# On the Secure Obfuscation of Deterministic Finite Automata

(Extended Abstract)

W. Erik Anderson*

Sandia National Laboratories

**Abstract**

In this paper, we show how to construct secure obfuscation for Deterministic Finite Automata, assuming non-uniformly strong one-way functions exist. We revisit the software protection approaches originally proposed by [6, 13, 16, 21] and revise them to the current obfuscation setting of Barak et al. [2]. Under this model, we introduce an efficient oracle that retains some "small" secret about the original program. Using this secret, we can construct an obfuscator and two-party protocol that securely obfuscates Deterministic Finite Automata against *malicious* adversaries. The security of this model retains the strong "virtual black box" property originally proposed in [2] while incorporating the stronger condition of dependent auxiliary inputs in [19]. Additionally, we further show that our techniques remain secure under concurrent self-composition with adaptive inputs and that Turing machines are obfuscatable under this model.

**Keywords:** Obfuscation, deterministic finite automata, state machines, Turing machines, authenticated encryption, oracle machines, provable security, game-playing.

## 1 Introduction

Program obfuscation, if possible and practical, would have a considerable impact on the way we protect software systems today. It would be instrumental in protecting intellectual property, preventing software piracy, and managing use-control applications. Since its inception, many practitioners have relied on using heuristic notions of security [11]. These heuristics often provide a false sense of security, as they are usually informal and lack a rigorous framework. Only recently has a formalized framework of obfuscation been given.

The work of Barak et al. [2] initiated the first formal study of obfuscation. They define an obfuscator $\mathcal{O}$ to be an efficient, probabilistic compiler that takes a program $P$ and transforms it into a functionally equivalent yet unintelligible program $\mathcal{O}(P)$. Unintelligible is defined in the strictest sense, to imply that the program $\mathcal{O}(P)$ behaves ideally like a "virtual black box". That is, whatever can be efficiently extracted from the obfuscated program can also be extracted efficiently when given only oracle access to $P$.

*Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000. Email: weander@sandia.gov

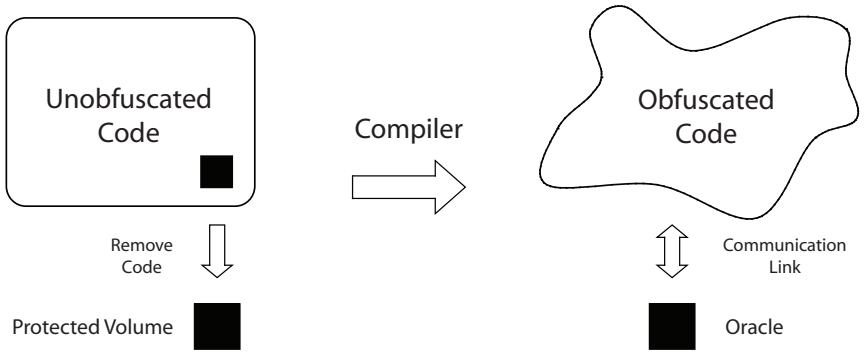


Figure 1: Obfuscation with respect to oracle machines

Unfortunately, in [2] it was proven that obfuscation in general is impossible. Namely, there exist a family of functions that are unobfuscatable under the "virtual black box" notion of security. This would seem to suggest that having physical access to the program is a much stronger capability than having only access to its input and output behavior. In addition to this main impossibility result, the authors also prove that if secure symmetric key encryption schemes exist, pseudorandom functions exist, or message authentication schemes exist, then so do unobfuscatable versions of each. Therefore, it would appear that the "virtual black box" property is inherently flawed, and if we hope for any positive obfuscation results, then either this model needs to be abandoned or we need to accept that many programs are not obfuscatable [2].

Numerous other impossibility results have also shed light on the problem of obfuscation. For example, Goldwasser et al. [19] showed that many natural circuit classes are unobfuscatable when auxiliary inputs are added to the obfuscation model. Auxiliary inputs provide for a more robust model of obfuscation, since the adversary is assumed to have some a priori information about the underlying program. This additional layer of security is useful in practice, since it is likely that the obfuscated code will be utilized in a large system, and the system may inadvertently reveal partial information about the functionality of the code. A formal definition with respect to dependent auxiliary inputs is given Section 1.2.

In spite of the numerous impossibility results, other works such as Lynn et al. [22] have examined alternative models of obfuscation in the hope of achieving meaningful possibility results. Under the random oracle model of obfuscation, they assume that both the obfuscator and obfuscated code have access to a public random oracle. Under this assumption, they are able to show that both point functions and complex access control schemes are obfuscatable. Similar results were obtained by [7, 10, 25] under a slightly weaker notion of "virtual black box" obfuscation (without random oracles). For example, Wee showed in [25] that point functions are (weakly) obfuscatable, provided that strong one-way permutations exist.

In this paper, we introduce a new model of obfuscation that has wide and meaningful possibility results beyond those described above. To demonstrate the utility of this model, we show that both deterministic finite automata and Turing machines are securely obfuscatable, provided non-uniformly strong one-way functions exist. We call this model of obfuscation *obfuscation w.r.t. oracle machines*.

Unlike the "virtual black box" model of obfuscation, where we assume an adversary has full access

to the obfuscated code, we instead consider the case where a small portion of the computation remains hidden and is only accessible via black box. See Figure 1 for an illustration. A compiler in this case takes a program $P$ and returns two outputs, the obfuscated code $\mathcal{O}(P)$ which is given to the adversary, and a small secret which is given to the oracle (black box). An execution of the obfuscated code takes an input $x$ and computes $\mathcal{O}(P)(x)$, via a two-party protocol. To avoid certain trivialities, we impose restrictions on the oracle's computational resources. Namely, we will consider only the case when the oracle's space resources are asymptotically smaller than the program itself. In practice, the oracle may be implemented as a computationally limited device, such as a smart card or crypto processor.

### 1.1 Our Contribution

We generalize the software protection model originally proposed by Goldreich [13] into the current setting of virtual black box obfuscation with dependent auxiliary input. We prove several necessary conditions for obfuscating non-resettable deterministic finite automata and show they are securely obfuscatable provided that non-uniformly strong one-way functions exist. We further extend this model to the composition setting and show our techniques remain secure under concurrent self-composition with adaptive inputs. Using the obfuscation techniques derived for deterministic finite automata, we also prove that Turing machines (hence all programs) are obfuscatable under this model.

### 1.2 Related work

**Obfuscation w.r.t. Auxiliary Inputs.** In [19], Goldwasser and Kalai introduced the notion of *obfuscating w.r.t. auxiliary inputs*. Under this setting, the adversary is given some additional a priori (auxiliary) information in addition to the obfuscated code. They consider two types of auxiliary inputs in their paper, dependent and independent. For our examination here, we will review only the dependent case.

Dependent auxiliary input includes the case when the a priori information may depend on the underlying program. Whatever the auxiliary information happens to be, it should not provide the adversary any additional advantage in extracting information from the obfuscated code than would be the case if that same auxiliary information was available to an adversary with only black box access to the machine. A formal definition is given below.

**Definition 1 (Obfuscation w.r.t. Dependent Auxiliary Input)** *A probabilistic polynomial-time algorithm $\mathcal{O}$ is said to be an obfuscator of the family $\mathcal{F} = \{\mathcal{F}_k\}_{k\in\mathbb{N}}$ w.r.t. dependent auxiliary input, if the following three conditions hold:*

- *(Approximate Functionality) There exists a negligible function $\mu$ such that for all $k$ and $M \in \mathcal{F}_k$, $\mathcal{O}(M, 1^k)$ describes an TM that computes the same function as $M$ with probability at least $1 - \mu(k)$.*
- *(Polynomial Slowdown) The description length and running time of $\mathcal{O}(M, 1^k)$ is at most polynomial larger than that of $M$. That is, there exists a polynomial $p$ such that for all $k$ and $M \in \mathcal{F}_k$, $|\mathcal{O}(M, 1^k)| \leq p(k)$ and if $M$ takes $t$ time steps on an input $x$, then $\mathcal{O}(M, 1^k)$ takes at most $p(k + t)$ time steps on $x$.*

- *(Virtual Black Box) For every PPT $A$, there is a PPT simulator $S$ and a negligible function $\nu$ such that, for all $k$ and $M \in \mathcal{F}_k$, and every polynomial $q$ with bounded auxiliary input $z$ of size $q(k)$ we have*

$$\left|\Pr[A(\mathcal{O}(M, 1^k), 1^k, z) = 1] - \Pr[S^M(1^{|M|}, 1^k, z) = 1]\right| \leq \nu(k).$$

Under the dependent case, the authors prove that if the class of point-filter functions[1] can be (weakly) obfuscated with respect to dependent auxiliary input, then every class of circuits with super-polynomial pseudo-entropy cannot be (weakly) obfuscated w.r.t. dependent auxiliary input. This would imply that many cryptographic tasks, such as pseudorandom functions, encryption schemes, and signature algorithms cannot be obfuscated [19].

### 1.3 Notation

We will use the notation PPT to stand for probabilistic polynomial-time Turing machine. If $A$ is a PPT, B an oracle, and $x$ an input to $A$, then by $A^B(x)$ we mean the algorithm that runs on input $x$ using oracle access to $B$. We will often refer to $A$ as a PPT oracle machine. When writing $x \Leftarrow A$ we mean the value $x$ is returned by $A$. Additionally, $A(1^k)$ implies $A$ is given the value $k$. In our algorithm descriptions, we make use of the statements **return** $y$ and **Return** $z$. When using the syntax **return**, we imply that the value $y$ is returned internally to the algorithm (such as the output of a function call), while by using **Return**, we imply that the value $z$ is written to the output tape. As usual, we use the notation $\{0,1\}^k$ to denote the set of all $k$-bit binary strings, and by $x \xleftarrow{\$} \{0,1\}^k$ we mean $x$ is uniformly chosen from $\{0,1\}^k$. We also use the conventional notation of $\|$ and $\oplus$ to denote the string operators *concatenate* and *exclusive or*. Unless explicitly stated otherwise, we will assume all references to log are based 2. A function $\mu : \mathbb{N} \to \mathbb{R}^+$ is said to be *negligible*, if for any positive polynomial $p$ there exists an integer $N$ such that for any $k > N$, $\mu(k) < 1/p(k)$. We will sometimes use the notation $neg(\cdot)$ to denote an arbitrary negligible function.

## 2 Obfuscation with respect to Oracle Machines

In this section we introduce the framework for *obfuscating w.r.t. oracle machines*. We model obfuscation under this new framework as a two-party protocol, where one party represents the obfuscated code and the other an oracle containing some "small" secret. The communication between the two parties is characterized using *interactive Turing machines* introduced by [17]. Under this framework, we assume the adversary has complete control over both the obfuscated code and message scheduling. We further assume the adversary is *malicious*, and may deviate from the protocol in any way. This allows the adversary to adaptively query the oracle with messages of its own choice. We define an interactive Turing machine as follows.

**Interactive Turing Machines.** An interactive Turing machine (ITM) is a Turing machine that has an additional communication tape, together with its read-only input tape, read-only random

[1] The class of point-filter functions $\Delta^L = \{\Delta_n^L\}_{n\in\mathbb{N}}$ for a language $L \in$ NP, is defined as the set of functions $\Delta_n^L := \{\delta_{x,b}\}_{x\in\{0,1\}^n, b\in\{0,1\}}$ where $\delta_{x,b}(w) = (x, b)$ if $w$ is a valid witness to $x$ in $R_L$ and $\delta_{x,b}(w) = x$ otherwise.

tape, write-only output tape, and read-and-write work tape. The communication tape consists of two tapes, a write-only *outgoing communication tape* and a read-only *incoming communication tape*. When the incoming and outgoing communication tape of one ITM are shared with the outgoing and incoming communication tapes of the other ITM we call this pair an *interactive pair of Turing machines*.

We denote a pair of interactive Turing machines $M$ and $N$ as the tuple $(M, N)$. The pair $(M, N)$ is assumed to be ordered in the sense that at any one time only one Turing machine is active. The active Turing machine can compute on its internal work tapes, read from its input tapes, write to its output tape, and send a message to the other Turing machine on its outgoing communication tape. When one Turing machine has completed its computation, it transfers control over to the other. This process continues until one machine reaches a halt state.

Informally, we view the oracle as a computationally limited device containing some "small" secret related to the original program. The oracle's internal computations are assumed to be hidden, so that, behaviorally, it appears to be a black box.

**Oracle Model.** The obfuscation oracle $\mathcal{R}$ is modeled as an interactive Turing machine with one additional read-and-write tape called *internal_state*. The tape *internal _state* has a unique feature called *persistence* that distinguishes it from the other tapes in the oracle. We say a tape is *persistent* if the tape's contents are preserved between each successive execution of $\mathcal{R}$. The other internal working tapes do not share this property and are assumed to be blank after each execution. Given a particular *input* and *internal_state*, the oracle $\mathcal{R}$ computes an *output* (which may be $\perp$) on its outgoing communication tape along with a new internal state, *internal_state*$'$.

$$(output, internal\_state') \leftarrow \mathcal{R}(input, internal\_state)$$

If $\mathcal{R}$ does not have access to a random tape then we say $\mathcal{R}$ is deterministic.

Before we finalize the oracle's computational model, we need to capture the idea of a resource-limited device. This will help clarify our meaning of an oracle maintaining a "small" secret. To explore this idea more throughly, we consider the following two illustrations. In our first example, we examine the case when the *internal_state* tape is assumed to be very large, so large in fact that it can store the entire program that is being obfuscated. In this instance, we can create a trivial obfuscator that loads the entire program into the oracle's *internal_state* at setup. This simulates a true black box and maintains the security properties we are after. However, in practice very large programs may not physically fit on a device, or it may be prohibitively expensive to do so, especially if the device requires tamper and read-proof protection.

As another example, we consider the parallel case when the input tape is very large. In this case we can devise a protocol that loads an authenticated encrypted version of the program onto the oracle's work tape (for each input query), which the oracle later decrypts, authenticates, and runs. Having a large input tape does not make sense in practice, as the input is usually comparatively smaller than the program size. To avoid these trivialities we consider placing bounds on both the size of the *internal_state* and input tape of the oracle. Under this supposition, we assume there exists a polynomial $s(\cdot)$ such that for each $k \in \mathbb{N}$, both tapes are polynomial bounded by $s(k)$. In this framework, we will consider only the non-trivial case when $s(k) = o(f(k))$,[2] where the device's

[2]The size of each $M \in \mathcal{F}_k$ is polynomial bounded by $f(k)$.

resources are asymptotically smaller than the program itself. In the special case when $s(k) = O(k)$, we will say that the oracle maintains a "small" internal state.

**Definition 2 (Obfuscation w.r.t. Oracle Machines)** *A probabilistic polynomial time algorithm $\mathcal{O}$ and oracle $\mathcal{R}$ are said to comprise an obfuscator of the family $\mathcal{F} = \{\mathcal{F}_k\}_{k\in\mathbb{N}}$ w.r.t. polynomial-time bounded oracle machines, if the following three conditions hold:*

- *(Approximate Functionality) There exists a negligible function $\mu$ such that for all $k$ and $M \in \mathcal{F}_k$, $\mathcal{O}^{\mathcal{R}}(M, 1^k)$ describes an ITM that computes the same function as $M$ with probability at least $1 - \mu(k)$.*
- *(Polynomial Slowdown) The description length and running time of $\mathcal{O}^{\mathcal{R}}(M, 1^k)$ is at most polynomial larger than that of $M$. That is, there exists a polynomial $p$ such that for all $k$ and $M \in \mathcal{F}_k$, $|\mathcal{O}(M, 1^k)| \leq p(k)$ and if $M$ takes $t$ time steps on an input $x$, then $\mathcal{O}^{\mathcal{R}}(M, 1^k)$ takes at most $p(k + t)$ time steps on $x$.*
- *(Virtual Black Box) For every PPT $A$, there is a PPT simulator $S$ and a negligible function $\nu$ such that, for all $k$ and $M \in \mathcal{F}_k$, and every polynomial $q$ with bounded auxiliary input $z$ of size $q(k)$, we have*

$$\left|\Pr[A^{\mathcal{R}}(\mathcal{O}^{\mathcal{R}}(M, 1^k), 1^k, z) = 1] - \Pr[S^M(1^{|M|}, 1^k, z) = 1]\right| \leq \nu(k).$$

For convenience, when our family $\mathcal{F}$ is a family of Turing machines, we will adopt the convention that each program is represented by its binary string encoding for some fixed[3] polynomial time *universal* Turing machine. Therefore, the size of each Turing machine is measured as the size of its binary string representation. To simulate the adversaries view correctly, we will also leak to $S$ the number of steps taken on each oracle query to $M$. So not only will $S$ be able to observe the input and output behavior of $M$, but it will also observe its timing as well.

Before moving onto the next section we restate the definition of non-uniformly strong one-way functions.

