DE GRUYTER OPEN

Proceedings on Privacy Enhancing Technologies ; 2018 (2):85–104

Phillip Rogaway and Yusi Zhang

Onion-AE: Foundations of Nested Encryption

Abstract: Nested symmetric encryption is a well-known technique for low-latency communication privacy. But just what problem does this technique aim to solve? In answer, we provide a provable-security treatment for *onion authenticated-encryption* (onion-AE). Extending the conventional notion for authenticated-encryption, we demand indistinguishability from random bits and time-of-exit authenticity verification. We show that the encryption technique presently used in Tor does not satisfy our definition of onion-AE security, but that a construction by Mathewson (2012), based on a strong, tweakable, wideblock PRP, does do the job. We go on to discuss three extensions of onion-AE, giving definitions to handle inbound flows, immediate detection of authenticity errors, and corrupt ORs.

Keywords: Anonymity, authenticated encryption, onion routing, privacy, oracle silencing, provable security, Tor

DOI 10.1515/popets-2018-0014
Received 2017-08-31; revised 2017-12-15; accepted 2017-12-16.

1 Introduction

This paper is about formalizing one of the basic problems that underlie onion-routing and Tor [12, 16, 17, 31]. We call it the *onion-encryption problem*. Solutions to this problem usually go like this. A user holds a key $k_0 = (k_1, \ldots, k_n)$ while each "onion router" OR_i among $\mathrm{OR}_1, \ldots, \mathrm{OR}_n$ holds a key k_i. The user encrypts a message m for the circuit $(\mathrm{OR}_1, \ldots, \mathrm{OR}_n)$ by iteratively encrypting it in $k_n, k_{n-1}, \ldots, k_1$. She provides the resulting ciphertext c_0 to the "entry node" OR_1. It decrypts this using k_1 and passes the result onto OR_2, which decrypts using k_2, and so on. At the end of the circuit, the "exit node" OR_n decrypts its input to obtain the original message m. That message is usually imagined to exit the onion-routing network at this point.

Despite an extensive history (for a glimpse, see the "Related work" portion at the end of this introduction), clean cryptographic foundations for this onion-encryption problem do not exist. Most fundamentally, there is no clear definition for the problem it aims to solve. This paper aims to fill this gap, providing a self-contained, game-based, provable-security treatment.

Phillip Rogaway: Department of Computer Science, University of California, Davis. E-mail: rogaway@cs.ucdavis.edu
Yusi Zhang: Department of Computer Science, University of California, Davis. E-mail: ysizhang@ucdavis.edu

Our contributions. We begin with syntax, formalizing an onion-encryption scheme as a tuple of algorithms $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$. The key-distribution algorithm $\mathcal{K}$ defines the joint distribution on who gets what keys, $(k_0, k_1, \ldots, k_n) \twoheadleftarrow \mathcal{K}(n)$. Key k_0 will be held by the user and key k_i, for $i \geq 1$, by OR_i. How the circuit $(\mathrm{OR}_1, \ldots, \mathrm{OR}_n)$ gets constructed and how keys get where they need to be is, for us, conveniently out of scope. The user will encrypt using $\mathcal{E}(k_0, \cdots)$ while each OR_i will decrypt using $\mathcal{D}(k_i, \cdots)$. Crucially, both algorithms are stateful. This will be necessary for achieving the stringent security requirements we seek. To satisfy efficiency expectations reflected by Tor, we will insist that ciphertexts be of fixed length, independent of the circuit length n. This effectively means that plaintexts too have some fixed length, message segmentation and padding again being out-of-scope of our treatment.

While multiple security definitions *could* be given for onion encryption, some of which we will briefly explore, we focus on a relatively simple notion, *onion-AE security*, that guarantees indistinguishability from random bits (sometimes called ind\$-security) and time-of-exit detection of authenticity errors (integrity, or unforgeability, of ciphertexts). Our definition is related to that of *authenticated encryption* (AE) [4, 5, 23, 28], but now we are in the stateful setting [3, 6, 24] and a chain of parties must cooperate to decrypt. As with all treatments of AE, the underlying attack model permits a chosen-ciphertext attack (CCA): the adversary can obtain the encryption of whatever it wants, and it can experiment with compliant ORs, using them as decryption oracles.

To keep our definition simple we employ a new technique, *oracle silencing*, advanced by the authors in a separate paper [29]. To make our treatment self-contained, the present paper will explain what we need of the oracle-silencing technique.

Our definitional syntax is rich enough to describe what goes on in Tor's relay protocol [30], but that protocol, we will show, does not satisfy our definition of onion-AE security. Its susceptibility to the tagging attack of Raymond [14, 27] already implies absence of onion-AE security. (As will be explained later, this claim says little or nothing about the actual security of Tor.)

Despite the above, we show that schemes secure in the sense of onion-AE are easy to construct if one starts from a strong enough tool: a wideblock blockcipher that's secure as a strong, tweakable PRP (pseudorandom permutation). The protocol we analyze, LBE (large-block encryption), is one of two suggested by Mathewson in a specification document [25] for improving Tor's relay protocol. We adapt his construction to our syntax and prove it secure. Our security notion thus provides a cryptographic foundation for showing Mathewson's construction to be sound.

Following our provable-security treatment of onion-AE, we outline three different extensions. The first extension, a straightforward one, is to deal with inbound communication along the circuit. This is needed for a user's communication target to be able to return messages to him/her. Our second extension is an alternative to onion-AE in which one aims to immediately detect forged ciphertexts (onions), rather than detecting them at the circuit's endpoint. If bogus ciphertexts should be quashed immediately after their insertion, significantly different mechanisms and definitions must be used. Our third extension acknowledges the possibility of adversarially controlled relays and defines security despite them. It provides the adversary the ability to corrupt and control ORs along a circuit without compromising the circuit's overall utility (beyond what is inherently lost).

LIMITATIONS. Our aim in this work is intentionally limited: to provide and explore a self-contained game-based security notion for the sort of nested encryption used in onion routing. Our work is primarily theoretical, with little to say about the security of Tor.

Our work leaves many directions unexplored. For example: (1) We effectively disallow the early exit ("leaky pipe") possibility permitted by Tor—that a message employing circuit $(\mathrm{OR}_1, \ldots, \mathrm{OR}_n)$ might exit prior to OR_n. (2) The protocol we analyze, LBE, ignores the encryption and authentication between consecutive ORs achieved in Tor by its use of the TLS record layer. (3) We omit the inclusion of associated data (AD) in our treatment of onion-AE, despite this having been important for the utility of AE. Extensions in all of these directions would be worthwhile. None of the them are explored in depth, which would require fully specified protocols and proofs.

ORACLE SILENCING. A conventional game-based definition for onion-AE turns out to be rather complex; look ahead to Fig. 11. Most of the complexity stems from having to specify exactly when a decryption query should be disallowed (or, alternatively, its answer suppressed, or the game declared a loss). While complicated game-based definitions do appear in the literature, our own view is that when game code gets as complex as what is seen in Fig. 11, it is too hard to understand what is happening, and too easy to make mistakes.

To address this problem, related and concurrent work by the same authors develops a concept we call *oracle silencing* [29] and applies it to settings simpler than those of this paper. Oracle silencing can be used to simplify some definitions, including those for onion-AE. Our general treatment of oracle silencing is still evolving.

In a nutshell, oracle silencing works as follows. First, a *utopian* pair of games is given—games that are simple and capture what you might *wish* you could achieve. But the games don't actually "work," as the adversary has an attack that *can* trivially distinguish between them. One therefore passes from the pair of utopian games to a pair of *silenced* games, which the adversary *cannot* trivially distinguish. In the silenced games, certain oracle queries are effectively ignored: the oracle responds with an indication that it refuses to answer. Whether or not silencing happens is determined *automatically*, using the protocol's own *correctness condition* (formally, the class $\mathfrak{C}$ of valid protocols). We use oracle silencing to define onion-AE, and various extensions of onion-AE, via simple utopian games.

RELATED WORK. Much work on onion routing focuses on anonymity and key distribution, often employing approaches rooted in the UC framework or information theory. Our work is quite different, focusing on the semantic security of the symmetric encryption itself.

The intellectual roots for onion routing trace back to Chaum's concept of a *mixnet* [10]. In a mixnet, keys are asymmetric instead of symmetric and the routers, called *mixes*, are expected to buffer some number of incoming ciphertexts before passing them on to the next mix in line. Mixnets are intended for high-latency private communication; onion routing, for the low-latency setting. Despite these differences, the customary solutions are similar, using nested encryption.

An artifact like Tor does far more than nested encryption, including router discovery, the distribution of symmetric keys along a circuit, and provisioning of private channels between nodes using TLS. Due to such complexity, most definitional and analytical work on onion routing either abstracts away some protocol details or focuses only on a single aspect, like the key-distribution phase. Feigenbaum et al. [13] give an analysis of Tor based on a black-box modelling in UC framework [7]. Backes et al. [1] provide a complete UC definition for the ideal functionality they want an onion-routing protocol to realize. There is a considerable body of further work that focus on aspects of onion routing other than nested-encryption [8, 9, 15, 21, 22].

We ourselves find the approach of modeling onion routing as a piece of ideal functionality and then invoking UC heavy. We prefer to work more bottom-up, giving a precise and self-contained (so UC-free) definition for a lightweight primitive. Our emphasis on the encryption-scheme "detail" is shared with Möller [26] and Danezis and Goldberg [11]. But both of these deal with mixnets, not onion routing, and neither tries to bring forward a new and general kind of cryptographic object.

Bellare, Kohno, and Namprempre [3] were the first to introduce a notion of *stateful* authenticated encryption. Building on this and other intervening work, Boyd et al. [6] provide a framework for multiple notions of stateful AE, all in the game-based tradition. It is this branch of work to which our own is most closely related, despite these works having nothing to do with onion routing. In fact, the strongest definition in the hierarchy of Boyd et al. (level 4: no forgeries, replays, reordering, or dropped messages) can be viewed as a special case of onion-AE where only one OR is present.

2 Syntax of Onion-AE

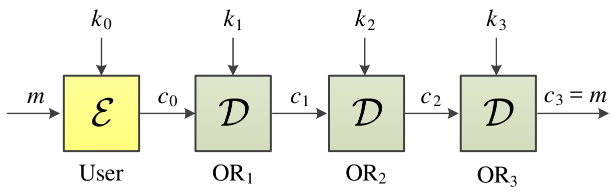
Conventions. We use lowercase letters to denote strings, integers and one-dimensional vectors (m, k, etc.), while uppercase letters to denote two-dimensional vectors, or matrices (S, D, etc.). We use uppercase letters of different fonts to refer to different objects ($\mathcal{K}, \mathcal{M}, \mathcal{C}$ for sets; and $\mathcal{E}, \mathcal{D}, \mathcal{A}$ for algorithms and adversaries) and sans-serif words for formal symbols and predicates (Enc, Dec, Fixed, etc.). For a one-dimensional vector s, we denote its length by $|s|$, its i-th element by s_i (first index 1). For a matrix S we use brackets to index its element like $S[i][j]$. We also use variables with fields named by keywords, like $x.\mathrm{type}$, the dot separating the variable name and the keyword. We use the word "tuple" as a synonym to "one-dimensional vector" and use parentheses-enclosed notation to explicitly denote a vector's elements, so $k = (k_1, k_2, \ldots, k_{|k|})$. For a string s, we denote by $s[i..j]$ the substring of s starting from index i and ending with index j (both inclusive). When both a and b are strings, we use $a \,||\, b$ to denote their concatenation. We write ε for the empty string. We use $a \leftarrow b$ to denote an assignment of b's value to a and $a \twoheadleftarrow B$ to denote a random assignment that assigns to a a value drawn from B (when B is a distribution) or uniformly sampled from B (when B is a set). When n is a positive integer we use $[n]$ to denote $\{1, 2, \ldots, n\}$. For an algorithm, by convention it always outputs $\Diamond$ whenever it takes an input outside its domain.

Syntax. An *onion-encryption* (OE) *scheme* is a tuple $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ where

- $\mathcal{K}\colon \mathbb{N} \to \mathcal{K}^*$ is a probabilistic algorithm that, given a *circuit size* $n \geq 1$, outputs a list of $n+1$ strings in the *key space* $\mathcal{K} \subseteq \{0,1\}^*$. We write $\mathcal{K}(n)$ for the distribution on vectors that $\mathcal{K}$ induces when its input is n.
- $\mathcal{E}\colon \mathcal{K} \times \mathcal{M} \times \mathcal{U} \to \mathcal{C} \times \mathcal{U}$ is a deterministic function that takes in a user key $k_0 \in \mathcal{K}$, a plaintext $m \in \mathcal{M}$ (the *message space*) and a user state $u \in \mathcal{U}$ (the *user state space*). It outputs a ciphertext (an *outermost onion*) $c \in \mathcal{C}$ (the *ciphertext space*) and an updated user state $u' \in \mathcal{U}$.
- $\mathcal{D}\colon \mathcal{K} \times \mathcal{C} \times \mathcal{S} \to (\mathcal{M} \cup \mathcal{C} \cup \{\Diamond\}) \times \mathcal{S}$ is a deterministic function. It takes in a key $k \in \mathcal{K}$, an input onion $c \in \mathcal{C}$, and an OR state $s \in \mathcal{S}$ (the *OR state space*). It returns a *decrypted result* d and an updated OR state $s' \in \mathcal{S}$.

