leftarrow\!\$\ \mathsf{E_f}(1^\lambda, pk, 0)$
                                                                         $c_n \leftarrow c_i$
                                                                         For $(p, c) \in \mathbf{fil}$ do
                                                                           $c_p \leftarrow\!\$\ \mathsf{E_f}(1^\lambda, pk, p)$; $c_c \leftarrow\!\$\ \mathsf{E_f}(1^\lambda, pk, c)$
                                                                           $c' \leftarrow\!\$\ \mathsf{Ev_f}(1^\lambda, pk, \mathsf{cmp}, c_f, c_p, c_c, c_r, c_n, c_i)$
                                                                           $c_r, c_n, c_i \leftarrow c'$
$p \leftarrow \mathsf{D_f}(1^\lambda, sk, c_r)$; $n \leftarrow \mathsf{D_f}(1^\lambda, sk, c_n)$   $\xleftarrow{c_r, c_n}$
                                          $\xrightarrow{n}$              $(p, c) \leftarrow \mathbf{fil}[n]$
                                          $\xleftarrow{p, c}$
If $n \neq 0$ then $c_1 \leftarrow \epsilon$; $k \leftarrow \mathsf{K}(1^\lambda, p, m)$
If $\mathsf{D}(1^\lambda, k, c) \neq m$ then ret $\perp$
Else
  $p \leftarrow\!\$\ \mathsf{P}(1^\lambda)$; $k \leftarrow \mathsf{K}(1^\lambda, p, m)$
  $c_1 \leftarrow\!\$\ \mathsf{E}(1^\lambda, k, m)$
$c_2 \leftarrow \mathbf{E}_{pk}(k)$
                                          $\xrightarrow{c_1, c_2, p, u, n}$   If $c_1 \neq \epsilon$ then
                                                                           $n_f \leftarrow n_f + 1$; $i \leftarrow n_f$; $\mathbf{fil}[i] \leftarrow (p, c_1)$
                                                                         $c_2 \leftarrow \mathtt{SiffE}(\mathbf{own}, (u, i), c_2)$
If $n \neq 0$ and $i \neq n$ then ret $\perp$ else ret $i$   $\xleftarrow{i}$
```

```
$\mathsf{Reg}[1](\epsilon)$                      $\mathsf{Reg}[2](\sigma_S)$
$(pk, sk) \leftarrow\!\$\ \mathsf{K_f}$   $\xrightarrow{\epsilon}$   $u \leftarrow\!\$\ \{0,1\}^\lambda \setminus \mathbf{U}$
Ret $(pk, sk, u)$   $\xleftarrow{u}$   $\mathbf{U} \leftarrow \mathbf{U} \cup \{u\}$
```

```
$\mathsf{Get}[1]((pk, u, sk), f)$                 $\mathsf{Get}[2](\sigma_S, \epsilon)$
                              $\xrightarrow{u, f}$    $c_2 \leftarrow \mathbf{own}[u, i]$
                                               $(p, c_1) \leftarrow \mathbf{fil}[i]$
If $c_1 = \perp$ then ret $\perp$   $\xleftarrow{c_1, c_2}$   If $c_2 = \perp$ then $c_1 \leftarrow \perp$
$k \leftarrow \mathsf{D_f}(1^\lambda, sk, c_2)$
Ret $\mathsf{D}(1^\lambda, k, c_1)$
```

Figure 5: The FCHECK iMLE scheme over $\mathsf{FHE} = (\mathsf{K_f}, \mathsf{E_f}, \mathsf{D_f}, \mathsf{Ev_f})$ and $\mathsf{MLEWC} = (\mathsf{P}, \mathsf{E}, \mathsf{K}, \mathsf{D})$.

require that advantage should be negligible for all PT A for all unpredictable PT S.

4 The FCHECK scheme

In this section, we describe the the FCHECK construction, which achieves soundness as well as security for messages that are both correlated and parameter-dependent, all in the standard model. As we noted in the introduction, prior to our work, achieving parameter-dependent correlated input security was open even in the random oracle model. We are able to exploit interactivity as a new ingredient to design a scheme that achieves security for parameter-dependent correlated messages.

Our approach starts by going after a new, seemingly weak primitive, one we call MLE-Without-Comparison (MLEWC). As the name indicates, MLEWC schemes are similar to MLE schemes in syntax and functionality, except that they do not support comparison between ciphertexts. We show that MLEWC can be realized in the standard model, starting from point function obfuscation [16] or, alternatively, UCE-secure hash function families [10]. However, comparison is essential to enable deduplication. To enable comparison, FCHECK employs an interactive protocol using a fully homomorphic encryption (FHE) scheme [18–20, 27, 36, 38, 55], transforming the MLEWC scheme into an iMLE scheme. We view FCHECK as a theoretical construction, and not an immediately practical iMLE scheme.

MLE Without Comparison (MLEWC). A scheme $\mathsf{MLEWC} = (\mathsf{P}, \mathsf{E}, \mathsf{K}, \mathsf{D})$ consists of four algorithms. Parameters are generated via $p \leftarrow\!\$\ \mathsf{P}(1^\lambda)$. Keys are generated via $k \leftarrow\!\$\ \mathsf{K}(1^\lambda, p, m)$, where

$m \in \{0,1\}^{\mu(\lambda)}$ is the plaintext. Encryption E takes p, k, m and returns a ciphertext $c \leftarrow\!\!\$\ \mathsf{E}(1^\lambda, k, m)$. Decryption D takes input k, c and returns $m \leftarrow \mathsf{D}(1^\lambda, k, c)$, or $\perp$. Correctness requires that $\mathsf{D}(1^\lambda, k, c) = m$ for all $k \in [\mathsf{K}(1^\lambda, p, m)]$, for all $c \in [\mathsf{E}(1^\lambda, k, m)]$, for all $p \in [\mathsf{P}(1^\lambda)]$, for all $m \in \{0,1\}^{\kappa(\lambda)}$ for all $\lambda \in \mathbb{N}$.

The WPRIV game with MLEWC, an auxiliary source S and an adversary A is described in Figure 17. The game runs S to get two vectors $\mathbf{m}_0, \mathbf{m}_1$, and forms $\mathbf{c}$ by encrypting one of the two vectors, using a fresh parameter for each component, or by picking random strings. A should guess which the case is. We associate advantage $\mathsf{Adv}^{\mathsf{wpriv}}_{\mathsf{MLEWC,S,A}}(\lambda) = 2\Pr[\mathrm{WPRIV}^{\mathsf{A,S}}_{\mathsf{MLEWC}}(1^\lambda)] - 1$. For MLEWC to be WPRIV-secure, advantage should be negligible for all PT adversaries A for all unpredictable PT auxiliary sources S. Note that unlike PRIV, here, a fresh parameter is picked for each encryption, and although we will end up using WPRIV-secure schemes to build parameter-dependent iMLE, in the WPRIV game, the source S is not provided the parameters.

Fully homomorphic encryption (FHE) [36]. An FHE scheme $\mathsf{FHE} = (\mathsf{K_f}, \mathsf{E_f}, \mathsf{D_f}, \mathsf{Ev_f})$ is a 4-tuple of algorithms. Key generation returns $(pk, sk) \leftarrow\!\!\$\ \mathsf{K_f}(1^\lambda)$, encryption takes pk, plaintext $m \in \mathbf{M}(\lambda)$, and returns ciphertext $c \leftarrow\!\!\$\ \mathsf{E_f}(1^\lambda, pk, m)$, and decryption returns $m' \leftarrow \mathsf{D_f}(1^\lambda, sk, c)$ on input sk and ciphertext c, where $m = \perp$ indicates an error. The set of valid ciphertexts is denoted by $\mathbf{C}(\lambda) = \{c : \mathsf{D_f}(1^\lambda, sk, c) \neq \perp, (pk, sk) \in [\mathsf{K_f}(1^\lambda)]\}$. Decryption correctness requires that $\mathsf{D_f}(1^\lambda, sk, \mathsf{E_f}(1^\lambda, pk, m)) = m$ for all $(pk, sk) \in [\mathsf{K_f}(1^\lambda)]$, for all $m \in \mathbf{M}(\lambda)$ for all $\lambda \in \mathbb{N}$.

Let $\langle . \rangle$ denote an encoding which maps boolean circuits f to strings denoted by $\langle f \rangle$ such that there exists PT Eval which satisfies $\mathsf{Eval}(\langle f \rangle, x) = f(x)$ for every valid input $x \in \{0,1\}^n$, where n is the input length of f. Evaluation $\mathsf{Ev_f}$ takes input a public key pk, a circuit encoding $\langle f \rangle$ and a tuple of ciphertexts $\mathbf{c}$ such that $|\mathbf{c}|$ is the input length of f and returns $c' \leftarrow\!\!\$\ \mathsf{Ev_f}(1^\lambda, pk, \langle f \rangle, \mathbf{c})$. Evaluation correctness requires that for random keys, on all functions and all inputs, $\mathsf{Ev_f}$ must compute the correct output when run on random coins, except with negligible error. More precisely, for all boolean circuits f, when $(pk, sk) \leftarrow\!\!\$\ \mathsf{K_f}(1^\lambda)$ and $c' \leftarrow\!\!\$\ [\mathsf{Ev_f}(1^\lambda, pk, \langle f \rangle, \mathbf{c})]$, if $|\mathbf{c}|$ is the input length of f, then it holds that the probability that $\mathsf{Eval}(f, \mathbf{x}) \neq y$ where $y \leftarrow \mathsf{D_f}(1^\lambda, sk, c')$ and $\mathbf{x}[i] \leftarrow \mathsf{D_f}(1^\lambda, sk, \mathbf{c}[i])$ is negligible for all $\mathbf{c}[1], \ldots, \mathbf{c}[|\mathbf{c}|] \in \mathbf{C}(\lambda)^{|\mathbf{c}|}$.

The FCHECK scheme. Let $\mathsf{FHE} = (\mathsf{K_f}, \mathsf{E_f}, \mathsf{D_f}, \mathsf{Ev_f})$ be an FHE scheme, and let $\mathsf{MLEWC} = (\mathsf{P}, \mathsf{E}, \mathsf{K}, \mathsf{D})$ be a MLEWC scheme where K is deterministic. The $\mathsf{FCHECK[FHE, MLEWC]}$ iMLE scheme is described in Figure 5. The Init algorithm is omitted: it lets $\mathbf{U} \leftarrow \emptyset$, and lets $\mathbf{fil}$ and $\mathbf{own}$ be empty tables.

In FCHECK, clients encrypt their plaintexts with MLEWC to be stored on the server, but pick a fresh parameter each time. The server's storage consists of a list of ciphertext-parameter pairs $\mathbf{c}[i], \mathbf{p}[i]$. When a client wants to put m, for each such $\mathbf{c}[i], \mathbf{p}[i]$, the server should generate a key $\mathbf{k}[i] \leftarrow \mathsf{K}(1^\lambda, \mathbf{p}[i], m)$ and check if $\mathsf{D}(1^\lambda, \mathbf{k}[i], \mathbf{c}[i]) = m$.

A match means that a duplicate ciphertext already exists on the server, while no match means that m is a fresh plaintext. The search for a match should be carried without the server learning m and is hence done over FHE ciphertexts of the components. The client sends pk and $c_f \leftarrow\!\!\$\ \mathsf{E_f}(1^\lambda, pk, m)$ and the server encrypts each $\mathbf{c}[i], \mathbf{p}[i]$ to get c_c and c_p and runs $\mathsf{Ev_f}$ on the cmp circuit described below with these values.

```
cmp(m, p, c, r, n, i)
  If $\mathsf{D}(1^\lambda, \mathsf{K}(1^\lambda, p, m), c) = m$ and $r = 0^{\kappa(\lambda)}$ then return $p, i+1, i+1$
  Else return $r, n, i+1$
```

The client is provided the encryptions of r and n in the end. If $n = 0$, no match was found, and the client picks $p \leftarrow\!\!\$\ \mathsf{P}(1^\lambda)$, computes $c \leftarrow \mathsf{E}(1^\lambda, \mathsf{K}(1^\lambda, p, m), m)$, and sends p, c to be stored on the server. Otherwise, n refers to the index of the match, and serves as the tag, and r refers to the parameter in the match. Now the client computes $k \leftarrow \mathsf{K}(1^\lambda, r, m)$, encrypts it under its private key, and stores the result on the server. The Reg and Get protocols proceed in a simple manner, and are described in Figure 5. It can be checked that FCHECK performs deduplication as expected, and we show this formally in Proposition E.1 of Appendix E.

```
$\mathsf{E}(1^\lambda, k_\mathsf{H}, k, m)$
$c_0 \leftarrow\$\ \mathsf{Obf}(1^\lambda, k, 0))$
For $i \in [|m|]$ do
  $c_i \leftarrow\$\ \mathsf{Obf}(1^\lambda, k \| \langle i, \ell \rangle \| m[i], 0)$
Ret $c_0, \ldots c_{|m|}$
```

```
$\mathsf{D}(1^\lambda, k_\mathsf{H}, k, c_0, \ldots c_n)$
If $\mathsf{Eval}(1^\lambda, c_0, k) = \perp$ then ret $\perp$
For $i \in [n]$ do
  If $\mathsf{Eval}(1^\lambda, c_i, k \| \langle i, \ell \rangle \| 0) = 1$ then
    $m_i \leftarrow 0$
  else $m_i \leftarrow 1$
Ret $m_1 \| \ldots \| m_n$
```

Figure 6: The HtO MLEWC scheme, with a CR hash HF and a point obfuscation scheme OS. Here, parameters are generated via $\mathsf{P}(1^\lambda)$ which runs $\mathsf{K_h}(1^\lambda)$ and returns the output, while message-derived keys are generated by letting $\mathsf{K}(1^\lambda, k_\mathsf{H}, m)$ return $k \leftarrow \mathsf{H}(1^\lambda, k_\mathsf{H}, m)$.

Theorem 4.1. *If* MLEWC *is a correct MLEWC scheme then* FCHECK[MLEWC, FHE] *is* Rec*-secure.*

Proof sketch. Observe that that whenever a client puts m, and a match is found in $\mathsf{Put}[2, 1]$, the client asks for the p, c pair corresponding to the index with the match, and checks by itself that this pair is a valid ciphertext for m. This, combined with the immutability of **fil** and **own** leads to perfect recovery correctness.

Theorem 4.2. *If* MLEWC *is* WPRIV*-secure and* FHE *is* CPA*-secure, then* FCHECK[MLEWC, FHE] *is* PDPriv*-secure.*

Proof sketch. We replace the c_2 components with encryptions of random strings, and use the CPA security of FHE to justify this. Now, only the $\mathbf{p}, \mathbf{c}$ pairs of the plaintexts reside on the server, and hence we can hope to show that if there exists an adversary A that can guess the challenge bit from only the $\mathbf{p}, \mathbf{c}$ values, then such an A can be used to build another adversary B which breaks WPRIV security of MLEWC.

But this cannot be accomplished right away. When A asks the game to run Put with some $\mathbf{m}_b[i]$, then B cannot simulate $\mathsf{Put}[2, 1]$ which looks through $\mathbf{p}, \mathbf{c}$ for a match for $\mathbf{m}_b[i]$ without knowing $\mathbf{m}_b[i]$. The proof first gets rid of the search step in $\mathsf{Put}[2, 1]$ and then builds B. We argue that the search step can be avoided. The adversary A, with no knowledge of the messages that the unpredictable source S produced, would have been able to use Msg to put a ciphertext for a $\mathbf{m}_b[i]$ only with negligible probability.

Constructing MLEWC schemes. To get an iMLE scheme via FCHECK, we still need to construct a MLEWC scheme. The lack of comparison means that MLEWC schemes should be easier to construct compared to MLE schemes, but constructions must still overcome two technical challenges: encrypting messages with keys derived from the messages themselves, and dealing with correlated messages. We explore two approaches to overcoming these two challenges. The first utilizes a special kind of point-function obfuscation scheme, and the second uses a UCE-secure [10] hash function. This construction, which we relegate to Appendix F, is straightforward. We start with a hash function family, $\mathsf{HF} = (\mathsf{K_h}, \mathsf{H})$. Parameter generation picks a hash key k_H. Given m, the key is generated as $k \leftarrow \mathsf{H}(1^\lambda, k_\mathsf{H}, m, 1^\lambda)$, and ciphertext as $c \leftarrow \mathsf{H}(1^\lambda, k_\mathsf{H}, k, 1^{|m|}) \oplus m$. Decryption, on input k, c removes the mask to recover m.

We now elaborate on the first approach, which builds a MLEWC scheme from a composable distributional indistinguishable point-function obfuscation scheme (CDIPFO) [16]. To give a high level idea for why CDIPFOs are useful, we note that point-function obfuscation is connected to encryption secure when keys and messages are related [24]. Moreover, CDIPFOs, due to their composability, remain secure even when obfuscations of several correlated points are provided and thus enable overcoming the two challenges described above.

Let $\alpha, \beta \in \{0, 1\}^*$. We let $\phi_{\alpha,\beta} : \{0, 1\}^* \to \{\beta, \perp\}$ denote the function that on input $\gamma \in \{0, 1\}^*$ returns β if $\gamma = \alpha$, and $\perp$ otherwise. We call α the special input, and β the special output. A point function obfuscator $\mathsf{OS} = (\mathsf{Obf}, \mathsf{Eval})$ is a pair of algorithms. Obfuscation takes (α, β) and outputs $\mathsf{F} \leftarrow_{\$} \mathsf{Obf}(1^\lambda, (\alpha, \beta))$, while Eval takes F, and a point γ and returns $y \leftarrow_{\$} \mathsf{Eval}(1^\lambda, \mathsf{F}, \gamma)$. Correctness requires that $\mathsf{Eval}(1^\lambda, \mathsf{Obf}(1^\lambda, \alpha, \beta), \alpha) = \beta$ for all $\alpha, \beta \in \{0,1\}^*$, for all $\lambda \in \mathbb{N}$.

A PF source S outputs a tuple of point pairs $\mathbf{p}$, along with auxiliary information z. There exist $m : \mathbb{N} \to \mathbb{N}$ and $\ell : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that $|\mathbf{p}| = m(\lambda)$, and $|\mathbf{p}[i,0]| = \ell(\lambda, 0)$ and $|\mathbf{p}[i,1]| = \ell(\lambda, 1)$ for all $i \in [m(\lambda)]$. Guessing probability $\mathbf{GP}_{\mathsf{S}}(\lambda)$ is defined as $\max_i(\mathbf{GP}(\mathbf{p}[i,0]|z))$ when $(\mathbf{p}, z) \leftarrow_{\$} \mathsf{S}(1^\lambda)$. We say that S is unpredictable if $\mathbf{GP}_{\mathsf{S}}(\cdot)$ is negligible.

Distributional indistinguishability for point function obfuscators is captured by the CDIPFO game (Figure 17) associated with OS, an PF source S, and an adversary A. At a high level, the game either provides OS-obfuscations of point functions from S, or from a uniform distribution, and to win, the adversary A should guess which the case is. We associate advantage $\mathsf{Adv}^{\mathsf{cdipfo}}_{\mathsf{OS},\mathsf{S},\mathsf{A}}(\lambda) = 2\Pr[\mathrm{CDIPFO}^{\mathsf{A},\mathsf{S}}_{\mathsf{OS}}(1^\lambda)] - 1$ with OS, S and A and say that OS is CDIPFO-secure if advantage is negligible for all PT A for all unpredictable PT S. Bitansky and Canetti show that CDIPFOs can be built in the standard model, from the t-Strong Vector Decision Diffie Hellman assumption [16].

Let $\mathsf{HF} = (\mathsf{K_h}, \mathsf{H})$ denote a family of CR hash functions. The Hash-then-Obfuscate transform $\mathsf{HtO}[\mathsf{HF}, \mathsf{OS}] = (\mathsf{P}, \mathsf{E}, \mathsf{K}, \mathsf{D})$ associates an MLEWC scheme with HF and OS as in Figure 6, restricting the message space to ℓ-bit strings. At a high level, a key is generated by hashing the plaintext m with HF, and m is obfuscated bit-by-bit, with the hash as the special input. Decryption, given the hash, can recover m from the obfuscations. Correctness follows from the correctness of OS, and the following theorem shows WPRIV-security.

Theorem 4.3. *If* HF *is CR-secure, and* OS *is* CDIPFO*-secure, then* $\mathsf{HtO}[\mathsf{HF}, \mathsf{OS}]$ *is* WPRIV*-secure.*

The proof of the theorem and some remarks on HtO are provided in Appendix F.

5 Incremental updates

In this section, we define iMLE with incremental updates, and provide a construction which achieves this goal. Building MLE schemes which can support incremental updates turns out to be challenging. On the one hand, it is easy to show that deterministic MLE schemes cannot support incremental updates. We elaborate on this in Appendix B. . On the other hand, randomized MLE schemes seem to need complex machinery such as NIZK proofs of consistency [1] to support incremental updates while retaining the same level of security as deterministic schemes, which makes them unfit for practical usage. We show how interaction can be exploited to achieve incremental updates in a practical manner, by building an efficient ROM iMLE scheme IRCE that supports incremental updates. We fix Hamming distance as the metric. In Appendix C, we define incremental updates w.r.t edit distance, and extend IRCE to work in this setting.

An interactive message-locked encryption scheme iMLE with updates supports an additional protocol Upd along with the usual three protocols Reg, Put, and Get. The Upd protocol updates a ciphertext of a file m_1 stored on the server to a ciphertext of an updated file m_2. Here, $\mathsf{Upd}[1]$ (i.e. the client-side algorithm) takes inputs f, σ_C, and two plaintexts m_1, m_2, and outputs a new identifier $f_2 \in \{0,1\}^*$.

Now, the Rec game (Figure 3) which asks for correct recovery of files also imposes conditions on update, namely that if a legitimate client puts a file on the server, it should be able to get the file later along with updates made to the file. This is captured by letting the adversary pick Upd as the protocol in the Init procedure. The WinCheck procedure, which checks if the adversary has won, is now invoked at successful runs of Upd additionally. It infers the value of f used in the update protocol as well as the updated plaintext m_2 and sets $T[f] \leftarrow m_2$, thus letting the adversary to win if a get at f does not return m_2.

We say that a scheme iMLE has incremental updates if the communication cost of updating a ciphertext for m_1 stored on the server to a ciphertext for m_2 is a linear function of $\mathsf{HAMM}(m_1, m_2)$