**Definition 3 (Non-Uniformly Strong One-Way Functions):** *A polynomial-time computable function $f : \{0,1\}^* \to \{0,1\}^*$ is called non-uniformly strong one-way if for every non-uniform PPT $A$ there is a negligible function $neg(\cdot)$ such that for sufficiently large $k$,*

$$\Pr_{x \xleftarrow{\$} \{0,1\}^k}[\, f^{-1}(f(x)) \ni y \leftarrow A(f(x), 1^k)\,] \leq neg(k).$$

## 2.1 Non-Resettable Deterministic Finite Automata

We define a *Deterministic Finite Automaton* (DFA) as a machine $\Psi = (Q, \Sigma, \delta, s_0, G)$ with a finite set of states $Q$, finite alphabet $\Sigma$, transition function $\delta$, initial state $s_0 \in Q$, and accepting states $G$. The structure of the DFA is determined by its transition function $\delta$, which maps each state and a given input symbol to a new state. The output function (which imitates black box behavior) of the DFA $\Psi$ is defined as

$$\Psi(s, \alpha) := \begin{cases} 1 & \text{if } \delta(s, \alpha) \in G \\ 0 & \text{if } \delta(s, \alpha) \notin G \end{cases}$$

[3]The description size of a program between any two universal Turing machines differs by at most a constant.

where the "user"-selectable input is $\alpha$, and $s$ is the current "internal" state. We note that the user does not have control of the state input. Rather $\Psi$ must internally maintain the state over each execution. We will often write just $\Psi(\alpha)$.

When modeling DFAs, it is often convenient (unless stated otherwise) to assign a *reset* capability, which allows the DFA to transition back to its initial state. In practice, having a reset capability is not always a desired characteristic, especially when developing software use-control applications, such as subscription policies and digital rights management. To differentiate between DFAs that have a reset capability and those that don't, we define a *non-resettable* DFA to be a deterministic finite automaton that is not resettable. We note that we can always build in resettability if we add an additional reset symbol to every state.

A topic of related interest that has been actively studied over the years has been on the problem of developing efficient learning algorithms. A learning algorithm takes a given input-output sample (i.e., transcript) and tries to construct a DFA that is *consistent* with this sample. Finding a minimum-state DFA that is consistent for a given sample was shown by Gold [12] to be NP-Hard. This holds for any passively observable learning algorithm of an unknown DFA (with a particular representation). Angulin extends this result and shows that active learning that allows user selectable inputs, is equally hard [1]. Specifically, one can construct a family of DFAs that cannot be learned in less than exponential time. A common interest to this line of work has been the relationship of resettability and learning. Non-resettability under certain frameworks can sometimes lead to efficient learning algorithms [23]. In general, though, learning a non-resettable DFA is a very difficult problem.

### 2.2 Necessary Conditions for Obfuscating a DFA

In this section we develop several necessary conditions for securely obfuscating a non-resettable DFA. In particular, we show that obfuscation is feasible only provided that the oracle's internal state is both read-proof and non-static. To facilitate the proof in Proposition 1, we begin by constructing a family of non-resettable DFAs that are hard to characterize given only black box access, yet easy given some additional power, such as reset. We define the family of non-resettable DFAs $\Psi_{i,j}$, $i, j \in \{0, 1\}$ to be the set of machines with the following characteristics: $Q = \{0, 1, 2\}$, $|\Sigma| \geq 2$, initial state 0, accept states $G_{i,j} = \{1 \text{ iff } i = 1, 2 \text{ iff } j = 1\}$, and transition function

$$\delta(q, \lambda) := \begin{cases} 0 & \text{if } q = 0 \text{ and } \lambda \in \Sigma - \{\alpha, \beta\} \\ 1 & \text{if } (q = 0 \text{ and } \lambda = \alpha) \text{ or } q = 1 \\ 2 & \text{if } (q = 0 \text{ and } \lambda = \beta) \text{ or } q = 2. \end{cases}$$

The family described above branches into two distinct states, depending on whether the first input symbol is $\alpha$ or $\beta$. Clearly, one can learn the DFA's full description if resets are allowed. We exploit this simple observation in the following result.

**Proposition 1** *If non-resettable DFAs are obfuscatable w.r.t. oracle machines then the following conditions must hold:*

*(1) The oracle's internal state tape cannot be static.*

*(2) The oracle must be read-proof.*

**Proof:** For the first condition we let $\mathcal{O}$ be any secure non-resettable DFA obfuscator. We show that having a static internal state gives the adversary a non-negligible advantage. Suppose $|\Sigma| \geq 2$ and consider the non-resettable DFAs $\Psi_{i,j}$ described above with alphabet symbols $\alpha, \beta \in \Sigma$. Since $\mathcal{O}$ is a secure obfuscator we must have for every PPT $A$ and auxiliary input $z$, the existence of a PPT simulator $S$ satisfying

$$\left|\Pr[A^{\mathcal{R}}(\mathcal{O}^{\mathcal{R}}(M_{\Psi_{i,j}}, 1^k), 1^k, z) = 1] - \Pr[S^{M_{\Psi_{i,j}}}(1^{|M_{\Psi_{i,j}}|}, 1^k, z) = 1]\right| \leq \nu(k)$$

Let $A$ be the adversary that takes the original obfuscated code $\mathcal{O}(\Psi_{i,j})$, stores a copy $C \leftarrow \mathcal{O}(\Psi_{i,j})$ and runs $i \leftarrow C^{\mathcal{R}}(\alpha)$ and $j \leftarrow C^{\mathcal{R}}(\beta)$. The distinguishing bit returned by $A$ is $b \leftarrow i \oplus j$. Now since

$$1 - \Pr[S^{\Psi_{i,j}}(1^{|M_{\Psi_{i,j}}|}, 1^k, z) = 1] \leq \nu(k) \quad \text{for } i \neq j$$

and

$$\Pr[S^{\Psi_{i,j}}(1^{|M_{\Psi_{i,j}}|}, 1^k, z) = 1] \leq \nu(k) \quad \text{for } i = j$$

we must have

$$1 - \nu(k) \leq \frac{1}{2} \sum_{i \neq j} \Pr[S^{\Psi_{i,j}}(1^{|M_{\Psi_{i,j}}|}, 1^k, z) = 1]$$

and

$$\frac{1}{2} \sum_{i=j} \Pr[S^{\Psi_{i,j}}(1^{|M_{\Psi_{i,j}}|}, 1^k, z) = 1] \leq \nu(k).$$

But the following equality

$$\sum_{i \neq j} \Pr[S^{\Psi_{i,j}}(1^{|M_{\Psi_{i,j}}|}, 1^k, z) = 1] = \sum_{i=j} \Pr[S^{\Psi_{i,j}}(1^{|M_{\Psi_{i,j}}|}, 1^k, z) = 1]$$

implies $1/2 \leq \nu(k)$ for $k$ sufficiently large, contradicting our assumption that $\nu$ is negligible.

For the second condition we assume the oracle is not read-proof, which implies the entire internal state can be extracted. Therefore, the adversary can simulate an exact copy of the oracle on its own. Using the same arguments as above we reach a contradiction. □

# 3 DFA Obfuscation

Following the framework described in Section 2, we show how to construct a DFA obfuscator that is secure with respect to dependent auxiliary inputs. Our goal is to develop a compact, yet very efficient DFA obfuscator that is not only of theoretical interest, but useful in practice as well. To obtain our results, we use a simple authenticated encryption scheme to hide the structure of the DFA and authenticate the execution of the protocol. As noted earlier, we view a Turing machine as a program running on a *universal TM*. Therefore, when describing our DFA representations, we will informally write their descriptions as pseudocode.

**Representation.** We model each DFA $\Psi$ as a polynomial-time Turing machine $M_\Psi$ with an additional *persistent* read-and-write tape, called *internal_state*. The *internal_state* maintains a record of the values needed to compute the DFA, such as the DFA's current state. Each $M_\Psi$ is represented by a table where, $\forall\ \alpha \in \Sigma$, $\forall\ s \in Q$, there is a table entry containing $\alpha$, $s$, $\delta(s, \alpha)$, and *acpt* (which

equals 1 iff $\delta(s, \alpha) \in G$). Without any loss of functionality, we compress the table by employing an injective map that encodes each $\alpha \in \Sigma$ to a string in $\{0,1\}^{\lceil \log |\Sigma| \rceil}$. Using the table described, we can create a program $M_\Psi$ that simulates $\Psi$'s output behavior. The program consists of the DFA table, high-level code, and two persistent variables *current_state* and *current_acpt*. The high-level code describes the programming language used, table lookup algorithm, alphabet $\Sigma$, and function calls that manipulate the persistent variables. The program $M_\Psi$ works as follows: On user input $\alpha$, the table lookup algorithm searches each table entry for the pair $\alpha, \mathit{current\_state}$. If a match is found, the *acpt* bit is updated and $\delta(\mathit{current\_state}, \alpha)$ is recorded temporarily. The program continues to search the rest of the table for a match. At the end of the table search, the user is given the recorded *acpt* bit, and the variable $\mathit{current\_state} \leftarrow \delta(\mathit{current\_state}, \alpha)$ is updated. After the *acpt* bit has been returned, the DFA is ready to accept its next input.

Following this description, our next goal is to define an encoding scheme of $M_\Psi$. Our choice of encoding is important for several reasons. First, it allows us to calculate the size of $|M_\Psi|$, which is needed for evaluating the polynomial slowdown property. And second, depending on our choice of encoding, the size of $|M_\Psi|$ may drastically affect the simulator's ability to simulate the obfuscated code. We formalize our encoding scheme as follows.

**Encoding.** We begin our encoding by splitting up the description of $M_\Psi$ into its individual components: high-level code and DFA table (which is further broken down by individual table entries). We create a parsing scheme that takes the bit description of each component and adds a trailing bit of a 1 or 0 to the end of each individual bit. The trailing bit allows the parser to recognize the end of a component's description. For example if the high-level code has a bit description $h_0 \dots h_m$ then its new bit description is $h_0 0 h_1 0 \dots h_m 1$. Adopting this encoding scheme, we can find a $t \geq 0$ such that the size of each table entry satisfies $2^t \leq |\mathit{table\ entry}| < 2^{t+1}$. Given $t$, we pad each table entry with the string $00 \dots 01$ (which is a multiple of two in length) until its length is exactly $2^{t+1}$. If the number of tables entries is even, we pad the last table entry with an additional $2^{t+1}$ bits of the form $00 \dots 01$ and add a single 1 bit value on the end. If, on the other hand, the number of table entries is already odd, then we do nothing. For convenience we denote the number of edges in $\Psi$ as $|E(\Psi)|$. By prefixing the parser to the encoded $M_\Psi$, it follows that $|M_\Psi| = |\text{Parser}| + |\text{High-level code}| + |\text{Table}|$, where $|\text{Table}| = 2^{t+1}|E(\Psi)|$ if the number of table entries is odd and $2^{t+1}(|E(\Psi)| + 1) + 1$ if the number of table entries is even.

Since both the size of the parser and the high-level code are public, it follows that knowing the size of $|M_\Psi|$ implies that one also knows the size of |Table|. But one can efficiently extract the number of edges $|E(\Psi)|$ based on our encoding above. We use this deduction later in the proof of Proposition 2 to swap the simulator's input $1^{|M_\Psi|}$ with $1^{|E(\Psi)|}$.

Based on the encoding above, we define the family $\mathcal{F}_{\text{DFA}} := \{\mathcal{F}_k\}_{k \in \mathbb{N}}$ to be the set of all polynomial bounded $M_\Psi$ satisfying

$$\mathcal{F}_k := \{M_\Psi \mid |M_\Psi| \leq f(k) \text{ and } 2 \log |\mathit{States}(\Psi)| + \log |\Sigma| + 1 < k\}$$ [4]

for some fixed polynomial $f(k)$. The parameter $k$ is called the *security parameter*.

[4] The condition $2 \log |\mathit{States}(\Psi)| + \log |\Sigma| + 1 < k$ may be removed by modifying the encryption scheme in Figure 3 to have more than one call to $F_K$ per table entry. This is a relatively easy fix since we need at most $m = \lceil (2t \log(ck) + 1)/k \rceil$ constant calls to $F_K$ given $|M_\Psi| \leq f(k) \leq ck^t$ some fixed $c, t > 0$. This condition was added to simplify the obfuscation algorithm.

**Setup**$(M_\Psi, k)$**:**

INPUT: $M_\Psi, 1^k$

KEY GENERATION:
$K \leftarrow \mathcal{K}(k)$

GENERATE STATE TABLE:
STATETABLE$(\Psi)$ :
$s \leftarrow 0$
$|m|^* \leftarrow \lceil \log_2 |\Sigma| \rceil + 2\lceil \log_2 |Q| \rceil + 1$
**for** $state \leftarrow 0$ **to** $|Q| - 1$ **do**
  **for** $symbol \leftarrow 0$ **to** $|\Sigma| - 1$ **do**
    $s_\alpha \leftarrow \alpha_{symbol}$
    $s_{state} \leftarrow state$
    $s_{\delta(state,\alpha)} \leftarrow \delta(state, \alpha_{symbol})$
    $s_{acpt} \leftarrow 1$ iff $s_{\delta(state,\alpha)} \in G$, 0 else
    $\mathbb{T}^*_{state}[s] \leftarrow$
      $s_\alpha \| s_{state} \| s_{\delta(state,\alpha)} \| s_{acpt} \| 0^{k-|m|^*}$
    $s \leftarrow s + 1$
$|Table|^* \leftarrow s$
**return** $(|m|^*, |Table|^*, \mathbb{T}^*_{state})$

ENCRYPT STATE TABLE ENTRIES:
$\mathcal{E}_{F_K}(\mathbb{T}^*_{state})$ :
$X_1 \leftarrow 1^k$
$Auth \leftarrow F_K(X_1)$
**for** $s \leftarrow 0$ **to** $|Table|^* - 1$ **do**
  $X_0 \leftarrow s \| 0$
  $Y \leftarrow F_K(X_0)$
  $\mathbb{T}^*_C[s] \leftarrow Y \oplus \mathbb{T}_{state}[s]$
  $X_1 \leftarrow Auth \oplus \mathbb{T}^*_C[s]$
  $Auth \leftarrow F_K(X_1)$
$Auth^* \leftarrow Auth$
**return** $\mathbb{T}^*_C \| Auth^*$

**Return** $(K, |m|^*, |Table|^*, \mathbb{T}^*_C, Auth^*)$

**Algorithm** $\mathcal{O}(|Table|^*, \mathbb{T}^*_C, Auth^*)$:

INPUT: $|Table|^*, \mathbb{T}^*_C, Auth^*$

INITIALIZATION:
$|Table| \leftarrow |Table|^*$
$\mathbb{T}_C \leftarrow \mathbb{T}^*_C$
$Auth \leftarrow Auth^*$
$State \leftarrow$ **Transition_Query**

STATE TRANSITIONS:
**Case**$(State)$

**Transition_Query:**
  $\alpha \leftarrow scan\_input$
  **Query oracle** $\mathcal{R}$ with $\alpha$
  $State \leftarrow$ **State_Update**

**State_Update:**
  **for** $s \leftarrow 0$ **to** $|Table| - 1$ **do**
    **if** $s \neq |Table| - 1$ **then**
      **Query oracle** $\mathcal{R}$ with $\mathbb{T}_C[s]$
    **if** $s = |Table| - 1$ **then**
      **Query oracle** $\mathcal{R}$ with $\mathbb{T}_C[s] \| Auth$

  **if** $acpt \Leftarrow \mathcal{R}$
    **Return** $acpt$
    $State \leftarrow$ **Transition_Query**
  **if** $auth\_fail \Leftarrow \mathcal{R}$
    $State \leftarrow$ **Transition_Query**

Figure 2: Algorithm **Setup** and $\mathcal{O}$.

**Obfuscation.** To simplify our description of the DFA obfuscator, we split the obfuscation into three separate algorithms, Setup, $\mathcal{O}$, and $\mathcal{R}$. The Setup algorithm, shown in Figure 2, takes a DFA encoding $M_\Psi$ and generates inputs for both the obfuscated code and oracle. Without loss of generality, we view our encoding of $M_\Psi$ to be the DFA state transition table of $\Psi$. The parsing operation and high-level code was left out for simplicity.

The obfuscated code $\mathcal{O}$, also shown in Figure 2, can be described as a protocol template. The template takes as input the encrypted table $\mathbb{T}_C$, authentication tag $Auth$, and table size $|Table|$ returned by the Setup algorithm. During the **Transition_Query** phase the obfuscated code scans in the user's input $\alpha$, queries the oracle $\mathcal{R}$, and enters a new phase called **State_Update**. Dur-

**Algorithm** $\mathcal{R}(K, |m|^*, |Table|^*)$:

INPUT: $K, |m|^*, |Table|^*$

INITIALIZATION:
$|Table| \leftarrow |Table|^*$
$|m| \leftarrow |m|^*$
$acpt \leftarrow \perp$
$current\_state \leftarrow 0$
$Auth' \leftarrow \perp$
$temp_\alpha \leftarrow \perp$
$temp_{cs} \leftarrow \perp$
$s \leftarrow \perp$
$State \leftarrow$ **Transition_Query**

STATE TRANSITIONS:
**Case**$(State)$

**Transition_Query:**
**On query** $\alpha$ **do**
$temp_\alpha \leftarrow \alpha$
$s \leftarrow 0$
$X_1 \leftarrow 1^k$
$Auth' \leftarrow F_K(X_1)$
$State \leftarrow$ **State_Update**

**State_Update:**
**On query** $\mathbb{T}_C[s]$ **or** $\mathbb{T}_C[s] \| Auth$ **do**

STATE AUTHENTICATION:
$X_1 \leftarrow Auth' \oplus \mathbb{T}_C[s]$
$Auth' \leftarrow F_K(X_1)$
**if** $s = |Table| - 1$ **and** $Auth' \neq Auth$
**then**
**Return** *auth_fail*
$State \leftarrow$ **Transition_Query**

COMPARE TABLE ENTRIES:
$X_0 \leftarrow s \| 0$
$Y \leftarrow F_K(X_0)$
$M_s \leftarrow Y \oplus \mathbb{T}_C[s]$
$s_\alpha \| s_{state} \| s_{\delta(state,\alpha)} \| s_{acpt} \leftarrow M_{s[k-1:k-|m|]}$

**if** $s_{state} = current\_state$ **and** $s_\alpha = temp_\alpha$
**then**
$temp_{cs} \leftarrow s_{\delta(state,\alpha)}$
$acpt \leftarrow s_{acpt}$

UPDATE ORACLE STATE & COUNTER:
**if** $s = |Table| - 1$ **then**
$current\_state \leftarrow temp_{cs}$
**Return** *acpt*
$State \leftarrow$ **Transition_Query**
$s \leftarrow s + 1$

Figure 3: Oracle $\mathcal{R}$.

ing **State_Update**, the obfuscated code submits the table $\mathbb{T}_C$ along with the authentication tag *Auth*. The oracle processes $\mathbb{T}_C$ one table entry at a time and verifies the table's integrity. If the authentication passes, the oracle returns an accept value corresponding to whether the new state is an accept state.