We assume that for all $n \in \mathbb{N}$, $c \in \mathcal{C}$, and $s \in \mathcal{S}$, we have that $(k_0, k_1, \ldots, k_n) \twoheadleftarrow \mathcal{K}(n)$ implies $\mathcal{D}(k_i, c, s)$ is in $\mathcal{C} \times \mathcal{S}$ when $1 \leq i < n$, while in $(\mathcal{M} \cup \{\Diamond\}) \times \mathcal{S}$ when $i = n$. For simplicity, and to match efficiency expectations associated to Tor, and to avoid issues of length revelation, we assume that $\mathcal{M} = \{0,1\}^{l_1}$ and $\mathcal{C} = \{0,1\}^{l_2}$ with $1 \leq l_1 < l_2$.

Explanation. See Fig. 1 for naming conventions and an illustration of an OE scheme's usage. After a user has constructed a circuit of n ORs and generated keys, he adopts some method to distribute to each OR_i the key k_i. He keeps for himself the key k_0. The user applies $\mathcal{E}$ to encrypt his message into an outermost onion (ciphertext) c_0 and sends that to the first OR (entry guard). Each OR in the circuit, upon receiving an incoming onion, applies $\mathcal{D}$ to get a decrypted result. The result might be a plaintext, a ciphertext, or an indica-



procedure $\text{Correct}_\Pi(k, m)$
  $(k_0, k_1, \ldots, k_n) \leftarrow k; \quad (m_1, \ldots, m_\ell) \leftarrow m$
  $u, s_1, \ldots, s_n \leftarrow \varepsilon$
  for $i \leftarrow 1$ **to** ℓ **do**
    $(c_0^i, u) \leftarrow \mathcal{E}(k_0, m_i, u)$
    for $j \leftarrow 1$ **to** n **do** $(c_j^i, s_j) \leftarrow \mathcal{D}(k_j, c_{j-1}^i, s_j)$
  return $\bigwedge_{1 \le i \le \ell} (m_i = c_n^i)$

Fig. 1. Labeling conventions and basic correctness. Top: A user encrypts a message to get a ciphertext that, when iteratively decrypted, yields the original message. **Bottom:** Formalizing this correctness condition.

tion of invalidity. Authentication failures result in the last possibility.

For simplicity, we assume a fixed point of exit: the only position where a message is imagined to egress and be routed to its ultimate destination is the last OR (the exit relay). This is in contrast to what the Tor designers called a *leaky pipe* topology [12], which allows the user to choose the exit position on a per-message basis. The feature is little used in Tor [25, line 196–199] and would significantly complicate our treatment of onion-AE.

An implication of the fixed-exit-point assumption is that the exit node is the only point where authenticity checking is needed or expected. If an onion should get forged at an intermediate point in the circuit, we do not expect to detect the problem until the corresponding onion reaches the exit node. Such an approach is called *end-to-end* integrity checking [12] and is adopted by Tor's relay protocol. An alternative approach would be to have intermediate routers detect forgeries right away (see the threads following [32] for a discussion of this), which we call *eager* authentication. In fact, Tor's use of TLS channels between adjacent ORs should provide for eager authentication. Still, our initial treatment of onion-AE ignores this TLS-induced aspect of Tor, and eager authentication more broadly. In this way we more closely model Tor's relay protocol, which sits above TLS. See Section 8 for a formalization of eager authentication using oracle silencing. The treatment could serve as a starting point for modelling this TLS-based aspect of Tor.

Correctness. Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an OE scheme with message space $\mathcal{M}$ and ciphertext space $\mathcal{C}$. We say that Π is *correct* if, for any $n \ge 1$, $k \in \mathcal{K}(n)$, and $m \in \mathcal{M}^*$, the predicate $\text{Correct}_\Pi(k, m)$ defined in Fig. 1 returns true. Informally, the correctness condition just says that if values are relayed in the manner of a wire, the last OR always outputs what it should.

3 Oracle Silencing

Before we define onion-AE, we need to describe the idea of *oracle silencing*. This technique starts from the following intuitive idea: if an adversary *knows* the answer it will receive in response to some oracle query, that answer need not be returned—it can be suppressed, instead. If you automatically suppress oracle responses in this way, then giving definitions can be simplified.

We work in the game-based formulation of security notions. We treat a game G as a function that takes in the adversary's previous queries and the game's underlying randomness, and returns the response of the last query in the input. That is, given a sequence of queries $(x_1, x_2, \ldots, x_i) \in \{0,1\}^{**}$ and a string of coins $\Gamma \in \{0,1\}^\infty$, we let $G(x_1, x_2, \ldots, x_i; \Gamma) = y_i$ be the response to x_i. Here we assume some standard encoding of the queries. The game function is deterministic and defined by the game's code.

To define oracle silencing, we start from a pair of *utopian games* G and H. These correspond to the real and ideal worlds in a typical indistinguishability-based notion. Usually, the two games depend on some underlying cryptographic scheme $\Pi \in \mathfrak{C}$, where $\mathfrak{C}$ denotes the class of all *correct* schemes. We use G_Π to denote the specific instantiation of G with the underlying scheme Π. Given G and $\mathfrak{C}$, oracle silencing defines a binary predicate SILENCE on the set of *query histories* $(x_1, y_1, x_2, y_2, \ldots, x_i)$. See Fig. 2 for its formula.

The SILENCE predicate depends on a predicate Fixed that describes all query histories for which *all* schemes in $\mathfrak{C}$ will generate fixed responses, regardless of the randomness. In addition, SILENCE requires that the history must be generable by some scheme $\Pi \in \mathfrak{C}$ under some randomness, a condition we call the *nondegeneracy condition* (expressed by the predicate Nondegenerate).

More specifically, Fixed is defined as all schemes $\Pi \in \mathfrak{C}$ satisfying a Silence predicate, which in turn is defined, for a particular scheme, as all game randomness giving identical responses to the current query x_i when the game randomness can make the history pre-

$\mathsf{SILENCE}_{G,\mathfrak{C}}(x_1, y_1, \ldots, x_j) =$
$\quad \mathsf{Fixed}_{G,\mathfrak{C}}(x_1, y_1, \ldots, x_j) \bigwedge$
$\quad \mathsf{Nondegenerate}_{G,\mathfrak{C}}(x_1, y_1, \ldots, x_{j-1}, y_{j-1})$

$\mathsf{Fixed}_{G,\mathfrak{C}}(x_1, y_1, \ldots, x_j) =$
$\quad \forall \Pi \in \mathfrak{C}\ [\mathsf{Silence}_{G,\Pi,\mathfrak{C}}(x_1, y_1, \ldots, x_j)]$

$\mathsf{Nondegenerate}_{G,\mathfrak{C}}(x_1, y_1, \ldots, x_{j-1}, y_{j-1}) =$
$\quad \exists \Pi \in \mathfrak{C}\ \exists \Gamma \in \{0,1\}^\infty$
$\qquad [\mathsf{Gives}_{G,\Pi,\mathfrak{C}}(x_1, y_1, \ldots, x_{j-1}, y_{j-1}; \Gamma)]$

$\mathsf{Silence}_{G,\Pi,\mathfrak{C}}(x_1, y_1, \ldots, x_j) =$
$\quad \forall \Gamma_1, \Gamma_2 \in \{0,1\}^\infty$
$\qquad \big[\mathsf{Gives}_{G,\Pi,\mathfrak{C}}(x_1, y_1, \ldots, x_{j-1}, y_{j-1}; \Gamma_1) \wedge$
$\qquad \mathsf{Gives}_{G,\Pi,\mathfrak{C}}(x_1, y_1, \ldots, x_{j-1}, y_{j-1}; \Gamma_2)\big] \Rightarrow$
$\qquad G_\Pi(x_1, x_2, \ldots, x_j; \Gamma_1) = G_\Pi(x_1, x_2, \ldots, x_j; \Gamma_2)$

$\mathsf{Gives}_{G,\Pi,\mathfrak{C}}(x_1, y_1, \ldots, x_j, y_j; \Gamma) =$
$\quad \mathsf{Gives}_{G,\Pi,\mathfrak{C}}(x_1, y_1, \ldots, x_{j-1}, y_{j-1}; \Gamma) \bigwedge$
$\qquad \big[y_j = G_\Pi(x_1, x_2, \ldots, x_j; \Gamma)$
$\qquad \vee (y_j = \perp \wedge \mathsf{SILENCE}_{G,\mathfrak{C}}(x_1, y_1, \ldots, x_j))\big]$

$\mathsf{Gives}_{G,\Pi,\mathfrak{C}}(\varepsilon; \Gamma) = \mathsf{true}$

Fig. 2. Definition of SILENCE, used for defining oracle silencing. The definitions are recursive.

fix $(x_1, y_1, \ldots, x_{i-1}, y_{i-1})$ occur. The "make the history prefix occur" idea is again formalized as a Gives predicate, recursively calling SILENCE for those already silenced responses.

With SILENCE defined we can easily map a tuple $(G, H, \mathfrak{C})$ to a pair of *silenced games* $(\overline{G}, \overline{H})$, where $\overline{G}$ and $\overline{H}$ are identical to G and H except that before each query response y_i is returned, if $\mathsf{SILENCE}_{G,\mathfrak{C}}(x_1, y_1, \ldots, x_i)$ is true then y_i is replaced by the symbol $\perp$. Neither G nor H are themselves allowed to return that symbol. The definitional paradigm has one provide explicit utopian games G (the "real" utopian game) and H (the "ideal" utopian game), along with a class of schemes $\mathfrak{C}$. One then inherits the silenced games $\overline{G}$ and $\overline{H}$. Security (more precisely, insecurity) is measured by an adversary's ability to distinguish between the silenced games.

There are a variety of related but alternative ways to define games $(\overline{G}, \overline{H})$ from $(G, H, \mathfrak{C})$. We continue to investigate them, in search of the cleanest approach [29].

4 Defining Onion-AE

To define onion-AE we give the adversary access to two oracles, Enc and Dec, that model, respectively, the user's encryption and any chosen OR's decryption. We describe real and ideal worlds for instantiating these oracles. In the former, the Enc oracle takes in a message and outputs the result of running $\mathcal{E}$. The Dec oracle takes in an input onion and an OR index. It outputs the result obtained by running $\mathcal{D}$. In contrast, for the ideal world, both the Enc oracle and the Dec oracle output independent random strings in $\mathcal{C}$ *except* for the case of a Dec query pointing to the exit OR. For that case $\Diamond$ is returned. See Fig. 3.

The two games are utopian in that an adversary can easily distinguish them: it simply forwards an onion along the circuit and observes whether the last Dec query returns a message (real world) or $\Diamond$ (ideal world). For the rest of the paper, we call such a trivial kind of game interaction an *honest execution* and the oracle queries corresponding to it *honest queries*.

We apply oracle silencing to games OE1, OE0, and the class of correct OE schemes $\mathfrak{C}$. The resulting pair of silenced games is denoted $\overline{\text{OE1}}$ and $\overline{\text{OE0}}$. An adversary $\mathcal{A}$'s advantage in breaking an onion encryption scheme Π is defined as $\mathbf{Adv}^{\mathsf{oe}}_{\Pi}(\mathcal{A}) = \mathbf{Pr}[\mathcal{A}^{\overline{\text{OE1}}_\Pi} \to 1] - \mathbf{Pr}[\mathcal{A}^{\overline{\text{OE0}}_\Pi} \to 1]$. Informally, the scheme Π is onion-AE secure if for all $\mathcal{A}$ employing a reasonable amount of resource the advantage $\mathbf{Adv}^{\mathsf{oe}}_{\Pi}(\mathcal{A})$ is small.

Implications of onion-AE. Formalizing a notion of indistinguishability from random bits, onion-AE directly captures privacy. It also captures an "authenticity-at-exit" notion: whenever an adversary tries to deviate from an honest execution, the silencing does not occur and the last Dec oracle outputs $\Diamond$ with overwhelming probability.