```
Init(1^λ)
  $p \leftarrow_\$ \{0,1\}^{\kappa(\lambda)}$; $\mathbf{U} \leftarrow \emptyset$; $\mathbf{fil} \leftarrow \emptyset$; $\mathbf{own} \leftarrow \emptyset$
  Ret $\sigma_S = (p, \mathbf{U}, \mathbf{fil}, \mathbf{own})$

Reg[1]($\epsilon$)                                      Reg[2]($\sigma_S$)
  $k \leftarrow_\$ \{0,1\}^{\kappa(\lambda)}$   $\xrightarrow{\epsilon}$     $u \leftarrow_\$ \{0,1\}^\lambda \setminus \mathbf{U}$
                                   $\xleftarrow{u, p}$    $\mathbf{U} \leftarrow \mathbf{U} \cup \{u\}$
  Ret $(k, u, p)$

Get[1]($(k, u, p), t$)                                  Get[2]($\sigma_S$)
                                   $\xrightarrow{u, t}$      $(c_1, c_2) \leftarrow \mathbf{fil}[t]$
                                                 $c_3 \leftarrow \mathbf{own}[u, t]$
                                                 If $c_3 = \bot$ then
  If $c_1 = \bot$ then ret $\bot$      $\xleftarrow{c_1, c_2, c_3}$      $(c'_1, c'_2) \leftarrow (\bot, \bot)$
  $k_2 \leftarrow \mathsf{D}(1^\lambda, k, c_3)$
  Ret $\mathsf{D}(1^\lambda, k_2 \oplus c_2, c_1)$
```

Figure 7: The Init algorithm, and Reg and Get protocols of the IRCE iMLE scheme.

and $\log |m_2|$. More formally, there exists a linear function $u : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that for all client parameters σ_C, for all server-side states $\sigma_S \in \{0,1\}^*$, for all plaintexts $m_1, m_2 \in \{0,1\}^*$ such that $|m_1| = |m_2|$, for all coins r_1, r_2, for all $f \in \{0,1\}^*$, if $(m_1, \sigma'_S) \leftarrow \mathsf{Run}(\mathsf{Get}, (\sigma_C, f), \sigma_S; r_1)$, and $(f', \sigma''_S) \leftarrow \mathsf{Run}(\mathsf{Upd}, (\sigma_C, m_1, m_2), \sigma_S; r_2)$, and $f' \neq \bot$, then $|\mathsf{Msgs}(\mathsf{Upd}, (\sigma_C, m_1, m_2), \sigma_S; r_2)| \leq \mathsf{HAMM}(m_1, m_2) u(\log |m_1|, \lambda)$.

Preliminaries. A deterministic symmetric encryption (D-SE) scheme $\mathsf{SE} = (\mathsf{E}, \mathsf{D})$ is a pair of algorithms, where encryption returns $c \leftarrow \mathsf{E}(1^\lambda, k, m)$ on input plaintext $m \in \{0,1\}^*$ and key $k \in \{0,1\}^{\kappa(\lambda)}$, and decryption returns $m \leftarrow \mathsf{D}(1^\lambda, k, c)$. Correctness requires $\mathsf{D}(1^\lambda, k, \mathsf{E}(1^\lambda, k, m)) = m$ for all plaintexts $m \in \{0,1\}^*$ for all keys $k \in \{0,1\}^{\kappa(\lambda)}$ for all $\lambda \in \mathbb{N}$. We say that SE supports incremental updates w.r.t Hamming distance if there exists an algorithm U such that $\mathsf{U}(1^\lambda, \mathsf{E}(1^\lambda, k, m_1), \mathsf{diff}(m_1, m_2)) = \mathsf{E}(1^\lambda, k, m_2)$ for all plaintexts $m_1, m_2 \in \{0,1\}^*$ for all keys $k \in \{0,1\}^{\kappa(\lambda)}$ for all $\lambda \in \mathbb{N}$.

Key-recovery security is defined through game $\mathrm{KR}^{\mathsf{A}}_{\mathsf{SE}}(1^\lambda)$ which lets adversary A query an oracle Enc with a plaintext m then picks $k \leftarrow_\$ \{0,1\}^{\kappa(\lambda)}$ and returns $\mathsf{E}(1^\lambda, k, m)$; A wins if it can guess k.

The CPA security game $\mathrm{CPA}^{\mathsf{A}}_{\mathsf{SE}}(1^\lambda)$, picks $b \leftarrow_\$ \{0,1\}$ and $k \leftarrow_\$ \kappa(\lambda)$, runs A with access to Enc, and responds to queries m by returning $c \leftarrow \mathsf{E}(k, m)$ if $b = 1$ and returning a random $|c|$-bit string if $b = 0$. To win, the adversary should guess b. We define advantages $\mathsf{Adv}^{\mathsf{kr}}_{\mathsf{SE,A}}(\lambda) = \Pr[\mathrm{KR}^{\mathsf{A}}_{\mathsf{SE}}(1^\lambda)]$ and $\mathsf{Adv}^{\mathsf{cpa}}_{\mathsf{SE,A}}(\lambda) = 2 \cdot \Pr[\mathrm{CPA}^{\mathsf{A}}_{\mathsf{SE}}(1^\lambda)] - 1$ and say that SE is KR-secure (resp. CPA-secure) if $\mathsf{Adv}^{\mathsf{kr}}_{\mathsf{SE,A}}(\cdot)$ (resp. $\mathsf{Adv}^{\mathsf{cpa}}_{\mathsf{SE,A}}(\cdot)$) is negligible for all PT A. The CTR mode of operation over a blockcipher, with a fixed IV is an example of a D-SE scheme with incremental updates, and KR and CPA security.

A hash function H with $\kappa(\lambda)$-bit keys is a PT algorithm that takes $p \in \{0,1\}^{\kappa(\lambda)}$ and a plaintext m returns hash $h \leftarrow \mathsf{H}(p, m)$. Collision resistance is defined through game $\mathrm{CR}^{\mathsf{A}}_{\mathsf{H}}(1^\lambda)$, which picks $p \leftarrow_\$ \{0,1\}^{\kappa(\lambda)}$, runs adversary $\mathsf{A}(1^\lambda, p)$ to get m_0, m_1, and returns True if $m_0 \neq m_1$ and $\mathsf{H}(p, m_1) = \mathsf{H}(p, m_2)$. We say that H is collision resistant if $\mathsf{Adv}^{\mathsf{cr}}_{\mathsf{H,A}}(\lambda) = \Pr[\mathrm{CR}^{\mathsf{A}}_{\mathsf{H}}(1^\lambda)]$ is negligible for all PT A.

A table T is immutable if each entry $T[t]$ can be assigned only one value after initialization. Immutable tables supports the Set-iff-empty, or `SiffE` operation, which takes inputs a table T, an index f, and a value m. If $T[f] = \bot$ then $T[f] \leftarrow m$ and m is returned; otherwise $T[f]$ is returned.

The IRCE scheme. Let H denote a hash function with $\kappa(\lambda)$-bit keys and $\kappa(\lambda)$-bit outputs, and let $\mathsf{SE} = (\mathsf{E}, \mathsf{D})$ denote a D-SE scheme with $\kappa(\lambda)$-bit keys, where ciphertexts have same lengths as plaintexts and incremental updates are supported through an algorithm U. The IMLE scheme $\mathsf{IRCE}[\mathsf{SE}, \mathsf{H}]$ is described in figures 7 and 8. We call the construction IRCE, expanding to interactive randomized convergent encryption. since it resembles the randomized convergent encryption (RCE) scheme of [11].

To describe how IRCE works, let us consider a IMLE scheme built around RCE. In RCE, to put m on the server, the client encrypts m with a random key ℓ to get c_1, and then encrypts ℓ with $k_m = \mathsf{H}(p, m)$ to get c_2, where p is a system-wide public parameter. Then, k_m is hashed once more to get the tag $t = \mathsf{H}(p, k_m)$. The client sends t, c_1, c_2 and the server stores c_1, c_2 in a table $\mathbf{fil}$ at index t. If another client starts with m, it will end up with the same t, although it will derive a different c'_1, c'_2, as ℓ is picked at random. However, when this client sends t, c'_1, c'_2, the server knows that $\mathbf{fil}[t]$ is not empty, meaning a duplicate exists, and hence will drop c'_1, c'_2, thereby achieving deduplication. The second client should be able to recover m by sending t to the server, receiving c_1, c_2, recovering ℓ from c_2 and decrypting c_1. However, the problem with RCE is that, when the first client sends t, c_1, c_2, the server has no way of checking whether c_1, c_2 is a proper ciphertext of m, or a corrupted one. Thus, the second client, in spite of storing a ciphertext of m on the server might not be able to recover m — this violates our soundness requirement. We will now fix this issue with interaction.

The Put protocol in IRCE differs in that, if the server finds that $\mathbf{fil}[t] \neq \bot$ then it responds with h, c'_2, where $(c'_1, c'_2) \leftarrow \mathbf{fil}[t]$ and $h \leftarrow \mathsf{H}(p, c'_1)$. Now, the client can check that $\mathsf{H}(p, \mathsf{E}(1^\lambda, c'_2 \oplus k_m, m)) = h$ which means that whenever deduplication happens, the client can check the validity of the duplicate ciphertext, which in turn guarantees soundness. The Put protocol is specified in Figure 8, and is a bit more involved than our sketch here. Specifically, the clients are assigned unique identifiers which are provided during Put. The message-derived key k_m is also encrypted to get c_3 (under per-client keys) and stored on the server, in a separate table **own**, which enables checking that a client starting a get protocol with an identifier did put the file earlier. If the client is the first to put a ciphertext with tag t, then the server still returns $\mathsf{H}(p, c_1), c_2, c_3$ so that external adversaries cannot learn if deduplication occurred. We note that in Figure 8, the **fil** and **own** tables are immutable, and this will help in arguing soundness of the scheme.

The Init algorithm (Figure 7) sets up the **fil** and **own** tables, and additional server-side state, and picks a key p for H, which becomes the public-parameter of the system. The Reg protocol (Figure 8) sets up a new client by creating a unique client identifier u, and providing the client p. The client also picks a secret key k without the involvement of the server. The Get protocol (Figure 8) recovers a plaintext from the identifier, which in the case of IRCE is the tag.

IRCE supports incremental updates, as described in Figure 8. If the client wants to update m to m_2, it does not have to resend all of c_1, c_2, c_3. Instead, it can use the same key ℓ and incrementally update c_1, and compute new values for c_2 and c_3, along with the new tag t_2. If the server finds that $\mathbf{fil}[t_2]$ is not empty, the same check as in Put is performed.

Propositions C.1 and C.2 of Appendix C show that IRCE performs deduplication, and supports incremental updates, and their proofs proceed in a straightforward manner. The following theorem, with proof in Appendix C shows that IRCE is REC-secure (which, along with deduplication, establishes soundness).

Theorem 5.1. *If* H *is collision resistant and* SE *is a correct D-SE scheme, then* $\mathsf{IRCE}[\mathsf{H}, \mathsf{SE}]$ *is* REC-*secure.*

Proof sketch. To win the REC game, the adversary A must put a plaintext m on the server, possibly update it to some m', complete a Get instance with the identifier for m or m' and show that the result is incorrect.

The proof uses the immutability of **fil** and **own** to argue that the ciphertext stored in the server could not have changed between the failed Get instance and the last time the plaintext was put/updated. However, Put and Upd both ensure that the hash of the ciphertext stored on the server matches with the hash of a correctly formed ciphertext for the plaintext being put/updated. Consequently, whenever A breaks REC-security, it is in effect finding a pair of colliding inputs, namely the hash inputs involved in the comparison. A CR adversary B can be built which has the same advantage as the REC-advantage of A.

The following theorem (with proof in Appendix C) shows that IRCE is PRIV-secure in the ROM, assuming that SE is secure. Let $\mathsf{IRCE_{RO}}$ denote the ROM analogue of IRCE, formed by modelling H as a random oracle.

Theorem 5.2. *If* SE *is* CPA-*secure and* KR-*secure, then* $\mathsf{IRCE_{RO}}[\mathsf{SE}]$ *is* PRIV-*secure.*

Proof sketch. In PRIV, the source S outputs $\mathbf{m}_0, \mathbf{m}_1$, the game picks $b \leftarrow\!\!\$\ \{0, 1\}$ and adversary A can

```
$\underline{\mathsf{Put}[1]((k,u,p),m)}$                                   $\underline{\mathsf{Put}[2](\sigma_S)}$
$\ell \leftarrow_{\$} \{0,1\}^{\kappa(\lambda)}$; $c_1 \leftarrow \mathsf{E}(1^\lambda, \ell, m)$
$k_m \leftarrow \mathsf{H}(p,m)$; $c_2 \leftarrow k_m \oplus \ell$; $c_3 \leftarrow \mathsf{E}(1^\lambda, k, k_m)$
$t \leftarrow \mathsf{H}(p,k_m)$
                                $\xrightarrow{u, c_1, c_2, c_3, t}$   $(c_1, c_2) \leftarrow \mathtt{SiffE}(\mathbf{fil}, t, c_1, c_2)$; $h \leftarrow \mathsf{H}(p, c_1)$
                                                                    $c_3 \leftarrow \mathtt{SiffE}(\mathbf{own}, (u,t), c_3)$
                                $\xleftarrow{h, c_2', c_3'}$
If $c_3 \neq c_3'$ then ret $\perp$
$\ell' \leftarrow c_2' \oplus k_m$ ; $c_1'' \leftarrow \mathsf{E}(1^\lambda, \ell', m)$; $h' \leftarrow \mathsf{H}(p, c_1'')$
If $h = h'$ then ret $t$ Else ret $\perp$
```