The oracle $\mathcal{R}$, shown in Figure 3, describes the oracle's behavior. Just like the obfuscated code $\mathcal{O}$, the oracle is nothing more than a protocol with a symmetric key and a few additional variables. Other than the padding length $|m|$, table size $|Table|$, and *current_state*, the oracle maintains no information about the DFA.

**Proposition 2** *If non-uniformly strong one-way functions exist, then non-resettable DFAs are obfuscatable with respect to oracle machines.*

**Proof:** Let $f(k)$ be some positive polynomial and consider the family $\mathcal{F}_{\text{DFA}}$ defined over $f(k)$. We will assume without loss of generality for the remainder of the proof that $k$ is sufficiently large so that the inequality $2\log|States(\Psi)| + \log|\Sigma| + 1 < k$ is satisfied for every $M_\Psi \in \mathcal{F}_k$. This assumption follows from the fact that every $M_\Psi \in \mathcal{F}_k$ is polynomial bounded, and therefore there exists a fixed $t > 0$ with $|M_\Psi| \leq k^t$ for every $k$ sufficiently large. Thus $|States(\Psi)||\Sigma| < |M_\Psi| \leq k^t$ implies $\log|States(\Psi)| + \log|\Sigma| < t\log k$ whence $2\log|States(\Psi)| + \log|\Sigma| + 1 \leq 2t\log k + 1 < k$ for $k$ sufficiently large. This last restriction was added to guarantee that the size of each table entry is

no larger than the size of the pseudorandom function's output.

To prove that the obfuscator in Figure 2 and 3 obfuscates non-resettable DFAs, we need to show that the aforementioned three conditions hold: *Approximate Functionality*, *Polynomial Slowdown*, and *Virtual Black Box*.

*Approximate Functionality:* The oracle may be in only one of two states at any one time. In the first state, **Transition_Query**, the user submits a transition symbol to the oracle, which the oracle internally stores on its *internal state* tape. After this value has been written, the oracle's state is updated to its second state, called **State_Update**. In **State_Update** the user transmits the encrypted table to the oracle (from top to bottom). The oracle checks the ciphertext integrity in each table entry and compares the underlying plaintext with the current state and stored transition symbol. If a match occurs, the oracle stores the new transition state and accept bit in its *internal state*. Provided that the protocol has been executed faithfully, the oracle will return the *acpt* bit on the last query. After this stage has been completed, the oracle reverts back to its **Transition_Query** state, and this cycle repeats indefinitely. Given this short description, it is not difficult to verify that the obfuscated DFA computes the original DFA with a probability of 1.

*Polynomial Slowdown:* In order to show that the obfuscator satisfies *polynomial slowdown* we must prove there exists a polynomial $p$ satisfying: for all $k$ and $M_\Psi \in \mathcal{F}_k$ the description length satisfies $|\mathcal{O}(M_\Psi, 1^k)| \leq p(k)$ and if $M_\Psi$ takes $t$ time steps on an input $x$ then $\mathcal{O}^{\mathcal{R}}(M_\Psi, 1^k)$ takes at most $p(k+t)$ time steps on $x$. We do this by constructing two polynomials, one that bounds the description size of the obfuscated code and the other bounding the number of steps. We then construct a suitable polynomial from both of these that satisfies the above requirement.

Observe that the size of the obfuscated code is asymptotically bounded above by $|\mathcal{O}(M_\Psi, 1^k)| = O(|\text{High-level Code}| + k|E(\Psi)|)$. Since $M_\Psi \in \mathcal{F}_k$, we must have $|E(\Psi)| \leq |M_\Psi| \leq f(k)$. But this implies $|\mathcal{O}(M_\Psi, 1^k)| = O(kf(k))$; hence the description length is polynomial bounded. For the time complexity, observe that the string comparisons for each table entry under $M_\Psi$s takes at least $\lceil \log |\Sigma| \rceil + \lceil \log |States(\Psi)| \rceil \geq \log |E(\Psi)|$ steps. Since this operation is repeated $|E(\Psi)|$ times it follows that the total number of steps needed to compute $M_\Psi$ on any input is at least $t \geq |E(\Psi)| \log |E(\Psi)|$. On the other hand the number of steps needed for $\mathcal{O}^{\mathcal{R}}(M_\Psi, 1^k)$ to send the table to the oracle is at most $O(k|E(\Psi)| \log |E(\Psi)|)$, while the oracle, which is polynomial time computable, takes at most $q(k)$ polynomial number of steps per query. Therefore, the total number of steps taken on any input (including the number of steps for the oracle) is at most $O(k\, q(k)|E(\Psi)| \log |E(\Psi)|)$. Without loss of generality, we may assume that both polynomials $f(k)$ and $q(k)$ absorb the constants for the asymptotic bounds of the description length and time complexity. Therefore, we can find a suitable $n, c > 0$ such that for all $k$, $\max\{f(k), q(k)\} \leq ck^n$. We claim that $p(k) := ck^{n+2}$ satisfies the *polynomial slowdown* requirement. This is clear since the description length $|\mathcal{O}(M_\Psi, 1^k)| \leq kf(k) \leq ck^{n+2}$ and the time complexity of $\mathcal{O}^{\mathcal{R}}(M_\Psi, 1^k)$ is at most $k\, q(k)|E(\Psi)| \log |E(\Psi)| \leq c(k+t)^{n+2}$. Our claim follows.

*Virtual Black Box:* To simplify the notation in the proof we omit the input $1^k$. We also replace the simulator input $1^{|M_\Psi|}$ with $1^{|E(\Psi)|}$ which can be extracted (based on our encoding of $M_\Psi$). This reduces the virtual black box inequality to Equation (1).

We begin our analysis by breaking up Equation (1) into four separate problems, each problem representing the indistinguishability of obfuscating with different oracles. Other than the first oracle $\mathcal{R}_{F_K}$ we do not place any computational assumptions on the others. This allows them to

maintain a much larger internal state.

$$\left|\Pr[A^{\mathcal{R}_{F_K}}(\mathcal{O}^{\mathcal{R}_{F_K}}(\Psi), z) = 1] - \Pr[S^{\Psi}(1^{|E(\Psi)|}, z) = 1]\right| \tag{1}$$

$$\leq \left|\Pr[A^{\mathcal{R}_{F_K}}(\mathcal{O}^{\mathcal{R}_{F_K}}(\Psi), z) = 1] - \Pr[A^{\mathcal{R}_{\text{Fun}}}(\mathcal{O}^{\mathcal{R}_{\text{Fun}}}(\Psi), z) = 1]\right| \tag{2}$$

$$+ \left|\Pr[A^{\mathcal{R}_{\text{Fun}}}(\mathcal{O}^{\mathcal{R}_{\text{Fun}}}(\Psi), z) = 1] - \Pr[A^{\mathcal{R}^*_{\text{Fun}}}(\mathcal{O}^{\mathcal{R}^*_{\text{Fun}}}(\Psi), z) = 1]\right| \tag{3}$$

$$+ \left|\Pr[A^{\mathcal{R}^*_{\text{Fun}}}(\mathcal{O}^{\mathcal{R}^*_{\text{Fun}}}(\Psi), z) = 1] - \Pr[A^{\mathcal{R}^*_{\text{Rand}}}(\mathcal{O}^{\mathcal{R}^*_{\text{Rand}}}(\Psi), z) = 1]\right| \tag{4}$$

$$+ \left|\Pr[A^{\mathcal{R}^*_{\text{Rand}}}(\mathcal{O}^{\mathcal{R}^*_{\text{Rand}}}(\Psi), z) = 1] - \Pr[S^{\Psi}(1^{|E(\Psi)|}, z) = 1]\right|. \tag{5}$$

In Equation (2) we introduce the oracle $\mathcal{R}_{\text{Fun}}$ in order to measure the pseudorandomness of $\mathcal{R}_{F_K}$. Both $\mathcal{R}_{\text{Fun}}$ and $\mathcal{R}_{F_K}$ have the same description, except every call to $F_K$ in $\mathcal{R}_{F_K}$ is replaced with a similar call to a random function (independent of $z$) with the same input and output size. For convenience we refer to this random function as Fun. Using algorithms $\mathcal{E}$ and $\mathcal{V}$ shown in Figure 10 (with the $IV's$ removed), we can reduce the distinguishability of Equation (2) to the distinguishability of the pair of oracles $(\mathcal{E}_{F_k}, \mathcal{V}_{F_k})$ and $(\mathcal{E}_{\text{Fun}}, \mathcal{V}_{\text{Fun}})$. We base this reduction on adversary $B_{A,\Psi}$ given in Figure 4.

In our description of $B_{A,\Psi}$, we use the parameter $\Psi$ to indicate the hardwiring of $B$'s oracle query to $\mathcal{E}$ (which is dependent on StateTable($\Psi$)). $B_{A,\Psi}$ uses $\mathcal{E}$'s response to construct the obfuscated code, which is given to $A$. Using $A$, $B_{A,\Psi}$ simulates $A$'s query-response interaction with the oracle. The distinguishing bit $b$ returned by $B_{A,\Psi}$ is the same bit returned by $A$. Therefore Equation (6) reduces to Equation (7). If we replace every oracle call to $\mathcal{E}$ and $\mathcal{V}$ with multiple calls to either $F_K$ or Fun then we can reduce Equation (7) even further. We denote this simulation by $B_{A,\Psi}{}'$ to distinguish itself from $B_{A,\Psi}$. Therefore Equation (7) reduces to Equation (8). But this last equation is just the pseudorandom distinguishability of $F_K$ given auxiliary input $z$. Using our assumption that non-uniformly strong one-way functions exist, we can use the Goldreich et al. construction in [14] to generate a pseudorandom function that is secure against non-uniform PPT adversaries (denoted as prf-nu). If the adversary $A$ makes no more than $q_v$ distinct[5] **State_Update** queries, then the total number of queries made to $F_K$ or Fun by $B_{A,\Psi}{}'$ is no more than $(q_v + 2)|E(\Psi)| + 1$. Therefore Equation (6) reduces to Equation (10), which is negligible following our assumption.

$$\left|\Pr[A^{\mathcal{R}_{F_K}}(\mathcal{O}^{\mathcal{R}_{F_K}}(\Psi), z) = 1] - \Pr[A^{\mathcal{R}_{\text{Fun}}}(\mathcal{O}^{\mathcal{R}_{\text{Fun}}}(\Psi), z) = 1]\right| \tag{6}$$

$$= \left|\Pr[B_{A,\Psi}^{\mathcal{E}_{F_K}, \mathcal{V}_{F_K}}(z) = 1] - \Pr[B_{A,\Psi}^{\mathcal{E}_{\text{Fun}}, \mathcal{V}_{\text{Fun}}}(z) = 1]\right| \tag{7}$$

$$= \left|\Pr[B_{A,\Psi}^{F_K}{}'(z) = 1] - \Pr[B_{A,\Psi}^{\text{Fun}}{}'(z) = 1]\right| \tag{8}$$

$$= \mathbf{Adv}^{\text{prf}}_{F_K, B_{A,\Psi}{}'}(k, z) \tag{9}$$

$$\leq \mathbf{Adv}^{\text{prf-nu}}_{F_K}(k, (q_v + 2)|E(\Psi)| + 1). \tag{10}$$

For Equation (3), we would like to perform a similar reduction as performed for Equation (8) except, instead of measuring the pseudorandomness of $F_K$, we would like to measure the unforgeability provided by the verifier $\mathcal{V}$. To do this, we introduce the oracle $\mathcal{R}^*_{\text{Fun}}$. Internally, the oracle $\mathcal{R}^*_{\text{Fun}}$ looks identical to $\mathcal{R}_{\text{Fun}}$ except during the state authentication process. Instead of computing a partial authentication tag for each **State_Update** query, as is done in Figure 3, it collectively

[5]Each $q_v$ represents a complete chain of **State_Update** queries (i.e., the user has submitted the entire encrypted table with *Auth* tag).

**Setup of $B_{A,\Psi}$:**

INPUT: $1^k, z$

GENERATE STATE TABLE:
$(|m|, |Table|, \mathbb{T}_{state}) \leftarrow \text{STATETABLE}(\Psi)$

ENCRYPT STATE TABLE ENTRIES:
**Query oracle** $\mathcal{E}$ with $\mathbb{T}_{state}$
$(\mathbb{T}_C, Auth) \Leftarrow \mathcal{E}(\mathbb{T}_{state})$

$A \Leftarrow \mathcal{O}(|Table|, \mathbb{T}_C, Auth), z$

**Simulation of Oracle $\mathcal{R}$:**

INPUT: $1^k, |m|, |Table|, \mathbb{T}_{state}, \mathbb{T}_C, Auth$

INITIALIZATION:
$current\_state \leftarrow 0$
$acpt \leftarrow \perp$
$temp_\alpha \leftarrow \perp$
$temp_{cs} \leftarrow \perp$
$flag_{auth} \leftarrow \perp$
$C \leftarrow \perp$
$s \leftarrow \perp$
$State \leftarrow$ **Transition_Query**

**Case**$(State)$
**Transition_Query:**
**When** $A$ makes a query $\alpha$ **do**
$temp_\alpha \leftarrow \alpha$
$flag_{auth} \leftarrow false$
$C \leftarrow \perp$
$s \leftarrow 0$
$State \leftarrow$ **State_Update**

**State_Update:**
**When** $A$ makes a query $\mathbb{T}'_C[s]$ **or** $\mathbb{T}'_C[s] \| Auth'$ **do**

STATE AUTHENTICATION:
$C \leftarrow C \| \mathbb{T}'_C[s]$
**if** $\mathbb{T}'_C[s] \neq \mathbb{T}_C[s]$ **or** $(s = |Table| - 1$ **and** $Auth' \neq Auth)$ **then**
$flag_{auth} \leftarrow true$
**if** $s = |Table| - 1$ **and** $flag_{auth} = true$ **then**
**Query oracle** $\mathcal{V}$ with $(C, Auth)$
**if** $0 \Leftarrow \mathcal{V}(C, Auth)$ **then**
$A \Leftarrow auth\_fail$
$State \leftarrow$ **Transition_Query**

COMPARE TABLE ENTRIES:
$M'_s \leftarrow \mathbb{T}'_C[s] \oplus (\mathbb{T}_C[s] \oplus \mathbb{T}_{state}[s])$
$s_\alpha \| s_{state} \| s_{\delta(state,\alpha)} \| s_{acpt} \leftarrow M'_{s[k-1:k-|m|]}$
**if** $s_{state} = current\_state$ **and** $s_\alpha = temp_\alpha$ **then**
$temp_{cs} \leftarrow s_{\delta(state,\alpha)}$
$acpt \leftarrow s_{acpt}$

UPDATE ORACLE STATE:
**if** $s = |Table| - 1$ **then**
$current\_state \leftarrow temp_{cs}$
$A \Leftarrow acpt$
$State \leftarrow$ **Transition_Query**
$s \leftarrow s + 1$

Figure 4: Adversary $B_{A,\Psi}$.

gathers all of the ciphertext queries and final authentication tag and submits them to a verifier $\mathcal{V}^*$. To do this, $\mathcal{R}^*_{\text{Fun}}$ stores the values $(\mathbb{T}_C, Auth)$ returned by the initial **Setup**$(M_\Psi, k)$ algorithm. During the **State_Update** phase, the oracle checks to see if the table entries queried by the user are the same entries as those in $\mathbb{T}_C$. If any of the table entries are incorrect, including the final authentication tag, or if they are queried in a different order, the oracle $\mathcal{R}^*_{\text{Fun}}$ returns *auth_fail*. This is equivalent to querying $\mathcal{V}^*$,

$$1 \leftarrow \mathcal{V}^*(C \| Auth) \quad \text{iff} \quad C \| Auth \text{ was a response of } \mathcal{E}, 0 \text{ else}$$

where $C$ is the concatenation of the queried table entries. Reusing $B_{A,\Psi}$ we can reduce Equation (3) to inequality (11) by simulating the distinguishability with oracles $(\mathcal{E}_{\text{Fun}}, \mathcal{V}_{\text{Fun}})$ and $(\mathcal{E}_{\text{Fun}}, \mathcal{V}^*)$. We call this advantage IND-VERF, since it measures the indistinguishability between the two verifiers.