Onion-AE also formalizes one form of anonymity. We give an informal argument for this (no other being possible in the absence of first providing a definition for the anonymity of an OE scheme). Assume first an inactive adversary: it does not modify any onions (ciphertexts). We demanded that all intermediate ciphertexts be indistinguishable from random bits (ind\$-security), whence the ciphertexts produced by any one party, no matter what *it* is encrypting, must be indistinguishable from the ciphertexts produced by any other party (Recall that our ciphertexts all have the same length.). So passive-adversary anonymity seems clear.

Game OE1 (Real)	Game OE0 (Ideal)
$\underline{\text{KEY}(n')}$	$\underline{\text{KEY}(n')}$
if $n \neq \perp$ **then return** Err	**if** $n \neq \perp$ **then return** Err
$n \leftarrow n'$	$n \leftarrow n'$
$(k_0, \ldots, k_n) \twoheadleftarrow \mathcal{K}(n)$	
$\underline{\text{ENC}(m)}$	$\underline{\text{ENC}(m)}$
if $n = \perp$ **then return** Err	**if** $n = \perp$ **then return** Err
$(c, u) \leftarrow \mathcal{E}(k_0, m, u)$	$c \twoheadleftarrow \mathcal{C}$
return c	**return** c
$\underline{\text{DEC}(c, i)}$	$\underline{\text{DEC}(c, i)}$
if $n = \perp$ **then return** Err	**if** $n = \perp$ **then return** Err
$(d, s_i) \leftarrow \mathcal{D}(k_i, c, s_i)$	**if** $i = n$ **then** $d \leftarrow \lozenge$
return d	**else** $d \twoheadleftarrow \mathcal{C}$
	return d

Fig. 3. Utopian games used for defining onion-AE. The real game OE1 executes real encryption and decryption while the ideal game OE0 samples fresh random strings for encryption and decryption, except for a decryption at the exit, in which case $\lozenge$ is returned.

Now suppose that the adversary goes in and mucks with some ciphertexts in an attempt to garner identity-related information. This will fail because, regardless of identities, the same thing happens when onions are mauled. Namely, the modified onion continues to traverse the onion-routing network towards the exit node, with each new intermediate onion being indistinguishable from random bits, and therefore uncorrelated to any player identities. Then, at the exit node, an authentication failure will occur: in our model, an indication of failure almost always happens (up to the advantage bound proven by the protocol). So the behavior is, yet again, independent of player identities.

In short, the combination of indistinguishability from random bits for all onions and unforgeability of ciphertexts, enforced at a single, determinate place, dooms all tagging attacks (see Section 5) and, more broadly, all other approaches that might expose identity information yet fall within the scope of the onion-encryption model. We emphasize, however, that the above arguments do not apply to attacks that leverage timing information, say traffic confirmation, because our model does not express anything about time.

5 Tor's Relay Protocol Does Not Achieve Onion-AE Security

We now recast the Tor relay protocol in terms of our syntax, and show that it does not achieve onion-AE security. In particular, we argue that the technique used in a tagging attack is enough to violate the onion-AE security notion. For more on tagging attacks, see [14] for a real-world experiment and [27] for a summary of related attacks.

MECHANISM. Tor attaches several header fields to a message before it applies AES counter mode multiple times to encrypt it. Decryption correspondingly peels off one layer of encryption from an input onion and bases its decision on the value of two fields: a 2-byte *recognized field* and a 4-byte *digest field*. At the time of a user's encryption, it sets the recognized field to all-zeros and the digest field to the first four bytes of a running digest of all the bytes that have been destined for this OR, seeded from the symmetric key between them [30]. At the time of an OR's decryption, the OR checks if the recognized field is all-zero. If it is, the OR regards that as a signal of having been chosen as the desired point of exit. It then checks whether the digest matches and treats the decrypted result as the plaintext message if it does. It reports an authentication failure otherwise. Conversely, the OR treats a non-zero recognized field as indicating that there are further layers of encryption in the onion,

$\underline{\mathcal{K}(n)}$

for $i \leftarrow 1$ **to** n **do**
  $k_i.\text{key} \twoheadleftarrow \mathcal{K}$
  $k_i.\text{exit} \leftarrow 0$
$k_n.\text{exit} \leftarrow 1$
$k_0 \leftarrow (k_1, \ldots, k_n)$
return $(k_0, \ldots, k_n)$

$\underline{\mathcal{E}(k, m, u)}$

$n \leftarrow |k|$
$c \leftarrow m \,\|\, 0^{l_2 - l_1}$
for $i \leftarrow n$ **downto** 1 **do**
  $c \leftarrow \mathbb{E}(k_i.\text{key}, u_i, c)$
  $u_i \leftarrow u_i \,\|\, c$
$u \leftarrow (u_1, u_2, \ldots, u_n)$
return (c, u)

$\underline{\mathcal{D}(k, c, s)}$

$c' \leftarrow \mathbb{D}(k.\text{key}, s, c)$
$s \leftarrow s \,\|\, c$
if $k.\text{exit} = 1$ **and**
  $c'[l_1 + 1..l_2] = 0^{l_2 - l_1}$ **then**
  return $(c'[1..l_1], s)$
if $k.\text{exit} = 1$ **then**
  return $(\Diamond, s)$
return (c', s)

Fig. 4. Definition of the scheme LBE**, adapted from Mathewson [25].** The construction depends on a parameter l_1 (the message length) and a tweakable wideblock blockcipher $\mathbb{E}$ operating on l_2-bit blocks (the onion length), where $l_1 < l_2$.

so it forwards the decrypted result as an onion to the OR's successor—assuming there is one.

For the sake of brevity, we have omitted several details that are irrelevant to our current purpose. For example, the seed of the hash digest is not the key itself but something derived from it.

As just explained, an OR might regard itself as the final hop for some cell despite not being the circuit's last node [12]. This supports Tor's leaky pipe design, but falls out-of-scope of our own formalization. In the current section we ignore this possibility, as Tor's relay mechanism does not achieve onion-AE security even if we insist that packets exit at the final OR.

Tagging attacks break onion-AE. Let us translate what happens with a tagging attack into our framework, showing that it violates onion-AE security for the Tor relay protocol. Specifically, consider an onion-AE adversary performing the following queries:

1. Key(3). The adversary chooses a circuit size of 3.
2. Enc(m) $\to c_1$. The adversary asks the user to encrypt an arbitrary message m. He sees c_1 as the returned string.
3. Dec($c_1, 1$) $\to c_2$. An honest execution of OR's decryption. The adversary sees c_2 as the decrypted result.
4. Dec($c_2 \oplus \mathit{tag}, 2$) $\to c_3$. The adversary xors in a self-composed string $\mathit{tag} \neq 0$ before handing c_2 to OR_2.
5. Dec($c_3 \oplus \mathit{tag}, 3$) $\to m'$. The adversary xors in tag to c_3 before forwarding it to OR_3, and checks whether the returned result m' equals to m. If yes, the adversary outputs 1 otherwise 0.

As the last query is not silenced (which is true when $\mathit{tag} \neq 0$), the adversary sees m in the real world and $\Diamond$ in the ideal world, so the attack always succeeds. We conclude that the Tor relay protocol does not satisfy onion-AE security.

Significance for Tor. The attack above does not demonstrate that the Tor relay protocol is broken in any meaningful sense. To begin with, it has been much debated if tagging attacks ever matter, and if they accomplish more than timing attacks do. (We don't know, but see reference [32] for an argument that they do.) Beyond this, however, Tor communication between successive ORs is provisioned using TLS, whose record layer provides AE. A complete description of how cells get encrypted in Tor would have to fold in this extra encryption, and the extra keys associated to it (between each OR_i and OR_{i+1}). Once this is accounted for, the attack above fails to work, and it is possible that the enlarged protocol actually *is* secure, although under a definition (see Section 8) that demands early detection of authenticity errors. The most one can conclude from what we have shown is that the standard way of drawing abstraction boundaries for Tor leaves one with a mechanism whose cryptographic security is both weak (since the notion of onion-AE is not all that strong) and unformalized.

Given that tagging attack might not be a severe practical issue, one could fairly argue that the onion-AE notion is not that meaningful. To some extent, we agree, yet we would like to defend the soundness of onion-AE as follows: it serves as a simple way to instantiate our general definitional idea, that users and ORs are treated as stateful entities and that onion-AE in such a model essentially becomes a generalization of stateful AE. This idea, when instantiated differently, can bring upon different extensions of onion-AE that deal with inbound directions, eager authentication, and so on. See Section 7, 8 and 9 for details.

6 Achieving Onion-AE

In this section we analyze the OE scheme LBE proposed by Mathewson [25] ("Design 1: Large-block encryption") and prove its onion-AE security. LBE's high-level idea is to employ a tweakable wideblock blockcipher $\mathbb{E}\colon \mathcal{K}\times\{0,1\}^*\times\mathcal{C}\to\mathcal{C}$ and use the tweak to encode the ciphertext history so far. In this way, any mauling of onions will pollute the tweak and result in garbage for the result, which will almost certainly trigger an authentication failure at the exit-OR (assuming the difference l_2-l_1 is large). When we say that $\mathbb{E}\colon \mathcal{K}\times\{0,1\}^*\times\mathcal{C}\to\mathcal{C}$ is a tweakable blockcipher (TBC) we mean that $\mathbb{E}(k,t,\cdot)$ is a permutation on $\mathcal{C}$ for all k and t in its domain. Let $\mathbb{D}=\mathbb{E}^{-1}$ denote the inverse of $\mathbb{E}$, meaning that $\mathbb{D}(k,t,y)$ is the unique x such that $\mathbb{E}(k,t,x)=y$.

Algorithm LBE$[\mathbb{E},l_1]$ is described in Fig. 4. The user's state is a vector of strings, one for each OR. The innermost layer appends an all-zero string, the *redundancy*, to the input message. The outer layers apply the blockcipher directly to the input onion. Decryption applies $\mathbb{D}$ and returns a decrypted result based on the key's *exit-node flag*. This flag, dropped into an OR's key by the key-generation algorithm, tells the OR if it is the exit node. When the flag is one, it is, and the decryption algorithm performs the needed integrity check; otherwise, the check is skipped.

The original construction proposed by Mathewson does not have the exit-node flag, and for any OR, as long as the redundancy field is all zero, the OR will process the message as a recipient. This is for compatibility with the existing leaky-pipe design. For our definitions and algorithm, we need the flag for satisfying correctness.

We now define security for a TBC in the sense of a strong, tweakable PRP. The "strong" refers to the adversary having both forward and backward access to the primitive. Let $\mathrm{CCA}_{\mathbb{E}}$ denote the following game for a TBC $\mathbb{E}\colon \mathcal{K}\times\{0,1\}^*\times\mathcal{C}\to\mathcal{C}$. The game runs as follows. First a bit $b \twoheadleftarrow \{0,1\}$ is flipped, a key $k\twoheadleftarrow\mathcal{K}$ is selected, and a function $\pi\colon \mathcal{K}\times\{0,1\}^*\times\mathcal{C}\to\mathcal{C}$ is chosen uniformly at random such that $\pi(k,t,\cdot)$ is a permutation for all k and t. An adversary is provided oracles Encipher and Decipher that behave as follows. If $b=1$ then Encipher realizes $\mathbb{E}(k,\cdot,\cdot)$ and Decipher realizes its inverse. If $b=0$ then Encipher realizes $\pi(k,\cdot,\cdot)$ and Decipher realizes its inverse. When the adversary is done it outputs a bit b' and *wins*—the game outputs 1—if $b'=b$. We define the strong, tweakable PRP-advantage of an adversary $\mathcal{A}$ attacking $\mathbb{E}$ as $\mathbf{Adv}^{\mathsf{sprp}}_{\mathbb{E}}(\mathcal{A}) := 2\,\mathbf{Pr}[\mathrm{CCA}^{\mathcal{A}}_{\mathbb{E}}\to 1]-1$.

The following theorem states that LBE$[\mathbb{E},l_1]$ is onion-AE secure if $\mathbb{E}$ is secure in the sense just defined and l_2-l_1 is substantial (like 64–128 bits). The proof is made more complex because a fundamental component of the security definition (the SILENCE predicate) does not have a computational specification. To get around this we transform the abstractly defined games, $\overline{\mathrm{OE1}}$ and $\overline{\mathrm{OE0}}$, into equivalent but more complicated concretely defined games, cOE1 and cOE0 (look ahead to Fig. 11). See Appendix B for the proof.