```
$\underline{\mathsf{Upd}[1]((k,u,p),t,m_1,m_2)}$                             $\underline{\mathsf{Upd}[2](\sigma_S)}$
$k_1 \leftarrow \mathsf{H}(p,m_1)$; $k_2 \leftarrow \mathsf{H}(p,m_2)$; $\delta \leftarrow \mathsf{diff}(m_1,m_2)$
$t_2 \leftarrow \mathsf{H}(p,k_2)$; $c_d \leftarrow k_1 \oplus k_2$; $t_1 \leftarrow \mathsf{H}(p,k_1)$
$c_3 \leftarrow \mathsf{E}(1^\lambda, k, k_2)$
                                $\xrightarrow{u, t_1, t_2, c_3, c_d, \delta}$   If $\mathbf{own}[u, t_1] \neq \perp$ then
                                                                      $c_3 \leftarrow \mathtt{SiffE}(\mathbf{own}, (u,t), c_3)$
                                                                      $c_1, c_2 \leftarrow \mathbf{fil}[t_1]$
                                                                      $c_1' \leftarrow \mathsf{patch}(c_1, \delta)$; $c_2' \leftarrow c_2 \oplus c_d$
                                                                      $(c_1', c_2') \leftarrow \mathtt{SiffE}(\mathbf{fil}, t_2, c_1', c_2')$
                                                                    Else $(c_1', c_2') \leftarrow (\perp, \perp)$
                                                                    $h \leftarrow \mathsf{H}(p, c_1')$
                                $\xleftarrow{h, c_2', c_3'}$
If $c_3 \neq c_3'$ then ret $\perp$
$c_1'' \leftarrow \mathsf{E}(1^\lambda, c_2' \oplus k_2, m_2)$; $h' \leftarrow \mathsf{H}(p, c_1'')$
If $h = h'$ then ret $t_2$; Else ret $\perp$
```

Figure 8: The Put and Upd protocols of the IRCE iMLE scheme. The **fil** and **own** tables are immutable, and support the set-iff-empty operation (SiffE) explained in text.

put and update components of $\mathbf{m}_b$, and finally gets to learn the server-side state. To win, A should guess b.

First, the c_3 components are changed to encrypt random strings instead of message-derived keys $\mathbf{k}_m[i]$; CPA security of SE makes this change indistinguishable by A. The proof then moves to a game where RO responses are no longer consistent with the keys and tags being generated. For instance, if S or A queries the RO at $p \| \mathbf{m}_b[i]$, it gets a response different from $\mathbf{k}_m[i]$. The remainder of the proof involves two steps. First, we show that once we stop maintaining RO consistency, the adversary gets no information about the ℓ values used to encrypt the messages, and hence guessing b means breaking either the CPA security or key recovery security of SE. Second, we argue that neither S nor A can detect that RO responses are inconsistent. This is because S does not know p, a prefix to the key and tag generation queries. An A that detects the inconsistency will break the CPA security of SE.

6 References

[1] M. Abadi, D. Boneh, I. Mironov, A. Raghunathan, and G. Segev. Message-locked encryption for lock-dependent messages. In R. Canetti and J. A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 374–391. Springer, Aug. 2013. (Cited on page 3, 4, 5, 6, 9, 13.)

[2] A. Adya, W. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. Douceur, J. Howell, J. Lorch, M. Theimer, and R. Wattenhofer. Farsite: Federated, available, and reliable storage for an incompletely trusted environment. *ACM SIGOPS Operating Systems Review*, 36(SI):1–14, 2002. (Cited on page 3, 4.)

[3] Amazon. S3. `http://aws.amazon.com/s3/pricing/`. (Cited on page 4.)

[4] P. Anderson and L. Zhang. Fast and secure laptop backups with encrypted de-duplication. In *Proc. of USENIX LISA*, 2010. (Cited on page 3, 4.)

[5] C. Batten, K. Barr, A. Saraf, and S. Trepetin. pStore: A secure peer-to-peer backup system. *Unpublished report, MIT Laboratory for Computer Science*, 2001. (Cited on page 3, 4.)

[6] M. Bellare, A. Boldyreva, and A. O'Neill. Deterministic and efficiently searchable encryption. In A. Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 535–552. Springer, Aug. 2007. (Cited on page 3.)

[7] M. Bellare, R. Canetti, and H. Krawczyk. A modular approach to the design and analysis of authentication and key exchange protocols (extended abstract). In *30th ACM STOC*, pages 419–428. ACM Press, May 1998. (Cited on page 7.)

[8] M. Bellare, O. Goldreich, and S. Goldwasser. Incremental cryptography: The case of hashing and signing. In Y. Desmedt, editor, *CRYPTO'94*, volume 839 of *LNCS*, pages 216–233. Springer, Aug. 1994. (Cited on page 4, 13.)

[9] M. Bellare, O. Goldreich, and S. Goldwasser. Incremental cryptography and application to virus protection. In *27th ACM STOC*, pages 45–56. ACM Press, May / June 1995. (Cited on page 4, 13.)

[10] M. Bellare, V. T. Hoang, and S. Keelveedhi. Instantiating random oracles via uces. Cryptology ePrint Archive, Report 2013/424, 2013. Preliminary version in Crypto 2013. (Cited on page 3, 4, 5, 10, 12, 34.)

[11] M. Bellare, S. Keelveedhi, and T. Ristenpart. Message-locked encryption and secure deduplication. In T. Johansson and P. Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 296–312. Springer, May 2013. (Cited on page 3, 4, 5, 6, 8, 14, 20.)

[12] M. Bellare and T. Kohno. A theoretical treatment of related-key attacks: RKA-PRPs, RKA-PRFs, and applications. In E. Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 491–506. Springer, May 2003. (Cited on page 10.)

[13] M. Bellare and D. Micciancio. A new paradigm for collision-free hashing: Incrementality at reduced cost. In W. Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 163–192. Springer, May 1997. (Cited on page 4, 13.)

[14] M. Bellare and P. Rogaway. Entity authentication and key distribution. In D. R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 232–249. Springer, Aug. 1993. (Cited on page 7.)

[15] M. Bellare and P. Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In S. Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 409–426. Springer, May / June 2006. (Cited on page 6, 31.)

[16] N. Bitansky and R. Canetti. On strong simulation and composable point obfuscation. In T. Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 520–537. Springer, Aug. 2010. (Cited on page 5, 10, 12, 13, 33.)

[17] Bitcasa. Bitcasa inifinite storage. `http://blog.bitcasa.com/tag/patented-de-duplication/`. (Cited on page 3, 4.)

[18] Z. Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In R. Safavi-Naini and R. Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 868–886. Springer, Aug. 2012. (Cited on page 5, 10.)

[19] Z. Brakerski and V. Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In R. Ostrovsky, editor, *52nd FOCS*, pages 97–106. IEEE Computer Society Press, Oct. 2011. (Cited on page 5, 10.)

[20] Z. Brakerski and V. Vaikuntanathan. Fully homomorphic encryption from ring-LWE and security for key dependent messages. In P. Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 505–524. Springer, Aug. 2011. (Cited on page 5, 10.)

[21] E. Buonanno, J. Katz, and M. Yung. Incremental unforgeable encryption. In M. Matsui, editor, *FSE 2001*, volume 2355 of *LNCS*, pages 109–124. Springer, Apr. 2001. (Cited on page 4.)

[22] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, Oct. 2001. (Cited on page 7.)

[23] R. Canetti and R. R. Dakdouk. Obfuscating point functions with multibit output. In N. P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 489–508. Springer, Apr. 2008. (Cited on page 10.)

[24] R. Canetti, Y. T. Kalai, M. Varia, and D. Wichs. On symmetric encryption and point obfuscation. In D. Micciancio, editor, *TCC 2010*, volume 5978 of *LNCS*, pages 52–71. Springer, Feb. 2010. (Cited on page 10, 12.)

[25] Ciphertite. Ciphertite data backup. `https://www.cyphertite.com/faq.php`. (Cited on page 3, 4.)

[26] J. Cooley, C. Taylor, and A. Peacock. ABS: the apportioned backup system. *MIT Laboratory for Computer Science*, 2004. (Cited on page 3, 4.)

[27] J.-S. Coron, A. Mandal, D. Naccache, and M. Tibouchi. Fully homomorphic encryption over the integers with shorter public keys. In P. Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 487–504. Springer, Aug. 2011. (Cited on page 5, 10.)

[28] L. P. Cox, C. D. Murray, and B. D. Noble. Pastiche: making backup cheap and easy. *SIGOPS Oper. Syst. Rev.*, 36:285–298, Dec. 2002. (Cited on page 3, 4.)

[29] Y. Dodis, T. Ristenpart, and S. P. Vadhan. Randomness condensers for efficiently samplable, seed-dependent sources. In R. Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 618–635. Springer, Mar. 2012. (Cited on page 33.)

[30] J. Douceur, A. Adya, W. Bolosky, D. Simon, and M. Theimer. Reclaiming space from duplicate files in a serverless distributed file system. In *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*, pages 617–624. IEEE, 2002. (Cited on page 3.)

[31] Dropbox. Deduplication in Dropbox. `https://forums.dropbox.com/topic.php?id=36365`. (Cited on page 3.)

[32] T. Duong and J. Rizzo. Here come the ninjas. *Unpublished manuscript*, 2011. (Cited on page 4.)

[33] M. Dutch. Understanding data deduplication ratios. In *SNIA Data Management Forum*, 2008. (Cited on page 7.)

[34] M. Fischlin. Incremental cryptography and memory checkers. In W. Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 293–408. Springer, May 1997. (Cited on page 4.)

[35] Flud. The Flud backup system. `http://flud.org/wiki/Architecture`. (Cited on page 3, 4.)

[36] C. Gentry. Fully homomorphic encryption using ideal lattices. In M. Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009. (Cited on page 5, 10, 11.)

[37] C. Gentry and S. Halevi. Implementing Gentry's fully-homomorphic encryption scheme. In K. G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 129–148. Springer, May 2011. (Cited on page 10.)

[38] C. Gentry, S. Halevi, and N. P. Smart. Fully homomorphic encryption with polylog overhead. In D. Pointcheval and T. Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 465–482. Springer, Apr. 2012. (Cited on page 5, 10.)

[39] GNUnet. GNUnet, a framework for secure peer-to-peer networking. `https://gnunet.org/`. (Cited on page 3, 4.)

[40] Google. Blob store. `https://developers.google.com/appengine/docs/pricing`. (Cited on page 4.)

[41] Google. Google Drive. `http://drive.google.com`. (Cited on page 3.)

[42] V. Goyal, A. O'Neill, and V. Rao. Correlated-input secure hash functions. In Y. Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 182–200. Springer, Mar. 2011. (Cited on page 10.)

[43] S. Halevi, D. Harnik, B. Pinkas, and A. Shulman-Peleg. Proofs of ownership in remote storage systems. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 491–500. ACM, 2011. (Cited on page 7.)

[44] D. Harnik, B. Pinkas, and A. Shulman-Peleg. Side channels in cloud services: Deduplication in cloud storage. *Security & Privacy, IEEE*, 8(6):40–47, 2010. (Cited on page 7.)

[45] J. Katz and V. Vaikuntanathan. Round-optimal password-based authenticated key exchange. In Y. Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 293–310. Springer, Mar. 2011. (Cited on page 7.)

[46] M. Killijian, L. Courtès, D. Powell, et al. A survey of cooperative backup mechanisms, 2006. (Cited on page 3, 4.)

[47] L. Marques and C. Costa. Secure deduplication on mobile devices. In *Proceedings of the 2011 Workshop on Open Source and Design of Communication*, pages 19–26. ACM, 2011. (Cited on page 3, 4.)

[48] D. Meister and A. Brinkmann. Multi-level comparison of data deduplication in a backup scenario. In *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*, page 8. ACM, 2009. (Cited on page 7.)

[49] Microsoft. Windows Azure. `http://www.windowsazure.com/en-us/pricing/details/storage/`. (Cited on page 4.)

[50] I. Mironov, O. Pandey, O. Reingold, and G. Segev. Incremental deterministic public-key encryption. In D. Pointcheval and T. Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 628–644. Springer, Apr. 2012. (Cited on page 4, 13, 21.)

[51] NetApp. NetApp. `http://www.netapp.com/us/products/platform-os/dedupe.aspx`. (Cited on page 3.)

[52] A. Rahumed, H. Chen, Y. Tang, P. Lee, and J. Lui. A secure cloud backup system with assured deletion and version control. In *Parallel Processing Workshops (ICPPW), 2011 40th International Conference on*, pages 160–167. IEEE, 2011. (Cited on page 3, 4.)

[53] T. Ristenpart, H. Shacham, and T. Shrimpton. Careful with composition: Limitations of the indifferentiability framework. In K. G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 487–506. Springer, May 2011. (Cited on page 6.)

[54] M. Storer, K. Greenan, D. Long, and E. Miller. Secure data deduplication. In *Proceedings of the 4th ACM international workshop on Storage security and survivability*, pages 1–10. ACM, 2008. (Cited on page 3, 4.)

[55] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan. Fully homomorphic encryption over the integers. In H. Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 24–43. Springer, May 2010. (Cited on page 5, 10.)

```
$\underline{\mathsf{Run}(1^\lambda, \mathsf{P}, \mathsf{inp})}$
$T \leftarrow \emptyset$; $n \leftarrow 1$; $\mathtt{M} \leftarrow \epsilon$
For $i = 1$ to $n$ do $\mathbf{a}[i,1] \leftarrow \mathsf{inp}[i]$; $\mathtt{rd}[i] \leftarrow 1$
While $T \neq [n]$ do
  If $n \in T$ then return $\bot$
  $i \leftarrow \mathtt{rd}[n]$
  $(\mathbf{a}[n, i+1], \mathtt{M}, \mathtt{N}, \mathtt{T}) \leftarrow\!\$\; \mathsf{P}[n,i](1^\lambda, \mathbf{a}[n,i], \mathtt{M})$
  If $\mathtt{T} = \mathsf{True}$ then $T \leftarrow T \cup \{n\}$
  $\mathtt{rd}[n] \leftarrow \mathtt{rd}[n] + 1$; $n \leftarrow \mathtt{N}$
For $i = 1$ to $n$ do $\mathsf{outp}[i] \leftarrow \mathtt{last}(\mathbf{a}[i])$
Ret $\mathsf{outp}$
```

```
$\underline{\mathsf{Msgs}(1^\lambda, \mathsf{P}, \mathbf{a}, r)}$
$T \leftarrow \emptyset$; $n \leftarrow 1$; $\mathtt{M} \leftarrow \epsilon$; $j \leftarrow 1$
For $i = 1$ to $n$ do $\mathbf{a}[i,1] \leftarrow \mathsf{inp}[i]$; $\mathtt{rd}[i] \leftarrow 1$
While $T \neq [n]$ do
  If $n \in T$ then return $\bot$
  $i \leftarrow \mathtt{rd}[n]$
  $(\mathbf{a}[n, i+1], \mathtt{M}, \mathtt{N}, \mathtt{T}) \leftarrow\!\$\; \mathsf{P}[n,i](1^\lambda, \mathbf{a}[n,i], \mathtt{M}; r[n,i])$
  If $\mathtt{T} = \mathsf{True}$ then $T \leftarrow T \cup \{n\}$
  $\mathtt{rd}[n] \leftarrow \mathtt{rd}[n] + 1$; $n \leftarrow \mathtt{N}$; $\mathbf{M}[j] \leftarrow \mathtt{M}$; $j \leftarrow j + 1$
Ret $\mathbf{M}$
```

Figure 9: **Left**: Running a protocol P. **Right**: The Msgs procedure returns the messages exchanged during the protocol when invoked with specified inputs and coins.

A Interactive protocols

Consider a protocol P with n-players, where each player is invoked for a maximum of q-times. We represent such a protocol as a $n \times q$-tuple $(\mathsf{P}[i,j])_{i \in [n], j \in [q]}$ of algorithms. The algorithm $\mathsf{P}[i,j]$ represents the action of the i-th player, when invoked for the j-th time. Each algorithm is invoked with the security parameter 1^λ, an input $\mathbf{a}$, and a message $\mathtt{M} \in \{0,1\}^*$, and returns a 4-tuple consisting of an output $\mathbf{a}'$, an outgoing message $\mathtt{M}' \in \{0,1\}^*$, the index $\mathtt{N} \in \mathbb{N}$ of the next algorithm to run, and a boolean $\mathtt{T}$ to indicate termination. When all algorithms of a protocol have terminated, the protocol is said to have terminated. We denote the n-players of the protocol by $\mathsf{P}[1], \ldots, \mathsf{P}[n]$. The execution of a protocol P is captured by the Run algorithm, which takes inputs inp and returns outp, both tuples of n-elements, is described in Figure 9. We say that P is run on inp, or that $\mathsf{P}[i]$ is invoked with $\mathsf{inp}[i]$ for $i \in [n]$ to mean that the input to $\mathsf{P}[i,1]$ is set to $\mathsf{inp}[i]$ for $i \in [n]$. We say that P returns outp or that $\mathsf{P}[i]$ gets output $\mathsf{outp}[i]$ for $i \in [n]$ to mean that $\mathsf{Run}(1^\lambda, \mathsf{P}, \mathsf{inp})$ returns outp. The Msgs procedure of Figure 9 returns the protocol messages when invoked with specified inputs and coins.

B Deterministic MLE schemes cannot support incremental updates

An updatable MLE scheme $\mathsf{MLE} = (\mathsf{Pg}, \mathsf{K}, \mathsf{E}, \mathsf{D}, \mathsf{T}, \mathsf{U}_1, \mathsf{U}_2)$ is a seven-tuple of PT algorithms. The first five algorithms work as in regular MLE schemes. The two update algorithms U_1 and U_2 work as follows. On input parameters p and two messages m_1, m_2 where m_2 is the newer version of m_1, the U_1 algorithm outputs a string $c_u \in \{0,1\}^*$. On input parameters p, update string c_u, and original ciphertext c, the U_2 algorithm produces updated ciphertext c'. We say that MLE is deterministic if $\mathsf{K}, \mathsf{E}, \mathsf{U}_1$ and U_2 are deterministic.

We say that MLE is incremental if there exist functions $a : \mathbb{N} \to \mathbb{N}$ and $b : \mathbb{N} \to \mathbb{N}$ such that for for all $p \in [\mathsf{P}(1^\lambda)]$, for all $m_1, m_2 \in \{0,1\}^*$, it holds that $|c_u| \leq a(\lambda) \log(|m_1| + |m_2|)\delta + b(\lambda)$ for all $c_u \in [\mathsf{U}_1(1^\lambda, p, m_1, m_2)]$, where $\delta = \mathsf{HAMM}(m_1, m_2)$.

We now show that if a deterministic MLE scheme supports incremental updates, then it cannot even satisfy PRV-CDA security, the weakest among the security notions of [11]. Note that the attacker does not have the ability to update ciphertexts. Informally, the result is achieved using the fact that highly correlated plaintexts will produce similar ciphertexts, while independently chosen plaintexts will produce different-looking ciphertexts, and this can be used to guess the bit in the PRV-CDA game.

Theorem B.1. *Let* MLE *denote a deterministic MLE scheme which supports incremental updates with bound* $u : \mathbb{N} \to \mathbb{N}$. *Then* MLE *is not* PRV-CDA *secure.*

Proof. Consider $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2)$ where A_1 picks m_1, m_3, m_4 at random from $\{0,1\}^{\mu}(\lambda)$ and picks m_2 such that $\texttt{dist}(m_1, m_2) = 1$. Now, A_1 outputs $(m_1, m_2), (m_3, m_4)$ as its output tuples. Informally, the two components of $\mathbf{m}_0$ are distance 1-apart and hence their ciphertexts should be close. On the other hand the two components of $\mathbf{m}_1$ are unlikely to be close, as they are picked independently at random, and A_2 uses this difference to infer the bit of the game. Specifically, $\mathsf{A}_2(1^\lambda, p, \mathbf{c})$ returns 0 if $\mathsf{HAMM}(\mathbf{c}[0], \mathbf{c}[1]) \leq u(1)$, and 1 otherwise. Clearly, if A_2 outputs 0 whenever $b = 0$ in the game. The probability that the ciphertexts of m_3 and m_4 are less than $u(1)$ units apart isis negligible. Moreover, $\mathbf{GP}_{\mathsf{A}_1} = 2^{1-\mu(\lambda)}$, making it an unpredictable source. Thus, A is a valid PRV-CDA adversary with advantage negligibly away from 1, meaning that MLE is not PRV-CDA secure. □

We remark that a similar result applies for deterministic PKE schemes, with regards to PRIV security. Mironov, Pandey, Reingold, and Segev [50] model incremental deterministic PKE schemes, but they restrict to attention to PRIV1 security, which does not consider correlated messages.

C Incremental updates: Proofs and extensions

Proposition C.1. *Let* H *denote a deterministic hash function and* $\mathsf{SE} = (\mathsf{E}, \mathsf{D})$ *denote a deterministic symmetric encryption scheme. Then* $\mathsf{IRCE}[\mathsf{H}, \mathsf{SE}]$ *supports deduplication.*

Proof. When a client with id u and parameters σ_C puts a plaintext m on the server, then, in **fil**, an entry c_1, c_2, is added at index $t = \mathsf{H}(p, \mathsf{H}(p, m))$ where $c_1 \leftarrow \mathsf{E}(1^\lambda, \ell, m)$, and $k_m \leftarrow \mathsf{H}(p, m)$, and $c_2 \leftarrow k_m \oplus \ell$. Now, if another client with id u' and parameters σ'_C tries to put m, and sends across a c'_1, c'_2, t (we need H to be deterministic, to ensure that the same tag is generated both times) to the server, the server detects a duplicate at $\mathbf{fil}[t]$ and drops c'_1, c'_2. A fresh copy of c_3 is still stored at $\mathbf{own}[u', t]$, but the size of c_3 is bounded by $\kappa(\lambda)$, and the increase in σ_S is bounded by $|u'| + |t| + |\kappa(\lambda)|$, which is independent of $|m|$. □

Proposition C.2. *Let* H *denote a cryptographic hash function and* $\mathsf{SE} = (\mathsf{E}, \mathsf{D})$ *denote a deterministic symmetric encryption scheme, which supports incremental updates w.r.t Hamming distance. Then* $\mathsf{IRCE}[\mathsf{H}, \mathsf{SE}]$ *also supports incremental updates w.r.t Hamming distance.*

Proof. By inspecting the Upd protocol when invoked on plaintexts m_1, m_2, we can check that the total length of the transmitted messages is $\lambda + 6\kappa(\lambda) + |\delta|$, where $\delta = \mathsf{diff}(m_1, m_2)$. Letting $\kappa(\lambda) = \lambda$, and noting that δ is the list of positions where m_1 and m_2 differ. Since $\log |m_1|$ bits are needed to represent one position, the total size of δ can be bounded by $\mathsf{HAMM}(m_1, m_2) \log |m_1|$. The total length of messages is bounded by $\mathsf{HAMM}(m_1, m_2)(\log |m_1|) + 7\lambda$, proving the proposition.

□

C.1 Proof of Theorem 5.1

Proof. Consider games G_1 and G_2 of Figure 10. Here G_1 is essentially the Rec game with IRCE, except that G_1 maintains tables $\mathbf{C}_v$, $\mathbf{C}_s$, and $\mathbf{C}_r$. When the adversary completes a instantiation of $\mathsf{Put}(m)$ through calls to Init and Step, the ciphertexts c''_1, c'_2, c_3 are stored in $\mathbf{C}_v[m]$. Note that c''_1, c'_2, c_3 is always valid encryption of m, and hence we store the tuple in $\mathbf{C}_v$, the set of valid ciphertexts. The tuple (c'_1, c'_2, c'_3), all values returned by the server through $\mathsf{Put}[2, 1]$ are stored in $\mathbf{C}_s[m]$, the table of server ciphertexts. The same steps are also performed during Upd, except that here the index into $\mathbf{C}_v$ and $\mathbf{C}_s$ is the updated ciphertext. During $\mathsf{Get}[1, 2]$, the c_1, c_2, c_3 tuple returned by the server are added to $\mathbf{C}_r[m]$, the table of recovered ciphertexts, where m is the recovered plaintext value.

Note that, in G_1, the $\mathsf{Put}[2, 1]$ and $\mathsf{Upd}[2, 1]$ algorithms return the ciphertexts c'_1 along with the hashes. This change does not affect the outcome of the game as these values are ignored by the $\mathsf{Put}[1, 2]$