**Setup of $(B_{A,\Psi})_p$:**

ENCRYPT STATE TABLE ENTRIES:
  **When** $B_{A,\Psi}$ makes a query $M$ to oracle $\mathcal{E}_{\text{Fun}}$ **do**
    $(C, Auth) \leftarrow \mathcal{E}_{\text{Fun}}(M)$
    $B_{A,\Psi} \Leftarrow (C, Auth)$

**Simulation of Oracle $\mathcal{R}$:**

STATE AUTHENTICATION:
  **for** $i \leftarrow 1$ **to** $q_v$ **do**
    **When** $B_{A,\Psi}$ makes a query $(C', Auth')$ to oracle ($\mathcal{V}_{\text{Fun}}$ or $\mathcal{V}^*$) **do**
      **if** $(C', Auth') \neq (C, Auth)$ **then**
        $B_{A,\Psi} \Leftarrow 0$
      **else** $B_{A,\Psi} \Leftarrow 1$
  $b' \Leftarrow B_{A,\Psi}$
  **Return** $b'$

Figure 5: Adversary $(B_{A,\Psi})_p$

Therefore

$$\begin{aligned}
&\left|\Pr[A^{\mathcal{R}_{\text{Fun}}}(\mathcal{O}^{\mathcal{R}_{\text{Fun}}}(\Psi), z) = 1] - \Pr[A^{\mathcal{R}^*_{\text{Fun}}}(\mathcal{O}^{\mathcal{R}^*_{\text{Fun}}}(\Psi), z) = 1]\right| \\
&\quad = \left|\Pr[B^{\mathcal{E}_{\text{Fun}},\mathcal{V}_{\text{Fun}}}_{A,\Psi}(z) = 1] - \Pr[B^{\mathcal{E}_{\text{Fun}},\mathcal{V}^*}_{A,\Psi}(z) = 1]\right| \\
&\quad = \mathbf{Adv}^{\text{ind-verf}}_{\mathcal{SE}_{\text{Fun}},B_{A,\Psi}}(k, q_e, q_v, \eta_e, \eta_v, z). \qquad (11)
\end{aligned}$$

with $q_e = 1$ denoting the number of encryption queries and $\eta_e = \eta_v - 1 = |E(\Psi)|$ the maximum number of $k$-bit blocks each encryption or verification query may have. We claim this advantage is bounded above by the INT-CTXT-$m$ security of $\mathcal{SE}_{\text{Fun}}$. See Appendix A.1 for more details on the security definition of INT-CTXT-$m$.

**Claim 1** $\mathbf{Adv}^{\text{ind-verf}}_{\mathcal{SE}_{\text{Fun}},B_{A,\Psi}}(k, q_e, q_v, \eta_e, \eta_v, z) \leq \mathbf{Adv}^{\text{int-ctxt-m}}_{\mathcal{SE}_{\text{Fun}},(B_{A,\Psi})_{\text{ctxt}}}(k, q_e, q_v, \eta_e, \eta_v, z)$

**Proof:** We will assume throughout the rest of this claim that $q_e = 1$ and $\eta_e = \eta_v - 1 = |E(\Psi)|$. To simplify the notation, we omit writing the variables $q_e, \eta_e$, and $\eta_v$. Let $E$ be the event (over the randomness of Fun and $A$) that $B_{A,\Psi}$ submits at least one ciphertext authentication pair that passes verification (after at most $q_v$ distinct **State_Update** queries) and was never a response from $\mathcal{E}_{\text{Fun}}$. In Figure 5 we define a new adversary $(B_{A,\Psi})_p$ that simulates $B_{A,\Psi}$'s interaction with the verifier $\mathcal{V}^*$. Given the event $\overline{E}$ it follows that both $(B_{A,\Psi})_p$ and $B_{A,\Psi}$ return the same distinguishing bit $b'$. Therefore

$$\begin{aligned}
\Pr[\, b = b' \leftarrow B_{A,\Psi} \wedge \overline{E}\,] &= \Pr[\, b = b' \leftarrow B_{A,\Psi} \mid \overline{E}\,] \cdot \Pr[\,\overline{E}\,] \\
&= \Pr[\, b = b' \leftarrow (B_{A,\Psi})_p \mid \overline{E}\,] \cdot \Pr[\,\overline{E}\,] \\
&\leq \Pr[\, b = b' \leftarrow (B_{A,\Psi})_p\,] \\
&= \frac{1}{2}\mathbf{Adv}^{\text{ind-verf}}_{\mathcal{SE}_{\text{Fun}},(B_{A,\Psi})_p}(k, q_v, z) + \frac{1}{2}.
\end{aligned}$$

But $\mathbf{Adv}^{\text{ind-verf}}_{\mathcal{SE}_{\text{Fun}},(B_{A,\Psi})_p}(k,q_v,z)$ must be equal to $\left|\Pr[B^{\mathcal{E}_{\text{Fun}},\mathcal{V}^*}_{A,\Psi}(z)=1]-\Pr[B^{\mathcal{E}_{\text{Fun}},\mathcal{V}^*}_{A,\Psi}(z)=1]\right|$ which is 0. Hence

$$\begin{aligned}\frac{1}{2}\mathbf{Adv}^{\text{ind-verf}}_{\mathcal{SE}_{\text{Fun}},B_{A,\Psi}}(k,q_v,z)+\frac{1}{2} &= \Pr[\,b=b'\leftarrow B_{A,\Psi}]\\ &= \Pr[\,b=b'\leftarrow B_{A,\Psi}\wedge E\,]+\Pr[\,b=b'\leftarrow B_{A,\Psi}\wedge\overline{E}\,]\\ &\leq \Pr[\,E\,]+\Pr[\,b=b'\leftarrow (B_{A,\Psi})_p\,]\\ &= \frac{1}{2}\Pr[\,E\mid b=0]+\frac{1}{2}\mathbf{Adv}^{\text{ind-verf}}_{\mathcal{SE}_{\text{Fun}},(B_{A,\Psi})_p}(k,q_v,z)+\frac{1}{2}\\ &= \frac{1}{2}\mathbf{Adv}^{\text{int-ctxt-m}}_{\mathcal{SE}_{\text{Fun}},(B_{A,\Psi})_{\text{ctxt}}}(k,q_v,z)+\frac{1}{2}.\end{aligned}$$

and our claim follows. □

Now that we have bounded Equation (3) by the INT-CTXT-$m$ security of $\mathcal{SE}_{\text{Fun}}$ we are now ready to move onto Equation (4).

In Equation (4) we measure the chosen plaintext distinguishability between encrypting with either $\mathcal{E}_{\text{Fun}}$ or $\mathcal{E}_{\text{Rand}}$, where $\mathcal{E}_{\text{Rand}}(M)$ is a random string of length $|M|$. The oracles $\mathcal{R}^*_{\text{Rand}}$ and $\mathcal{R}^*_{\text{Fun}}$ are identical except for their calls to $\mathcal{E}_{\text{Fun}}$ or $\mathcal{E}_{\text{Rand}}$. As before we will use the $*$ in $\mathcal{R}^*_{\text{Rand}}$ to denote that verifier $\mathcal{V}^*$ is used. We define $B^*_{A,\Psi}$ to be the algorithm $B_{A,\Psi}$ that uses $\mathcal{V}^*$ as its verifier (which can be easily *simulated* given the output of $\mathcal{E}$). Therefore Equation (4) reduces to inequality (12)

$$\begin{aligned}&\left|\Pr[A^{\mathcal{R}^*_{\text{Fun}}}(\mathcal{O}^{\mathcal{R}^*_{\text{Fun}}}(\Psi),z)=1]-\Pr[A^{\mathcal{R}^*_{\text{Rand}}}(\mathcal{O}^{\mathcal{R}^*_{\text{Rand}}}(\Psi),z)=1]\right|\\ &\quad=\left|\Pr[B^{*\,\mathcal{E}_{\text{Fun}}}_{A,\Psi}(z)=1]-\Pr[B^{*\,\mathcal{E}_{\text{Rand}}}_{A,\Psi}(z)=1]\right|\\ &\quad=\mathbf{Adv}^{\text{ind\$-cpa}}_{\mathcal{SE}_{\text{Fun}},B^*_{A,\Psi}}(k,q_e,\eta_e,z).\end{aligned}\tag{12}$$

with $q_e=1$ and $\eta_e=|E(\Psi)|$.

In the final Equation (5), we introduce the simulator $S$, which as you recall has only black box access to $\Psi$. In order for $S$ to properly simulate $A$'s view, it needs to know the number of edges $|E(\Psi)|$. This can be easily extracted knowing just the size of $M_\Psi$ based on our encoding. Given the number of edges $|E(\Psi)|$, $S$ can easily simulate $A$'s view of the obfuscated code by giving $A$ a copy of $\mathcal{O}(|E(\Psi)|,\mathbb{T}_C,Auth)$, where $\mathbb{T}_C$ is a random table of the appropriate size (dependent on $|E(\Psi)|$ and $k$) and $Auth$ a $k$-bit random string. Using its oracle access to $\Psi$, $S$ can simulate $A$'s interaction with $\mathcal{R}^*_{\text{Rand}}$ using the values $|E(\Psi)|,\mathbb{T}_C$, and $Auth$. Therefore, the entire simulation, which we denote by $S_A$, consists of passing $A$ the obfuscated code $\mathcal{O}(|E(\Psi)|,\mathbb{T}_C,Auth)$ and simulating the interaction between $\mathcal{R}^*_{\text{Rand}}$ and $A$ using oracle $\Psi$. The full description of simulator $S_A$ is given in Figure 6.

To help with the analysis, we model adversary $A^{\mathcal{R}^*_{\text{Rand}}}(\mathcal{O}^{\mathcal{R}^*_{\text{Rand}}}(\Psi),z)$ as we did in Equation (4) by replacing it with $B^{*\,\mathcal{E}_{\text{Rand}}}_{A,\Psi}(z)$. From this we have $\Pr[A^{\mathcal{R}^*_{\text{Rand}}}(\mathcal{O}^{\mathcal{R}^*_{\text{Rand}}}(\Psi),z)=1]=\Pr[B^{*\,\mathcal{E}_{\text{Rand}}}_{A,\Psi}(z)=1]$. Notice that during the **State_Update** phase of $B^*_{A,\Psi}$, in order for the final query to reach Update Oracle State and return an output other than *auth_fail*, $\mathcal{R}^*_{\text{Rand}}$ must pass the verifier $\mathcal{V}^*$. This implies that the adversary submits the table $\mathbb{T}_C$ free of modifications. Hence the operations under Compare Table Entries may be completely replaced with a simulated oracle call to the DFA in much the same way simulator $S_A$ does. Replacing this code, we obtain a new $B^{*\,\prime}_{A,\Psi}$ which

**Setup of $S_A$:**

INPUT: $1^k, 1^{|E(\Psi)|}, z$

GENERATE STATE TABLE:
  **for** $s \leftarrow 0$ **to** $|E(\Psi)| - 1$ **do**
    $\mathbb{T}_{state}[s] \leftarrow 0^k$

ENCRYPT STATE TABLE ENTRIES:
  **Query** $\mathcal{E}_{\text{Rand}}$ with $\mathbb{T}_{state}$
  $(\mathbb{T}_C, Auth) \Leftarrow \mathcal{E}_{\text{Rand}}(\mathbb{T}_{state})$

  $A \Leftarrow \mathcal{O}(|E(\Psi)|, \mathbb{T}_C, Auth), z$

**Simulation of Oracle $\mathcal{R}^*_{\text{Rand}}$:**

INPUT: $|E(\Psi)|, \mathbb{T}_C, Auth$

INITIALIZATION:
  $acpt \leftarrow \perp$
  $temp_\alpha \leftarrow \perp$
  $flag_{auth} \leftarrow \perp$
  $C \leftarrow \perp$
  $s \leftarrow \perp$
  $State \leftarrow$ **Transition_Query**

**Case**$(State)$
**Transition_Query:**
  **When** $A$ makes a query $\alpha$ **do**
    $temp_\alpha \leftarrow \alpha$
    $flag_{auth} \leftarrow false$
    $C \leftarrow \perp$
    $s \leftarrow 0$
    $State \leftarrow$ **State_Update**

**State_Update:**
  **When** $A$ makes a query $\mathbb{T}'_C[s]$ **or**
    $\mathbb{T}'_C[s] \| Auth'$ **do**

  STATE AUTHENTICATION:
    $C \leftarrow C \| \mathbb{T}'_C[s]$
    **if** $\mathbb{T}'_C[s] \neq \mathbb{T}_C[s]$ **or** $(s = |E(\Psi)| - 1$ **and**
      $Auth' \neq Auth)$ **then**
      $flag_{auth} \leftarrow true$
    **if** $s = |E(\Psi)| - 1$ **and** $flag_{auth} = true$ **then**
      **Query** $\mathcal{V}^*$ with $(C, Auth)$
      **if** $0 \Leftarrow \mathcal{V}^*(C, Auth)$ **then**
        $A \Leftarrow auth\_fail$
        $State \leftarrow$ **Transition_Query**

  QUERY DFA ORACLE:
    **if** $s = |E(\Psi)| - 1$ **then**
      **Query oracle** $\Psi$ with $temp_\alpha$
      $acpt \leftarrow \Psi(temp_\alpha)$

  UPDATE ORACLE STATE:
    **if** $s = |E(\Psi)| - 1$ **then**
      $A \Leftarrow acpt$
      $State \leftarrow$ **Transition_Query**
    $s \leftarrow s + 1$

Figure 6: Simulator $S_A$.

is functionally equivalent to $B^*_{A,\Psi}$. Since the variables *current_state* and $temp_{cs}$ are no longer needed as they are used in the simulation of oracle $\Psi$, we can remove them. Finally, observe that an oracle call to $\mathcal{E}_{\text{Rand}}$ in ENCRYPT STATE TABLE ENTRIES returns random strings regardless of the particular input. Therefore encrypting with the real state table $\mathbb{T}_{state}$ or one containing all zeroes provides a random output that is of the same size. Hence it follows ${B^*_{A,\Psi}}'$ and $S_A$ have a distinguishability of 0. Thus

$$\begin{aligned}
&\left|\Pr[A^{\mathcal{R}^*_{\text{Rand}}}(\mathcal{O}^{\mathcal{R}^*_{\text{Rand}}}(\Psi), z) = 1] - \Pr[S_A^{\Psi}(1^{|E(\Psi)|}, z) = 1]\right| \\
&\quad = \left|\Pr[{B^*_{A,\Psi}}^{\mathcal{E}_{\text{Rand}}\prime}(z) = 1] - \Pr[S_A^{\Psi}(1^{|E(\Psi)|}, z) = 1]\right| \\
&\quad = 0.
\end{aligned}$$

Using the bounds derived in Appendix A with $q_e = 1$ and $\eta_e = \eta_v - 1 = |E(\Psi)|$, we have the

following result

$$\begin{aligned}
&\left|\Pr[A^{\mathcal{R}_{F_K}}(\mathcal{O}^{\mathcal{R}_{F_K}}(\Psi), z) = 1] - \Pr[S^{\Psi}(1^{|E(\Psi)|}, z) = 1]\right| \qquad (13)\\
&\quad \leq \mathbf{Adv}^{\text{prf-nu}}_{F_K}(k, (q_v + 2)|E(\Psi)| + 1)\\
&\qquad\qquad + \mathbf{Adv}^{\text{int-ctxt-m}}_{\mathcal{SE}_{\text{Fun}},(B_{A,\Psi})_{\text{ctxt}}}(k, q_e, q_v, \eta_e, \eta_v, z) + \mathbf{Adv}^{\text{ind\$-cpa}}_{\mathcal{SE}_{\text{Fun}},B^*_{A,\Psi}}(k, q_e, \eta_e, z)\\
&\quad \leq \mathbf{Adv}^{\text{prf-nu}}_{F_K}(k, (q_v + 2)|E(\Psi)| + 1)\\
&\qquad\qquad + q_v(|E(\Psi)|^2 + |E(\Psi)|)2^{-k} + \frac{1}{2}(3|E(\Psi)|^2 + |E(\Psi)|)2^{-k}.
\end{aligned}$$

□

The amount of persistent state needed to obfuscate the DFA in the above Proposition is in fact quite small. In the next Proposition we show that we need at most $O(k)$-bits. This is especially ideal if the oracle is implemented on a computationally limited device with a minimal amount of tamper protection.

**Corollary 1** *If non-uniformly strong one-way functions exist, then non-resettable DFAs are obfuscatable with respect to oracle machines with small internal state.*

**Proof:** In Proposition 2 we used the Goldreich et al. construction in [14] to generate a pseudorandom function that is secure against non-uniform PPT adversaries. The key generated for this construction is the same size as the security parameter $k$. But this implies that the size of the oracle's internal state is no more than $O(\log|State(\Psi)| + \log|\Sigma| + k) = O(k)$, following our definition of $\mathcal{F}_k$. □

### 3.1 Composition of Obfuscations

So far, we have looked at the case of DFA obfuscation in a stand-alone setting, where the obfuscated code is operating in isolation. Suppose now we allow multiple obfuscations to execute alongside one another, all sharing the same oracle as shown in Figure 7. If we compose obfuscations in such a manner, is the resulting scheme any less secure? That is, does running multiple obfuscations provide any more information that couldn't otherwise be efficiently extracted by running their respective black boxes? Using a simple modification to the obfuscation algorithm presented earlier, we show that it is possible to securely compose obfuscations in this manner.