Theorem 6.1. *Fix $1\le l_1<l_2$ and $\mathcal{M}=\{0,1\}^{l_1}$ and $\mathcal{C}=\{0,1\}^{l_2}$. Fix a TBC $\mathbb{E}\colon\mathcal{K}\times\{0,1\}^*\times\mathcal{C}\to\mathcal{C}$. Let $\Pi=\mathrm{LBE}[\mathbb{E},l_1]$ and let $\mathcal{A}$ be an adversary attacking Π. Suppose its running time is t and it asks at most q_E encryption queries and q_D decryption queries. Suppose $\mathcal{A}$ calls* Key *with input n. Then the proof of this theorem specifies, in a black-box manner, an adversary $\mathcal{B}$ that attacks $\mathbb{E}$ and whose advantage satisfies*

$$\mathbf{Adv}^{\mathsf{sprp}}_{\mathbb{E}}(\mathcal{B}) \;\ge\; (1/n)\,\mathbf{Adv}^{\mathsf{oe}}_{\Pi}(\mathcal{A})-\frac{q_E}{2^{l_2-1}}-\frac{4\,q_D}{n\,2^{l_2-l_1}}\,.$$

Adversary $\mathcal{B}$ runs in time $t+c(nq_E+q_D)$ for some absolute constant c, and it asks at most q_E encryption queries and q_D decryption queries.

Instantiation. Mathewson already gave some guidance on what a practical instantiation of LBE would look like. In particular, there is no need for ORs to grow their state with each onion received, nor for the user to grow his state with each ciphertext sent. Instead, the state u_i can be of constant length, like 20 bytes. To accomplish this, one must select a good realization of the TBC $\mathbb{E}$.

In Tor, the ciphertext length (in Tor's terminology, the cell payload length) is 509 bytes. There are no widely-deployed TBCs for that length. Still, constructions are out there, including EME2 (also called EME*) [18, 19], a wideblock TBC based on AES, and AEZ [20], an arbitrary-input-length TBC based on the AES round-function. Both of these constructions are incremental [2] for the tweak in the following sense: given $\mathbb{E}(k,t,x)$ and a fixed-length string s saved during its computation, one can compute $\mathbb{E}(k,t\,\|\,t',x')$ in time linear to $|t'|+|x'|$.

While one would expect almost any well-designed TBC to have the property just named, in fact, one can always engineer what is needed into a TBC with the help of a collision-resistant hash function. Namely, given a collision resistant hash-function $H\colon\{0,1\}^*\to\{0,1\}^t$ and a TBC $\mathbb{E}:\mathcal{K}\times\{0,1\}^t\times\mathcal{C}\to\mathcal{C}$, define the TBC $\mathbb{E}^*:\mathcal{K}\times(\{0,1\}^t)^+\times\mathcal{C}\to\mathcal{C}$ by as-

procedure $\mathrm{Correct}'_\Pi(k', m')$
$(k'_0, k'_1, \ldots, k'_n) \leftarrow k'; \quad (m'_1, \ldots, m'_\ell) \leftarrow m'$
$u', s'_1, \ldots, s'_n \leftarrow \varepsilon$
for $i \leftarrow 1$ **to** ℓ **do**
  $c'_i \leftarrow m'_i$
  for $j \leftarrow n$ **to** 1 **do** $(c'_i, s'_j) \leftarrow \mathcal{E}'(k'_j, c'_i, s'_j)$
  $(d'_i, u') \leftarrow \mathcal{D}'(k'_0, c'_i, u')$
return $\bigwedge_{1 \le i \le \ell} (m'_i = d'_i)$

Fig. 5. Inbound correctness condition.

serting that $\mathbb{E}^*(k, c_1 \cdots c_m, x) = \mathbb{E}(k, t, x)$ where $t = H(\cdots H(H(H(c_1) \,||\, c_2) \,||\, c_3) \cdots \,||\, c_m)$.

Translating into English, every time a ciphertext comes in, the OR hashes the concatenation of its current state with the incoming onion. That is the tweak which is used for the TBC, as well as the OR's updated state. It is straightforward to prove that the collision resistance of H and the strong, tweakable PRP security of $\mathbb{E}$ suffice for the strong, tweakable PRP security of $\mathbb{E}^*$ constructed in this way. In fact, this is exactly the approach suggested by Mathewson in his proposal.

7 Extension 1: Inbound Flows

So far our modeling only covers the outbound direction of onion routing. We describe how to extend our syntax and games to model the inbound direction as well. We do not prove the security of (an appropriately modified variant of) LBE with respect to the resulting notion, but doing so would appear to be routine.

SYNTAX. We begin by adding two new algorithms $\mathcal{E}'$ and $\mathcal{D}'$ into our syntax, defining an *extended* OE scheme Π as a five-tuple $(\mathcal{K}, \mathcal{E}, \mathcal{D}, \mathcal{E}', \mathcal{D}')$. The new algorithms are deterministic and have the following syntax:

– $\mathcal{E}'\colon \mathcal{K} \times (\mathcal{M} \cup \mathcal{C}) \times \mathcal{S} \to \mathcal{C} \times \mathcal{S}$
– $\mathcal{D}'\colon \mathcal{K} \times \mathcal{C} \times \mathcal{U} \to (\mathcal{M} \cup \{\Diamond\}) \times \mathcal{U}$

In brief, the pair $(\mathcal{E}', \mathcal{D}')$ is the counterpart of $(\mathcal{E}, \mathcal{D})$ for inbound flows: each OR_i receives either a message m from the outside world (the exit OR) or an *input onion* c from OR_{i+1} (the intermediate OR). It uses an *inbound key* k'_i and an *inbound state* s'_i to compute an *output onion* $c' \leftarrow \mathcal{E}'(k'_i, m, s'_i)$ (or $c' \leftarrow \mathcal{E}'(k, c, s'_i)$ for intermediate ORs). It delivers c' to OR_{i-1} (OR_0 is treated as the user here). The user, after receiving an inbound onion c from OR_1, uses his own inbound key k'_0 and inbound state u' to compute a decrypted result $m \leftarrow \mathcal{D}'(k'_0, c, u')$, which is either a message in $\mathcal{M}$ or an authentication failure symbol $\Diamond$.

We have assumed that for both the user and the ORs there is an inbound key, OR state, and user state. The initialization of these is the same as their outbound counterparts: the states are set to ε and the keys are sampled according to $\mathcal{K}$. Thus the outbound and inbound directions are completely separated. Although such a definitional choice is not mandatory, it allows an easier formalization and reflects real-world applications.

CORRECTNESS. We formulate the inbound correctness experiment $\mathrm{Correct}'_\Pi$ in a way similar to the existing outbound one. See Fig. 5 for details. We say an extended OE scheme Π is *correct* if for all $n \in \mathbb{N}$, all $k \in \mathcal{K}(n)$, and all $m \in \mathcal{M}^*$, both $\mathrm{Correct}_\Pi(k, m)$ in Fig. 1 and $\mathrm{Correct}'_\Pi(k, m)$ in Fig. 5 return true. That is, the scheme is correct only if it correctly decrypts onions in both directions.

INBOUND SECURITY GAMES. One can extend the prior security games to cover the inbound direction. Instead of modifying the code of Fig. 3 we choose to write a new pair of *inbound games* iOE1 and iOE0 from scratch, and define an extended OE scheme Π to be secure if both the pair $(\overline{\mathrm{OE1}_\Pi}, \overline{\mathrm{OE0}_\Pi})$ and the pair $(\overline{\mathrm{iOE1}_\Pi}, \overline{\mathrm{iOE0}_\Pi})$ are indistinguishable, both pairs of games silenced using the class of schemes $\mathfrak{C}'$ corresponding to the new correctness condition. For any adversary with reasonable resources we want both $\mathbf{Adv}^{\mathsf{oe}}_\Pi(\mathcal{A}) = \mathbf{Pr}[\mathcal{A}^{\overline{\mathrm{OE1}_\Pi}} \to 1] - \mathbf{Pr}[\mathcal{A}^{\overline{\mathrm{OE0}_\Pi}} \to 1]$ and $\mathbf{Adv}^{\mathsf{r\text{-}oe}}_\Pi(\mathcal{A}) = \mathbf{Pr}[\mathcal{A}^{\overline{\mathrm{iOE1}_\Pi}} \to 1] - \mathbf{Pr}[\mathcal{A}^{\overline{\mathrm{iOE0}_\Pi}} \to 1]$ to be small. See Fig. 6 for the code of iOE1 and iOE0.

The separation of security into two notions needs justification, since one must ensure security against attacks where messages are routed in either directions and in an arbitrary, interleaved fashion. Let us informally explain why this is not a concern. Since we provide a definition where the two flow directions use separate variables, both keys and states, even if one constructs a composite pair of games where both directions are modeled altogether, the indistinguishability of them will end up equivalent to the indistinguishability of the two pairs of games. A standard hybrid argument would establish that. The separate keys and states allow one to analyze the security notions one at a time without a need to worry about their interaction.

Game iOE1 (Real)

$\underline{\textsc{Key}(n')}$

if $n \neq \perp$ **then return** Err
$n \leftarrow n'$
$(k_0, \ldots, k_n) \twoheadleftarrow \mathcal{K}(n)$

$\underline{\textsc{Enc}(c, i)}$

if $n = \perp$ **then return** Err
$(c', s_i) \leftarrow \mathcal{E}'(k_i, c, s_i)$
return c'

$\underline{\textsc{Dec}(c)}$

if $n = \perp$ **then return** Err
$(d, u) \leftarrow \mathcal{D}'(k_0, c, u)$
return d

Game iOE0 (Ideal)

$\underline{\textsc{Key}(n')}$

if $n \neq \perp$ **then return** Err
$n \leftarrow n'$

$\underline{\textsc{Enc}(c, i)}$

if $n = \perp$ **then return** Err
$c' \twoheadleftarrow \mathfrak{C}$
return c'

$\underline{\textsc{Dec}(c)}$

if $n = \perp$ **then return** Err
$d \leftarrow \Diamond$
return d

Fig. 6. Utopian games for defining inbound onion-AE security. Games iOE1 and iOE0 have semantics almost identical to that of OE1 and OE0 but now $\textsc{Enc}$ models encryptions by the ORs and $\textsc{Dec}$ models decryption by the user.

Game eOE1 (Real)

$\underline{\textsc{Key}(n')}$

if $n \neq \perp$ **then return** Err
$n \leftarrow n'$
$(k_0, \ldots, k_n) \twoheadleftarrow \mathcal{K}(n)$

$\underline{\textsc{Enc}(m)}$

if $n = \perp$ **then return** Err
$(c_0, c_1, \ldots, c_{n-1}, u) \leftarrow \mathcal{E}(k_0, m, u)$
return $(c_0, c_1, \ldots, c_{n-1})$

$\underline{\textsc{Dec}(c, i)}$

if $n = \perp$ **then return** Err
$(d, s_i) \leftarrow \mathcal{D}(k_i, c, s_i)$
return d

Game eOE0 (Ideal)

$\underline{\textsc{Key}(n')}$

if $n \neq \perp$ **then return** Err
$n \leftarrow n'$

$\underline{\textsc{Enc}(m)}$

if $n = \perp$ **then return** Err
else $(c_0, c_1, \ldots, c_{n-1}) \twoheadleftarrow \mathfrak{C}^n$
return $(c_0, c_1, \ldots, c_{n-1})$

$\underline{\textsc{Dec}(c, i)}$

if $n = \perp$ **then return** Err
return $\Diamond$

Fig. 7. Utopian games for defining eager-OE. Game eOE1 outputs everything honestly, while game eOE0 outputs freshly sampled random strings for $\textsc{Enc}$ queries and $\Diamond$ for $\textsc{Dec}$ queries (after the circuit size has been initialization by a $\textsc{Key}$ query). The games are transformed by oracle silencing using the class of schemes $\mathfrak{C}^*$ associated to strong correctness.

8 Extension 2: Eager Authenticity

The authenticity notion captured by onion-AE is an "only-at-exit" one: to satisfy our security definition, a forged onion, regardless of its point of insertion, should traverse the entire circuit until it reaches the exit node. Only then is it quashed. This is implied by the game logic of OE0, which always demands pseudorandom intermediate onions. In this section we explore an alternative aim: to immediately detect and quash a forged ciphertext. We call this alternative *eager-OE*. We intend, with eager-OE, that deviation from the honest execution should result in a decryption failure as soon as possible. Our task here is to formalize this deceptively simple-to-state requirement.