```
$\underline{\textsc{Main}(1^\lambda)}$   // $G_1^{\mathsf{A}}(1^\lambda)$, $G_2^{\mathsf{A}}(1^\lambda)$
$\sigma_S \leftarrow\!\!\$\ \mathsf{Init}(1^\lambda)$; $(\sigma_C, \sigma_S) \leftarrow\!\!\$\ \mathsf{Run}(\mathsf{Reg}, \epsilon, \sigma_S)$
$\mathsf{A}^{\textsc{Init},\textsc{Step},\textsc{Msg},\textsc{State}}(1^\lambda, \sigma_C)$; Ret $\mathsf{win}$

$\underline{\mathsf{Get}[1,2](\mathbf{a})}$
$(c_1, c_2, c_3) \leftarrow \mathtt{M}$; If $c_1 = \perp$ then ret $\perp$; $k_2 \leftarrow \mathsf{D}(1^\lambda, k, c_3)$
$m' \leftarrow \mathsf{D}(1^\lambda, k_2 \oplus c_2, c_1)$; $\mathbf{C}_r[m'] \leftarrow (c_1, c_2, c_3)$
Ret $m', \epsilon, \mathsf{True}$

$\underline{\mathsf{Upd}[1,1](\mathbf{a} = (t, m_1, m_2), \mathtt{M})}$
$k_1 \leftarrow \mathsf{H}(p, m_1)$; $k_2 \leftarrow \mathsf{H}(p, m_2)$; $t_2 \leftarrow \mathsf{H}(p, k_2)$
$\delta \leftarrow \mathsf{diff}(m_1, m_2)$; $c_3 \leftarrow \mathsf{E}(1^\lambda, k, k_2)$
$c' \leftarrow (k_1 \oplus k_2, c_3, \delta)$; $\mathtt{M} \leftarrow (u, t_1, t_2, c')$
$\mathbf{a} \leftarrow (m_1, m_2, k_1, k_2, t_1, t_2, c_1, c_2, c_3)$; Ret $(\mathbf{a}, \mathtt{M}, \mathsf{False})$

$\underline{\mathsf{Upd}[2,1](\sigma_S, \mathtt{M})}$
$(u, t_1, t_2, c_d, c_3, \delta) \leftarrow \mathtt{M}$; $(p, \mathbf{U}, \mathbf{fil}, \mathbf{own}) \leftarrow \sigma_S$
If $\mathbf{own}[u, t_1] = \perp$ then $(c_1', c_2') \leftarrow (\perp, \perp)$
Else
  If $\mathbf{fil}[t_2] = \perp$ then
    $(c_1, c_2) \leftarrow \mathbf{fil}[t_1]$; $c_1' \leftarrow \mathsf{patch}(c_1, \delta)$
    $c_2' \leftarrow c_2 \oplus c_d$; $\mathbf{fil}[t_2] \leftarrow (c_1', c_2')$
  Else $(c_1', c_2') \leftarrow \mathbf{fil}[t_2]$
  If $\mathbf{own}[t_2, u] = \perp$ then $\mathbf{own}[t_2, u] = c_3$
  Else $c_3 \leftarrow \mathbf{own}[t_2, u]$
$h \leftarrow \mathsf{H}(p, c_1')$; $\mathtt{M} \leftarrow (c_1', h, c_2', c_3)$
Ret $(p, \mathbf{U}, \mathbf{fil}, \mathbf{own}), \mathtt{M}, \mathsf{True}$

$\underline{\mathsf{Upd}[1,2](\mathbf{a}, \mathtt{M})}$
$(m_1, m_2, k_1, k_2, t_1, t_2, c_1, c_2, c_3) \leftarrow \mathbf{a}$; $(h, c_1', c_2', c_3') \leftarrow \mathtt{M}$
If $c_3 \neq c_3'$ then $\mathbf{a} \leftarrow \perp$
Else $c_1'' \leftarrow \mathsf{E}(1^\lambda, c_2' \oplus k_2, m_2)$; $h' \leftarrow \mathsf{H}(p, c_1'')$
  If $h = h'$ then
    $\mathbf{C}_v[m] \leftarrow (c_1'', c_2', c_3)$; $\mathbf{C}_s[m] \leftarrow (c_1', c_2', c_3')$; $\mathbf{a} \leftarrow t_2$
  Else $\mathbf{a} \leftarrow \perp$; Ret $(\mathbf{a}, \epsilon, \mathsf{True})$

$\underline{\mathsf{Get}[2,1](t)}$
$(p, \mathbf{U}, \mathbf{fil}, \mathbf{own}) \leftarrow \sigma_S$; $(c_1, c_2) \leftarrow \mathbf{fil}[t]$; $c_3 \leftarrow \mathbf{own}[t, u]$
If $c_3 = \perp$ then $(c_1', c_2') \leftarrow (\perp, \perp)$
$\mathtt{M} \leftarrow (c_1, c_2, c_3)$; $\sigma_S \leftarrow (p, \mathbf{U}, \mathbf{fil}, \mathbf{own})$; Ret $\sigma_S, \mathtt{M}, \mathsf{True}$
```

```
$\underline{\mathsf{Put}[1,1](m, \mathtt{M})}$
$\ell \leftarrow\!\!\$\ \{0,1\}^{\kappa(\lambda)}$; $c_1 \leftarrow \mathsf{E}(1^\lambda, \ell, m)$; $k_m \leftarrow \mathsf{H}(p, m)$
$t \leftarrow \mathsf{H}(p, k_m)$; $c_2 \leftarrow k_m \oplus \ell$; $c_3 \leftarrow \mathsf{E}(1^\lambda, k, k_m)$
$c \leftarrow (c_1, c_2, c_3)$; $\mathtt{M} \leftarrow (u, c, t)$
$\mathbf{a} \leftarrow (m, \ell, c_1, c_2, c_3, t)$; Ret $\mathbf{a}, \mathtt{M}, \mathsf{False}$

$\underline{\mathsf{Put}[2,1](\sigma_S, \mathtt{M})}$
$(p, \mathbf{U}, \mathbf{fil}, \mathbf{own}) \leftarrow \sigma_S$; $(u, c, t) \leftarrow \mathtt{M}$; $(c_1, c_2, c_3) \leftarrow c$
If $\mathbf{fil}[t] = \perp$ then $\mathbf{fil}[t] = (c_1, c_2)$ Else
  $(c_1, c_2) \leftarrow \mathbf{fil}[t]$
$h \leftarrow \mathsf{H}(p, p \| c_1)$
If $\mathbf{own}[t, u] = \perp$ then $\mathbf{own}[t, u] = c_3$
Else $c_3 \leftarrow \mathbf{own}[t, u]$
$\mathtt{M} \leftarrow (h, c_2, c_3)$
$\sigma_S \leftarrow (p, \mathbf{U}, \mathbf{fil}, \mathbf{own})$
Ret $\mathtt{M}, \sigma_S, \mathsf{True}$

$\underline{\mathsf{Put}[1,2](\mathbf{a}, \mathtt{M})}$
$(m, \ell, c_1, c_2, c_3, t) \leftarrow \mathbf{a}$; $(h, c_1', c_2', c_3') \leftarrow \mathtt{M}$
$\ell' \leftarrow c_2' \oplus k_m$ ; $c_1'' \leftarrow \mathsf{E}(1^\lambda, \ell', m)$
$h' \leftarrow \mathsf{H}(p, c_1'')$; $\mathbf{a} \leftarrow \perp$
If $h = h'$ then
  $\mathbf{C}_v[m] \leftarrow (c_1'', c_2', c_3)$; $\mathbf{C}_s[m] \leftarrow (c_1', c_2', c_3')$
$\mathbf{a} \leftarrow t$
Ret $(\mathbf{a}, \epsilon, \mathsf{True})$

$\underline{\textsc{WinCheck}(j)}$   // $G_2$
If $\mathbf{PS}[j] = \mathsf{Put}$ then
  $(\sigma_C, m) \leftarrow \mathtt{first}(\mathbf{a}[j, 1])$; $f \leftarrow \mathtt{last}(\mathbf{a}[j, 1])$
  $T[f] \leftarrow m$
If $\mathbf{PS}[j] = \mathsf{Upd}$ then
  $(\sigma_C, f, m_1, m_2) \leftarrow \mathtt{first}(\mathbf{a}[j, 1])$
  $f \leftarrow \mathtt{last}(\mathbf{a}[j, 1])$
  $T[f] \leftarrow m_2$
If  $\mathbf{PS}[j] = \mathsf{Get}$ then
  $(\sigma_C, f) \leftarrow \mathtt{first}(\mathbf{a}[j, 1])$; $m' \leftarrow \mathtt{last}(\mathbf{a}[j, 1])$
  $m \leftarrow T[f]$
  If $m' \neq m$ then
    If $\mathbf{C}_r[m'] = \mathbf{C}_s[m] = \mathbf{C}_v[m]$ then $\mathsf{win}_1 \leftarrow \mathsf{True}$
    Else If $\mathbf{C}_r[m'] \neq \mathbf{C}_s[m]$ then $\mathsf{win}_2 \leftarrow \mathsf{True}$ else
$\mathsf{win}_3 \leftarrow \mathsf{True}$
$\mathsf{win} \leftarrow \mathsf{win}_1 \vee \mathsf{win}_2 \vee \mathsf{win}_3$
```

Figure 10: Games G_1 and G_2 of Theorem 5.1. The Init, Step, State and Msg procedures for G_1 and G_2, and WinCheck for G_1 are the same as Rec and hence omitted. Also omitted is the trivial $\mathsf{Get}[1,1]$ procedure, which simply sets its outgoing message to its input f.

and $\mathsf{Upd}[1,2]$, except for updating $\mathbf{C}_v$, $\mathbf{C}_s$, and $\mathbf{C}_r$. However, maintaining $\mathbf{C}_v$, $\mathbf{C}_s$, and $\mathbf{C}_r$ itself has no effect on setting win and hence on the outcome of G_1. Thus, we have $\Pr[\textsc{Rec}^{\mathsf{A}}_{\mathsf{IRCE}}(1^\lambda)] = \Pr[G_1^{\mathsf{A}}(1^\lambda)]$.

Game G_2 differs from G_1 only on how WinCheck works. In G_1, as in Rec, the game maintains a table T through WinCheck. When the adversary runs a put or an update protocol to completion through Step, the games set $T[m] = f$, where m is the put/updated plaintext and f is the returned identifier. When the adversary runs $\mathsf{Get}(f)$ to completion, the games check if the recovered plaintext m' matches $m = T[f]$, and if not, set win to true. However, in G_2, when such a mismatch happens, the game goes through a series of steps before setting win. If $\mathbf{C}_v[m] = \mathbf{C}_s[m] = \mathbf{C}_r[m']$, the game sets win_1. If $\mathbf{C}_s[m] \neq \mathbf{C}_r[m']$, the game sets win_2. Otherwise, win_3 is set. It can be observed that if $m \neq m'$, one of $\mathsf{win}_1, \mathsf{win}_2, \mathsf{win}_3$ will get set, and since the winning condition in G_2 is $\vee_{i=1}^{i=3}\mathsf{win}_i$, it follows that $\Pr[G_1^{\mathsf{A}}(1^\lambda)] = \Pr[G_2^{\mathsf{A}}(1^\lambda)]$.

Now, we show that the probabilities of setting win_1 and win_2 are zero, leaving win_3 as the only way for A to break soundness. If $(c_1^r, c_2^r, c_3^r) = (c_1^v, c_2^v, c_3^v)$, then, m' has to be m, by the correctness of SE, since the latter is a valid ciphertext for m. Hence, if $m' \neq m$, the three ciphertext triples cannot be equal, meaning that $\Pr[G_2^{\mathsf{A}}(1^\lambda) \text{ sets } \mathsf{win}_1] = 0$.

If $\mathbf{C}_s[m] \neq \mathbf{C}_r[m']$, the game sets win_2, but note that both these triples are $\mathbf{fil}[f]$ and $\mathbf{own}[u, f]$, the difference being that $\mathbf{C}_s[m]$ was derived during $\mathsf{Put}[2, 1]$, or $\mathsf{Upd}[2, 1]$ while $\mathbf{C}_r[m']$ was derived during a run of $\mathsf{Get}[2, 1]$. However, given that $\mathbf{fil}$ and $\mathbf{own}$ are both immutable arrays, it follows that the two values have to be equal, meaning that $\Pr[G_2^{\mathsf{A}}(1^\lambda) \text{ sets } \mathsf{win}_2] = 0$.

In the setting of win_3, we have $\mathbf{C}_s[m] = \mathbf{C}_r[m']$, and $\mathbf{C}_s[m] \neq \mathbf{C}_v[m]$. Let $(c_1^s, c_2^s, c_3^s) \leftarrow \mathbf{C}_s[m]$ and $(c_1^v, c_2^v, c_3^v) \leftarrow \mathbf{C}_v[m]$. We know that $c_3^v = c_3^s$ because $\mathsf{Put}[1, 2]$ and $\mathsf{Upd}[1, 2]$ check for this condition, returning error on failure. Moreover, $c_2^v = c_2^s$ by construction, which means that $c_1^v \neq c_1^s$. But we know that $\mathsf{H}(p, c_1^v) = \mathsf{H}(p, c_1^s)$ because $\mathsf{Put}[1, 2]$ and $\mathsf{Upd}[1, 2]$ check for this condition as well, once again returning error on failure. This of course means that a collision has been found in HF. It is straightforward to describe an adversary B such that $\mathsf{Adv}^{\mathsf{cr}}_{\mathsf{HF},\mathsf{B}} = \Pr[G_2^{\mathsf{A}}(1^\lambda) \text{ sets } \mathsf{win}_3]$. Adding the above equations, we have $\mathsf{Adv}^{\mathsf{rec}}_{\mathsf{IRCE},\mathsf{A}}(\lambda) = \mathsf{Adv}^{\mathsf{cr}}_{\mathsf{HF},\mathsf{B}}(\lambda)$. □

C.2 Proof of Theorem 5.2

Proof. Let S be an unpredictable PT source which outputs $m(\lambda)$ plaintexts with length $\ell(\lambda, \cdot)$. and A be a PT adversary. Let $n : \mathbb{N} \to \mathbb{N}$ denote a bound on the total number of messages stored by the adversary (including both Put and Upd), and let $q_{\mathsf{S}}(\lambda) : \mathbb{N} \to \mathbb{N}$ denote a bound on the number of RO_1 queries made by the source.

Consider games G_1, G_2, G_3 and G_4 of Figure 11. Game G_1 is identical to the $\text{Priv}^{\mathsf{S},\mathsf{A}}_{\mathsf{IRCE}}$ game. In G_1, when the adversary imitates a Put protocol, and runs the first step through Step, the game derives the ciphertext c_1, c_2, c_3. Here, the key should be derived as $k_m \leftarrow \mathsf{RO}(m)$, and the tag should be derived as $t \leftarrow \mathsf{RO}(m)$. Instead, G_1 picks k_m and t as random $\kappa(\lambda)$-bit strings. However, if $p\|k_m$ or $p\|m$ have already been queried at the RO by S or A, then G_1 ensures consistency by using the existing values. This event sets the bad flag in G_1. When the adversary initiates an Update protocol, G_1 follows a similar procedure with k_2 and t_2.

Moreover, G_1 ensures that subsequent queries to RO at $p\|m$ or $p\|k_m$ are replied with k_m or t respectively. Such events also set the bad flag in G_1. Although $\mathsf{Put}[1, 1]$ and $\mathsf{Upd}[1, 1]$ are implemented differently in G_1, this does not affect the outcome of the game. Game G_2 does away with maintaining consistency in the RO replies, but remains identical-until-bad to G_1. We have

$$\Pr[\text{Priv}^{\mathsf{S},\mathsf{A}}_{\mathsf{IRCE}}(1^\lambda)] = \Pr[G_1^{\mathsf{S},\mathsf{A}}(1^\lambda)] \leq \Pr[G_2^{\mathsf{S},\mathsf{A}}(1^\lambda)] + \Pr[G_2^{\mathsf{S},\mathsf{A}}(1^\lambda) \text{ sets bad}]. \tag{1}$$

Game G_3 is identical to G_2, and G_4 differs from G_3 in that the c_3 components of ciphertexts are encryptions of random strings, instead of encryptions of k_m and k_2 during $\mathsf{Put}[1, 1]$ and $\mathsf{Upd}[1, 1]$. Here, parts of the $\mathsf{Put}[1, 1]$ procedures of G_3 (Left) and G_4 (Right) are provided for comparison. The $\mathsf{Upd}[1, 1]$ procedures are similarly related. There exists a CPA adversary B_1, and a KR adversary B_2