We model the composed DFA obfuscations as a system of ITMs whose communication tapes are connected via a polynomial-time computable control function. The control function interfaces with the oracle's input and output communication tapes and delegates the order in which messages are sent to the oracle. In practice, the control function may implement a quality of service scheduling algorithm that gives certain DFAs a higher priority over others.

The notion of composition we use here is similar in flavor to the one used in secure multi-party protocols. While we don't completely generalize our security claims to the protocol framework, which includes an *environment* distinguisher that distinguishes between a real protocol execution from an simulated ideal process (i.e. black box access), the proofs can be modified to this case (since all of the reductions use the adversary as an oracle). Unfortunately, even with these modifications,

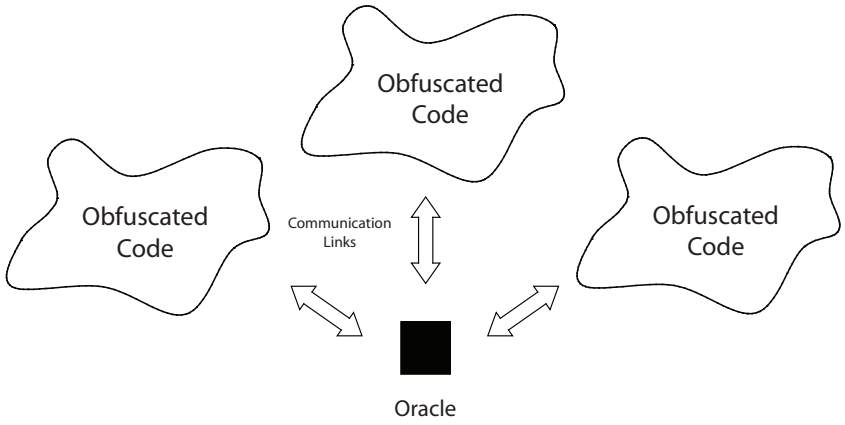


Figure 7: Composition of obfuscations w.r.t. single oracle

general composition cannot be maintained because a symmetric key is used. Our composition assumptions are stated below. For a more thorough introduction to the taxonomy of composition see [9].

**Concurrent Composition.** Any interleaving of messages to the oracle is allowed. Multiple DFA executions operate independently of one another and submit messages to the oracle at their own discretion.

**Adaptively Chosen Inputs.** The inputs into each DFA execution are determined adaptively by the environment. No assumptions are placed on the inputs.

**Self-composition.** The number of DFA executions is fixed in advance, but may be chosen arbitrarily from the same family $\mathcal{F}_k$ for a given security parameter $k$.

Using the definitions above, we now present the main composition result.

**Proposition 3** *If non-uniformly strong one-way functions exist, then there exists a DFA obfuscator that remains secure under concurrent self-composition with adaptively chosen inputs.*

**Proof:** The DFA obfuscator used in Proposition 2 can easily be modified to account for composition. Let $\{M_{\Psi_i}\}_{i=1,\ldots,t}$ be a finite family of DFAs in $\mathcal{F}_k$ with the same encoding scheme as described in Section 3. Our goal is to show that the following inequality is negligible:

$$\Big|\Pr[A^{\mathcal{R}_{F_K}}(\mathcal{O}^{\mathcal{R}_{F_K}}(\Psi_1),\ldots,\mathcal{O}^{\mathcal{R}_{F_K}}(\Psi_t),z)=1] - \Pr[S^{\Psi_1,\ldots,\Psi_t}(1^{|E(\Psi_1)|},\ldots,1^{|E(\Psi_t)|},z)=1]\Big|. \quad (14)$$

To make sure the messages sent between the oracle and the obfuscated DFAs are properly routed, we assign a unique *ID* to each of them. This allows the oracle to distinguish the messages sent from each party. The following changes were made to the obfuscation algorithms **Setup**, $\mathcal{O}$, and $\mathcal{R}$ in Figure 2:

- **Setup**$(M_{\Psi_1}, \ldots, M_{\Psi_t}, k)$
  - The scheme $\mathcal{E}_{F_K}$ under ENCRYPT STATE TABLE ENTRIES is replaced with the encryption scheme in Figure 10.
  - A unique $ID_i$ is assigned to each $M_{\Psi_i}$, corresponding to the $IV$ used for encryption.
- $\mathcal{O}(ID_i, |Table|_i^*, \mathbb{T}_{C,i}^*, Auth_i^*), i = 1, \ldots, t$
  - All communications are prefixed with the DFA's unique $ID$.
  - If a message is received with an $ID$ different than it's own, it is ignored.
- $\mathcal{R}(K, ID_1, |m|_1^*, |Table|_1^*, \ldots, ID_t, |m|_t^*, |Table|_t^*)$
  - Each $\mathcal{O}(ID_i, |Table|_i^*, \mathbb{T}_{C,i}^*, Auth_i^*)$ is assigned it's own set of persistent variables $|Table|_{ID_i}$, $|m|_{ID_i}$, $acpt_{ID_i}$, $current\_state_{ID_i}$, $Auth'_{ID_i}$, $temp_{\alpha,ID_i}$, $temp_{cs,ID_i}$, $s_{ID_i}$, and $State_{ID_i}$.
  - All outgoing messages are prefixed with the input message $ID$.
  - If an incoming message uses an unrecognized $ID$, an $ID \| invalid\_id$ message is returned.
  - The assignment of $X_1$ under **Transition_Query** is changed to $X_1 \leftarrow ID \| 1^{k-|ID|}$, $|ID| < k$.
  - The assignment of $X_0$ under COMPARE TABLE ENTRIES is changed to $X_0 \leftarrow ID \| s \| 0$.

We begin our analysis by breaking up inequality (14) into four separate problems in much the same way as we did in Proposition 2. The oracles $\mathcal{R}_{\text{Fun}}$, $\mathcal{R}_{\text{Fun}}^*$, and $\mathcal{R}_{\text{Rand}}^*$ are reused with the above $ID$ modifications. Since obfuscations may operate concurrently, we must show that this additional capability does not give an adversary a non-negligible advantage. In our particular case, concurrency implies only that messages are interleaved, since a single oracle can process messages only sequentially. In Proposition 2, the adversary $B_{A,\Psi}$ in Figure 4 was able to assemble a chain of **State_Update** queries to construct a single verification query. This was easily achieved since only a single DFA obfuscation was communicating with the oracle at a time. We would like to use this same basic idea to help us here; unfortunately things are more complicated since messages are now interleaved. To mitigate this issue, we create an adversary $A'$ (using $A$ as a subprotocol) that untangles the messages and resubmits them to the oracle in an orderly fashion. A description of adversary $A'$ is given in Figure 8.

Since $A$ receives an output message from the oracle only when an obfuscated DFA has submitted its final **State_Update** query, we can simulate the oracle's output by holding back all of $A$'s queries until a complete chain of **State_Update** queries have been submitted. Therefore, we can untangle $A$'s queries and resubmit them in the following order $(\text{Transition\_Query}_{ID_{i_1}}, \text{State\_Update}_{ID_{i_1}})$ $, \ldots, (\text{Transition\_Query}_{ID_{i_m}}, \text{State\_Update}_{ID_{i_m}})$[6]. Hence it follows that

$$\begin{aligned}\Pr[A^{\mathcal{R}_{F_K}}(\mathcal{O}^{\mathcal{R}_{F_K}}(\Psi_1), \ldots, \mathcal{O}^{\mathcal{R}_{F_K}}(\Psi_t), z) = 1] \\ = \Pr[A'^{\mathcal{R}_{F_K}}(\mathcal{O}^{\mathcal{R}_{F_K}}(\Psi_1), \ldots, \mathcal{O}^{\mathcal{R}_{F_K}}(\Psi_t), z) = 1].\end{aligned}$$

Throughout the rest of the proof we will denote adversary $A'$ as $A$ and assume $A$ submits oracle queries only as described above.

[6] $(\text{Transition\_Query}_{ID_i}, \text{State\_Update}_{ID_i})$ denotes the single **Transition_Query** and complete chain of **State_Update** queries made by $ID_i$.

**Adversary $A'$:**

INPUT: $\mathcal{O}^{\mathcal{R}_{F_K}}(\Psi_1), \ldots, \mathcal{O}^{\mathcal{R}_{F_K}}(\Psi_t)$

EXTRACT TABLE SIZE:
  **for** $i \leftarrow 1$ **to** $t$ **do**
    $|Table|_{ID_i} \leftarrow$ EXTRACTTABLESIZE$(\mathcal{O}^{\mathcal{R}_{F_K}}(\Psi_i))$

REORDER AND RESUBMIT:
  **When** $A$ makes a **Transition_Query** query $ID_i \| \alpha$ **do**
    $temp_{\alpha,ID_i} \leftarrow \alpha$
    $s_{ID_i} \leftarrow 0$
  **When** $A$ makes a **State_Update** query $ID_i \| \mathbb{T}'_C[s]$ **or** $ID_i \| \mathbb{T}'_C[s] \| Auth'_{ID_i}$ **do**
    $\mathbb{T}_{ID_i}[s_{ID_i}] \leftarrow \mathbb{T}'_C[s]$
    **if** $s_{ID_i} = |Table|_{ID_i} - 1$ **then**
      **Query oracle** $\mathcal{R}_{F_K}$ with $ID_i \| temp_{\alpha,ID_i}$
      **for** $s \leftarrow 0$ **to** $|Table|_{ID_i} - 1$ **do**
        **if** $s \neq |Table|_{ID_i} - 1$ **then**
          **Query oracle** $\mathcal{R}_{F_K}$ with $ID_i \| \mathbb{T}_{ID_i}[s]$
        **if** $s = |Table|_{ID_i} - 1$ **then**
          **Query oracle** $\mathcal{R}_{F_K}$ with $ID_i \| \mathbb{T}_{ID_i}[s] \| Auth'_{ID_i}$
          $A \Leftarrow$ Oracle's output
    $s_{ID_i} \leftarrow s_{ID_i} + 1$

  $b' \Leftarrow A$
  **Return** $b'$

Figure 8: Adversary $A'$

Now that the oracle messages have been untangled, we can use adversary $B_{A,\Psi}$ to help complete our analysis. To account for the multiple DFA obfuscations, we relabel $B_{A,\Psi}$ as $B_{A,(\Psi_1,\ldots,\Psi_t)}$ and make the following modifications in Figure 4:

- Under GENERATE STATE TABLE replace the existing code with:
  **for** $i \leftarrow 1$ **to** $t$ **do**
    $(|m|_i, |Table|_i, \mathbb{T}_{state,i}) \leftarrow$ STATETABLE$(\Psi_i)$

- Under ENCRYPT STATE TABLE ENTRIES replace the existing code with:
  **for** $i \leftarrow 1$ **to** $t$ **do**
    **Query oracle** $\mathcal{E}$ with $\mathbb{T}_{state,i}$
    $(ID_i, \mathbb{T}_{C,i}, Auth_i) \Leftarrow \mathcal{E}(\mathbb{T}_{state,i})$

  $A \Leftarrow \mathcal{O}(ID_i, |Table|_i, \mathbb{T}_{C,i}, Auth_i), \ldots, \mathcal{O}(ID_t, |Table|_t, \mathbb{T}_{C,t}, Auth_t), z$

- Under **Simulation of Oracle** $\mathcal{R}$: INPUT include the inputs $ID_i$, $|m|_i$, $|Table|_i$, $\mathbb{T}_{state,i}$, $\mathbb{T}_{C,i}$, $Auth_i$ for $i = 1, \ldots t$.

- Under **Simulation of Oracle** $\mathcal{R}$: INITIALIZATION assign each $ID_i$ a unique copy of the variables listed.

- Modify all incoming and outgoing messages to account for $ID$s.

Following Proposition 2, we replace every oracle call made to $\mathcal{E}$ and $\mathcal{V}$ in $B_{A,(\Psi_1,\ldots,\Psi_t)}$with multiple calls to either $F_K$ or Fun and call this simulation $B_{A,(\Psi_1,\ldots,\Psi_t)}{}'$. Therefore, given that adversary $A$ makes no more than $q_v$ distinct **State_Update** queries, it follows that the total number of queries made to $F_K$ or Fun by $B_{A,(\Psi_1,\ldots,\Psi_t)}{}'$ is no more than $(q_v + 2t)\max_i |E(\Psi_i)| + t$. Hence

$$\begin{aligned}
&\Big|\Pr[A^{\mathcal{R}_{F_K}}(\mathcal{O}^{\mathcal{R}_{F_K}}(\Psi_1),\ldots,\mathcal{O}^{\mathcal{R}_{F_K}}(\Psi_t),z)=1] \\
&\qquad\qquad -\Pr[A^{\mathcal{R}_{\mathrm{Fun}}}(\mathcal{O}^{\mathcal{R}_{\mathrm{Fun}}}(\Psi_1),\ldots,\mathcal{O}^{\mathcal{R}_{\mathrm{Fun}}}(\Psi_t),z)=1]\Big| \\
&= \Big|\Pr[B^{\mathcal{E}_{F_K},\mathcal{V}_{F_K}}_{A,(\Psi_1,\ldots,\Psi_t)}(z)=1] - \Pr[B^{\mathcal{E}_{\mathrm{Fun}},\mathcal{V}_{\mathrm{Fun}}}_{A,(\Psi_1,\ldots,\Psi_t)}(z)=1]\Big| \\
&= \Big|\Pr[B^{F_K}_{A,(\Psi_1,\ldots,\Psi_t)}{}'(z)=1] - \Pr[B^{\mathrm{Fun}}_{A,(\Psi_1,\ldots,\Psi_t)}{}'(z)=1]\Big| \\
&= \mathbf{Adv}^{\mathrm{prf}}_{F_K,B_{A,(\Psi_1,\ldots,\Psi_t)}{}'}(k,z) \\
&\leq \mathbf{Adv}^{\text{prf-nu}}_{F_K}(k,(q_v+2t)\max_i |E(\Psi_i)|+t).
\end{aligned}$$

Similarly it follows that

$$\begin{aligned}
&\Big|\Pr[A^{\mathcal{R}_{\mathrm{Fun}}}(\mathcal{O}^{\mathcal{R}_{\mathrm{Fun}}}(\Psi_1),\ldots\mathcal{O}^{\mathcal{R}_{\mathrm{Fun}}}(\Psi_t),,z)=1] \\
&\qquad\qquad -\Pr[A^{\mathcal{R}^*_{\mathrm{Fun}}}(\mathcal{O}^{\mathcal{R}^*_{\mathrm{Fun}}}(\Psi_1),\ldots,\mathcal{O}^{\mathcal{R}^*_{\mathrm{Fun}}}(\Psi_t),z)=1]\Big| \\
&= \Big|\Pr[B^{\mathcal{E}_{\mathrm{Fun}},\mathcal{V}_{\mathrm{Fun}}}_{A,(\Psi_1,\ldots,\Psi_t)}(z)=1] - \Pr[B^{\mathcal{E}_{\mathrm{Fun}},\mathcal{V}^*}_{A,(\Psi_1,\ldots,\Psi_t)}(z)=1]\Big| \\
&= \mathbf{Adv}^{\text{ind-verf}}_{\mathcal{SE}_{\mathrm{Fun}},B_{A,(\Psi_1,\ldots,\Psi_t)}}(k,q_e,q_v,\eta_e,\eta_v,z) \\
&\leq \mathbf{Adv}^{\text{int-ctxt-m}}_{\mathcal{SE}_{\mathrm{Fun}},(B_{A,(\Psi_1,\ldots,\Psi_t)})_{\mathrm{ctxt}}}(k,q_e,q_v,\eta_e,\eta_v,z).
\end{aligned}$$

and

$$\begin{aligned}
&\Big|\Pr[A^{\mathcal{R}^*_{\mathrm{Fun}}}(\mathcal{O}^{\mathcal{R}^*_{\mathrm{Fun}}}(\Psi_1),\ldots,\mathcal{O}^{\mathcal{R}^*_{\mathrm{Fun}}}(\Psi_t),z)=1] \\
&\qquad\qquad -\Pr[A^{\mathcal{R}^*_{\mathrm{Rand}}}(\mathcal{O}^{\mathcal{R}^*_{\mathrm{Rand}}}(\Psi_1),\ldots,\mathcal{O}^{\mathcal{R}^*_{\mathrm{Rand}}}(\Psi_t),z)=1]\Big| \\
&= \Big|\Pr[B^{*\,\mathcal{E}_{\mathrm{Fun}}}_{A,(\Psi_1,\ldots,\Psi_t)}(z)=1] - \Pr[B^{*\,\mathcal{E}_{\mathrm{Rand}}}_{A,(\Psi_1,\ldots,\Psi_t)}(z)=1]\Big| \\
&= \mathbf{Adv}^{\text{ind\$-cpa}}_{\mathcal{SE}_{\mathrm{Fun}},B^*_{A,(\Psi_1,\ldots,\Psi_t)}}(k,q_e,\eta_e,z).
\end{aligned}$$

where $q_e = t$ denotes the number of encryption queries made and $\eta_e = \eta_v - 1 = \max_i |E(\Psi_i)|$ the maximum number of $k$-bit blocks each encryption and verification query may have.