STRONG CORRECTNESS. We first describe what we expect the utopian games to look like. The "real" utopian behavior, game eOE1, does what it has to do: it follows

$$
\begin{array}{l}
\textbf{procedure } \mathrm{Correct}^{*}_{\Pi}(k, m) \\
\quad (k_0, k_1, \ldots, k_n) \leftarrow k; \quad (m_1, \ldots, m_\ell) \leftarrow m \\
\quad u, s_1, \ldots, s_n \leftarrow \varepsilon \\
\quad \textbf{for } i \leftarrow 1 \textbf{ to } \ell \textbf{ do} \\
\quad\quad c^i_n \leftarrow m_i;\ (c^i_0, c^i_1, \ldots, c^i_{n-1}, u) \leftarrow \mathcal{E}(k_0, c^i_n, u) \\
\quad\quad d^i_0 \leftarrow c^i_0 \\
\quad\quad \textbf{for } j \leftarrow 1 \textbf{ to } n \textbf{ do } (d^i_j, s_j) \leftarrow \mathcal{D}(k_j, d^i_{j-1}, s_j) \\
\quad \textbf{return } \bigwedge_{1 \le i \le \ell} \bigwedge_{1 \le j \le n} (c^i_j = d^i_j)
\end{array}
$$

Fig. 8. Strong correctness condition. The syntax of $\mathcal{E}$ is extended so that it outputs a vector of intermediate onions (from the outermost to the innermost).

$$
\begin{array}{l}
\textbf{procedure } \mathrm{Correct}^{**}_{\Pi}(k, m) \\
\quad (k_1, \ldots, k_n) \leftarrow k; \quad (m_1, \ldots, m_\ell) \leftarrow m \\
\quad u_1, \ldots, u_n, s_1, \ldots, s_n \leftarrow \varepsilon \\
\quad \textbf{for } i \leftarrow 1 \textbf{ to } \ell \textbf{ do} \\
\quad\quad c^i_n \leftarrow m_i; \\
\quad\quad \textbf{for } j \leftarrow n \textbf{ to } 1 \textbf{ do } (c^i_{j-1}, u_j) \leftarrow \mathcal{E}(k_j, c^i_j, u_j) \\
\quad\quad d^i_0 \leftarrow c^i_0 \\
\quad\quad \textbf{for } j \leftarrow 1 \textbf{ to } n \textbf{ do } (d^i_j, s_j) \leftarrow \mathcal{D}(k_j, d^i_{j-1}, s_j) \\
\quad \textbf{return } \bigwedge_{1 \le i \le \ell} \bigwedge_{1 \le j \le n} (c^i_j = d^i_j)
\end{array}
$$

Fig. 9. Layered correctness condition. The syntax is that of layered OE schemes.

the protocol. The "ideal" utopian behavior, game eOE0, should instead do this: encryption queries should return random bits, while decryption queries should return an indication of an authentication failure, which we'll continue to write as $\Diamond$.

Of course the specified games *are* utopian for the same reason as OE: an honest execution is enough for an adversary to determine if it is operating in the real or ideal games—it simply observes whether the decryption response is a string or $\Diamond$.

In the case of OE, we were able to delegate the logic of exception handling to oracle silencing (honest decryption at the exit OR). In the silenced games, oracles would behave as instructed, following the utopian games, except when some generic condition, determined by the correctness condition, demanded that the oracles return a distinguished value $\perp$. That won't work for eager-OE, since now the exceptional cases include honest decryption at intermediate ORs, which is not silenced with respect to the correctness condition we used so far. We thus need a stronger correctness condition, one that demands the equivalence of not only the final decrypted message, but also the intermediate onion values. We approach this by extending the syntax of $\mathcal{E}$ so that it outputs not only an outermost onion c_0, but also intermediate onions $c_1, c_2, \ldots, c_{n-1}$, with the number of encryption layers from $n-1$ to 1: $(c_0, c_1, \ldots, c_{n-1}; u') \leftarrow \mathcal{E}(k_0, m, u)$.

The new correctness condition is formalized in Fig. 8. We denote the class of all OE schemes Π (with the modified encryption syntax) satisfying this condition (for all $n \in \mathbb{N}$, all $k \in \mathcal{K}(n)$ and all $m \in \mathcal{M}^*$, $\mathrm{Correct}^*_\Pi(k, m)$ in Fig. 8 is true) as $\mathfrak{C}^*$. We call this the *strong* correctness condition. We then call the original correctness condition (Fig. 1) the *weak* correctness condition.

eOE GAMES. We are now ready to write the code of eOE1 and eOE0, which, with the help of the strong correctness condition becomes almost trivial. See Fig. 7.

The interesting change is the ENC oracle. With $\mathcal{E}$ now outputting a vector of onions, the ENC oracle in both worlds are changed accordingly—the real world outputs the whole vector of real onions while the ideal world samples a vector of mutually independent strings in $\mathcal{C}^n$. Note that such a change from a single outermost onion to the whole vector is necessary for oracle silencing to capture honest decryption on intermediate ORs—we need to include intermediate onions in the query history.

The changed oracle syntax does not model the real-world setting closely; in practice, an adversary does not learn the intermediate onions by coaxing a user into encrypting a chosen message. But these values can be learned by forwarding onions along the circuit, making honest DEC queries. The change still seems meaningful. The ENC oracle now provides the adversary not only what the user would output to OR_1, but, also, the intermediate onions to $\mathrm{OR}_2, \ldots, \mathrm{OR}_n$ that it would learn from an honest run. In this way, honest DEC queries do not give an adversary useful information, as they return onions already included in previous ENC responses, and will always be silenced.

OPEN QUESTIONS. Beside LBE, Mathewson [25] also introduced a second encryption scheme for Tor's relay protocol called short-MAC-and-pad. Unlike LBE, this algorithm apparently targets eager authentication. It should be possible to analyze it against the eager-OE notion. We leave its analysis as a future work, but conjecture that it should satisfy eager-OE under reasonably standard assumptions. There would appear to be further alternatives for achieving eager-OE, including an

Game xOE1 (Real)

$\underline{\textsc{Cor}(n', T')}$

if $n \neq \perp$ **then return** Err
if $T \nsubseteq [n']$ **then return** Err
$(n, T) \leftarrow (n', T')$
$(k_1, \ldots, k_n) \twoheadleftarrow \mathcal{K}(n)$
return $\{(i, k_i) : i \in T\}$

$\underline{\textsc{Enc}(m)}$

if $n = \perp$ **then return** Err
$c_n \leftarrow m$
for $i \leftarrow n$ **to** 1 **do**
    $(c_{i-1}, u_i) \leftarrow \mathcal{E}(k_i, c_i, u_i)$
return $(c_0, c_1, \ldots, c_{n-1})$

$\underline{\textsc{Dec}(c, i)}$

if $n = \perp$ **then return** Err
if $i \in T$ **then return** Err
$(d, s_i) \leftarrow \mathcal{D}(k_i, c, s_i)$
return d

Game xOE0 (Ideal)

$\underline{\textsc{Cor}(n', T')}$

if $n \neq \perp$ **then return** Err
if $T \nsubseteq [n']$ **then return** Err
$(n, T) \leftarrow (n', T')$
$(k_1, \ldots, k_n) \twoheadleftarrow \mathcal{K}(n)$
return $\{(i, k_i) : i \in T\}$

$\underline{\textsc{Enc}(m)}$

if $n = \perp$ **then return** Err
for $i \leftarrow n$ **to** 1 **do**
    if $i \in T$ **then** $(c_{i-1}, u_i) \leftarrow \mathcal{E}(k_i, c_i, u_i)$
    else $c_{i-1} \twoheadleftarrow \mathcal{C}$
return $(c_0, c_1, \ldots, c_{n-1})$

$\underline{\textsc{Dec}(c, i)}$

if $n = \perp$ **then return** Err
if $i \in T$ **then return** Err
if $i = n$ **then return** $\Diamond$
$c \twoheadleftarrow \mathcal{C}$
return c

Fig. 10. Utopian games for defining OE in the face of static corruption of ORs. Games xOE1 and xOE0 extend OE1 and OE0 by replacing $\textsc{Key}$ with $\textsc{Cor}$, which combines the modeling of key generation and static corruption together. In these games, $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ is a layered OE scheme.

embellishment of LBE that adds pairwise AE between successive ORs.

9 Extension 3: Corrupted ORs

As our final extension to OE, we discuss an approach to model corrupted ORs. Our corruption model is static: the adversary can ask one and only one oracle-corruption query, corrupting at that time all the ORs it wishes to. Corruption is done prior to asking encryption or decryption queries.

Layered OE scheme. We first introduce a further syntactic change to an OE scheme: based on the idea of strong correctness, we consider a model where a user's encryption not only outputs a vector of intermediate onions, but also keeps a vector of separate encryption keys and states. Effectively, we remove the abstraction of user's encryption key as a single element k_0. A user in this new model would instead share n secret keys $(k_1, \ldots, k_n)$ with the ORs and the encryption operation can now be expressed as a composition of multiple layers. Concretely speaking, an OE scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ has its syntax changed to what we call a *layered* OE scheme, as follows:

- $\mathcal{K}\colon \mathbb{N} \to \mathcal{K}^*$ is a probabilistic algorithm that, given a circuit size $n \geq 1$, outputs a list of n strings in $\mathcal{K} \subseteq \{0,1\}^*$. Note the difference from ordinary OE schemes where $n+1$ strings are generated.
- $\mathcal{E} : \mathcal{K} \times (\mathcal{M} \cup \mathcal{C}) \times \mathcal{U} \to \mathcal{C} \times \mathcal{U}$ is a deterministic function that takes in a *partial* user key $k_i \in \mathcal{K}$, either a plaintext $m \in \mathcal{M}$ or an intermediate onion $c \in \mathcal{C}$, and a user state $u \in \mathcal{U}$. It outputs an onion $c' \in \mathcal{C}$ with an additional layer and an updated user state $u' \in \mathcal{U}$.
- $\mathcal{D}\colon \mathcal{K} \times \mathcal{C} \times \mathcal{S} \to (\mathcal{M} \cup \mathcal{C} \cup \{\Diamond\}) \times \mathcal{S}$ is a deterministic function. It takes in a key $k \in \mathcal{K}$, an input onion $c \in \mathcal{C}$, and an OR state $s \in \mathcal{S}$. It returns a *decrypted result* d and an updated OR state $s' \in \mathcal{S}$. This is the only algorithm whose semantics remains unchanged from ordinary OE schemes.

The correctness condition of a layered OE scheme is defined in the same way as strong correctness condition: in an honest execution, all the intermediate onions must be equivalent. Formally, a layered OE scheme Π satisfies the *layered correctness condition* if for all $n \in \mathbb{N}$, all $k \in \mathcal{K}(n)$, and all $m \in \mathcal{M}^*$, the predicate $\mathrm{Correct}_\Pi^{**}(k, m)$ in Fig. 9 is true. The class of all correct layered OE schemes is denoted $\mathfrak{C}^{**}$.

Ideal encryption oracle. With the modified syntax and layered correctness condition in place, the code of utopian games modeling static corruption, xOE1 and xOE0, are given in Fig. 10. A layered encryption scheme Π is OE-secure with respect to static corruption if any adversary with a reasonable amount of resource cannot distinguish $\overline{\mathrm{xOE1}}_\Pi$ and $\overline{\mathrm{xOE0}}_\Pi$, the silencing done with respect to $\mathfrak{C}^{**}$.

Most parts of the code are self-explanatory. The most noteworthy point is the Enc oracle's behavior in the ideal world: it computes real encryption for the corrupted layers and samples fresh random strings for the honest layers. For corrupted layers there is no hope for achieving any semantic security, due to the leak of secret keys. In fact, this is this definitional choice that drove us to the formulation of a layered correctness condition.

Open questions. We leave the analysis of LBE against the notion described, xOE, as a future problem. We conjecture that it should satisfy xOE, with an analysis similar to that in Appendix B.

10 Conclusions

We end with some brief remarks on the limitations of our formalization for onion encryption, and our basic view as to what we have done.

First, we remind the reader that our setting is different from the actual one used in Tor insofar as our syntax disallows plaintexts to exit anywhere but at the end of a circuit. Modeling Tor's "leaky-pipe" possibility (which we did in earlier versions of this paper) added considerable complexity, and we questioned its value. With early exit the correctness condition becomes a game that the adversary can win by getting onions to exit where they ought not. The added complexity infects everything.

We acknowledge that our formulation of oracle silencing is by no means the only one possible. Several alternative formulations to that of Fig. 2 are possible, and we don't yet know how they compare.

While oracle silencing appears to be a powerful technique for simplifying or justifying some security notions, it is no magic bullet. We don't yet have very good intuition for precisely what one gets when one defines a utopian game and then "mods out" by a correctness condition. Also, in order to *use* a definition obtained by oracle silencing it seems that one will usually need to provide an alternative and more concrete characterization. It is a fair complaint that some of the simplicity of oracle silencing is deceptive, some of the complexity being pushed off into the concrete game that is later needed and the proof of its equivalence to the silenced game, and with further complexity hidden in the formal definition of when silencing occurs. Then again, the same complaints can be made of other powerful definitional frameworks, like UC.