```
$\textsc{Main}(1^\lambda)$   // $\boxed{G_1^{\mathsf{S,A}}(1^\lambda), G_3^{\mathsf{S,A}}(1^\lambda)}$, $G_2^{\mathsf{S,A}}(1^\lambda), G_4^{\mathsf{S,A}}(1^\lambda)$
$\sigma_S \leftarrow_{\$} \mathsf{Init}(1^\lambda)$; $b \leftarrow_{\$} \{0,1\}$; $\vec{m_0}, \vec{m_1} \leftarrow_{\$} \mathsf{S}^{\mathsf{RO}}(1^\lambda, \epsilon)$
$b' \leftarrow_{\$} \mathsf{A}^{\textsc{Reg},\textsc{Put},\textsc{Upd},\textsc{Step},\textsc{Msg},\textsc{State},\mathsf{RO}}(1^\lambda)$; Ret $(b = b')$
```

```
$\mathsf{Put}[1,1](m, \mathtt{M})$
$\mathtt{n} \leftarrow \mathtt{n} + 1$; $k_m \leftarrow_{\$} \{0,1\}^{\kappa(\lambda)}$; $t \leftarrow_{\$} \{0,1\}^{\kappa(\lambda)}$
If $p \| m \in T$ then $\mathsf{bad} \leftarrow \mathsf{True}$ $\boxed{; k_m \leftarrow T[p \| m]}$
If $p \| k_m \in T$ then $\mathsf{bad} \leftarrow \mathsf{True}$ $\boxed{; t \leftarrow T[p \| k_m]}$
$\mathbf{k}[\mathtt{n}] \leftarrow k_m$; $\mathbf{m}[\mathtt{n}] \leftarrow m$; $\mathbf{t}[\mathtt{n}] \leftarrow t$
$\ell \leftarrow_{\$} \{0,1\}^{\kappa(\lambda)}$; $c_1 \leftarrow \mathsf{E}(1^\lambda, \ell, m)$
$c_2 \leftarrow k_m \oplus \ell$; $c_3 \leftarrow \mathsf{E}(1^\lambda, k, k_m)$; $c \leftarrow (c_1, c_2, c_3)$
$\mathtt{M} \leftarrow (u, c, t)$; $\mathbf{a} \leftarrow (m, \ell, c_1, c_2, c_3, t)$; Ret $\mathbf{a}, \mathtt{M}, 2, \mathsf{False}$
```

```
$\mathsf{Put}[2,1](\sigma_S, \mathtt{M})$
$(p, \mathbf{U}, \mathbf{fil}, \mathbf{own}) \leftarrow \sigma_S$; $(u, c, t) \leftarrow \mathtt{M}$; $(c_1, c_2, c_3) \leftarrow c$
If $\mathbf{fil}[t] = \perp$ then $\mathbf{fil}[t] = (c_1, c_2)$ Else $(c_1, c_2) \leftarrow \mathbf{fil}[t]$
If $\mathbf{own}[t, u] = \perp$ then $\mathbf{own}[t, u] = c_3$ else $c_3 \leftarrow \mathbf{own}[t, u]$
$h \leftarrow \mathsf{H}(p, p \| c_1)$; $\mathtt{M} \leftarrow (h, c_2, c_3)$; $\sigma_S \leftarrow (p, \mathbf{U}, \mathbf{fil}, \mathbf{own})$
Ret $\mathtt{M}, \sigma_S, 1, \mathsf{True}$
```

```
$\mathsf{Put}[1,2](\mathbf{a}, \mathtt{M})$
$(m, \ell, c_1, c_2, c_3, t) \leftarrow \mathbf{a}$; $(h, c_1', c_2', c_3') \leftarrow \mathtt{M}$
$\ell' \leftarrow c_2' \oplus k_m$ ; $c_1'' \leftarrow \mathsf{E}(1^\lambda, \ell', m)$; $h' \leftarrow \mathsf{H}(p, c_1'')$
If $h = h'$ then
  $\mathbf{C}_v[m] \leftarrow (c_1'', c_2', c_3)$; $\mathbf{C}_s[m] \leftarrow (c_1', c_2', c_3')$; $\mathbf{a} \leftarrow t$
Else $\mathbf{a} \leftarrow \perp$
Ret $(\mathbf{a}, \epsilon, 2, \mathsf{True})$
```

```
$\mathsf{RO}(x)$
For $i = 1$ to $\mathtt{n}$ do
  If $x = \mathbf{k}[i]$ then $\mathsf{bad} \leftarrow \mathsf{True}$$\boxed{; \text{Ret } \mathbf{t}[i]}$
  If $x = \mathbf{m}[i]$ then $\mathsf{bad} \leftarrow \mathsf{True}$$\boxed{; \text{Ret } \mathbf{k}[i]}$
If $x \notin T$ then $T[x] \leftarrow_{\$} \{0,1\}^{\kappa(\lambda)}$
Ret $T[x]$
```

```
$\mathsf{Put}[1,1](m, \mathtt{M})$   // $\boxed{G_3^{\mathsf{S,A}}(1^\lambda)}$, $G_4^{\mathsf{S,A}}(1^\lambda)$
$t \leftarrow_{\$} \{0,1\}^{\kappa(\lambda)}$; $\mathbf{k}[\mathtt{n}] \leftarrow k_m$; $\ell \leftarrow_{\$} \{0,1\}^{\kappa(\lambda)}$
$c_1 \leftarrow \mathsf{E}(1^\lambda, \ell, m)$
$k_m \leftarrow_{\$} \{0,1\}^{\kappa(\lambda)}$; $c_2 \leftarrow_{\$} \ell \oplus k_m$; $k' \leftarrow_{\$} \{0,1\}^{\kappa(\lambda)}$
$c_3 \leftarrow \mathsf{E}(1^\lambda, k, k')$; $\boxed{c_3 \leftarrow \mathsf{E}(1^\lambda, k, k_m)}$; $c \leftarrow (c_1, c_2, c_3)$
$\mathtt{M} \leftarrow (u, c, t)$; $\mathbf{a} \leftarrow (m, \ell, c_1, c_2, c_3, t)$; Ret $\mathbf{a}, \mathtt{M}, 2, \mathsf{False}$
```

```
$\mathsf{Upd}[1,1](\mathbf{a} = (t, m_1, m_2), \mathtt{M})$
$\mathtt{n} \leftarrow \mathtt{n} + 1$; $k_2 \leftarrow_{\$} \{0,1\}^{\kappa(\lambda)}$; $t_2 \leftarrow_{\$} \{0,1\}^{\kappa(\lambda)}$
$\mathbf{k}[\mathtt{n}] \leftarrow k_2$; $\mathbf{m}[\mathtt{n}] \leftarrow m_2$
$\mathbf{t}[\mathtt{n}] \leftarrow t_2$; $\delta \leftarrow \mathsf{diff}(m_1, m_2)$
If $p \| m_2 \in T$ then $\mathsf{bad} \leftarrow \mathsf{True}$ $\boxed{; k_2 \leftarrow T[p \| m_2]}$
If $p \| k_2 \in T$ then $\mathsf{bad} \leftarrow \mathsf{True}$ $\boxed{; t_2 \leftarrow T[p \| k_2]}$
$c_3 \leftarrow \mathsf{E}(1^\lambda, k, k_2)$; $c' \leftarrow (k_1 \oplus k_2, c_3, \delta)$
$\mathtt{M} \leftarrow (u, t_1, t_2, c')$
$\mathbf{a} \leftarrow (m_1, m_2, k_1, k_2, t_1, t_2, c_1, c_2, c_3)$
Ret $(\mathbf{a}, \mathtt{M}, 2, \mathsf{False})$
```

```
$\mathsf{Upd}[2,1](\sigma_S, \mathtt{M})$
$(u, t_1, t_2, c_d, c_3, \delta) \leftarrow \mathtt{M}$; $(p, \mathbf{U}, \mathbf{fil}, \mathbf{own}) \leftarrow \sigma_S$
If $\mathbf{own}[u, t_1] = \perp$ then $(c_1', c_2') \leftarrow (\perp, \perp)$
Else
  If $\mathbf{fil}[t_2] = \perp$ then
    $(c_1, c_2) \leftarrow \mathbf{fil}[t_1]$; $c_1' \leftarrow \mathsf{patch}(c_1, \delta)$
    $c_2' \leftarrow c_2 \oplus c_d$; $\mathbf{fil}[t_2] \leftarrow (c_1', c_2')$
  Else $(c_1', c_2') \leftarrow \mathbf{fil}[t_2]$
  If $\mathbf{own}[t_2, u] = \perp$ then $\mathbf{own}[t_2, u] = c_3$
Else $c_3 \leftarrow \mathbf{own}[t_2, u]$
$h \leftarrow \mathsf{H}(p, c_1')$; $\mathtt{M} \leftarrow (c_1', h, c_2', c_3)$
$\sigma_S \leftarrow (p, \mathbf{U}, \mathbf{fil}, \mathbf{own})$; Ret $\sigma_S, \mathtt{M}, 1, \mathsf{True}$
```

```
$\mathsf{Upd}[1,2](\mathbf{a}, \mathtt{M})$
$(m_1, m_2, k_1, k_2, t_1, t_2, c_1, c_2, c_3) \leftarrow \mathbf{a}$
$(h, c_1', c_2', c_3') \leftarrow \mathtt{M}$
If $c_3 \neq c_3'$ then $\mathbf{a} \leftarrow \perp$
Else
  $c_1'' \leftarrow \mathsf{E}(1^\lambda, c_2' \oplus k_2, m_2)$; $h' \leftarrow \mathsf{H}(p, c_1'')$
  If $h = h'$ then $\mathbf{C}_v[m] \leftarrow (c_1'', c_2', c_3)$
    $\mathbf{C}_s[m] \leftarrow (c_1', c_2', c_3')$; $\mathbf{a} \leftarrow t_2$
  Else $\mathbf{a} \leftarrow \perp$
Ret $(\mathbf{a}, \epsilon, 2, \mathsf{True})$
```

```
$\mathsf{Upd}[1,1](\mathbf{a} = (t, m_1, m_2), \mathtt{M})$   //   $\boxed{G_3^{\mathsf{S,A}}(1^\lambda)}$, $G_4^{\mathsf{S,A}}(1^\lambda)$
$k_2 \leftarrow_{\$} \{0,1\}^{\kappa(\lambda)}$; $t_2 \leftarrow_{\$} \{0,1\}^{\kappa(\lambda)}$
$\delta \leftarrow \mathsf{diff}(m_1, m_2)$
$k' \leftarrow_{\$} \{0,1\}^{\kappa(\lambda)}$; $c_3 \leftarrow \mathsf{E}(1^\lambda, k, k')$
$\boxed{c_3 \leftarrow \mathsf{E}(1^\lambda, k, k_m)}$
$c' \leftarrow (k_1 \oplus k_2, c_3, \delta)$; $\mathtt{M} \leftarrow (u, t_1, t_2, c')$
$\mathbf{a} \leftarrow (m_1, m_2, k_1, k_2, t_1, t_2, c_1, c_2, c_3)$
Ret $(\mathbf{a}, \mathtt{M}, 2, \mathsf{False})$
```

Figure 11: Games G_1, G_2, G_3 and G_4. The boxed code is part of G_1 and G_3.

```
$\textsc{Main}(1^\lambda)$  // $\boxed{\boxed{H_1^{\mathsf{S},\mathsf{A}}(1^\lambda)}}$, $\boxed{H_2^{\mathsf{S},\mathsf{A}}(1^\lambda)}$, $H_3^{\mathsf{S},\mathsf{A}}(1^\lambda)$
$\sigma_S \leftarrow\!\!\$\ \mathsf{Init}(1^\lambda)$; $b \leftarrow\!\!\$\ \{0,1\}$; $\vec{m_0}, \vec{m_1} \leftarrow\!\!\$\ \mathsf{S}^{\mathsf{RO}_1}(1^\lambda, \epsilon)$
$b' \leftarrow\!\!\$\ \mathsf{A}^{\textsc{Reg},\textsc{Put},\textsc{Upd},\textsc{Step},\textsc{Msg},\textsc{State},\mathsf{RO}_2}(1^\lambda)$
Ret $\mathsf{bad}$

$\mathsf{Put}[1,1](m, \mathtt{M})$
$\mathtt{n} \leftarrow \mathtt{n} + 1$; $k_m \leftarrow\!\!\$\ \{0,1\}^{\kappa(\lambda)}$; $t \leftarrow\!\!\$\ \{0,1\}^{\kappa(\lambda)}$
$\mathbf{k}[\mathtt{n}] \leftarrow k_m$; $\mathbf{m}[\mathtt{n}] \leftarrow m$; $\mathbf{t}[\mathtt{n}] \leftarrow t$
$\boxed{\boxed{\text{If } p\|m \in T_1 \text{ or } p\|k_m \in T_1 \text{ then } \mathsf{bad} \leftarrow \mathsf{True}}}$
$\boxed{\text{If } p\|m \in T_2 \text{ or } p\|k_m \in T_2 \text{ then } \mathsf{bad} \leftarrow \mathsf{True}}$
$\ell \leftarrow\!\!\$\ \{0,1\}^{\kappa(\lambda)}$; $c_1 \leftarrow \mathsf{E}(1^\lambda, \ell, m)$
$c_2 \leftarrow k_m \oplus \ell$; $c_3 \leftarrow \mathsf{E}(1^\lambda, k, k_m)$
$c \leftarrow (c_1, c_2, c_3)$
$\mathtt{M} \leftarrow (u, c, t)$; $\mathbf{a} \leftarrow (m, \ell, c_1, c_2, c_3, t)$
Ret $\mathbf{a}, \mathtt{M}, 2, \mathsf{False}$

$\mathsf{Upd}[1,1](\mathbf{a} = (t, m_1, m_2), \mathtt{M})$
$\mathtt{n} \leftarrow \mathtt{n} + 1$; $k_2 \leftarrow\!\!\$\ \{0,1\}^{\kappa(\lambda)}$; $t_2 \leftarrow\!\!\$\ \{0,1\}^{\kappa(\lambda)}$
$\mathbf{k}[\mathtt{n}] \leftarrow k_2$; $\mathbf{m}[\mathtt{n}] \leftarrow m_2$
$\mathbf{t}[\mathtt{n}] \leftarrow t_2$; $\delta \leftarrow \mathsf{diff}(m_1, m_2)$
$\boxed{\boxed{\text{If } p\|m_2 \in T_1 \text{ or } p\|k_2 \in T_1 \text{ then } \mathsf{bad} \leftarrow \mathsf{True}}}$
$\boxed{\text{If } p\|m_2 \in T_2 \text{ or } p\|k_2 \in T_2 \text{ then } \mathsf{bad} \leftarrow \mathsf{True}}$
$c_3 \leftarrow \mathsf{E}(1^\lambda, k, k_2)$; $c' \leftarrow (k_1 \oplus k_2, c_3, \delta)$
$\mathtt{M} \leftarrow (u, t_1, t_2, c')$
$\mathbf{a} \leftarrow (m_1, m_2, k_1, k_2, t_1, t_2, c_1, c_2, c_3)$
Ret $(\mathbf{a}, \mathtt{M}, 2, \mathsf{False})$

$\mathsf{RO}_1(x)$
If $x \notin T$ then $T[x] \leftarrow\!\!\$\ \{0,1\}^{\kappa(\lambda)}$
$T_1[x] \leftarrow T[x]$; Ret $T[x]$

$\mathsf{RO}_2(x)$
If $x \in \mathbf{k} \cup \mathbf{m}$ then $\mathsf{bad} \leftarrow \mathsf{True}$
If $x \notin T$ then $T[x] \leftarrow\!\!\$\ \{0,1\}^{\kappa(\lambda)}$
$T_2[x] \leftarrow T[x]$; Ret $T[x]$
```

Figure 12: Games H_1, H_2 and H_3.

such that

$$\mathsf{Adv}^{\mathsf{cpa}}_{\mathsf{SE},\mathsf{B}_1}(\lambda) = \Pr[G_4^{\mathsf{S},\mathsf{A}}(1^\lambda)] - \Pr[G_3^{\mathsf{S},\mathsf{A}}(1^\lambda)] \ \ , \ \ \Pr[G_4^{\mathsf{S},\mathsf{A}}(1^\lambda)] \le m(\lambda)\mathsf{Adv}^{\mathsf{cpa}}_{\mathsf{SE},\mathsf{B}_2}(\lambda) + m(\lambda)\mathsf{Adv}^{\mathsf{kr}}_{\mathsf{SE},\mathsf{B}_3}(\lambda) \ , \tag{2}$$

where m is the bound on the size of S output along with the total number of procedure calls made by A.

Consider games H_1, H_2 and H_3 of Figure 12, where, H_1 is G_2, except that setting bad wins H_1. Both the source S and the adversary A can set bad. Given that the source is not provided p, the probability that S sets bad in H_1 is bounded by $q_{\mathsf{S}}(\lambda)n(\lambda)/2^{\kappa(\lambda)}$. In H_2, the source cannot set bad, as the RO_1 query points of S are not taken into account while testing for bad. We have

$$\Pr[H_2^{\mathsf{S},\mathsf{A}}(1^\lambda) \text{ sets bad}] - \Pr[H_1^{\mathsf{S},\mathsf{A}}(1^\lambda) \text{ sets bad}] \le \frac{q_{\mathsf{S}}(\lambda)(\lambda)n(\lambda)}{2^{\kappa(\lambda)}}. \tag{3}$$

Consider an adversary A' which runs A, keeps track of all the RO queries of A and when A finishes, repeats all the queries. By running A', testing for bad can be localized to RO_2 and dropped in $\mathsf{Put}[1,1]$ and $\mathsf{Upd}[1,1]$. This change is implemented in H_3. We have

$$\Pr[H_2^{\mathsf{S},\mathsf{A}}(1^\lambda) \text{ sets bad}] \le \Pr[H_3^{\mathsf{S},\mathsf{A}'}(1^\lambda) \text{ sets bad}] \ . \tag{4}$$

In H_4, as in G_4, the c_3 components of the ciphertexts are replaced by encryptions of random strings. In H_5, the ciphertexts c_1 are derived as encryptions of random strings, and not as encryptions of messages output by S. There exist adversaries $\mathsf{B}_3, \mathsf{B}_4$ and B_5 such that

$$|\Pr[H_5^{\mathsf{S},\mathsf{A}'}(1^\lambda) \text{ sets bad}] - \Pr[H_3^{\mathsf{S},\mathsf{A}'}(1^\lambda) \text{ sets bad}]| \le \mathsf{Adv}^{\mathsf{cpa}}_{\mathsf{SE},\mathsf{B}_4}(\lambda) + n(\lambda)(\mathsf{Adv}^{\mathsf{cpa}}_{\mathsf{SE},\mathsf{B}_4}(\lambda) + \mathsf{Adv}^{\mathsf{kr}}_{\mathsf{SE},\mathsf{B}_5}(\lambda)\ ). \tag{5}$$

Finally, in H_5, the adversary receives no information about the messages output by S, and it follows that

$$\Pr[H_5^{\mathsf{S},\mathsf{A}}(1^\lambda) \text{ sets bad}] \le \frac{q_{\mathsf{A}}(\lambda)n(\lambda)}{2^{\kappa(\lambda)}} + \le q_{\mathsf{S}}(\lambda)n(\lambda)\mathbf{GP}_{\mathsf{S}}(\lambda) \ . \tag{6}$$

where $q_{\mathsf{A}}(\lambda) : \mathbb{N} \to \mathbb{N}$ is a bound on the number of RO_2 queries made by A. Adding the above, we