Making similar changes to the simulator given in Figure 6 as those made to $B_{A,\Psi}$ above, we may construct our final simulator $S_A^{\Psi_1,\ldots,\Psi_t}$. As before, we let each of the tables $\mathbb{T}_{state,i}$ consist of all zeroes. Therefore, following the arguments made in Proposition 2 we have

$$\begin{aligned}
&\Big|\Pr[A^{\mathcal{R}^*_{\mathrm{Rand}}}(\mathcal{O}^{\mathcal{R}^*_{\mathrm{Rand}}}(\Psi_1),\ldots,\mathcal{O}^{\mathcal{R}^*_{\mathrm{Rand}}}(\Psi_t),z)=1] \\
&\qquad\qquad -\Pr[S_A^{\Psi_1,\ldots,\Psi_t}(1^{|E(\Psi_1)|},\ldots,1^{|E(\Psi_t)|},z)=1]\Big| \\
&= \Big|\Pr[B^{*\,\mathcal{E}_{\mathrm{Rand}}}_{A,(\Psi_1,\ldots,\Psi_t)}{}'(z)=1] - \Pr[S_A^{\Psi_1,\ldots,\Psi_t}(1^{|E(\Psi_1)|},\ldots,1^{|E(\Psi_t)|},z)=1]\Big| \\
&= 0.
\end{aligned}$$

Using the bounds derived in Appendix A with $q_e = t$ and $\eta_e = \eta_v - 1 = \max_i |E(\Psi_i)|$ we have the following result

$$\begin{aligned}
&\Big|\Pr[A^{\mathcal{R}_{F_K}}(\mathcal{O}^{\mathcal{R}_{F_K}}(\Psi_1),\ldots,\mathcal{O}^{\mathcal{R}_{F_K}}(\Psi_t),z)=1] \\
&\qquad\qquad -\Pr[S^{\Psi_1,\ldots,\Psi_t}(1^{|E(\Psi_1)|},\ldots,1^{|E(\Psi_t)|},z)=1]\Big| \\
&\leq \mathbf{Adv}^{\text{prf-nu}}_{F_K}(k,(q_v+2t)\max_i|E(\Psi_i)|+t) \\
&\quad +\mathbf{Adv}^{\text{int-ctxt-m}}_{\mathcal{SE}_{\text{Fun}},(B_{A,(\Psi_1,\ldots,\Psi_t)})_{\text{ctxt}}}(k,q_e,q_v,\eta_e,\eta_v,z) \\
&\quad +\mathbf{Adv}^{\text{ind\$-cpa}}_{\mathcal{SE}_{\text{Fun}},B^*_{A,(\Psi_1,\ldots,\Psi_t)}}(k,q_e,\eta_e,z) \\
&\leq \mathbf{Adv}^{\text{prf-nu}}_{F_K}(k,(q_v+2t)\max_i|E(\Psi_i)|+t) \\
&\quad +\frac{5}{2}q_v(t+1)^2(\max_i|E(\Psi_i)|^2)2^{-k} \\
&\quad +\frac{t^2}{2}(3\max_i|E(\Psi_i)|^2+\max_i|E(\Psi_i)|)2^{-k}.
\end{aligned} \tag{15}$$

which is negligible. □

### 3.2 Beyond DFA Obfuscation

Now that we have shown how to obfuscate DFAs w.r.t. oracle machines, we would like to investigate if these same results can be extended to more complex computational models. Namely, we are interested in determining whether Turing machines are obfuscatable as well. Since a Turing machine is a DFA with access to an infinite tape, we need to figure out how to integrate tapes into our DFA obfuscation techniques. This, as we will show, is quite easy.

We define a *Turing machine* as a machine $\Phi = (Q, \Sigma, \Gamma, \delta, s_0, G)$ with finite input alphabet $\Sigma$ (not including the blank symbol $\sqcup$), finite tape alphabet $\Gamma$ (with $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$), transition function $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$, initial state $s_0 \in Q$, and accepting states $G$. The read and write tape is assumed to be infinite in both the left and right directions. For this discussion we will associate an integer to each cell with the initial cell being 0 and the left and right cells being -1 and 1, respectively.

To obfuscate the Turing machine $\Phi$, we follow a similar approach to that taken in Section 3 and break up the obfuscation protocol into three different states, **User_Input**, **State_Update**, and **Tape_Update**. The **User_Input** stage takes an input $\alpha_0 \ldots \alpha_{t-1}$ provided by the user and submits it to the oracle one symbol at a time. On receipt of each input symbol $\alpha_i$, the oracle concatenates a pointer and cell hit counter giving, $\alpha_i \| ptr \| cell\_hits$. The pointer references the numbered cell the tape symbol resides in while the cell hit counter counts the number of times this cell has been accessed. The oracle generates an encryption and intermediate authentication tag (stored persistently) of the concatenation above for $i = 0, \ldots, t-1$ and sends the resulting encryption back to the user. The authenticated-encryption scheme used here is similar to the one used in the DFA obfuscation protocol with the exception that the final authentication tag is not returned to the user but rather is stored by the oracle in its persistent state. After the last input symbol has been submitted to the oracle, the oracle changes to **State_Update**.

For the **State_Update** stage we follow the same procedure as before for obfuscating a DFA except that we add two additional variables to each DFA table element $s_{cell\_write}$ and $s_{LR}$, giving

| Step | Head Position | Head Movement | Authenticated-Encrypted Tape |
|---|---|---|---|
| 1 | 0 | R | $\alpha_0 \| 0 \| 0$ $\alpha_1 \| 1 \| 0$ |
| 2 | 1 | L | $\alpha_0 \| 0 \| 0$ $\alpha_1 \| 1 \| 0$ $\alpha_2 \| 0 \| 1$ |
| 3 | 0 | L | $\alpha_0 \| 0 \| 0$ $\alpha_1 \| 1 \| 0$ $\alpha_2 \| 0 \| 1$ $\alpha_3 \| 1 \| 1$ |
| 4 | -1 | L | $\alpha_0 \| 0 \| 0$ $\alpha_1 \| 1 \| 0$ $\alpha_2 \| 0 \| 1$ $\alpha_3 \| 1 \| 1$ $\alpha_4 \| 0 \| 2$ |
| 5 | -2 | R | $\alpha_0 \| 0 \| 0$ $\alpha_1 \| 1 \| 0$ $\alpha_2 \| 0 \| 1$ $\alpha_3 \| 1 \| 1$ $\alpha_4 \| 0 \| 2$ $\alpha_5 \| -1 \| 0$ |

Figure 9: **Tape_Update** with input $\alpha_0\alpha_1$.

$s_\alpha \| s_{state} \| s_{\delta(state,\alpha)} \| s_{cell\_write} \| s_{LR} \| s_{acpt}$. The variable $s_{cell\_write}$ stores the tape symbol to be written in the current cell while $s_{LR}$ stores the heads transition (either 1=left or 0=right). In addition, if the authentication fails during this stage rather than going back to **User_Input**, the oracle loops to the beginning of **State_Update**, and the user resubmits the authenticated-encrypted table. If the authentication does pass, then the oracles changes to **Tape_Update**.

During the final stage **Tape_Update**, the user submits the encrypted tape to the oracle one encrypted cell at a time. The oracle decrypts and scans through each cell, looking for the cell that has the correct pointer and largest number of cell hits (which corresponds to the most recent symbol written to that cell). If the oracle does not find the pointer in the authenticated-encrypted tape, then the pointer must be pointing to a previously unaccessed cell and therefore the tape symbol must be blank. In Figure 9 we give an example run of the **Tape_Update** stage over multiple steps. To avoid unnecessary complications, a description of the DFA is not given. In the first step the authenticated-encrypted tape contains just the encrypted cells returned by the **User_Input** stage. The symbol $\alpha_0$ has a pointer 0 since it is in the initial cell position, while $\alpha_1$ has pointer 1 since it is to the right of $\alpha_0$. Both symbols have cell hits of 0 since neither of them have been accessed before. We will assume in this example that the previous state **State_Update** determined that the current cell's symbol is to be replaced by $\alpha_2$ and the tape head would move one position to the right. During the first **Tape_Update**, the oracle decrypts each cell and scans for the current pointer 1 (cell position 0 moved one position to the right) with the largest number of cell hits. It finds $\alpha_1 \| 1 \| 0$ and stores the tape symbol $\alpha_1$ in its persistent state, which will be used by the next DFA **State_Update**. At the end of the scan and authentication check, the oracle encrypts the cell $\alpha_2 \| 0 \| 1$ and updates the authentication tag for the new encrypted tape. The encrypted cell is then returned to the user and the oracle's state changes to **State_Update**. This procedure repeats indefinitely until a DFA accept state has been reached, thus allowing the user to submit a new input. The authenticated-encrypted tape given in step 2 is the end result of **Tape_Update** in step 1. The other steps follow based on the above arguments.

A more complete description of this protocol is given below as well as a list of the primary persistent variables used by the oracle.

**Persistent variables:**

$K$: Encryption key.
$ctr$: Changes after each TM execution. Allows TM to execute multiple times.

*State*: **User_Input**, **State_Update**, **Tape_Update**.
*acpt*: Current state accept status.
*tape_symbol*: Stores current cell's tape symbol.
*ptr*: Current tape position.
*cell_hits*: Number of cell hits for current head position.
$cell\_hits_{temp}$: Number of scanned cell hits for new head position.
$|Tape|$: Current size of tape.
$|Table|$: Size of DFA table.
*LR*: Tape head movement.
*cell_write*: Tape symbol to be written in current cell.
$temp_{cs}$: Temporary storage of current state.
*current_state*: Current state of DFA.
$temp_{\alpha}$: Temporary storage of tape symbol when scanning DFA.
$Auth_{tape}$: Intermediate authentication tag for encrypted tape.
$Auth^{*}_{tape}$: Final authentication tag for encrypted tape.
$Auth_{DFA}$: Intermediate authentication tag for encrypted DFA table.

**TM obfuscation protocol:**

- **Input_Query:**
  1. User transmits input symbols $\alpha_0, \ldots, \alpha_{t-1}$ to oracle with the initial input symbol $\alpha_0$ sent first.
  2. For each received input symbol $\alpha_i$ oracle performs the following:
     - Stores first input symbol $tape\_symbol \leftarrow \alpha_0$ in persistent state.
     - Generates encrypted tape cell $T_i \leftarrow \alpha_i \| i \| 0 \oplus F_K(ctr \| i \| 01)$ and returns to user.
     - Stores intermediate authentication tag $Auth_{tape} \leftarrow F_K(Auth_{tape} \oplus T_i)$ and final tag $Auth^{*}_{tape} \leftarrow F_K(Auth_{tape} \oplus T_{t-1})$ in persistent state (with initial tag $Auth_{tape} \leftarrow F_K(ctr \| 10)$).

     In addition on the final input symbol oracle does the following:
     - Set persistent variables $ptr \leftarrow 0$, $cell\_hits_{temp} \leftarrow 0$, $cell\_hits \leftarrow 1$, $|Tape| \leftarrow t$, and $State \leftarrow$ **State_Update**.
- **State_Update:**
  Following the DFA obfuscator in Section 3
  1. User transmits encrypted DFA table and authentication tag.
  2. Oracle decrypts each table entry and scans for *current_state* and *tape_symbol*.
     - If *current_state* and *tape_symbol* are found then oracle updates persistent variables $ptr \leftarrow ptr + (-1)^{s_{LR}}$, $LR \leftarrow s_{LR}$, $cell\_write \leftarrow s_{cell\_write}$, $acpt \leftarrow s_{acpt}$, and $temp_{cs} \leftarrow s_{\delta(current\_state, tape\_symbol)}$.
     - If DFA table authentication passes then oracle updates persistent variables $current\_state \leftarrow temp_{cs}$ and $cell\_hits_{temp} \leftarrow 0$.
       - If $acpt = 1$ then return *acpt* to user and set $ctr \leftarrow ctr + 1$ and $State \leftarrow$ **Input_Query**, else $State \leftarrow$ **Tape_Update**.

– If DFA table authentication does not pass then return *auth_fail* to user and set *State* $\leftarrow$ **State_Update**. User must retransmit encrypted DFA table and authentication tag.

- **Tape_Update:**

  1. User transmits encrypted tape cells $T_0, \ldots, T_{t-1}$ to oracle.
  2. Oracle decrypts each encrypted cell $T_i$ one at a time and scans for *ptr* and *cell_hits*.
     – If *ptr* matches decrypted $ptr'$ and the number of $cell\_hits_{temp}$ is less than or equal to the decrypted $cell\_hits'$ then update persistent variables $tape\_symbol \leftarrow tape\_symbol'$ and $cell\_hits_{temp} \leftarrow cell\_hits'$.
     – Oracle updates intermediate authentication tag $Auth_{tape} \leftarrow F_K(Auth_{tape} \oplus T_i)$ in persistent state (with initial tag $Auth_{tape} \leftarrow F_K(ctr\|10)$).
     – If tape authentication passes then oracle updates persistent variables in the following order $T_{|Tape|} \leftarrow cell\_write\|ptr - (-1)^{LR}\|cell\_hits \oplus F_K(ctr\|\|Tape|\|\|01)$, $Auth^*_{tape} \leftarrow F_K(Auth^*_{tape} \oplus T_{|Tape|})$, and $|Tape| \leftarrow |Tape| + 1$.
       – If *ptr* matched a decrypted $ptr'$ during tape scan then set $cell\_hits \leftarrow cell\_hits_{temp} + 1$, else set $cell\_hits \leftarrow 0$ and $tape\_symbol \leftarrow \sqcup$.

       Set *State* $\leftarrow$ **State_Update**
     – If tape authentication does not pass then return *auth_fail* to user and set $cell\_hits_{temp} \leftarrow 0$ and *State* $\leftarrow$ **Tape_Update**. User must retransmit encrypted tape.

It is important to observe that the above protocol makes use of several encryption tweaks in the authenticated-encryption scheme for both the DFA table and tape. For the tape encryption we used the two-bit tweak 01 while for the seed feeding the initial authentication we used 10. Similarly for the DFA table we used the tweaks 00 (different than the original tweak) and 11 for the encryption and authentication, respectively. These tweaks guarantee that the inputs into the pseudorandom function are unique with high probability.

Also observe that many of the variables may grow exponentially large (if the TM never halts) over time and may overload the number of bits originally assigned to them, thereby potentially breaking the *approximate functionality* requirement for obfuscation. To work around this technicality, the oracle may release the encryption key once these variables are overloaded. The obfuscated code could then decrypt itself and run in the open. This does not break the virtual black box security requirement since the adversary has a running time polynomial in $k$ and thus will never overload these variables for $k$ sufficiently large.

Following a similar proof as in Proposition 2 we have the following result.

**Proposition 4** *If non-uniformly strong one-way functions exist, then Turing machines are obfuscatable with respect to oracle machines with small internal state.*

## References

[1] D. Angluin "A note on the Number of Queries needed to Identify Regular Languages", *Information and Control*, 51:76-87, 1981.

[2] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, K. Yang "On the (Im)possibility of Obfuscating Programs", *Advances in Cryptology - Crypto 2001*, 1-18, 2001.

[3] M. Bellare, O. Goldreich, A. Mityagin "The Power of Verification in Message Authentication and Authenticated Encryption", *Crypto ePrint Archive*, 2004/309.

[4] M. Bellare, C. Namprempre "Authenticated Encryption: Relations among notions and analysis of the generic composition paradigm", *Advances in Cryptology - Asia Crypt 2000*, Lecture notes in Computer Science Vol. 1976, T. Okamoto ed, Springer Verlag, 2000.

[5] M. Bellare, P. Rogaway "Code-Based Game-Playing Proofs and the Security of Triple Encryption" *Advances in Cryptology - EUROCRYPT 2006*, Vol. 4004, 409-426, Springer-Verlag, May 2006.

[6] R. Best "Microprocessor for Executing Encrypted Programs", US Patent 4,168,396. Issued September 1979.

[7] R. Canetti "Towards Realizing Random Oracles: Hash Functions that Hide all Partial Information", *Advances in Cryptology - Crypto 1997*, 455-469, 1997.

[8] R. Canetti "Universally Composable Security: A New Paradigm for Cryptographic Protocols", In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001*, 136-145 2001.

[9] R. Canetti "Security and Composition of Cryptographic Protocols: A Tutorial", *Crypto ePrint Archive*, 2006/1218.

[10] R. Canetti, D. Micciancio, O. Reingold "Perfect One-way Probabilistic Hash Function", *Proceedings of STOC*, 1998.

[11] C. Colberg, C. Thomborson "Watermarking, Tamper-Proofing, and Obfuscation - Tools for Software Protection", Technical Report TR00-03, Department of Computer Science, University of Arizona, February 2000.

[12] E. M. Gold "Complexity of Automaton Identification from given data", *Information and Control*, 37:302-370, 1978.

[13] O. Goldreich "Towards a Theory of Software Protection and Simulation by Oblivious RAMS", *Proceedings of the seventeenth annual ACM Theory of Computing*, 291-304, 1985.

[14] O. Goldreich, S. Goldwasser, S. Micali, "How to Construct Random Functions", *J. of the ACM*, Vol. 33, 792-807, 1986.