We believe that Tor is the most important privacy tool currently in existence, and find it unfortunate that what was arguably the most basic aspect of it, its use of nested encryption, has lacked a compelling cryptographic definition. We have wanted to help set this right. We believe in the importance of foundations for real-world privacy problems. Cryptographic experience in a diverse set of domains suggests to us that a crucial first step towards improving practice may be to back away from the the realm of techniques and figure out what problem those techniques aim to solve. Nested encryption, properly realized, is the answer to a question that has eluded being asked: the question of how to construct a scheme that realizes authenticated encryption in a world where decryption happens using a chain of separately keyed entities.

Acknowledgments

Our work was inspired by a conversation with Nick Mathewson from several years back. Thanks also to the anonymous referees for their many excellent comments, including pointing us to Mathewson's writeup about improving Tor's relay protocol [25]. Our research was supported by NSF grants CNS 1314885 and CNS 1717542. Opinions, findings, conclusions, motivations, and recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] M. Backes, I. Goldberg, A. Kate, and E. Mohammadi. Provably secure and practical onion routing. Cryptology ePrint Archive, Report 2011/308, 2011. http://eprint.iacr.org/2011/308.

[2] M. Bellare, O. Goldreich, and S. Goldwasser. Incremental cryptography: The case of hashing and signing. In Y. Desmedt, editor, *Advances in Cryptology – CRYPTO'94*, volume 839 of *Lecture Notes in Computer Science*, pages 216–233, Santa Barbara, CA, USA, Aug. 21–25, 1994. Springer, Heidelberg, Germany.

[3] M. Bellare, T. Kohno, and C. Namprempre. Authenticated encryption in SSH: Provably fixing the SSH binary packet protocol. In V. Atluri, editor, *ACM CCS 02: 9th Conference on Computer and Communications Security*, pages 1–11, Washington D.C., USA, Nov. 18–22, 2002. ACM Press.

[4] M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In T. Okamoto, editor, *Advances in Cryptology – ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 531–545, Kyoto, Japan, Dec. 3–7, 2000. Springer, Heidelberg, Germany.

[5] M. Bellare and P. Rogaway. Encode-then-encipher encryption: How to exploit nonces or redundancy in plaintexts for efficient cryptography. In T. Okamoto, editor, *Advances in Cryptology – ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 317–330, Kyoto, Japan, Dec. 3–7, 2000. Springer, Heidelberg, Germany.

[6] C. Boyd, B. Hale, S. F. Mjølsnes, and D. Stebila. From stateless to stateful: Generic authentication and authenticated encryption constructions with application to TLS. In K. Sako, editor, *Topics in Cryptology – CT-RSA 2016*, volume 9610 of *Lecture Notes in Computer Science*, pages 55–71, San Francisco, CA, USA, Feb. 29 – Mar. 4, 2016. Springer, Heidelberg, Germany.

[7] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science*, pages 136–145, Las Vegas, NV, USA, Oct. 14–17, 2001. IEEE Computer Society Press.

[8] D. Catalano, M. Di Raimondo, D. Fiore, R. Gennaro, and O. Puglisi. Fully non-interactive onion routing with forward-secrecy. In J. Lopez and G. Tsudik, editors, *ACNS 11: 9th International Conference on Applied Cryptography and Network Security*, volume 6715 of *Lecture Notes in Computer Science*, pages 255–273, Nerja, Spain, June 7–10, 2011. Springer, Heidelberg, Germany.

[9] D. Catalano, D. Fiore, and R. Gennaro. Certificateless onion routing. In E. Al-Shaer, S. Jha, and A. D. Keromytis, editors, *ACM CCS 09: 16th Conference on Computer and Communications Security*, pages 151–160, Chicago, Illinois, USA, Nov. 9–13, 2009. ACM Press.

[10] D. L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

[11] G. Danezis and I. Goldberg. Sphinx: A compact and provably secure mix format. In *2009 IEEE Symposium on Security and Privacy*, pages 269–282, Oakland, CA, USA, May 17–20, 2009. IEEE Computer Society Press.

[12] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. Technical report, DTIC Document, 2004.

[13] J. Feigenbaum, A. Johnson, and P. Syverson. Probabilistic analysis of onion routing in a black-box model. *ACM Transactions on Information and System Security (TISSEC)*, 15(3):14, 2012.

[14] X. Fu, Z. Ling, J. Luo, W. Yu, W. Jia, and W. Zhao. One cell is enough to break tor's anonymity. In *Proceedings of Black Hat Technical Security Conference*, pages 578–589, 2009.

[15] I. Goldberg, D. Stebila, and B. Ustaoglu. Anonymity and one-way authentication in key exchange protocols. *Designs, Codes and Cryptography*, pages 1–25, 2012.

[16] D. Goldschlag, M. Reed, and P. Syverson. Hiding routing information. In *Information Hiding*, pages 137–150. Springer, 1996.

[17] D. M. Goldschlag, M. G. Reed, and P. F. Syverson. Onion routing. *Commun. ACM*, 42(2):39–41, 1999.

[18] S. Halevi. EME*: Extending EME to handle arbitrary-length messages with associated data. In A. Canteaut and K. Viswanathan, editors, *Progress in Cryptology - INDOCRYPT 2004: 5th International Conference in Cryptology in India*, volume 3348 of *Lecture Notes in Computer Science*, pages 315–327, Chennai, India, Dec. 20–22, 2004. Springer, Heidelberg, Germany.

[19] S. Halevi and P. Rogaway. A parallelizable enciphering mode. In T. Okamoto, editor, *Topics in Cryptology – CT-RSA 2004*, volume 2964 of *Lecture Notes in Computer Science*, pages 292–304, San Francisco, CA, USA, Feb. 23–27, 2004. Springer, Heidelberg, Germany.

[20] V. T. Hoang, T. Krovetz, and P. Rogaway. Robust authenticated-encryption AEZ and the problem that it solves. In E. Oswald and M. Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 15–44, Sofia, Bulgaria, Apr. 26–30, 2015. Springer, Heidelberg, Germany.

[21] A. Kate, G. Zaverucha, and I. Goldberg. Pairing-based onion routing with improved forward secrecy. Cryptology ePrint Archive, Report 2008/080, 2008. http://eprint.iacr.org/2008/080.

[22] A. Kate, G. M. Zaverucha, and I. Goldberg. Pairing-based onion routing. In *Privacy Enhancing Technologies, 7th International Symposium, PET 2007 Ottawa, Canada, June 20-22, 2007, Revised Selected Papers*, pages 95–112, 2007.

[23] J. Katz and M. Yung. Unforgeable encryption and chosen ciphertext secure modes of operation. In B. Schneier, editor, *Fast Software Encryption – FSE 2000*, volume 1978 of *Lecture Notes in Computer Science*, pages 284–299, New York, NY, USA, Apr. 10–12, 2001. Springer, Heidelberg, Germany.

[24] T. Kohno, A. Palacio, and J. Black. Building secure cryptographic transforms, or how to encrypt and MAC. *IACR Cryptology ePrint Archive*, 2003:177, 2003.

[25] N. Mathewson. Two improved relay encryption protocols for Tor cells. https://gitweb.torproject.org/torspec.git/tree/proposals/202-improved-relay-crypto.txt, June 2012. Accessed: 2017-07-02.

[26] B. Möller. Provably secure public-key encryption for length-preserving chaumian mixes. In M. Joye, editor, *Topics in Cryptology – CT-RSA 2003*, volume 2612 of *Lecture Notes in Computer Science*, pages 244–262, San Francisco, CA, USA, Apr. 13–17, 2003. Springer, Heidelberg, Germany.

[27] J.-F. Raymond. Traffic analysis: Protocols, attacks, design issues, and open problems. In *Designing Privacy Enhancing Technologies*, pages 10–29. Springer, 2001.

[28] P. Rogaway. Authenticated-encryption with associated-data. In V. Atluri, editor, *ACM CCS 02: 9th Conference on Computer and Communications Security*, pages 98–107, Washington D.C., USA, Nov. 18–22, 2002. ACM Press.

[29] P. Rogaway and Y. Zhang. Simplifying game-based definitions: Correctness-adjusted indistinguishability. Manuscript, 2017.

[30] N. M. Roger Dingledine. Tor protocol specification. https://gitweb.torproject.org/torspec.git/tree/tor-spec.txt. Accessed: 2017-02-25.

[31] P. F. Syverson, D. M. Goldschlag, and M. G. Reed. Anonymous connections and onion routing. In *1997 IEEE Symposium on Security and Privacy*, pages 44–54, Oakland, CA, USA, 1997. IEEE Computer Society Press.

[32] The 23 Raccoons. Analysis of the relative severity of tagging attacks. http://archives.seul.org/or/dev/Mar-2012/msg00019.html. Accessed: 2017-08-12.

A Concrete OE Games

In this section we develop concrete games cOE1 and cOE0 corresponding to the silenced games OE1 and OE0. This will be necessary for establishing the security of LBE. See Fig. 11 for the concrete games' code. The games are identical to the utopian OE1 and OE0 except that each oracle records its queries and responses into a query history, and the Dec oracle, before responding, first computes a *concrete silencing predicate* Ψ to decide whether or not to silence the response.

Concrete silencing predicate. We explain the semantics of Ψ in Fig. 11. At a high level, it identifies those Dec queries at the exit OR (line 1319) such that all queries so far form a *stack of end-to-end chains*. Specifically, the code first runs through all (x_i, y_i) pairs and records them in several tables. These tables share the same indexing convention: the first index models time and gets incremented for various types of queries, while the second index refers to OR indices. When the second index is j, it either refers to entities decrypted "out of" OR_j (as with entries of D) or onions with outermost layer for OR_{j+1} (as with entries of S). After the recording, the final if-clauses inspect them to check whether at all previous rows the "chain" actually forms—there is an Enc query at the row (line 1320) and Dec queries join end-to-end with the Enc query's response (line 1321 and 1322). Once this *chaining condition* is met, the predicate also makes sure there is no *chained-*$\Diamond$: a previous Dec query at the last OR that returns $\Diamond$ but satisfies the chaining condition (this can only occur in the ideal world). Above all, the predicate returns true when the chaining condition (line 1319 to 1322) is met, and there is no chained-$\Diamond$ (line 1323). See Fig. 12 for a graphical illustration of the chaining condition.

We write $\mathsf{Chain1}(n)$ to denote the chaining condition: for $1 \le i \le n$ and query history $(x_1, y_1, \ldots, x_q)$, the predicate $\mathsf{Chain1}(i)$ is defined as:

$$\begin{aligned}\mathsf{Chain1}(i) = \big(x_q.\mathrm{idx} = i\big) \wedge \big(v \ge w_i\big) \wedge \\ \big((\forall t \in [w_i])\, C_t = S[t][0]\big) \wedge \\ \big((\forall t \in [w_i]\ \forall j \in [i-1])\, D[t][j] = S[t][j]\big),\end{aligned}$$

where v, w, C, S and D are as defined in the bottom row of Fig. 11. The predicate $\mathsf{Chain2}$ is identical to Ψ in Fig. 11 except that line 1323 is negated:

$$\mathsf{Chain2} = \mathsf{Chain1}(n) \wedge (\exists t \in [w_n - 1]) D[t][n] = \Diamond.$$

Alternative characterization of onion-AE. The following lemma proves useful in order to work with our rather abstract definition onion-AE security.

Main Lemma. [Equivalence of $\overline{\mathrm{OE}}$ and cOE] *Let* $\mathfrak{C}$ *be the class of all correct OE schemes and let* $(\overline{\mathrm{OE1}}, \overline{\mathrm{OE0}})$ *be the silenced games of* $(\mathrm{OE1}, \mathrm{OE0})$ *with respect to* $\mathfrak{C}$*. Then* $\overline{\mathrm{OE1}}$ *is identical to* cOE1 *and* $\overline{\mathrm{OE0}}$ *is identical to* cOE0*.*

The lemma effectively corroborates the correctness of the complex logic embodied in the code of cOE1 and cOE0. It does this by equating the games to the abstract silenced ones defined, more convincingly, by the oracle-silencing technique. We expect this to be a typical usage of oracle silencing: the abstract nature of the silenced games makes the definition hard to work with, so one must first propose a concrete version and prove it "right" by equating the two definitions. From a different point of view, the tractability of the oracle-silencing function (the fact that the adversary itself could easily compute it) addresses the generic concern that oracle-silencing ought not be used with a correctness condition that leads to the adversary being provided information (in the silencing or non-silencing of oracles) that the adversary could not compute on its own.