```
$\mathtt{dist}_e(s_1, s_2)$
For $i = 1$ to $|s_1|$ do $D[i, 0] \leftarrow i$
For $j = 1$ to $|s_2|$ do $D[0, j] \leftarrow j$
For $i = 1$ to $|s_1|$ do
  For $j = 1$ to $|s_2|$ do
    If $s_1[i] = s_2[j]$ then $D[i, j] \leftarrow D[i-1, j-1]$
    Else
      $D[i, j] \leftarrow \min(D[i-1, j] + 1,$
                    $D[i, j-1] + 1, D[i-1, j-1] + 1)$
Ret $D[|s_1|, |s_2|]$
```

```
$\mathsf{patch}_e(S, s_1)$
For $(\alpha, \beta, \gamma) \in S$
  If $\alpha = \mathsf{r}$ then $s_1[\beta] \leftarrow \gamma$
  If $\alpha = \mathsf{i}$ then $s_1 \leftarrow s_1[1, \beta] + \gamma + s_1[\beta + 1, |s_1|]$
  If $\alpha = \mathsf{d}$ then $s_1 \leftarrow s_1[1, \beta - 1] + s_1[\beta + 1, |s_1|]$
Ret $s_1$
```

```
$\mathsf{diff}_e(D, s_1, s_2)$
$i \leftarrow |s_1|$; $j \leftarrow |s_2|$; $S \leftarrow \epsilon$
While $i > 0$ and $j > 0$
  $d \leftarrow D[i-1, j-1]$; $t \leftarrow D[i-1, j]$
  $l \leftarrow D[i, j-1]$
  If $d < t$ and $d < l$ then
    If $d < D[i, j]$ then $S \leftarrow S \cup \{(\mathsf{r}, i, s_2[j])\}$
    $i \leftarrow i - 1$; $j \leftarrow j - 1$; continue
  If $l < t$ then and $l \leq D[i, j]$
    $S \leftarrow S \cup \{(\mathsf{i}, i, s_2[j])\}$; $j \leftarrow j - 1$; continue
  $S \leftarrow S \cup \{(\mathsf{d}, i)\}$; $i \leftarrow i - 1$
Ret $S$
```

Figure 13: The $\mathtt{dist}_e, \mathsf{diff}_e$, and patch_e algorithms for edit distance.

have

$$\mathsf{Adv}^{\mathsf{priv}}_{\mathsf{IRCE[SE],S,A}}(\lambda) \leq 2(n(\lambda)+1)\mathsf{Adv}^{\mathsf{cpa}}_{\mathsf{SE,C_1}}(\lambda) + 2n(\lambda)\mathsf{Adv}^{\mathsf{kr}}_{\mathsf{SE,C_2}}(\lambda) + \frac{(q_{\mathsf{S}}(\lambda)+q_{\mathsf{A}}(\lambda))n(\lambda)}{2^{\kappa(\lambda)}} + q_{\mathsf{A}}(\lambda)n(\lambda)\mathbf{GP}_{\mathsf{S}}(\lambda)\ .$$

where C_1 and C_2 are the ones among the CPA and KR adversaries with highest advantage. □

D The IRCE2 scheme

Edit distance. The edit distance between two strings s_1 and s_2 over $\mathbf{\Sigma}$ is the minimum number of single character modifications, including insertion, deletion, and substitution that need to be performed to convert s_1 to s_2. We define edit distance $\mathtt{dist}_e$ and associated algorithms diff_e and patch_e in Figure 13.

IVT. Let E be a blockcipher with blocksize $w(\lambda)$ and $\kappa(\lambda)$-bit keys. The $\mathsf{IVT[E]} = (\mathsf{E_{IVT}}, \mathsf{D_{IVT}})$ SE scheme is described in Figure 14. We assume that plaintext lengths are exact multiples of $w(\lambda)$, a restriction that can be circumvented via an appropriate padding.

At a high level, IVT is like CTR mode of operation with E, but instead of having a single starting point for the counter, it contains a table accompanying the ciphertext that says what counter value to use for each block of data. This table can be compressed down when the counter values are increasing incrementally, and encryption the scheme works just like CTR. But having this table enables inserting, deleting and changing blocks. For example, given a ciphertext of ℓ-blocks, if a block has to be inserted in the middle, it is XORed with the output of E on counter $\ell + 1$. The table is modified to indicate this aberration in the middle, but can still be compressed efficiently.

The $\mathsf{diff_{IVT}}$ algorithm (Figure 14), given two plaintexts m_1, m_2 computes the information S that needs to applied to a ciphertext c_1 of m_1 under k to change it to a ciphertext of m_2. The $\mathsf{patch_{IVT}}$ algorithm (Figure 14) takes S and c_1 and returns c_2, a ciphertext of m_2. Here, c_1 does not have to an output of $\mathsf{E_{IVT}}$; it could be a result of previous patching efforts and hence contain a modified IV table.

We state the following proposition, which is easy to verify from the pseudocode of IVT.

Proposition D.1. *For all $c = (c_1, c_2, \mathtt{iv}_l)$, for all $k \in \{0,1\}^{\kappa(\lambda)}$, if $\mathsf{D_{IVT}}(1^\lambda, k, c) = m$ then for all $m_2 \in \{0,1\}^*$, it holds that $\mathsf{D_{IVT}}(1^\lambda, k, \mathsf{patch_{IVT}}(1^\lambda, c, \mathsf{diff_{IVT}}(1^\lambda, k, m_1, m_2))) = m_2$.*

```
$\underline{\mathsf{E}_{\mathsf{IVT}}(1^\lambda, k, m)}$
For $i = 1$ to $|m|/w(\lambda)$ do $\mathsf{iv}[i] = i$
$c_1 \leftarrow \mathsf{CTR}[\mathsf{E}](k, m)$; $c_2 \leftarrow \texttt{compress}_{\mathsf{IVT}}(\mathsf{iv})$
$\texttt{iv}_l \leftarrow i$; Ret $(c_1, c_2, \texttt{iv}_l)$

$\underline{\texttt{compress}_{\mathsf{IVT}}(\mathsf{iv})}$
While $i < |\mathsf{iv}|$
  $s \leftarrow \mathsf{iv}[i]$; $j \leftarrow 0$; While $\mathsf{iv}[i+j] = s + j$; $j \leftarrow j + 1$
  $\mathsf{iv}_c \leftarrow \mathsf{iv}_c \cup \{s, j\}$
Ret $\mathsf{iv}_c$

$\underline{\mathsf{patch}_{\mathsf{IVT}}(1^\lambda, c, (\delta, \texttt{iv}_l))}$
$(c_1, c_2, \texttt{iv}_l) \leftarrow c$; $\mathsf{iv} \leftarrow \texttt{expand}_{\mathsf{IVT}}(c_2)$
For $(\alpha, \beta, \gamma) \in \delta$
  If $\alpha = \mathsf{r}$ then $c_1[\beta] \leftarrow c_1[\beta] \oplus \gamma$
  If $\alpha = \mathsf{i}$ then
    $c_1 \leftarrow c_1[1, \beta] \| \gamma[1] \| c_1[\beta + 1, |c_1|]$
    $\mathsf{iv} \leftarrow \mathsf{iv}[1, \beta] \| \gamma[2] \| \mathsf{iv}[\beta + 1, |\mathsf{iv}|]$
  If $\alpha = \mathsf{d}$ then
    $c_1 \leftarrow c_1[1, \beta - 1] \| c_1[\beta + 1, |c_1|]$; $\mathsf{iv} \leftarrow \mathsf{iv}[1, \beta] \| \mathsf{iv}[\beta + 1, |\mathsf{iv}|]$
$c_2 \leftarrow \texttt{compress}_{\mathsf{IVT}}(\mathsf{iv})$; Ret $(c_1, c_2, \texttt{iv}_l)$
```

```
$\underline{\mathsf{D}_{\mathsf{IVT}}(1^\lambda, k, c)}$
$(c_1, c_2, \texttt{iv}_l) \leftarrow c$; $\mathsf{iv} \leftarrow \texttt{expand}_{\mathsf{IVT}}(c_2)$
For $i = 1$ to $|c_1|/w(\lambda)$ do $m[i] \leftarrow \mathsf{E}(1^\lambda, k, \mathsf{iv}[i]) \oplus c_1[i]$
Ret $m$

$\underline{\texttt{expand}_{\mathsf{IVT}}(\mathsf{iv}_c)}$
$k \leftarrow 1$
For $(a, b)$ in $\mathsf{iv}_c$ do
  For $i = 1$ to $b$ do $\mathsf{iv}[k] \leftarrow a + i$; $k \leftarrow k + 1$
Ret $\mathsf{iv}$

$\underline{\mathsf{diff}_{\mathsf{IVT}}(1^\lambda, k, m_1, m_2, \texttt{iv}_l)}$
$D \leftarrow \texttt{dist}_e(m_1, m_2)$; $\delta \leftarrow \mathsf{diff}_e(D, m_1, m_2)$
$n \leftarrow |m_1|/w(\lambda)$
For $(\alpha, \beta, \gamma) \in \delta$
  If $\alpha = \mathsf{r}$ then $\alpha' \leftarrow \mathsf{r}$; $\beta' \leftarrow \beta$; $\gamma' \leftarrow m_1[\beta] \oplus m_2[\beta]$
  If $\alpha = \mathsf{d}$ then $\alpha' \leftarrow \mathsf{d}$; $\beta' \leftarrow \beta$
  If $\alpha = \mathsf{i}$ then
    $\alpha' \leftarrow \mathsf{i}$; $\beta' \leftarrow \beta$; $\texttt{iv}_l \leftarrow \texttt{iv}_l + 1$
  $\gamma'[1] \leftarrow \mathsf{E}(1^\lambda, k, \texttt{iv}_l) \oplus \gamma$; $\gamma'[2] \leftarrow \texttt{iv}_l$
  $S \leftarrow S \cup \{(\alpha', \beta', \gamma')\}$
Ret $S, \texttt{iv}_l$
```

Figure 14: **Bottom:** The IVT SE scheme with a blockcipher E of blocksize $w(\lambda)$.

We do not explicitly prove or require incremental encryption security of IVT, but we will reason about its security as part of the $\mathsf{IRCE2}$ construction. However, we do observe that the SE scheme $\mathsf{IVT}[\mathsf{E}] = (\mathsf{E}_{\mathsf{IVT}}, \mathsf{D}_{\mathsf{IVT}})$ is deterministic CPA secure, as encryption here simply runs the CTR mode with E and adds a few other elements which can be generated knowing only the length of the ciphertext.

THE $\mathsf{IRCE2}$ SCHEME. Let H denote a hash function H with $\kappa(\lambda)$-bit keys and $\kappa(\lambda)$-bit outputs. Let E be a blockcipher with blocksize E and $\kappa(\lambda)$-bit keys. The $\mathsf{IRCE2}[\mathsf{H}, \mathsf{E}] = (\mathsf{Init}, \mathsf{Reg}, \mathsf{Put}, \mathsf{Get}, \mathsf{Upd})$ iMLE scheme resembles IRCE, but uses $\mathsf{IVT}[\mathsf{E}] = (\mathsf{E}_{\mathsf{IVT}}, \mathsf{D}_{\mathsf{IVT}})$ as the SE scheme. The Reg protocol and the Init algorithm are the same as in IRCE (Figure 7). The $\mathsf{Put}, \mathsf{Get}$ and Upd protocols are described in Figure 15.

Proposition C.1 can be extended in a simple manner to argue that $\mathsf{IRCE2}$ supports deduplication, as the Put protocols in the two schemes are essentially the same. The difference in the Upd protocols does not play any role.

In Section 5, we define incremental updates w.r.t Hamming distance. That definition can be modified for edit distance by simply replacing the $u(\mathsf{HAMM}(m_1, m_2))$ bound with $u(\texttt{dist}_e(m_1, m_2))$. Specifically, we consider edit distance with alphabet $\mathbf{\Sigma} = \{0, 1\}^{w(\lambda)}$.

Proposition D.2. *Then* $\mathsf{IRCE2}[\mathsf{H}, \mathsf{E}]$ *supports incremental updates w.r.t edit distance.*

Proof. It is easy to observe that the IVT SE scheme supports incremental updates w.r.t edit distance: the $\mathsf{diff}_{\mathsf{IVT}}$ algorithm (Figure 14), given two plaintexts m_1, m_2 and a key k produces S such that $|S| \leq 2|\delta|$, where δ in turn consists of $\texttt{dist}_e(m_1, m_2)$ elements, each element no more in size than $\log(|m_1| + |m_2|)$. The $\mathsf{patch}_{\mathsf{IVT}}$ algorithm, given such an S, can convert a ciphertext for m_1 (under k) to a ciphertext for m_2. Now, by inspecting the Upd protocol, it can be checked that the total

```
Put[1]((k, u, p), m)                                              Put[2](σ_S)
$\ell \leftarrow_{\$} \{0,1\}^{\kappa(\lambda)}$; $c_1 \leftarrow \mathsf{E}_{\mathsf{IVT}}(1^\lambda, \ell, m)$
$k_m \leftarrow \mathsf{H}(p, m)$; $c_2 \leftarrow k_m \oplus \ell$; $c_3 \leftarrow$
$\mathsf{E}_{\mathsf{IVT}}(1^\lambda, k, k_m)$
$t \leftarrow \mathsf{H}(p, k_m)$
                        $\xrightarrow{u, c_1, c_2, c_3, t}$
                                  $(c_1, c_2) \leftarrow \mathtt{SiffE}(\mathbf{fil}, t, c_1, c_2)$; $h \leftarrow \mathsf{H}(p, c_1)$
                                  $c_3 \leftarrow \mathtt{SiffE}(\mathbf{own}, (t, u), c_3)$
                                  $\mathtt{SiffE}(\mathbf{upd}, (u, t), (c_2, \epsilon))$
                        $\xleftarrow{h, c_2', c_3'}$
If $c_3 \neq c_3'$ then ret $\perp$
$\ell' \leftarrow c_2' \oplus k_m$ ; $c_1'' \leftarrow \mathsf{E}_{\mathsf{IVT}}(1^\lambda, \ell', m)$
$h' \leftarrow \mathsf{H}(p, c_1'')$
If $h = h'$ then ret $t$ Else ret $\perp$

Upd[1]((k, u, p), t, m_1, m_2)                                    Upd[2](σ_S)
$k_1 \leftarrow \mathsf{H}(p, m_1)$; $t_1 \leftarrow \mathsf{H}(p, k_1)$
                        $\xrightarrow{u, t_1}$
                                  $(c_1, c_2) \leftarrow \mathbf{upd}[u, t_1]$; $(c_1', c_2', \mathtt{iv}_l) \leftarrow c_1$
                        $\xleftarrow{c_2, \mathtt{iv}_l}$
$\ell \leftarrow k_1 \oplus c_2$; $k_2 \leftarrow \mathsf{H}(p, m_2)$; $c_3 \leftarrow \mathsf{E}(1^\lambda, k, k_2)$
$\delta \leftarrow \mathsf{diff}_{\mathsf{IVT}}(\ell, m_1, m_2, \mathtt{iv}_l)$; $c_\delta \leftarrow k_1 \oplus k_2$
$t_2 \leftarrow \mathsf{H}(p, k_2)$
                        $\xrightarrow{u, t_1, t_2, \delta, c_3, c_\delta}$
                                  $(c_1, c_2) \leftarrow \mathbf{upd}[u, t_1]$; $c_1' \leftarrow \mathsf{patch}_{\mathsf{IVT}}(c_1, \delta)$
                                  $c_2' \leftarrow c_2 \oplus c_\delta$; $\mathtt{SiffE}(\mathbf{upd}, (u, t_2), (c_1', c_2'))$
                                  $c_3 \leftarrow \mathtt{SiffE}(\mathbf{own}, (u, t_2), c_3)$
```

Figure 15: The Put and Upd protocols of IRCE2. The **fil** and **own** tables are immutable, and support the set-iff-empty operation (SiffE) explained in text.

length of the transmitted messages is $2\lambda + 6\kappa(\lambda) + |S|$, where $S = \mathsf{diff}_{\mathsf{IVT}}(m_1, m_2)$ as above. Letting $\kappa(\lambda) = \lambda$, this in turn is bounded by $a \log(|m_1| + |m_2|) + b\lambda$ for small constants a and b, proving the proposition. □

The proofs closely follow those from IRCE, so we reuse parts where applicable, and highlight the differences.

Theorem D.3. *If* H *is collision resistant, then* IRCE2[H, E] *is* REC*-secure.*

Proof. As with IRCE, the adversary A cannot hope to get a non-error output by furnishing a new t for Get, as the server checks that the tag was previously handled by the client. Now, if t was returned during a previous Put instance, then the same argument we used with IRCE carries over, as Upd does not play a role. Specifically, using the immutability of **fil** and **own**, we argue that the ciphertext stored in the server cannot change between the failed Get request and the last time the file was put. However, Put ensures that the hash of the ciphertext stored on the server matches with the hash of a correctly formed ciphertext of the plaintext. Consequently, if t does correspond to a Put instance, then A is in effect finding a pair of colliding inputs, namely the hash inputs involved in the comparison. We can argue that a CR adversary B can be built which has the same advantage as A in this case. The other case is when t is due to a ciphertext formed from an update. Consider such an Upd instance, and let $m_1, c_1, t_1, m_2, c_2, t_2$ denote the original and updated plaintext-ciphertext-tags. If c_2 is a valid ciphertext for m_2, then by immutability and correctness of decryption, Get with t_2 will return m_2. If c_2 is not a valid ciphertext for m_2, then either c_1 is not a valid ciphertext for m_1, or the update process must have introduced an error. The latter is ruled out by Proposition D.1. Now, if c_1 was placed on the server via a Put instance, we fall back to the first case (meaning we have an invalid ciphertext from a Put instance), and can build a CR adversary with matching advantage. If c_1 was placed via Upd, then the same argument can be applied recursively, until we reach a ciphertext from a Put instance. □

Priv security with edit distance. The following theorem shows Priv-security. We consider a variant of the Priv game of Figure 4, where the δ supplied to Upd is interpreted in terms of edit distance. Here, supporting delete is tricky, as the adversary could potentially start with an unpredictable plaintext, but then delete a sufficient number of characters to make the result predictable, and then break security. To prevent this, we require that the messages formed as result of the updates should also be unpredictable. We enforce this condition by making the source S output a list of allowed diff_e-outputs along with two tuples of messages. The adversary can only pick updates from this list. An edit source S is an algorithm that on input 1^λ returns $(\mathbf{m}_0, \mathbf{m}_1, \mathbf{d})$. Here $\mathbf{d}$ is a list of diff_e-values. Consider source S' defined as below, where $\|$ denotes adding an element to a tuple.