[15] O. Goldreich, S. Micali, A. Wigderson "How to Play any Mental Game - A Completeness Theorem for Protocols with Honest Majority", In *19th STOC*, 218-229, 1987.

[16] O. Goldreich, R. Ostrovsky "Software Protection and Simulation on Oblivious RAMs", *JACM* Vol. 43, No. 3, 431-473, 1996.

[17] S. Goldwasser, S. Micali, C. Rackoff "The Knowledge Complexity of Interactive Proof Systems", *SIAM Journal on Computing*, Vol. 18, No. 1, 186-208, 1989.

[18] S. Goldwasser, G. N. Rothblum "On Best-Possible Obfuscation", *TCC 2007*, 194-213.

[19] S. Goldwasser, Y. Tauman Kalai "On the Impossibility of Obfuscation with Auxiliary Input", *FOCS 2005*, 553-562, Oct. 2005.

[20] S. Hada "Zero-Knowledge and Code Obfuscation", *Advances in Cryptology - Asia Crypt 2000*, 443-457, 2000.

[21] S.T. Kent "Protecting Externally Supplied Software in Small Computers", Ph.D. Thesis, MIT/LCS/TR-255 1980.

[22] B. Lynn, M. Prabhakaran, A. Sahai "Positive Results and Techniques for Obfuscation", *Advances in Cryptology - EUROCRYPT 2004*, Lecture Notes in Computer Science, Springer-Verlag, May 2004.

[23] R. Rivest, R. E. Schapire "Inference of Finite Automata Using Homing Sequences" In *21st ACM Symposium on Theory of Computing*, 411-420, 1989.

[24] N. Varnovsky, V. Zakharov "On the Possibility of Provably Secure Obfuscating Programs" *Perspectives of Systems Informatics, 5th International Andrei Ershov Memorial Conference*, Lecture Notes in Computer Science. Springer-Verlag, July 2003.

[25] H. Wee "On Obfuscating Point Functions", *Crypto ePrint Archive*, 2005/001.

[26] A. Yao "How to Generate and Exchange Secrets", In *27th FOCS*, 162-167,1986.

# A Supplementary Proofs

In this appendix we review the security definitions of INT-CTXT and IND$-CPA and prove the bounds used in inequality 13 and 15. All of the results proven below are based on the authenticated encryption scheme shown in Figure 10. This generalized scheme is used for composing obfuscations, with the DFA's *ID* corresponding to the encryption $IV$. We will assume the $IV$s are fixed in size, with size strictly less than the security parameter $k$.

## A.1 Integrity Awareness

In Proposition 2 and 3 we showed that the distinguishing advantage between the verifiers $\mathcal{V}_{\text{Fun}}$ and $\mathcal{V}^*$ (with the adversary also having access to $\mathcal{E}_{\text{Fun}}$) is bounded above by the strong unforgeability of the ciphertexts. We state the security definition formally below.

**Definition 4 (Integrity Awareness w.r.t. Auxiliary Input):** *Let $\mathcal{SE}_{\text{Fun}}$ be the symmetric encryption scheme in Figure 10 using random functions and $A_{\text{ctxt}}$ a PPT adversary with access to two oracles, $\mathcal{E}_{\text{Fun}}$ and $\mathcal{V}_{\text{Fun}}$. Consider the following experiment with $k \in \mathbb{N}$ and $z \in \{0,1\}^{q(k)}$ for some polynomial $q$*

Algorithm $\mathcal{E}_\rho(M)$

$C \leftarrow \perp$
$IV \leftarrow 0^{|IV|}$, $|IV| < k$
$M_0 \| \ldots \| M_{t-1} \leftarrow M$, $|M_i| = k$
$X_1^{-1} \leftarrow IV \| 1^{k-|IV|}$
$Auth \leftarrow \rho(X_1^{-1})$
**for** $s \leftarrow 0$ **to** $t-1$ **do**
  $X_0^s \leftarrow IV \| s \| 0$
  $Y \leftarrow \rho(X_0^s)$
  $C_s \leftarrow Y \oplus M_s$
  $C \leftarrow C \| C_s$
  $X_1^s \leftarrow Auth \oplus C_s$
  $Auth \leftarrow \rho(X_1^s)$
$\mathsf{size}_C(IV) \leftarrow |C|$
$\mathsf{prev}_{IV} \leftarrow \mathsf{prev}_{IV} \cup \{IV\}$
$IV \leftarrow IV + 1$
**Return** $(IV, C \| Auth)$.

Algorithm $\mathcal{V}_\rho(IV', C' \| Auth')$

${C_0}' \| \ldots \| {C_{t-1}}' \leftarrow C'$
$X_1^{-1} \leftarrow IV' \| 1^{k-|IV|}$
$Auth \leftarrow \rho(X_1^{-1})$
**for** $s \leftarrow 0$ **to** $t-1$ **do**
  ${X_1^s}' \leftarrow Auth \oplus {C_s}'$
  $Auth \leftarrow \rho(X_1^s)$
**if** $Auth = Auth'$ **and**
  $IV' \in \mathsf{prev}_{IV}$ **and**
  $\mathsf{size}_C(IV') = |C'|$ **then**
  **Return** 1, else **Return** 0.

Figure 10: Generalized Encryption and Verification Schemes.

Experiment $\mathbf{Exp}^{\text{int-ctxt-m}}_{\mathcal{SE}_{\text{Fun}}, A_{\text{ctxt}}}(k, z)$
  Fun $\xleftarrow{\$}$ $\mathsf{Fun}(k)$
  If $A_{\text{ctxt}}^{\mathcal{E}_{\text{Fun}}, \mathcal{V}_{\text{Fun}}}(k, z)$ makes a query $C$ to
  the oracle $\mathcal{V}_{\text{Fun}}$ such that
    - $\mathcal{V}_{\text{Fun}}(C) = 1$
    - $C$ was never a response to $\mathcal{E}_{\text{Fun}}$
  then Return 1 else Return 0.

We denote the winning probability in adversary $A_{\text{ctxt}}$ breaking INT-CTXT-$m$ as

$$\mathbf{Adv}^{\text{int-ctxt-m}}_{\mathcal{SE}_{\text{Fun}}, A_{\text{ctxt}}}(k, z) := \Pr[\mathbf{Exp}^{\text{int-ctxt-m}}_{\mathcal{SE}_{\text{Fun}}, A_{\text{ctxt}}}(k, z) = 1]$$

The INT-CTXT-$m$ advantage over all PPT adversaries $A_{\text{ctxt}}$ is defined as the maximum

$$\mathbf{Adv}^{\text{int-ctxt-m}}_{\mathcal{SE}_{\text{Fun}}}(k, q_e, q_v, \eta_e, \eta_v, z) := \max_{A_{\text{ctxt}}} \{\mathbf{Adv}^{\text{int-ctxt-m}}_{\mathcal{SE}_{\text{Fun}}, A_{\text{ctxt}}}(k, z)\}$$

where $q_e$ and $q_v$ denote the maximum number of oracle calls to $\mathcal{E}_{\text{Fun}}$ and $\mathcal{V}_{\text{Fun}}$, while $\eta_e$ and $\eta_v$ denote the maximum number of $k$-bit blocks per encryption and verification query. The scheme $\mathcal{SE}_{\text{Fun}}$ is said to be INT-CTXT-$m$ secure w.r.t. auxiliary input if the advantage $\mathbf{Adv}^{\text{int-ctxt-m}}_{\mathcal{SE}_{\text{Fun}}}$ is negligible over all PPT adversaries (with time-complexity polynomial bounded in $k$) given arbitrary auxiliary input.

In the special case where we allow only a single verification query $q_v = 1$, we define the advantage as INT-CTXT-1. It was shown by Bellare et al. in [3] that if an encryption scheme $\mathcal{SE}$ is INT-CTXT-1 secure (without an auxiliary input), then it is also INT-CTXT-$m$ secure. Adding auxiliary inputs is a trivial modification to the original proof. Since we will be using this result to simplify our analysis, we state it in the following lemma.

**Lemma 1 (INT-CTXT-1 ⇒ INT-CTXT-M [3])** *Let $\mathcal{SE}$ be any symmetric encryption scheme and $z$ any polynomial bounded string in $k$ with $k \geq 1$. Then*

$$\mathbf{Adv}^{\text{int-ctxt-m}}_{\mathcal{SE}}(k, q_e, q_v, \eta_e, \eta_v, z) \leq q_v \cdot \mathbf{Adv}^{\text{int-ctxt-1}}_{\mathcal{SE}}(k, q_e, \eta_e, \eta_v, z)$$

In the following Proposition we prove the scheme in Figure 10 is INT-CTXT-$m$ secure when $q_e = 1$. This result is used to help facilitate the proof in Proposition 2.

**Proposition 5** *Let $\mathcal{SE}_{\text{Fun}}$ be the scheme given in Figure 10 with $IV = \perp$. Let $z$ be any polynomial bounded string in $k$ with $q_e = 1$, $\eta_v = \eta_e + 1$, and $q_v, k \geq 1$. Then*

$$\mathbf{Adv}^{\text{int-ctxt-m}}_{\mathcal{SE}_{\text{Fun}}}(k, q_e, q_v, \eta_e, \eta_v, z) \leq q_v(4\eta_e^2 + \eta_e)2^{-k}$$

**Proof:** To prove the above inequality holds, we will use the game-playing techniques introduced by Bellare and Rogaway in [5]. Our goal is to incrementally construct a chain of games using simple transformation techniques so that the terminal game is bounded above by a negligible factor. To simplify our analysis we use the result of Lemma 1 and derive an upperbound for INT-CTXT-1. Once we have found a bound for INT-CTXT-1, the more general INT-CTXT-$m$ bound will follow. For the sake of this proof, we will also assume that our adversary $A$ is computationally unbounded and therefore deterministic (since it may deterministically choose its queries to maximize its advantage). The only restrictions we place on $A$ is the number of queries it can make.

We begin our analysis by giving a description of game $\mathsf{G1}$ shown in Figure 12. The scheme in $\mathsf{G1}$ is the same encryption scheme shown in Figure 10 with $IV = \perp$. Notice that since we assumed $IV = \perp$ the scheme $\mathcal{SE}_{\text{Fun}}$ is no longer stateful and therefore not IND-CPA secure. Having IND-CPA security is not essential to proving the claim (since $q_e = 1$). Also observe that we removed the checking of $\mathsf{size}_C$ in game $\mathsf{G1}$. We will instead assume without loss of generality that the ciphertext submitted for verification is the same size of the ciphertext returned by the encryption query. Let $\rho$ be a randomly (independent of $z$) chosen function from the set $\mathsf{Fun}(k)$. Observe that game $\mathsf{G1}$ has only two queries in its description: an encryption query and a verification query. The single encryption query ($q_e = 1$) simulates obfuscating a single DFA while the verification query ($q_v = 1$) is the result of restricting our analysis to INT-CTXT-1. Based on the description of game $\mathsf{G1}$ it follows that

$$\mathbf{Adv}^{\text{int-ctxt-1}}_{\mathcal{SE}_{\text{Fun}}}(k, q_e, \eta_e, \eta_v, z) = \Pr[\text{Game } \mathsf{G1} \text{ sets } \mathit{bad}]$$

with $q_e = 1$ and $\eta_e = \eta_v - 1 = t$.

To transform game $\mathsf{G1} \rightarrow \mathsf{G2}$, we add additional settings of *bad* in lines 208, 214, and 224. We also observe that during the second query, the *Auth* value after the first index $i$ where $C_i' \neq C_i$ is just $\rho(X_1^{i-1})$. Therefore, the modifications made in lines 219 through 225 are a direct result of this observation. Since the functionality of game $\mathsf{G1}$ and $\mathsf{G2}$ are equivalent with the exception of additional settings of *bad* it follows that $\Pr[\text{Game } \mathsf{G1}] \leq \Pr[\text{Game } \mathsf{G2}]$.

To go from game $\mathsf{G2} \rightarrow \mathsf{G3}$, we unroll the **for** loops in line 205 and 221 and postpone the recordings of the variable $X_1^s$ in $\text{Dom}(\rho)$. We also swap the assignment of the variable $X_1^s \leftarrow \mathit{Auth} \oplus C_s$ with a random sampling $X_1^s \xleftarrow{\$} \{0,1\}^k$, since the *Auth* variable used in the assignment of $X_1^s$ is randomly sampled during $s - 1$. Finally, the assignments occurring after the setting of $\mathit{bad} \leftarrow \mathit{true}$ are removed. Therefore, the changes made from game $\mathsf{G2}$ to $\mathsf{G3}$ are conservative (i.e. $\Pr[\text{Game } \mathsf{G2}] = \Pr[\text{Game } \mathsf{G3}]$).

Game G1

100 **On first query** $M_0 \| \ldots \| M_{t-1}$
101 $C \leftarrow \perp$
102 $X_1^{-1} \leftarrow 1^k$
103 $Auth \xleftarrow{\$} \{0,1\}^k$
104 $\rho(X_1^{-1}) \leftarrow Auth$
105 **for** $s \leftarrow 0$ **to** $t-1$ **do**
106 $X_0^s \leftarrow s \| 0$
107 $Y \xleftarrow{\$} \{0,1\}^k$
108 **if** $X_0^s \in \mathrm{Dom}(\rho)$ **then** $Y \leftarrow \rho(X_0)$
109 $\rho(X_0^s) \leftarrow Y$
110 $C_s \leftarrow Y \oplus M_s$
111 $C \leftarrow C \| C_s$
112 $X_1^s \leftarrow Auth \oplus C_s$
113 $Auth \xleftarrow{\$} \{0,1\}^k$
114 **if** $X_1^s \in \mathrm{Dom}(\rho)$ **then** $Auth \leftarrow \rho(X_1^s)$
115 $\rho(X_1^s) \leftarrow Auth$
116 **Return** $C \| Auth$

117 **On second query** $C' \| Auth'$
118 $C_0' \| \ldots \| C_{t-1}' \leftarrow C'$
119 $Auth \leftarrow \rho(X_1^{-1})$
120 **for** $s \leftarrow 0$ **to** $t-1$ **do**
121 $X_1^{s\prime} \leftarrow Auth \oplus C_s'$
122 $Auth \xleftarrow{\$} \{0,1\}^k$
123 **if** $X_1^{s\prime} \in \mathrm{Dom}(\rho)$ **then** $Auth \leftarrow \rho(X_1^{s\prime})$
124 $\rho(X_1^{s\prime}) \leftarrow Auth$
125 $b \leftarrow 0$
126 **if** $Auth = Auth'$ **then** $bad \leftarrow true$, $b \leftarrow 1$
127 **Return** $b$

Game G2

200 **On first query** $M_0 \| \ldots \| M_{t-1}$
201 $C \leftarrow \perp$
202 $X_1^{-1} \leftarrow 1^k$
203 $Auth \xleftarrow{\$} \{0,1\}^k$
204 $\rho(X_1^{-1}) \leftarrow Auth$
205 **for** $s \leftarrow 0$ **to** $t-1$ **do**
206 $X_0^s \leftarrow s \| 0$
207 $Y \xleftarrow{\$} \{0,1\}^k$
208 **if** $X_0^s \in \mathrm{Dom}(\rho)$ **then** $bad \leftarrow true$, $Y \leftarrow \rho(X_0^s)$
209 $\rho(X_0^s) \leftarrow Y$
210 $C_s \leftarrow Y \oplus M_s$
211 $C \leftarrow C \| C_s$
212 $X_1^s \leftarrow Auth \oplus C_s$
213 $Auth \xleftarrow{\$} \{0,1\}^k$
214 **if** $X_1^s \in \mathrm{Dom}(\rho)$ **then** $bad \leftarrow true$, $Auth \leftarrow \rho(X_1^s)$
215 $\rho(X_1^s) \leftarrow Auth$
216 **Return** $C \| Auth$

217 **On second query** $C' \| Auth'$
218 $C_0 \| \ldots \| C_{i-1} \| C_i' \| \ldots \| C_{t-1}' \leftarrow C'$
219 $i \leftarrow \min\{s \mid C_s' \neq C_s\}$
220 $Auth \leftarrow \rho(X_1^{i-1})$
221 **for** $s \leftarrow i$ **to** $t-1$ **do**
222 $X_1^{s\prime} \leftarrow Auth \oplus C_s'$
223 $Auth \xleftarrow{\$} \{0,1\}^k$
224 **if** $X_1^{s\prime} \in \mathrm{Dom}(\rho)$ **then** $bad \leftarrow true$, $Auth \leftarrow \rho(X_1^{s\prime})$
225 $\rho(X_1^{s\prime}) \leftarrow Auth$
226 $b \leftarrow 0$
227 **if** $Auth = Auth'$ **then** $bad \leftarrow true$, $b \leftarrow 1$
228 **Return** $b$

Figure 11: INT-CTXT-1 Games G1-G2.