Proof. Since the game logic of $\overline{\mathrm{OE1}}$ (resp. $\overline{\mathrm{OE0}}$) and cOE1 (resp. cOE0) are identical except for the condition of when to silence a query (silencing refers to replacing a query response with $\perp$), it suffices to show, for the two versions of games, that their silencing conditions (on the query history) are identical. In the following, we refer to the silencing condition in $\overline{\mathrm{OE1}}$ and $\overline{\mathrm{OE0}}$ the *abstract silencing condition*; while that in cOE1 and cOE0 the *concrete silencing condition*. Recall that

Game cOE1 (Real)

$\underline{\textsc{Key}(n')}$

if $n \neq \perp$ **then return** Err
$q \leftarrow 0$
$n \leftarrow n'$
$(k_0, \ldots, k_n) \twoheadleftarrow \mathcal{K}(n)$

$\underline{\textsc{Enc}(m)}$

if $n = \perp$ **then return** Err
$q{+}{+}$
$(c, u) \leftarrow \mathcal{E}(k_0, m, u)$
$(x_q.\text{type}, x_q.\text{msg}, y_q) \leftarrow (\mathsf{Enc}, m, c)$
return c

$\underline{\textsc{Dec}(c, i)}$

if $n = \perp$ **then return** Err
$q{+}{+}$
$(d, s_i) \leftarrow \mathcal{D}(k_i, c, s_i)$
$(x_q.\text{type}, x_q.\text{ctxt}, x_q.\text{idx}, y_q) \leftarrow (\mathsf{Dec}, c, i, d)$
if $\Psi(n, x_1, y_1, \ldots, x_{q-1}, y_{q-1}, x_q)$ **then** $y_q \leftarrow \perp$
return y_q

Game cOE0 (Ideal)

$\underline{\textsc{Key}(n')}$

if $n \neq \perp$ **then return** Err
$q \leftarrow 0$
$n \leftarrow n'$

$\underline{\textsc{Enc}(m)}$

if $n = \perp$ **then return** Err
$q{+}{+}$
$c \twoheadleftarrow \mathfrak{C}$
$(x_q.\text{type}, x_q.\text{msg}, y_q) \leftarrow (\mathsf{Enc}, m, c)$
return c

$\underline{\textsc{Dec}(c, i)}$

if $n = \perp$ **then return** Err
$q{+}{+}$
if $i = n$ **then** $d \leftarrow \Diamond$
else $d \twoheadleftarrow \mathfrak{C}$
$(x_q.\text{type}, x_q.\text{ctxt}, x_q.\text{idx}, y_q) \leftarrow (\mathsf{Dec}, c, i, d)$
if $\Psi(n, x_1, y_1, \ldots, x_{q-1}, y_{q-1}, x_q)$ **then** $y_q \leftarrow \perp$
return y_q

$\underline{\Psi(n, y_1, \ldots, x_{q-1}, y_{q-1}, x_q)}$:

$v \leftarrow 0$
for $i \in [n]$ **do** $w_i \leftarrow 0$
for $i \in [q-1]$ **do**
  if $x_i.\text{type} = \mathsf{Enc}$ **then** $v \leftarrow v + 1$;
    $(M_v, C_v) \leftarrow (x_i.\text{msg}, y_i)$
  else $j \leftarrow x_i.\text{idx}$; $w_j \leftarrow w_j + 1$;
    $(S[w_j][j-1], D[w_j][j]) \leftarrow (x_i.\text{ctxt}, y_i)$
$w_n{+}{+}$; $S[w_n][n-1] \leftarrow x_q.\text{ctxt}$
return $x_q.\text{idx} = n \;\wedge$
  $v \geq w_n \;\wedge$
  $(\forall t \in [w_n])\, C_t = S[t][0] \;\wedge$
  $(\forall t \in [w_n]\; \forall j \in [n-1])\, D[t][j] = S[t][j] \;\wedge$
  $(\forall t \in [w_n - 1])\, D[t][n] \neq \Diamond$

Fig. 11. Concrete games cOE1 **and** cOE0 **and the concrete silencing predicate** Ψ **on which they rely.** In both games, for $\textsc{Dec}$, we apply Ψ to the query history so far and silence the response if it evaluates to true.

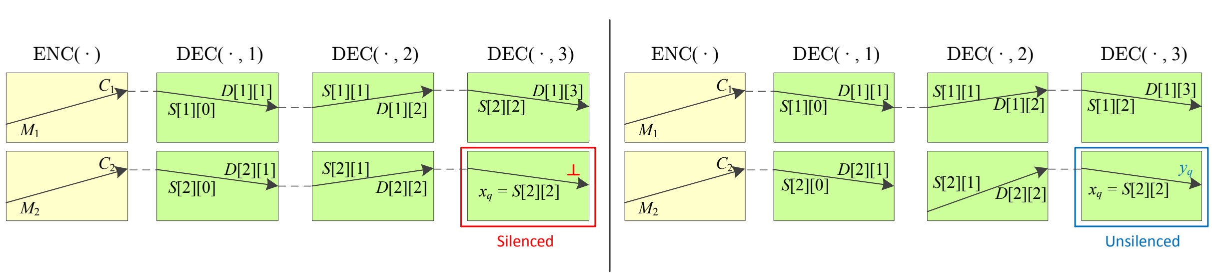


Fig. 12. Illustration of the chaining condition. The yellow boxes stand for $\textsc{Enc}$ queries and the green ones for $\textsc{Dec}$ queries. The dashed lines joining adjacent boxes imply that the left output and the right input are equal.

$\underline{\mathcal{K}(n)}$	$\underline{\mathcal{E}(k,m,u)}$	$\underline{\mathcal{D}(k,c,s)}$
for $i \leftarrow 1$ **to** n **do**	$n \leftarrow \lvert k \rvert$	$c' \leftarrow \mathbb{F}^{-1}(k.\text{key}, k.\text{id}, s, c)$
$\quad k_i.\text{key} \twoheadleftarrow \{0,1\}$	$c \leftarrow m \,\Vert\, 0^{l_2-l_1}$	$s \leftarrow s \,\Vert\, c$
$\quad k_i.\text{id} \leftarrow i$	**for** $i \leftarrow n$ **downto** 1 **do**	**if** $k.\text{exit} = 1$ **and**
$\quad k_i.\text{exit} \leftarrow 0$	$\quad c \leftarrow \mathbb{F}(k_i.\text{key}, i, u_i, c)$	$\quad c'[l_1+1..l_2] = 0^{l_2-l_1}$ **then**
$k_n.\text{exit} \leftarrow 1$	$\quad u_i \leftarrow u_i \,\Vert\, c$	$\quad$ **return** $(c'[1..l_1], s)$
$k_0 \leftarrow (k_1, \ldots, k_n)$	$u \leftarrow (u_1, u_2, \ldots, u_n)$	**if** $k.\text{exit} = 1$ **then**
return $(k_0, \ldots, k_n)$	**return** (c, u)	$\quad$ **return** $(\Diamond, s)$
		return (c', s)

Fig. 13. Code for $\text{AOE}[\mathbb{F}, l_1]$**, an artificial scheme used in the proof of the main lemma**. Here $\mathbb{F}$ is a tweakable permutation on $\{0,1\}^{l_2}$.

the abstract silencing condition is a conjugation of Fixed and Nondegenerate; the concrete silencing condition is a conjunction of Chain1(n) and $\neg$Chain2, where n is the circuit size as initialized by the KEY query.

We first show Chain1$(n) \wedge \neg$Chain2 $\Rightarrow$ Fixed $\wedge$ Nondegenerate. The easy part is Chain1$(n) \Rightarrow$ Fixed. Let $(x_1, y_1, \ldots, x_{j-1}, y_{j-1}, x_j)$ be the query history so far, by the correctness condition of OE, for any $\Pi \in \mathfrak{C}$, if Chain1(n) is satisfied by the history then for any Γ such that $\text{Gives}_{\text{OE1},\Pi,\mathfrak{C}}(x_1, y_1, \ldots, x_{j-1}, y_{j-1}; \Gamma)$, we have $\text{OE1}_\Pi(x_1, x_2, \ldots, x_j; \Gamma)$ equal to the message queried in the corresponding ENC query, which is determined by the history. That implies Fixed is satisfied.

It remains to show $\neg$Chain2 $\Rightarrow$ Nondegenerate. In fact, we are going to show a stronger result, that $\neg$Chain2 $\Leftrightarrow$ Nondegenerate and $\neg$Chain1$(n) \wedge \neg$Chain2 $\Rightarrow$ $\neg$Fixed. We establish this by constructing an artificial OE scheme AOE $\in \mathfrak{C}$ out of the query history. This will prove the reverse direction that the abstract silencing condition implies the concrete silencing condition as well, hence concluding the proof.

See Fig. 13 for the code of AOE. Basically, it uses a generic tweakable permutation $\mathbb{F}$ to perform the computation. In particular, the key space is simply $\{0,1\}$, while the tweak consists of the OR's ID (from 1 to n) and the concatenation of this OR's ciphertext history. For convenience, we sometimes subscript the key input of $\mathbb{F}$ and write $\mathbb{F}_0$ and $\mathbb{F}_1$ instead. Note that AOE is very similar to LBE and indeed LBE can be viewed as a specialization of AOE where $\mathbb{F}$ is independent of ORs' IDs. In the following, we will use the same variable notations as those in the bottom row of Fig. 11.

Suppose Chain2 is true, namely there is a DEC query for which the chaining condition is met yet its response is $\Diamond$. It is easy to see such an event can only occur in cOE0, for in cOE1 the response would have been either the original queried message or $\perp$ by the correctness condition and the game logic. Therefore, we conclude Chain2 $\Rightarrow$ $\neg$Nondegenerate.

Next suppose Chain2 is false, define $\mathbb{F}$ in terms of the query history in the following way. Initially, all the ORs are *synced*. By an OR_i being synced, we mean its encryption history u_i and decryption history s_i are identical. In fact, there will be two versions (for the two key values) of the user states $u_{i,0}$ and $u_{i,1}$ and OR states $s_{i,0}$ and $s_{i,1}$. We next specify, for each ENC query in turn, one or two input-output value pairs for each OR_i at certain tweak values of $\mathbb{F}_0$ and $\mathbb{F}_1$.

During the specification we also maintain the invariant that OR_j being synced implies all OR_i $(i \le j)$ being synced. The specification goes like this: for the t-th ENC query and response (M_t, C_t): for all synced OR_i $(1 \le i \le j)$, if $S[t][i-1] = D[t][i-1]$ (here $D[t][0]$ refers to C_t), then OR_i keeps synced and we specify both $\mathbb{F}_0(i, u_{i,0}, D[t][i]) \leftarrow S[t][i-1]$ and $\mathbb{F}_1(i, u_{i,1}, D[t][i]) \leftarrow S[t][i-1]$. (In case $D[t][i] \in \{\Diamond, \perp\}$ or does not exist, which is possible when $i = n$, let $D[t][i]$ be $M_t \,\Vert\, 0^{l_2-l_1}$ instead). Meanwhile, we update the states $u_{i,k}$ to $u_{i,k} \,\Vert\, S[t][i-1]$ accordingly for $k \in \{0,1\}$. For the first OR_m such that either $S[t][m-1] \neq D[t][m-1]$ (the input ciphertext deviates from the decrypted result from the predecessor OR or there is no corresponding DEC query hence no $S[t][m-1]$) or OR_m is unsynced, if $m < n$, choose two distinct $L_0[t][m]$ and $L_1[t][m]$ that are different from $S[t][m]$ and $D[t][m]$, and specify $\mathbb{F}_k(m, u_{m,k}, L_k[t][m]) \leftarrow D[t][m-1]$ for $k \in \{0,1\}$; if $m = n$, let both $L_0[t][m]$ and $L_1[t][m]$ be $M_t \,\Vert\, 0^{l_2-l_1}$ and do the same. Finally, for all remaining OR_i apply similar specifications: as long as the OR is not the exit one choose distinct $L_k[t][i]$ and specify $\mathbb{F}_k(i, u_{i,k}, L_k[t][i]) \leftarrow L_k[t][i-1]$ for $k \in \{0,1\}$; otherwise simply let $L_k[t][i]$ be $M_t \,\Vert\, 0^{l_2-l_1}$. Note in this way, the invariant that the synced ORs are always at the left-most are maintained, and its number can only decrease. Next, for each Dec query (c, d) at OR_i: if $d \in \{0,1\}^{l_2}$ let $\mathbb{F}_k(i, s_{i,k}, d) \leftarrow c$ for $k \in \{0,1\}$; if $d \in \{0,1\}^{l_1}$ let $\mathbb{F}_k^{-1}(i, s_{i,k}, c) \leftarrow d \,||\, 0^{l_2 - l_1}$ for $k \in \{0,1\}$; if $d = \lozenge$ let $\mathbb{F}_k^{-1}(i, s_{i,k}, c) \leftarrow \lozenge$ for $k \in \{0,1\}$. Last but not least, for the last query x_q, if it is an Enc query choose two distinct responses in $\mathcal{C}$ distinct from $S[v][0]$; if it is an Dec query on OR_i choose two distinct responses distinct from $S[v][i], L_0[v][i], L_1[v][i]$, and M_v if present.