```
$\underline{\mathsf{S}'(1^\lambda)}$
$(\mathbf{m}_0, \mathbf{m}_1, \mathbf{d}) \leftarrow\!\!\$\ \mathsf{S}(1^\lambda)$
For $b \in \{0,1\}$ and $m \in |\mathbf{m}_b|$ and $\delta \in \mathbf{d}$ do $\mathbf{m}_b \leftarrow \mathbf{m}_b \| \mathsf{patch}_e(1^\lambda, m, \delta)$
Ret $(\mathbf{m}_0, \mathbf{m}_1)$
```

We require that S' should satisfy the formatting restrictions of regular sources, including length and equality. We say that S is unpredictable if S' is unpredictable. We now sketch the Priv game for edit distances, with an edit source. Only the main, and Upd procedures differ. In main, the source S is executed to get $\mathbf{m}_0, \mathbf{m}_1, \delta$. In Upd$(i, \delta)$, before running the operations in Figure 4, a check that $\delta \in \mathbf{d}$ is performed.

PRF security. Let E be a blockcipher with blocksize $w(\lambda)$ and $\kappa(\lambda)$-bit keys. The PRF game with E and adversary A starts by picking $b \leftarrow\!\!\$\ \{0,1\}$ and $k \leftarrow\!\!\$\ \{0,1\}^{\kappa(\lambda)}$ and runs A with access to a Fn procedure, which A can query at points x. Upon such a query, if $b = 1$, the game returns $\mathsf{E}(1^\lambda, k, x)$; otherwise, it returns a random but consistent value. Finally A exits with output b' and wins the game if $b = b'$. We define advantage $\mathsf{Adv}^{\mathsf{PRF}}_{\mathsf{E},\mathsf{A}}(\lambda) = 2\Pr[\mathrm{PRF}^{\mathsf{A}}_{\mathsf{E}}(1^\lambda)] - 1$ and say that E is PRF-secure if no PT A has non-negligible advantage.

Theorem D.4. *If* E *is* PRF*-secure, then* $\mathsf{IRCE}_{\mathsf{RO}}[\mathsf{E}]$ *is* Priv*-secure.*

Proof. Let G_1 denote the $\mathrm{Priv}_{\mathsf{IRCE}}$ game included with an unpredictable edit source S. Let A denote a PT adversary. Without loss of generality, assume that A makes only permitted Upd queries. Let n denote a bound on the number of plaintexts put on the server by A and $q_{\mathsf{S}}(\lambda) : \mathbb{N} \to \mathbb{N}$ denote a bound on the number of RO queries made by S. Let G_2 denote the game similar to G_1, where the c_3, instead of being encryptions of the message-derived keys k_m, are replaced with encryptions of random strings. All the c_3 encryptions are performed under the legitimate client's secret key, which is never revealed to the server. There exists B such that

$$\mathsf{Adv}^{\mathsf{cpa}}_{\mathsf{IVT}[\mathsf{E}],\mathsf{B}}(\lambda) = \Pr[G_1^{\mathsf{A}}(1^\lambda)] - \Pr[G_2^{\mathsf{A}}(1^\lambda)].$$

It follows that there exists another adversary B' such that $\mathsf{Adv}^{\mathsf{cpa}}_{\mathsf{IVT}[\mathsf{E}],\mathsf{B}}(\lambda) = \mathsf{Adv}^{\mathsf{PRF}}_{\mathsf{E},\mathsf{B}'}(\lambda)$. Let G_3 denote the game where in $\mathsf{Put}[1,1]$, the key and tag, which should be derived as $k_m \leftarrow \mathsf{RO}(m)$, and $t \leftarrow \mathsf{RO}(m)$ are picked instead as random $\kappa(\lambda)$-bit strings, but if $p\|k_m$ or $p\|m$ have already been queried at the RO by S or A, then G_3 ensures consistency by using the existing values, but sets bad. When the adversary initiates an Update protocol, G_3 follows a similar procedure with k_2 and t_2. Subsequent queries to RO at $p\|m$ or $p\|k_m$ are replied with k_m or t respectively, but sets bad. In G_4, all the consistency measures are done away with, and the k_m and t values have no relations with the RO outputs on the associated $\mathbf{m}$ points. We have $\Pr[G_3^{\mathsf{A}}(1^\lambda)] \leq \Pr[G_4^{\mathsf{A}}(1^\lambda)] + \Pr[G_4^{\mathsf{A}}(1^\lambda) \text{ sets bad}]$.

Consider the PRF-game with n keys PRF$[n]$. We build a PRF$[n]$-adversary C which runs A on G_4. The n-keys of the game form the n-keys of A's plaintexts. Adversary C starts by running S to get $\mathbf{m}_0, \mathbf{m}_1, \mathbf{d}$. Then it picks a random bit b and use $\mathbf{m}_b$ in the rest of the simulation. When A calls Put(i), adversary C prepares a ciphertext for $\mathbf{m}_b[i]$. Here, c_2 is a random string, and c_1, supposed to be the output of $\mathsf{E}_{\mathsf{IVT}}$, is formed by C making queries to its Fn oracle. When A makes a (permitted) update query, then C forms the update plaintext m_2 and runs $\mathsf{diff}_{\mathsf{IVT}}$, with m, m_2, and $\mathtt{iv}_l$ which it can find from the server state it maintains using its FN oracle in lieu of the key. When C's FN oracle is implemented by E, it simulates A on G_4,

Consider G_5, where E is replaced with a different random function for each key. If A makes no UPD queries, only the c_1 components depend on b, But these are CTR mode encryptions with a random function. No queries to the random function are repeated, and hence the c_1 values can be picked as random strings, independent of b. When A does make update queries, deleting blocks and modifying blocks does not help towards finding b; inserting blocks could help, but in the IVT construction, in each ciphertext, a value $\mathtt{iv}_l$ is maintained, which keeps track of the last value of the counter used. When a new block is to be inserted, $\mathsf{patch}_{\mathsf{IVT}}$ increments IVT and uses a fresh counter value each time, meaning that these FN outputs can also be picked at random, independent of b and A cannot tell the difference. Overall, A learns no information about b in G_5, and hence, $\Pr[G_5^{\mathsf{A}}(1^\lambda)] = 1/2$. From a simple hybridization argument on $\mathrm{PRF}[n]$, we have $\Pr[G_4^{\mathsf{A}}(1^\lambda)] - \Pr[G_5^{\mathsf{A}}(1^\lambda)] = n(\lambda) \cdot \mathsf{Adv}^{\mathsf{PRF}}_{\mathsf{E},\mathsf{C}}(\lambda)$.

Consider games H_1, H_2 and H_3 described as follows. Game H_1 is the same as G_4, except that the winning condition in H_1 is setting bad. In game H_1, both the source S and the adversary A can set bad, by querying the random oracle at a $\mathbf{m}[i]$ or $\mathbf{k}[i]$ point. The probability it sets bad is bounded by $q_{\mathsf{S}}(\lambda)n(\lambda)/2^{\kappa(\lambda)}$. Game H_2 changes from H_1 only A query points are taken into account when testing for bad. In H_3, the ciphertexts c_1 are derived as encryptions of random strings. There exist adversaries C' and D such that

$$\Pr[H_3^{\mathsf{A}}(1^\lambda) \text{ sets bad}] - \Pr[H_2^{\mathsf{A}}(1^\lambda) \text{ sets bad}]| \leq n(\lambda)\mathsf{Adv}^{\mathsf{PRF}}_{\mathsf{E},\mathsf{C}}(\lambda) + n(\lambda)\mathsf{Adv}^{\mathsf{kr}}_{\mathsf{E},\mathsf{D}}(\lambda)\ .$$

Here, C' works like C, except that it keeps track of bad queries, and D checks if any of the RO queries are keys in its KR game. Finally, in H_3, the adversary learns nothing about the messages output by S, and we have

$$\Pr[H_3^{\mathsf{S},\mathsf{A}}(1^\lambda) \text{ sets bad}] \leq q_{\mathsf{S}}(\lambda)q(\lambda)2^{-\kappa(\lambda)} + q_{\mathsf{A}}(\lambda)q(\lambda)\mathbf{GP}_{\mathsf{S}}(\lambda)\ .$$

where $q(\lambda)_{\mathsf{A}} : \mathbb{N} \to \mathbb{N}$ is a bound on the number of RO queries made by A. Adding the equations so far, we have

$$\begin{aligned}\mathsf{Adv}^{\mathsf{priv}}_{\mathsf{IRCE2[E]},\mathsf{S},\mathsf{A}}(\lambda) \leq 2(n(\lambda)+1)\mathsf{Adv}^{\mathsf{PRF}}_{\mathsf{SE},\mathsf{C}_1}(\lambda) + 2n(\lambda)\mathsf{Adv}^{\mathsf{kr}}_{\mathsf{E},\mathsf{C}_2}(\lambda) + (q_{\mathsf{S}}(\lambda) + q_{\mathsf{A}}(\lambda))n(\lambda)2^{-\kappa(\lambda)} \\ + q_{\mathsf{A}}(\lambda)n(\lambda)\mathbf{GP}_{\mathsf{S}}(\lambda),\end{aligned}$$

where C_1 and C_2 are the ones among the PRF and KR adversaries with highest advantage. □

E Parameter-dependent security: Proofs and extensions

Proposition E.1. *If* FHE *has evaluation correctness, then* $\mathsf{FCHECK[FHE, MLEWC]}$ *scheme supports deduplication.*

Proof. We need to show that there exists a bound $\ell : \mathbb{N} \to \mathbb{N}$ such that for all server-side states $\sigma_S \in \{0,1\}^*$, for all valid client parameters (derived through Reg with fresh coins) σ_C, σ'_C, for all $m \in \{0,1\}^*$, the expected increase in size of σ''_S over σ'_S when $(f', \sigma'_S) \leftarrow\$ \mathsf{Run}(\mathsf{Put}, (\sigma_C, m), \sigma_S)$ and $(f', \sigma''_S) \leftarrow\$ \mathsf{Run}(\mathsf{Put}, (\sigma'_C, m), \sigma'_S)$ is bounded by $\ell(\lambda)$. In FCHECK, if a client with params σ_C runs Put with m, then, at the end, a parameter-ciphertext pair p, c (where $c = \mathsf{E}(1^\lambda, \mathsf{K}(1^\lambda, p, m), m)$) is stored on $\mathbf{fil}$. Now, when the second client with parameters σ'_C tries to put m, the search in $\mathsf{Put}[2,1]$ should find p, c. If the search happened over plaintexts, it is easy to see that the match will be detected, and as a result the client will only store an encryption of $\mathsf{K}(1^\lambda, p, m), m)$, which by assumption on MLEWC is independent of the size of m. But the search happens over FHE ciphertexts. We invoke evaluation correctness: as that the coins involved in $\mathsf{K_f}$ to generate the pk, sk in σ'_C, the coins in $\mathsf{E_f}$ to encrypt m, and the coins in $\mathsf{Ev_f}$ are all picked uniformly at random, except with negligible probability, the match is detected and the client decrypts to p, which leads the client to download p, c and hence stops the client from putting another ciphertext for m. □

E.1 Proof of Theorem 4.2

We now that for all PT A, for all unpredictable PT sources S, there exists a PT unpredictable source S' and adversaries B and C such that

$$\mathsf{Adv}^{\mathsf{ldpriv}}_{\mathsf{FCHECK[FHE,MLEWC],S,A}}(\lambda) \leq \mathsf{Adv}^{\mathsf{wpriv}}_{\mathsf{MLEWC,S',C}}(\lambda) + \mathsf{Adv}^{\mathsf{cpa}}_{\mathsf{FHE,B}}(\lambda) + 2m(\lambda)q(\lambda)\mathbf{GP}_{\mathsf{S}}(\lambda) \cdot \qquad (7)$$

where $q; \mathbb{N} \to \mathbb{N}$ is a bound on the total number of procedure queries made by A. Then, the theorem follows from the assumed CPA security of FHE, and from the WPRIV security of MLEWC. Consider games G_1 through G_4 of Figure 16. Without loss of generality, we assume that A does not repeat $\textsc{Put}$ queries. Here G_1 is the PDPRIV game with the code of S and $\mathsf{FCHECK[FHE, MLEWC]}$. We have

$$\Pr[G_1^{\mathsf{A}}(1^\lambda)] = \Pr[\mathrm{PDPRIV}^{\mathsf{S,A}}(1^\lambda)].$$

Game G_2, as in the proof of Theorem 4.1, performs the search not over ciphertexts, but instead over plaintexts. However, following the argument from the proof of Theorem 4.1, the correctness of FHE ensures that this does not affect the outcome of the game. We have $\Pr[G_1^{\mathsf{S,A}}(1^\lambda)] = \Pr[G_2^{\mathsf{S,A}}(1^\lambda)]$.

In game G_2, the c_2 components stored on the server correspond to the FHE encryptions of the MLEWC keys of the messages. These are replaced with random strings in G_3. Moreover, in $\mathsf{Put}[2, 1]$, when searching for a match for the put m, if a match is found, and the p, c pair for the match came as a result of a $\textsc{Msg}$ instance (i.e. not from $\textsc{Put}$), then the game sets bad. However, setting bad has not effect on the outcome of the game. Finally, in G_3, the algorithms in the Put protocol do not send the plaintext m in messages, but instead use the index of the plaintext in $\mathbf{m}_b$. This change does not affect the outcome of the game either, and only serves to simplify the code. There exists an adversary B such that

$$\Pr[G_2^{\mathsf{S,A}}(1^\lambda)] - \Pr[G_3^{\mathsf{S,A}}(1^\lambda)] \leq \frac{1}{2}\mathsf{Adv}^{\mathsf{cpa}}_{\mathsf{FHE,B}}(\lambda).$$

The description of B is straightforward, and we omit it here. Game G_4 is identical-until-bad to G_3. From the fundamental lemma of game-playing [15], we have

$$|\Pr[G_3^{\mathsf{S,A}}(1^\lambda)] - \Pr[G_4^{\mathsf{S,A}}(1^\lambda)]| \leq \Pr[G_4^{\mathsf{S,A}}(1^\lambda) \text{ sets bad}].$$

To set bad in G_4, adversary A must produce a p, c pair that is a valid encryption of some $\mathbf{m}_b[i]$, put it on the server with $\textsc{Msg}$, and subsequently run $\textsc{Put}(i)$ followed by $\textsc{Step}$ to make the search at $\mathsf{Put}[2, 1]$ find a match at p, c. However, A receives no information about the output of S, not even the ciphertexts. To see the ciphertexts, A must query $\textsc{State}$, but it can no longer query $\textsc{Step}$ after doing so. Thus, setting bad in G_4 can be bounded by $m(\lambda)q(\lambda)\mathbf{GP}_{\mathsf{S}}(\lambda)$.

Consider source S' and adversary C which work as follows. S' picks coins r at random and starts running G_4^{A} (except for picking the random bit b) with r as the only source of randomness up until the point when A makes a $\textsc{Ptxt}$ query with input d. At this point, S' runs $\mathsf{S}(1^\lambda, d)$ with fresh coins to get $\mathbf{m}_0, \mathbf{m}_1$ and exits with output $\mathbf{m}_0, \mathbf{m}_1, r$. Adversary C, when invoked with $\mathbf{p}, \mathbf{c}, r$ runs G_4^{A} with r, and when a $\textsc{Ptxt}(d)$ query is made, it lets $\mathbf{p}, \mathbf{c}$ play the role of the corresponding variables in G_4. When A finishes with output b', then C also exits with output b'. Together, S' and C simulate A in G_4, and hence it follows that $\Pr[\mathrm{WPRIV}^{\mathsf{S',C}}_{\mathsf{MLEWC}}(1^\lambda)] = \Pr[G_4^{\mathsf{S,A}}(1^\lambda)]$. Moreover, since S' runs S on fresh coins, which are not provided to C, it follows that S' is unpredictable if S is unpredictable. Adding up the above equations leads to Equation (7), completing the proof.

F MLEWC: Proofs and extensions

F.1 Proof of Theorem 4.3

Proof. Let S be an unpredictable auxiliary source and A be an adversary. Let $m : \mathbb{N} \to \mathbb{N}$ denote a bound on the length of message tuples output by S. Consider the constructions of PF source S' and B described in Figure 18. It can be checked that $\mathsf{Adv}^{\mathsf{wpriv}}_{\mathsf{HtO[HF},os],\mathsf{S,A}}(\lambda) \leq \mathsf{Adv}^{\mathsf{cdipfo}}_{\mathsf{OS,S',B}}(\lambda)$.

```
$\underline{\textsc{Main}(1^\lambda)\quad /\!/\ G_1^{\mathsf{A}}(1^\lambda) - G_4^{\mathsf{A}}(1^\lambda)}$
$b \xleftarrow{\$} \{0,1\};\ \sigma_S \xleftarrow{\$} \mathsf{Init}(1^\lambda)$
$b' \xleftarrow{\$} \mathsf{A}^{\textsc{Put},\textsc{Step},\textsc{Ptxt},\textsc{Msg},\textsc{Reg},\textsc{State}}(1^\lambda);$ Ret $(b = b')$

$\underline{\mathsf{Put}[1,2](m,\mathtt{M})}$
$p, i \leftarrow \mathsf{D_f}(1^\lambda, sk, c_r)$
If $p = 0^{\kappa(\lambda)}$ then
  $p \xleftarrow{\$} \mathsf{P}(1^\lambda);\ k \xleftarrow{\$} \mathsf{K}(1^\lambda, p, m);\ c_1 \leftarrow \mathsf{E}(1^\lambda, m)$
Else $k \xleftarrow{\$} \mathsf{K}(1^\lambda, p, m)$
$c_2 \leftarrow \mathbf{E}_{pk}(k)$ Ret $m, (c_1, c_2, p, u, i), 2, \mathsf{False}$

$\underline{\mathsf{Put}[2,2](\sigma_S,\mathtt{M})}$
$(n_c, \mathbf{U}, \mathbf{fil}, \mathbf{own}) \leftarrow \sigma_S;\ (u, p, c_1, c_2, i) \leftarrow \mathtt{M}$
If $c_1 \neq \epsilon$ then $n_f \leftarrow n_f + 1;\ i \leftarrow n_f;\ \mathbf{fil}[i] \leftarrow (p, c_1)$
$\mathbf{own}[u,i] \leftarrow c_2;\ \sigma_S \leftarrow (p, \mathbf{U}, \mathbf{fil}, \mathbf{own})$
$\mathtt{M} \leftarrow i;$ Ret $\sigma_S, \mathtt{M}, 1, \mathsf{True}$

$\underline{\textsc{Ptxt}(d)}\quad /\!/\ G_3^{\mathsf{A}}(1^\lambda),\ G_4^{\mathsf{A}}(1^\lambda)$
$\vec{m_0}, \vec{m_1} \xleftarrow{\$} \mathsf{S}(1^\lambda, d)$
For $i = 1$ to $|\mathbf{m}_b|$
  $\mathbf{p}[i] \xleftarrow{\$} \mathsf{P}(1^\lambda);\ \mathbf{k}[i] \leftarrow \mathsf{K}(1^\lambda, \mathbf{p}[i], \mathbf{m}_b[i])$
  $\mathbf{c}[i] \leftarrow \mathsf{E}(1^\lambda, \mathbf{k}[i], \mathbf{m}_b[i])$

$\underline{\mathsf{Put}[1,2](i,\mathtt{M})}\quad /\!/\ G_3^{\mathsf{A}}(1^\lambda),\ G_4^{\mathsf{A}}(1^\lambda)$
$p, i \leftarrow \mathsf{D_f}(1^\lambda, sk, c_r)$
If $p = 0^{\kappa(\lambda)}$ then $p \leftarrow \mathbf{p}[i];\ k \xleftarrow{\$} \mathbf{k}[i];\ c_1 \leftarrow \mathbf{c}[i]$
Else
  $k \leftarrow \mathsf{K}(1^\lambda, p, m)$
  $k' \xleftarrow{\$} \{0,1\}^{|k|};\ c_2 \leftarrow \mathbf{E}_{pk}(k')$
Ret $m, (c_1, c_2, p, u, i), \mathsf{False}$