For the final game $\mathsf{G3} \rightarrow \mathsf{G4}$ we begin by first swapping the random-assignment in line 305 with line 308 by replacing $Y \xleftarrow{\$} \{0,1\}^k$ and $C_s \leftarrow Y \oplus M_s$ with $C_s \xleftarrow{\$} \{0,1\}^k$ and $Y \leftarrow C_s \oplus M_s$. Since the variable $Y$ is no longer used, we may eliminate it from the game. Similarly, since the values recorded for $\rho(X_1^s)$ and $\rho(X_0^s)$ are never reused, they may be arbitrarily renamed as defined. The only prerecorded variable that is reused is $X_1^i$ on line 413. Given the above swapping it is easy to see that both $C$ and $Auth$ are random. Using the derandomization technique[7] we may replace them with constants $\mathtt{C} \| \mathtt{Auth}$. Since adversary $A$ is deterministic, there exist queries $\mathtt{M}_0 \| \ldots \| \mathtt{M}_{t-1}$ and $\mathtt{C}' \| \mathtt{Auth}'$ corresponding to output $\mathtt{C} \| \mathtt{Auth}$. By hardwiring these query-responses into game $\mathsf{G4}$, we may bound the probability of setting $bad$ as the maximum over all the possible query-responses

[7]Derandomization Technique: If a game $\mathsf{G}$ chooses a variable $X \xleftarrow{\$} \mathcal{X}$ and never redefines it, we may derandomize the variable by choosing a constant $\mathtt{x}$ to replace it. Given any adversary $A$, it follows that $\Pr[\text{Game } \mathsf{G}_A \text{ sets } bad] \leq \max_{\mathtt{x}} \Pr[\text{Game } \mathsf{G}_A^{\mathtt{x}} \text{ sets } bad]$.

Game G3

300 **On first query** $M_0 \| \ldots \| M_{t-1}$
301 $C \leftarrow \perp$
302 $X_1^{-1} \leftarrow 1^k$
303 **for** $s \leftarrow 0$ **to** $t-1$ **do**
304 $\quad X_0^s \leftarrow s \| 0$
305 $\quad Y \xleftarrow{\$} \{0,1\}^k$
306 $\quad$ **if** $X_0^s \in \mathrm{Dom}(\rho)$ **then** $bad \leftarrow true$
307 $\quad \rho(X_0^s) \leftarrow Y$
308 $\quad C_s \leftarrow Y \oplus M_s$
309 $\quad C \leftarrow C \| C_s$
310 $\quad X_1^s \xleftarrow{\$} \{0,1\}^k$
311 $\quad Auth \leftarrow X_1^s \oplus C_s$
312 $\quad \rho(X_1^{s-1}) \leftarrow Auth$
313 $\quad$ **if** $X_1^s \in \mathrm{Dom}(\rho)$ **then** $bad \leftarrow true$
314 $Auth \xleftarrow{\$} \{0,1\}^k$
315 $\rho(X_1^{t-1}) \leftarrow Auth$
316 **Return** $C \| Auth$

317 **On second query** $C' \| Auth'$
318 $C_0 \| \ldots \| C_{i-1} \| C_i{}' \| \ldots \| C_{t-1}{}' \leftarrow C'$
319 $i \leftarrow \min\{s \mid C_s{}' \neq C_s\}$
320 $Auth \leftarrow \rho(X_1^{i-1}) = X_1^i \oplus C_i$
321 $X_1^{i\prime} \leftarrow Auth \oplus C_i'$
322 **if** $X_1^{i\prime} \in \mathrm{Dom}(\rho)$ **then** $bad \leftarrow true$
323 **if** $i < t-1$ **then**
324 $\quad$ **for** $s \leftarrow i+1$ **to** $t-1$ **do**
325 $\quad\quad X_1^{s\prime} \xleftarrow{\$} \{0,1\}^k$
326 $\quad\quad Auth \leftarrow X_1^{s\prime} \oplus C_s{}'$
327 $\quad\quad \rho(X_1^{s-1\prime}) \leftarrow Auth$
328 $\quad\quad$ **if** $X_1^{s\prime} \in \mathrm{Dom}(\rho)$ **then** $bad \leftarrow true$
329 $Auth \xleftarrow{\$} \{0,1\}^k$
330 $\rho(X_1^{t-1\prime}) \leftarrow Auth$
331 **if** $Auth = Auth'$ **then** $bad \leftarrow true$
332 **Return** 0

Game G4

400 **Given** $\mathtt{M}_0 \| \ldots \| \mathtt{M}_{t-1}$
401 $X_1^{-1} \leftarrow 1^k$
402 **for** $s \leftarrow 0$ **to** $t-1$ **do**
403 $\quad X_0^s \leftarrow s \| 0$
404 $\quad$ **if** $X_0^s \in \mathrm{Dom}(\rho)$ **then** $bad \leftarrow true$
405 $\quad \rho(X_0^s) \leftarrow$ defined
406 $\quad X_1^s \xleftarrow{\$} \{0,1\}^k$
407 $\quad \rho(X_1^{s-1}) \leftarrow$ defined
408 $\quad$ **if** $X_1^s \in \mathrm{Dom}(\rho)$ **then** $bad \leftarrow true$
409 $\rho(X_1^{t-1}) \leftarrow$ defined

410 **Given** $\mathtt{C}' \| \mathtt{Auth}'$
411 $\mathtt{C}_0 \| \ldots \| \mathtt{C}_{i-1} \| \mathtt{C}_i{}' \| \ldots \| \mathtt{C}_{t-1}{}' \leftarrow \mathtt{C}'$
412 $i \leftarrow \min\{s \mid \mathtt{C}_s{}' \neq \mathtt{C}_s\}$
413 $Auth \leftarrow X_1^i \oplus \mathtt{C}_i$
414 $X_1^{i\prime} \leftarrow Auth \oplus \mathtt{C}_i' = X_1^i \oplus \delta$, some $\delta \neq 0$
415 **if** $X_1^{i\prime} \in \mathrm{Dom}(\rho)$ **then** $bad \leftarrow true$
416 **if** $i < t-1$ **then**
417 $\quad$ **for** $s \leftarrow i+1$ **to** $t-1$ **do**
418 $\quad\quad X_1^{s\prime} \xleftarrow{\$} \{0,1\}^k$
419 $\quad\quad \rho(X_1^{s-1\prime}) \leftarrow$ defined
420 $\quad\quad$ **if** $X_1^{s\prime} \in \mathrm{Dom}(\rho)$ **then** $bad \leftarrow true$
421 $Auth \xleftarrow{\$} \{0,1\}^k$
422 $\rho(X_1^{t-1\prime}) \leftarrow$ defined
423 **if** $Auth = \mathtt{Auth}'$ **then** $bad \leftarrow true$

Figure 12: INT-CTXT-1 Games G3-G4.

(thus removing the adaptivity of the adversary). It is not difficult to see that this maximum occurs when $t = \eta_e$, and the adversary submits a $t+1$-block authentication query with the first ciphertext block changed. Since there are $t+1$ non-random variables $X_0^{s=0,\ldots,t-1}, X_1^{-1}$ that do not collide with one another and $2t-1$ independent random variables $X_1^{s=0,\ldots,t-1}, X_1^{s=1,\ldots,t-1\prime}$ with a single dependent random variable $X_1^{0\prime} = X_1^0 \oplus \delta$ some fixed $\delta \neq 0$ recorded in $\mathrm{Dom}(\rho)$, it follows that the setting of *bad* based on these variables is

$$\Pr[\text{Variables in } \mathrm{Dom}(\rho) \text{ set } bad] \leq \left\{ \binom{3t+1}{2} - \binom{t+1}{2} - 1 \right\} 2^{-k}$$

which holds for any computationally unbounded adversary. Therefore, given $q_e = 1$, $\eta_v = \eta_e + 1$, and $\Pr[Auth \text{ sets } bad \text{ in line } 423] = 2^{-k}$ we have

$$\begin{aligned}
\mathbf{Adv}^{\text{int-ctxt-1}}_{\mathcal{SE}_{\text{Fun}}}(k, q_e, \eta_e, \eta_v, z) &\leq \Pr[\text{Game } \mathsf{G4} \text{ sets } bad] \\
&\leq \Pr[\text{Variables in Dom}(\rho) \text{ set } bad] \\
&\qquad + \Pr[Auth \text{ sets } bad \text{ in line } 423] \\
&\leq \left\{ \binom{3\eta_e + 1}{2} - \binom{\eta_e + 1}{2} \right\} 2^{-k} \\
&= (4\eta_e^2 + \eta_e) 2^{-k}.
\end{aligned}$$

□

In the case that $IV \neq \perp$ we may derive a more general result. By letting the $IV$'s represent the identity (which we denote as $ID$s) of each obfuscated DFA instance we may use the following generalization to prove the main composition result in Section 3.1.

**Proposition 6** *Let $\mathcal{SE}_{\text{Fun}}$ be the authenticated encryption scheme given in Figure 10 using random functions and $z$ any polynomial bounded string in $k$ with $q_e, q_v \geq 1$, $\eta_v = \eta_e + 1$, and $k \geq 1$. Then*

$$\mathbf{Adv}^{\text{int-ctxt-m}}_{\mathcal{SE}_{\text{Fun}}}(k, q_e, q_v, \eta_e, \eta_v, z) \leq \frac{5}{2} q_v \eta_e^2 (q_e + 1)^2 2^{-k}$$

**Proof:** To simplify our analysis we will reuse the result of Lemma 1 and derive an upperbound for INT-CTXT-1. Following the description in Figure 10 we modify the encryption scheme in games G1 through G4 (Proposition 5) to include $IV$s. Observe that the verifier shown in Figure 10 only accepts ciphertext queries that contain $IV$s previously returned by $\mathcal{E}_{\text{Fun}}$, such that the length of the new ciphertext match's the length of the original. Therefore an adversary gains no advantage by submitting a ciphertext query that contains an $IV$ never seen before or if the length of the ciphertext submission is different than the length of the original for that particular $IV$. To simplify the game descriptions we assume wlog that an adversary does not make these type of queries.

As in the last Proposition it is easy to see that an adversary maximizes their advantage by submitting encryption queries satisfying the bound $\eta_e$ with a single $\eta_e + 1$-block authentication query with the first ciphertext block changed (may choose any of the past $IV$s). It follows for any fixed chain of queries there are at most $q_e(\eta_e + 1)$ non-random variables $X^{-1}_{1,IV}, X^{s=0,\ldots,\eta_e-1}_{0,IV}$, $IV = 0, \ldots, q_e - 1$ that do not collide with one another and $\eta_e(q_e + 1) - 1$ independent random variables $X^{s=0,\ldots,\eta_e-1}_{1,IV}, X^{s=1,\ldots,\eta_e-1\prime}_{1,IV}$, $IV = 0, \ldots, q_e - 1$ with a single dependent random variable ${X^0_{1,IV}}' = X^0_{1,IV} \oplus \delta$ some fixed $\delta \neq 0$ recorded in Dom($\rho$). It follows that the setting of $bad$ for these variables is bounded above by

$$\begin{aligned}
\Pr[\text{Game } \mathsf{G4} \text{ sets } bad] &\leq \left\{ \binom{q_e(\eta_e + 1) + \eta_e(q_e + 1)}{2} - \binom{q_e(\eta_e + 1)}{2} - 1 \right\} 2^{-k} \\
&= \left\{ \binom{n_e(q_e + 1)}{2} + q_e \eta_e (q_e + 1)(\eta_e + 1) - 1 \right\} 2^{-k} \\
&\leq \left\{ \frac{5}{2} \eta_e^2 (q_e + 1)^2 - 1 \right\} 2^{-k}
\end{aligned}$$

which holds for any computationally unbounded adversary. Therefore, we have

$$\begin{aligned}\mathbf{Adv}^{\text{int-ctxt-m}}_{\mathcal{SE}_{\text{Fun}}}(k, q_e, q_v, \eta_e, \eta_v, z) &\leq \Pr[\text{Game } \mathsf{G4} \text{ sets } bad] \\ &\leq \Pr[\text{Variables in Dom}(\rho) \text{ set } bad] \\ &\quad + \Pr[Auth \text{ sets } bad \text{ in line 423}] \\ &\leq \frac{5}{2}\eta_e^2(q_e+1)^2 2^{-k}.\end{aligned}$$

□

## A.2 Indistinguishable from Random

In Proposition 2 and 3, we measured the indistinguishability between the schemes $\mathcal{E}_{\text{Fun}}$ and $\mathcal{E}_{\text{Rand}}$ under chosen plaintext attacks. The randomized scheme $\mathcal{E}_{\text{Rand}}$ as you recall took any message $M$ that was a multiple of $k$-bits ($k$ the security parameter) say $t$ and returned a random string of $(t+1)k$-bits along with an $IV$. In Proposition 2, $\mathcal{E}_{\text{Fun}}$ does not use an $IV$; therefore, in this case we take $IV = \perp$. Formally we define $\mathcal{E}_{\text{Rand}}$ as

Algorithm $\mathcal{E}_{\text{Rand}}(M)$
$M_0 \| \ldots \| M_{t-1} \leftarrow M$, $|M_i| = k$
$\text{Rand} \xleftarrow{\$} \{0,1\}^{(t+1)k}$
$IV \leftarrow IV + 1$
**Return** $(IV, \text{Rand})$.

For the definition of indistinguishable from random to make sense in our setting, we give the adversary an additional auxiliary input.

**Definition 5 (Indistinguishable from Random):** *Let $\mathcal{SE}_{\text{Fun}}$ be the symmetric encryption scheme in Figure 10 using random functions and $A_{\text{cpa}}$ a PPT adversary with access to two oracles, $\mathcal{E}_{\text{Fun}}$ and $\mathcal{E}_{\text{Rand}}$. Consider the following experiment with $k \in \mathbb{N}$ and $z \in \{0,1\}^{q(k)}$ for some polynomial $q$*

Experiment $\mathbf{Exp}^{\text{ind\$-cpa}}_{\mathcal{SE}_{\text{Fun}}, A_{\text{cpa}}}(k, z)$
$\text{Fun} \xleftarrow{\$} \mathsf{Fun}(k)$
$b \leftarrow A_{\text{cpa}}^{\mathcal{E}_{\text{Fun}}, \mathcal{E}_{\text{Rand}}}$
Return $b$

We denote the winning probability in the adversary breaking IND$-CPA as

$$\mathbf{Adv}^{\text{ind\$-cpa}}_{\mathcal{SE}_{\text{Fun}}, A_{\text{cpa}}}(k, z) := \Pr[\mathbf{Exp}^{\text{ind\$-cpa}}_{\mathcal{SE}_{\text{Fun}}, A_{\text{cpa}}}(k, z) = 1]$$

with the maximum over all possible PPT adversaries as

$$\mathbf{Adv}^{\text{ind\$-cpa}}_{\mathcal{SE}_{\text{Fun}}}(k, q_e, \eta_e, z) := \max_{A_{\text{cpa}}}\{\mathbf{Adv}^{\text{ind\$-cpa}}_{\mathcal{SE}_{\text{Fun}}, A_{\text{cpa}}}(k, z)\}$$

where $q_e$ denotes the maximum number of oracle calls to $\mathcal{E}_{\text{Fun}}$ or $\mathcal{E}_{\text{Rand}}$, and $\eta_e$ the maximum number of $k$-bit blocks per encryption query.

**Proposition 7** *Let $\mathcal{SE}_{\mathrm{Fun}}$ be the authenticated encryption scheme given in Figure 10 using random functions and $z$ any polynomial bounded string in $k$ with $q_e \geq 1$, and $k \geq 1$. Then*

$$\mathbf{Adv}^{\text{ind\$-cpa}}_{\mathcal{SE}_{\mathrm{Fun}}}(k, q_e, \eta_e, z) \leq \frac{q_e^2}{2}(3\eta_e^2 + \eta_e)2^{-k}$$

**Proof:** We can bound the IND$-CPA advantage using game G2 in Figure 11 if we remove the single authentication query and allow for more than one encryption query. This simulates both $\mathcal{SE}_{\mathrm{Fun}}$ and $\mathcal{SE}_{\mathrm{Rand}}$, which are identical until *bad* is set. Therefore, using the Fundamental Lemma of Game-Playing we have $\mathbf{Adv}^{\text{ind\$-cpa}}_{\mathcal{SE}_{\mathrm{Fun}}}(k, q_e, \eta_e, z) \leq \Pr[\text{Game 2 sets } bad]$. Following the same arguments as used in Proposition 5 (including the assumption that $A$ is deterministic and computationally unbounded), we may transform game G2 to G4. Since for any fixed chain of queries there are at most $q_e(\eta_e+1)$ non-random variables $X^{-1}_{1,IV}, X^{s=0,\ldots,\eta_e-1}_{0,IV}$, $IV = 0, \ldots, q_e-1$ that do not collide with one another and $q_e\eta_e$ independent random variables $X^{s=0,\ldots,\eta_e-1}_{1,IV}$, $IV = 0, \ldots, q_e - 1$ in $\mathrm{Dom}(\rho)$, it follows that the setting of *bad* in game G4 is bounded above by

$$\Pr[\text{Game G4 sets } bad] \leq \left\{ \binom{q_e(2\eta_e+1)}{2} - \binom{q_e(\eta_e+1)}{2} \right\} 2^{-k}$$

which holds for any computationally unbounded adversary. Therefore, it follows that

$$\begin{aligned}
\mathbf{Adv}^{\text{ind\$-cpa}}_{\mathcal{SE}_{\mathrm{Fun}}}(k, q_e, \eta_e, z) &\leq \Pr[\text{Game G4 sets } bad] \\
&\leq \left\{ \binom{q_e(2\eta_e+1)}{2} - \binom{q_e(\eta_e+1)}{2} \right\} 2^{-k} \\
&= \frac{q_e^2}{2}(3\eta_e^2 + \eta_e)2^{-k}.
\end{aligned}$$

□