The above specification only partially defines $\mathbb{F}$ at some domain points. For completing the definition, we simply specify arbitrary values on other undefined points. A simple induction on the number of Enc queries show that $\neg\mathsf{Chain2}$ implies the following are true:

- The partial specification is consistent: there are no two specifications for $\mathbb{F}$ such that the same input gets mapped to distinct outputs; nor are there any specification for $\mathbb{F}^{-1}$ that are contradictory to $\mathbb{F}$. This means the complete definition as mentioned above is well defined.
- When $\mathsf{Chain1}(n)$ is true, the given construction AOE can generate the history $(x_1, y_1, \ldots, x_{j-1}, y_{j-1})$ for both $k \in \{0,1\}$ in cOE1. This means $\mathsf{Chain1}(n) \wedge \neg\mathsf{Chain2} \Rightarrow \mathsf{Nondegenerate}$.
- When $\mathsf{Chain1}(n)$ is false, the given construction AOE not only satisfies the previous point, but also generates distinct responses for $k = 0$ and $k = 1$ for the last query x_j. This means $\neg\mathsf{Chain1}(n) \wedge \neg\mathsf{Chain2} \Rightarrow \neg\mathsf{Fixed}$.

The above three points conclude the proof. □

B Proof of Theorem 6.1

Proof. We will be constructing an adversary $\mathcal{B}$ for which

$$\mathbf{Adv}^{\mathsf{oe}}_{\Pi}(\mathcal{A}) \leq n \cdot \mathbf{Adv}^{\mathsf{sprp}}_{\mathbb{E}}(\mathcal{B}) + q_E \cdot \frac{n}{2^{l_2 - 1}} + q_D \left(\frac{3}{2^{l_2}} + \frac{1}{2^{l_2 - l_1}} \right), \qquad (1)$$

from which the result follows after a bit of manipulation. In broad outline, we employ a hybrid argument to replace the underlying TBC with a string-tweaked purely random permutation, and then we analyze that construction to show that no adversary will have any advantage unless it induces some bad events to occur. We bound the probability of those bad events.

By the hybrid argument, we construct $\mathcal{B}$ in the following way:

- Initially, run adversary $\mathcal{A}$ and get its intended circuit size n. Randomly choose an OR $j \twoheadleftarrow [n]$. Initialize u_i and s_i for each $i \in [n]$, and k_i for $i \in [n] - \{j\}$. Note that $\mathcal{B}$ can simulate $\mathrm{cOE1}_\Pi$ except computations involving k_j.
- Upon an Enc query m from $\mathcal{A}$, simulate a hybrid iterative encryption $\mathcal{E}$ as follows. For the layer before the j-th router, simulate a purely random string-tweaked permutation and use that to replace $\mathbb{E}$; for the honest layer after the j-th router, simulate the real $\mathbb{E}$; for the j-th layer, query the Encipher oracle with u_j as the tweak. Finally, update the user state and return the outermost onion to $\mathcal{A}$.
- Upon a Dec query (c, i) from $\mathcal{A}$, simulate it according to the code of $\mathrm{cOE1}_\Pi$ with all layers before j replaced by purely random string-tweaked permutations; all layers after j as real $\mathbb{D}$; and the very j-th by querying Decipher oracle with s_j as the tweak. Finally, update the server state s_i accordingly.
- Upon receiving the final bit b from $\mathcal{A}$, return to the challenger b.

Let us analyze the behavior of $\mathcal{B}$. First, note that when $\mathcal{B}$'s oracles realize the real TBC, what $\mathcal{A}$ sees is a variant of $\mathrm{cOE1}_\Pi$ where all layers before and *excluding* the j-th layer have their underlying TBCs replaced by purely random tweakable permutations. On the other hand, when $\mathcal{B}$'s oracles realize purely random string-tweaked permutations, what $\mathcal{A}$ sees is a similar variant, but this time with all layers before and *including* the j-th layer replaced.

We use G_i to denote the variant of $\mathrm{cOE1}_\Pi$ where all layers before and *including* the i-th layer are replaced. In this way, G_0 denotes the original $\mathrm{cOE1}_\Pi$ while G_n denotes a variant where all underlying TBCs are replaced. Applying standard hybrid argument:

$$\begin{aligned}\mathbf{Adv}^{\mathsf{sprp}}_{\mathbb{E}}(\mathcal{B}) &\geq \frac{1}{n}\big(\mathbf{Pr}[\mathcal{A}^{G_0} \Rightarrow 1] - \mathbf{Pr}[\mathcal{A}^{G_n} \Rightarrow 1]\big) \\ &= \frac{1}{n}\big(\mathbf{Adv}^{\mathsf{oe}}_{\Pi}(\mathcal{A}) - \big(\mathbf{Pr}[\mathcal{A}^{G_n} \Rightarrow 1] \\ &\qquad - \mathbf{Pr}[\mathcal{A}^{\mathrm{cOE0}} \Rightarrow 1]\big)\big)\end{aligned}$$

It remains to upper bound $\mathbf{Pr}[\mathcal{A}^{G_n} \Rightarrow 1] - \mathbf{Pr}[\mathcal{A}^{\mathrm{cOE0}} \Rightarrow 1]$. For this purpose, notice that in game G_n, all onion layers use purely random string-tweaked permutations. Since each Enc and Dec query appends the user and router states, the successive oracle queries of the same type never repeat the same tweak and hence are mutually independent. Therefore, for each OR, the corresponding permutation is sampled with the same tweak at most twice: one from an Enc query and the other from a Dec query. Our analysis makes use of that and rewrites most of the code into fresh string sampling, so that G_n can be made "identical-until-bad" to cOE0. See Fig. 14. We remark that although the re-writing does not alter the semantics of G_n, it does alter the semantics of cOE0 due to the assignment in line 1535. We will come to this point later.

To express the above idea in the code, we introduce a new variable T, indexed by a state string s and an OR-index i. Table $T[i, s]$ records the first sampled input-output pair of the tweakable permutation for tweak s and OR_i. This sampling results from either an Enc query or a Dec one, depending on which goes first in using s as the tweak.

It remains to bound the probability of bad events, which, in terms of code, are the conditions of the if-statements before every boxed statements in Fig. 14.

We begin with the Enc query.

Line 1519: Informally, this line is reached when previously there was a Dec query that sets an entry in T and the current Enc query happens to have a colliding input for the outermost layer, resulting in an output that is not freshly sampled. We now argue that this event is unlikely, based on the assumption that *no previous* Dec *query produced a decrypted result in* $\mathcal{M}$ *that was not silenced*. We analyze the probability of that event later, and merely assumes it has not taken place here.

In Line 1517, L_i has three possible origins: line 1511, line 1513, and line 1518 (before any boxed events occur). For line 1511, by the assumption of no previous unsilenced decrypted results in $\mathcal{M}$, a colliding L_{i+1} originating from line 1511 can only occur with probability at most 2^{-l_2}, because any information about this colliding L_{i+1} with $l_2 - l_1$ trailing zeros (sampled in line 1531 in a Dec query which first accessed $T[i, u[i]]$) must have been silenced. For the other cases, consider the layer $i + 1$, since line 1513 results in independent samplings, whenever L_{i+1} comes from such a sampling, the probability of collision is 2^{-l_2}. If L_{i+1} otherwise comes from line 1518 the problem reduces to bounding the probability of $L_{i+1} = L'_{i+1}$. An inductive argument shows that the final probability of collision is at most $nq_E/2^{l_2}$ throughout the game (on the assumption that no previous unsilenced decrypted results in $\mathcal{M}$, and no boxed events occurred so far).

Line 1522. The if-event is a simple collision of a fresh independent string, so its probability is bounded by $nq_E/2^{l_2}$ across the game.

We next analyze the Dec query.

Line 1535: Since m gets some value that is not freshly sampled, it is necessary to argue that such a non-random assignment does not allow the adversary to see any difference from cOE0. That is, if m is going to be returned to the adversary, it has to be uniformly random and independent from what the adversary has learned. Our argument goes in two steps:

- With probability at least $1 - 2^{-l_2}$, reaching to $m = m'$ in line 1535 implies $\mathsf{Chain1}(i)$.
- In the likely event of $\mathsf{Chain1}(i)$, the assigned value m' either is uniformly random and independent from what the adversary has learned, or it is going to be silenced.

For the first step, suppose at some time when line 1535 is reached $\mathsf{Chain1}(i)$ is false. Consider the first time when such an event takes place: the only possibility is that the adversary deviated from the "staircase order" and queried c directly on an inner layer. However, since Enc only returns the outermost sampling, and the inner layers can only be learned by asking Dec in the correct staircase order (the only line of Dec returning a "non-fresh" string is line 1535), the information about c' is hidden from the adversary and thus the probability of $c = c'$ is at most 2^{-l_2}.

The second argument follows partly from the above. Namely, apart from the innermost layer, m' must have come from a sampling in line 1513 of the Enc oracle (assuming no boxed events occurred so far), which is uniformly random. The remaining case is the innermost honest layer, for which we will show that with high probability, the predicate Ψ will be true, namely the response would be silenced. First, the chaining condition is satisfied by condition already, and therefore all previous Dec queries at the exit satisfy the chaining condition as well. Now suppose at some previous Dec query there is a chained-$\Diamond$, then at the point of this query, it cannot satisfy the chaining condition since otherwise that would contradict to the game's logic (a chained Dec query at the exit should return a message). By the same argument, such an event occurs with probability at most $1/2^{l_2}$. Applying union bound, we conclude for line 1535, the probability that the adversary gains any information-theoretic advantage throughout the game execution is bounded by $2q_D/2^{l_2}$.

Line 1536 and 1538: We have seen that as long as the response is not silenced, it always appears freshly uniform to the adversary. Therefore the two kinds of events together are bound by $q_D(2^{-l_2} + 2^{-(l_2-l_1)})$.

Finally, it remains to analyze the probability of unsilenced decrypted result in $\mathcal{M}$. Observe that this is ex-

$\underline{\textsc{Enc}(m)}$

1511 $L_n \leftarrow m \,||\, 0^{l_2 - l_1}$
1512 **for** $i \leftarrow n$ **downto** 1 **do**
1513 $L_{i-1} \twoheadleftarrow \{0,1\}^{l_2}$
1514   **if** $T[i, u_i] = \text{undef}$ **then** $T[i, u_i] \leftarrow (L_i, L_{i-1})$
1515   **else**
1516 $(L'_i, L'_{i-1}) \leftarrow T[i, u_i]$
1517   **if** $L_i = L'_i$ **then**
1518 $L_{i-1} \leftarrow L'_{i-1}$
1519   **if** $i = 1$ **then**
1520 $\boxed{L_{i-1} \twoheadleftarrow \{0,1\}^{l_2}}$
1521   **else**
1522   **if** $L_{i-1} = L'_{i-1}$ **then**
1523 $\boxed{L_{i-1} \twoheadleftarrow \{0,1\}^{l_2} / \{L'_{i-1}\}}$
1524 $u_i \leftarrow u_i \,||\, L_{i-1}$
1525 **return** L_0

$\underline{\textsc{Dec}(c, i)}$

1531 $m \twoheadleftarrow \{0,1\}^{l_2}$
1532 **if** $T[i, s_i] = \text{undef}$ **then** $T[i, s_i] \leftarrow (m, c)$
1533 **else**
1534 $(m', c') \leftarrow T[i, s_i]$
1535   **if** $c = c'$ **then** $m \leftarrow m'$
1536   **else if** $m = m'$ **then** $\boxed{m \twoheadleftarrow \{0,1\}^{l_2} - \{m'\}}$
1537 $s_i \leftarrow s_i \,||\, c$
1538 **if** $i = n$ **and** $m[l_1 + 1..l_2] = 0^{l_2 - l_1}$ **then**
1539 $\boxed{\textbf{return } m[1..l_1]}$
1540 **else if** $i = n$ **then return** $\Diamond$
1541 **return** m

Fig. 14. Rewritten code for $\mathbf{G_n}$ (with dashed box and without solid box) and OE0 (with solid box and without dashed box), used for bounding the adversarial advantage in distinguishing them. The semantics of G_n is preserved while the semantics of cOE0 is not due to line 1535.

actly the event of line 1539. So the bad events already contain unsilenced results in $\mathcal{M}$, thus no need to include it again.

Formula 1 is now established by summing the above probabilities and substituting that sum as the upper bound for $\mathbf{Pr}[\mathcal{A}^{G_n} \Rightarrow 1] - \mathbf{Pr}[\mathcal{A}^{\mathrm{OE0}} \Rightarrow 1]$. □