$\underline{\mathsf{Put}[1,1](m,\mathtt{M})}\quad /\!/\ G_2^{\mathsf{A}}(1^\lambda)$
Ret $m, m, \mathsf{False}$

$\underline{\mathsf{Put}[1,1](m,\mathtt{M})}\quad /\!/\ G_1^{\mathsf{A}}(1^\lambda)$
$c_f \xleftarrow{\$} \mathbf{E}_{pk}(m);$ Ret $m, c_f, \mathsf{False}$

$\underline{\mathsf{Put}[2,1](\sigma_S,\mathtt{M})}\quad /\!/\ G_1^{\mathsf{A}}(1^\lambda)$
$(n_c, \mathbf{U}, \mathbf{fil}, \mathbf{own}) \leftarrow \sigma_S$
$c_r \xleftarrow{\$} \mathsf{E_f}(1^\lambda, pk, 0^{\kappa(\lambda)});\ c_i \xleftarrow{\$} \mathsf{E_f}(1^\lambda, pk, 0);\ c_n \leftarrow c_i$
For $(p, c) \in \mathbf{fil}$ do
  $c_p \xleftarrow{\$} \mathsf{E_f}(1^\lambda, pk, p);\ c_c \xleftarrow{\$} \mathsf{E_f}(1^\lambda, pk, c)$
  $c_r, c_n, c_i \xleftarrow{\$} \mathsf{Ev_f}(1^\lambda, pk, \mathsf{cmp}, c_f, c_p, c_c, c_r, c_n, c_i)$
$\sigma_S \leftarrow (n_c, \mathbf{U}, \mathbf{fil}, \mathbf{own});\ \mathtt{M} \leftarrow c_r, c_n;$ Ret $\sigma_S, \mathtt{M}, \mathsf{False}$

$\underline{\mathsf{Put}[2,1](\sigma_S,m)}\quad /\!/\ G_2^{\mathsf{A}}(1^\lambda)$
$(n_c, \mathbf{U}, \mathbf{fil}, \mathbf{own}) \leftarrow \sigma_S;\ r \leftarrow 0^{\kappa(\lambda)};\ i \leftarrow 0;\ n \leftarrow 0$
For $(k, c) \in \mathbf{fil}$ do $(r, n, i) \leftarrow \mathsf{Eval}(\mathsf{cmp}, m, p, c, r, n, i)$
$\mathtt{M} \leftarrow (\mathsf{E_f}(1^\lambda, pk, r), \mathsf{E_f}(1^\lambda, pk, n));$ Ret $\sigma_S, \mathtt{M}, \mathsf{False}$

$\underline{\textsc{Put}(i)}\quad /\!/\ G_3^{\mathsf{A}}(1^\lambda),\ G_4^{\mathsf{A}}(1^\lambda)$
$\mathsf{p} \leftarrow \mathsf{p} + 1;\ \mathbf{PS}[\mathsf{p}] = \mathsf{Put};\ \mathbf{a}[\mathsf{p}, 1] \leftarrow i$
$\mathtt{N}[\mathsf{p}] \leftarrow 1;\ \mathtt{M}[\mathsf{p}] \leftarrow \epsilon;$ Ret $\mathsf{p}$

$\underline{\mathsf{Put}[1,1](i,\mathtt{M})}\quad /\!/\ G_3^{\mathsf{A}}(1^\lambda),\ G_4^{\mathsf{A}}(1^\lambda)$
Ret $i, i, \mathsf{False}$

$\underline{\mathsf{Put}[2,1](\sigma_S,j)}\quad /\!/\ G_3^{\mathsf{A}}(1^\lambda),\ G_4^{\mathsf{A}}(1^\lambda)$
$m \leftarrow \mathbf{m}_b[j];\ (n_c, \mathbf{U}, \mathbf{fil}, \mathbf{own}) \leftarrow \sigma_S;\ r \leftarrow 0^{\kappa(\lambda)}$
$i \leftarrow 0;\ n \leftarrow 0$
For $(p, c) \in \mathbf{fil}$ do
  $k \leftarrow \mathsf{K}(1^\lambda, p, m);\ c' \leftarrow \mathsf{E}(1^\lambda, k, m);\ i \leftarrow i + 1$
  If $c = c'$ and $p \neq \mathbf{p}[j]$ then $\mathsf{bad} \leftarrow \mathsf{True}$
$\boxed{;\ i_f \leftarrow i;\ p_f \leftarrow p}$
$\mathtt{M} \leftarrow (\mathsf{E_f}(1^\lambda, pk, p_f), \mathsf{E_f}(1^\lambda, pk, i_f));$ Ret $\sigma_S, \mathtt{M}, \mathsf{False}$
```

Figure 16: Games G_1 through G_4 of Theorem 4.2. Procedures with code unchanged from PDPRIV are omitted.

```
$\underline{\textsc{Main}(1^\lambda)\quad /\!/\ \text{WPRIV}^{\mathsf{S},\mathsf{A}}(1^\lambda)}$
$(\mathbf{m}_0, \mathbf{m}_1, z) \xleftarrow{\$} \mathsf{S}(1^\lambda, \epsilon);\ b \xleftarrow{\$} \{0,1\}$
For $i \in [|\mathbf{m}_b|]$ do
  $\mathbf{p}[i] \xleftarrow{\$} \mathsf{P}(1^\lambda);\ \mathbf{k}[i] \xleftarrow{\$} \mathsf{K}(1^\lambda, \mathbf{p}[i], \mathbf{m}_b[i])$
  $\mathbf{c}[i] \xleftarrow{\$} \mathsf{E}(1^\lambda, \mathbf{k}[i], \mathbf{m}_b[i])$
$b' \xleftarrow{\$} \mathsf{A}_2(1^\lambda, \mathbf{p}, \mathbf{c}, z);$ Ret $(b = b')$

$\underline{\textsc{Main}(1^\lambda)\quad /\!/\ \text{CDIPFO}_{\mathsf{OS}}^{\mathsf{S},\mathsf{A}}(1^\lambda)}$
$(\mathbf{p}, z) \xleftarrow{\$} \mathsf{S}(1^\lambda);\ b \xleftarrow{\$} \{0,1\}$
For $i \in [|\mathbf{p}|]$ do
  If $b = 1$ then $(\alpha, \beta) \leftarrow \mathbf{p}[i];\ \mathsf{F}[i] \xleftarrow{\$} \mathsf{Obf}(1^\lambda, (\alpha, \beta))$
  Else
    $(\alpha', \beta') \leftarrow \mathbf{p}[i]\ ;\ \alpha \xleftarrow{\$} \{0,1\}^{|\alpha'|};\ \beta \xleftarrow{\$} \{0,1\}^{|\beta'|}$
    $\mathsf{F}[i] \xleftarrow{\$} \mathsf{Obf}(1^\lambda, (\alpha, \beta))$
$b' \xleftarrow{\$} \mathsf{A}(1^\lambda, \mathsf{F}[i], z);$ Ret $(b = b')$
```

Figure 17: The WPRIV game on the left, and the and CDIPFO game on the right.

It remains to show that S' is unpredictable. Consider source S_2 works by running S to get $(\mathbf{m}_0, \mathbf{m}_1, z)$, then picks keys $\mathbf{k}_{\mathsf{H}}[i] \xleftarrow{\$} \mathsf{K_h}(1^\lambda)$ and a bit b, and for $i \in [|\mathbf{m}_b|]$, computes $\mathbf{k}_{\mathsf{H}}[i] \xleftarrow{\$} \mathsf{K_h}(1^\lambda)$

```
$\mathsf{S}'(1^\lambda)$
$b \leftarrow\!\!\$\ \{0,1\}$; $(\mathbf{m}_0, \mathbf{m}_1, z) \leftarrow\!\!\$\ \mathsf{S}(1^\lambda)$
For $i \in [|\mathbf{m}_b|]$ do
  $\mathbf{k}_{\mathsf{H}}[i] \leftarrow\!\!\$\ \mathsf{K}_{\mathsf{h}}(1^\lambda)$; $\mathbf{k}[i] \leftarrow \mathsf{H}(1^\lambda, \mathbf{k}_{\mathsf{H}}[i], \mathbf{m}_b[i])$
  For $j \in [|\mathbf{m}_b[i]|]$ do
    $\mathbf{m}'[i\ell(\lambda) + j] \leftarrow \mathbf{k}[i] \| \langle \ell, i \rangle \| \mathbf{m}_b[i, j]$
Ret $\mathbf{m}', (b, \mathbf{k}_{\mathsf{H}}, z)$
```

```
$\mathsf{B}(1^\lambda, \mathsf{F}, (b, \mathbf{k}_{\mathsf{H}}, z))$
For $i \in [m(\lambda)]$ do
  $\mathbf{c}[i] \leftarrow (\mathsf{F}[(i-1)\ell(\lambda) + 1], \ldots, \mathsf{F}[i\ell(\lambda)])$
$b' \leftarrow\!\!\$\ \mathsf{A}(1^\lambda, \mathbf{k}_{\mathsf{H}}, \mathbf{c}, z)$
If $b' = b$ return 1 else return 0
```

Figure 18: Source S' and adversary B of Theorem 4.3.

```
$\textsc{Main}(1^\lambda)$   // $\textsc{mUCE}^{\mathsf{S},\mathsf{D}}_{\mathsf{HF}}(1^\lambda)$
$(1^n, t) \leftarrow\!\!\$\ \mathsf{S}(1^\lambda, \epsilon)$; For $i = 1$ to $n$ do $\mathbf{k}_{\mathsf{H}}[i] \leftarrow\!\!\$\ \mathsf{K}(1^\lambda)(1^\lambda)$
$b \leftarrow\!\!\$\ \{0,1\}$; $L \leftarrow\!\!\$\ \mathsf{S}^{\textsc{Hash}}(1^n, t)$; $b' \leftarrow\!\!\$\ \mathsf{D}(1^\lambda, \mathbf{k}_{\mathsf{H}}, L)$
Return $(b' = b)$

$\textsc{Hash}(x, 1^\ell, i)$
If $T[x, \ell, i] = \perp$ then
  If $b = 1$ then $T[x, \ell, i] \leftarrow \mathsf{H}(1^\lambda, \mathbf{k}_{\mathsf{H}}[i], x, 1^\ell)$
Else $T[x, \ell, i] \leftarrow\!\!\$\ \{0,1\}^\ell$
Return $T[x, \ell, i]$
```

```
$\textsc{Main}(1^\lambda)$   // $\textsc{mPRED}^{\mathsf{P}}_{\mathsf{S}}(1^\lambda)$
$(1^n, t) \leftarrow\!\!\$\ \mathsf{S}(1^\lambda, \epsilon)$; $\mathsf{done} \leftarrow \mathsf{False}$
$\mathbf{P} \leftarrow \emptyset$; $L \leftarrow\!\!\$\ \mathsf{S}^{\textsc{Hash}}(1^n, t)$
$\mathsf{done} \leftarrow \mathsf{True}$; $\mathbf{P}' \leftarrow\!\!\$\ \mathsf{P}^{\textsc{Hash}}(1^\lambda, 1^n, L)$
Return $(\mathbf{P} \cap \mathbf{P}' \neq \emptyset)$

$\textsc{Hash}(x, 1^\ell, i)$
If $\mathsf{done} = \mathsf{False}$ then $\mathbf{P} \leftarrow \mathbf{P} \cup \{x\}$
If $T[x, \ell, i] = \perp$ then $T[x, \ell, i] \leftarrow\!\!\$\ \{0,1\}^\ell$
Return $T[x, \ell, i]$
```

Figure 19: The mUCE and mPRED games.

and $\mathbf{k}[i] \leftarrow \mathsf{H}(1^\lambda, \mathbf{k}_{\mathsf{H}}[i], \mathbf{m}_b[i])$. Then it outputs $\mathbf{k}, (\mathbf{k}_{\mathsf{H}}, b, z)$. Clearly, unpredictability of S' follows from that of S_2. Further, consider $\mathsf{S}_2[i]$ which operates as above, but only outputs $\mathbf{k}[i], (\mathbf{k}_{\mathsf{H}}[i], b, z)$. Once again, if $\mathsf{S}_2[i]$ is unpredictable for all $i \in [m(\lambda)]$, then S_2 is also unpredictable. Note that $\mathsf{S}_2[i]$ runs S, an unpredictable source, picks one of its outputs $\mathbf{m}_b[i]$, generates $k_{\mathsf{H}} \leftarrow\!\!\$\ \mathsf{K}_{\mathsf{h}}(1^\lambda)$ and outputs k_{H} and the hash $\mathsf{H}(1^\lambda, k_{\mathsf{H}}, \mathbf{m}_b[i])$. Knowing that CR hash functions are randomness condensers [29], it is easy to show that $\mathsf{S}_2[i]$ is unpredictable for $i \in [m(\lambda)]$. We omit the details.

□

Remarks on HtO. The HtO construction can be made more efficient by using a CDIPFO where the special output can depend on the special input, so that the plaintext can be obfuscated in one shot, instead of bit-by-bit. However, such obfuscators in the standard model come only from UCEs. The HtO construction can be modified so that ciphertext length is only additive, by changing E to encrypt m under an SE scheme with a fresh random key, and encrypting the key as above. Now OS should be secure against CDIPFO-secure against computationally unpredictable sources and this can be achieved from the construction in [16], by extending the t-Strong Vector Decision Diffie Hellman assumption to computationally unpredictable distributions.

F.2 PRV\$-CDA secure MLE from UCE

A family of functions $\mathsf{HF} = (\mathsf{K}_{\mathsf{h}}, \mathsf{H})$ is a pair of deterministic algorithms. Key generation $\mathsf{K}(1^\lambda)$ returns a key $k \in \{0,1\}^{\kappa(\lambda)}$ on input 1^λ, and evaluation H takes 1^λ, a key k, an input $m \in \{0,1\}^*$, and a unary encoding 1^ℓ of an output length to return an output $\mathsf{H}(1^\lambda, k, x, 1^\ell) \in \{0,1\}^\ell$.

We now recall the definition of statistical multi-key security UCE for hash functions. Consider the game mUCE of Figure 19. A source is an algorithm which begins by keys indicating n the number of instances, along with state t. The game creates n independent keys. The source gets a procedure Hash which is either implemented by HF with n-independent keys, or via n random functions depending on the bit b chosen in the game. Then S returns with output leakage $L \in \{0,1\}^*$

and the distinguisher D on input L and all keys should guess b to win. We associate advantage $\mathsf{Adv}^{\mathsf{m\text{-}uce}}_{\mathsf{HF},\mathsf{S},\mathsf{D}}(\lambda) = 2\Pr[\mathrm{MUCE}^{\mathsf{S},\mathsf{D}}_{\mathsf{HF}}(\lambda)] - 1$.

A statistical predictor P is an algorithm (not necessarily PT) if there exist polynomials q, s such that for all $\lambda \in \mathbb{N}$, in the mPRED game predictor P makes at most $q(\lambda)$ oracle queries and outputs a set $\mathbf{P}'$ of size at most $s(\lambda)$. A source S is multi-key statistically unpredictable if for all statistical predictors P, it holds that $\mathsf{Adv}^{\mathsf{pred}}_{\mathsf{S},\mathsf{P}}(\lambda) = \Pr[\mathrm{MPRED}^{\mathsf{P}}_{\mathsf{S}}(\lambda)]$ is negligible. We say that HF is statistical multi-key UCE secure ($\mathsf{UCE}[\mathcal{S}^{\mathrm{sup\text{-}m}}]$-secure), if it holds that for all $\mathsf{Adv}^{\mathsf{m\text{-}uce}}_{\mathsf{HF},\mathsf{S},\mathsf{D}}(\lambda)$ is negligible for all PT statistical unpredictable S, for all PT D.

Consider the MLE scheme $\mathsf{CE}[\mathsf{HF}] = (\mathsf{P}, \mathsf{K}, \mathsf{E}, \mathsf{D}, \mathsf{T})$ described below, where $\mathsf{T}(1^\lambda, p, c) = c$.

```
$\mathsf{P}(1^\lambda)$
$k_{\mathsf{H}} \leftarrow\!\!\$\ \mathsf{K_h}(1^\lambda)$
Return $k_{\mathsf{H}}$
```

```
$\mathsf{K}(1^\lambda, k_{\mathsf{H}}, m)$
$k \leftarrow \mathsf{H}(1^\lambda, k_{\mathsf{H}}, m, 1^\lambda)$
Return $k$
```

```
$\mathsf{E}(1^\lambda, k_{\mathsf{H}}, k, m)$
$c \leftarrow m \oplus \mathsf{H}(1^\lambda, k_{\mathsf{H}}, k, 1^{|m|})$
Return $c$
```

```
$\mathsf{D}(1^\lambda, k_{\mathsf{H}}, k, c)$
$m \leftarrow c \oplus \mathsf{H}(1^\lambda, k_{\mathsf{H}}, k, 1^{|c|})$
Return $m$
```

Correctness of the scheme is easy to check. The following theorem shows that $\mathsf{CE}[\mathsf{HF}]$ is WPRIV-secure if HF is statistical mUCE secure.

Theorem F.1. If HF is $\mathsf{UCE}[\mathcal{S}^{\mathrm{sup\text{-}m}}]$-secure, then $\mathsf{CE}[\mathsf{HF}]$ is WPRIV-secure.

Proof. Let A be a PT unpredictable WPRIV adversary with functions m, ℓ. Consider S, D described below.

```
$\mathsf{S}^{\mathrm{HASH}}(1^\lambda)$
$b \leftarrow\!\!\$\ \{0,1\}$; $\mathbf{m}_0, \mathbf{m}_1, z \leftarrow\!\!\$\ \mathsf{A}_1(1^\lambda)$
For $i = 1$ to $|\mathbf{m}_b|$ do
    $\mathbf{k}[i] \leftarrow \mathrm{HASH}(\mathbf{m}[i], 1^\lambda, i)$
    $\mathbf{c}[i] \leftarrow \mathbf{m}[i] \oplus \mathrm{HASH}(\mathbf{k}[i], 1^{|\mathbf{m}[i]|}, i)$
$L \leftarrow \mathbf{c}, b, z$; Return $L$
```

```
$\mathsf{D}(1^\lambda, \mathbf{k}_{\mathsf{H}}, L)$
$\mathbf{c}, b, z \leftarrow L$
$b' \leftarrow \mathsf{A}_2(1^\lambda, \mathbf{k}_{\mathsf{H}}, \mathbf{c}, z)$
If $b = b'$ then return 1 else return 0
```

It can be seen that $\mathsf{Adv}^{\mathsf{wpriv}}_{\mathsf{CE}[\mathsf{HF}],\mathsf{A}}(\lambda) = 2\mathsf{Adv}^{\mathsf{m\text{-}uce}}_{\mathsf{HF},\mathsf{S},\mathsf{D}}(\lambda)$. It remains to show that S is simple statistically unpredictable by Lemma 4.7 of [10]. If P is a simple predictor, it follows that $\mathsf{Adv}^{\mathsf{m\text{-}spred}}_{P,\mathsf{S}}(\lambda) \leq m \cdot (\mathbf{GP}_{\mathsf{A}}(\lambda) + 2^{-\lambda})$. Simple statistical unpredictability of S follows from unpredictability of A. □