Obfuscating Finite Automata

Steven D. Galbraith and Lukas Zobernig

Department of Mathematics, The University of Auckland, Auckland, New Zealand
{s.galbraith,lukas.zobernig}@auckland.ac.nz

Abstract We construct a VBB and perfect circuit-hiding obfuscator for *evasive deterministic finite automata* using a matrix encoding scheme with a limited zero-testing algorithm. We construct the matrix encoding scheme by extending an existing matrix FHE scheme. Using obfuscated DFAs we can for example evaluate secret *regular expressions* or disjunctive normal forms on public inputs. In particular, the possibility of evaluating regular expressions solves the open problem of *obfuscated substring matching*.

1 Introduction

There are several constructions of program obfuscation schemes for different evasive functions, such as hyperplane membership [11], boolean conjunctions and pattern matching with wildcards [10, 7, 6, 5], compute-and-compare programs [25, 19], fuzzy Hamming distance [14], and more [23]. These obfuscation schemes use different security notions such as *virtual black-box* (VBB) obfuscation, *input hiding* obfuscation, *perfect circuit-hiding* obfuscation, and *indistinguishability obfuscation* (iO).

All of the aforementioned obfuscation schemes target specific evasive functions [23]. General obfuscation schemes are less practical. There are candidates for generic iO schemes [15] and VBB branching program obfuscation, but none of them are practical. It seems that by restricting to evasive functions and special purpose obfuscation, it is possible to obtain feasible schemes.

In this work we consider a somewhat more general class of programs, namely *deterministic finite automata* (DFA). The theory of obfuscating a DFA has been considered before by Lynn et al. [23]. They give an obfuscator in the random oracle model for a special class of regular expressions for which the symbols are given by point functions. As open problems they ask whether regular languages can be obfuscated and whether there is any non-trivial obfuscation result without using the random oracle model. Kuzurin et al. [21] state that secure obfuscation of DFAs is one of the most challenging problems in the theory of program obfuscation. Note that we cannot simply apply existing circuit obfuscation solutions to the problem of obfuscating DFAs. Unlike a tree-like circuit which we can evaluate on an input by traversing it from the circuit root to one of the leaves, a DFA can cycle back to previous states. Furthermore, a DFA has a variable number of input symbols.

We give a VBB and perfect circuit-hiding obfuscator for *evasive* DFAs in the standard model. We will now explain why we do not consider arbitrary DFAs. It is a classical result that certain types of finite automata can be learned from their input/accept/reject behaviour, cf. Balcázar et al. [3]. We will also give another possible learning strategy for finite automata in Section 5.1 if certain information is given. Hence we will only consider those automata which, without loss of generality, reject almost all inputs. We will call such an automaton *evasive*. Any security claims we make are only for an adversary who does not know any accepting input.

Our solution to the problem of DFA obfuscation uses tools that were developed for homomorphic encryption and multilinear maps. These tools are often used to construct indistinguishability obfuscation schemes. Some of the general purpose iO schemes have questionable hardness assumptions or have been broken altogether, see Ananth et al. [1, 2, Appendix A]. To prove iO security, the underlying multilinear maps need to come with a hard generalised DDH problem.

For our application, we instead require a different hard computational problem: Distinguishing two related encodings given certain public zero-testing information should be intractable. We will consider only encodings of evasive DFAs for this problem to make sense. Instead of considering iO we will consider virtual black-box obfuscation. Since Barak et al. [4] showed that VBB obfuscation is equivalent to perfect circuit-hiding obfuscation for evasive functions, we indirectly prove that our obfuscator hides all information about the DFA description.

We believe that we have avoided all previous attacks on multilinear maps that exploit the zero-testing parameter, due to our restriction to evasive DFAs and due to our very limited zero-testing information. We discuss this further in Section 4.

Finite Automata. Using the matrix homomorphic encryption schemes by Hiromasa et al. [20] and Genise et al. [16] Alice may generate a private key and publish an encrypted secret finite automaton to Bob. A finite automaton is represented by a set of transition matrices, one matrix for each possible input symbol. The transition matrices themselves are $n \times n$ square matrices where n is the number of states of the automaton. The homomorphic properties allow Bob to evaluate the secret finite automaton on an arbitrary input. The result of this evaluation is an encrypted vector which Alice may decrypt using her private key. This can for example be use to evaluate secret *regular expressions* by a remote user while a central server can decide about the result.

Genise et al. [16] state the matching of anti-virus signatures as a possible application of such regular expressions. One problem with this setup is the need for interactivity. In their scheme, Alice uses a matrix FHE scheme to encrypt a virus signature represented by an automaton and sends it to Bob. Bob then applies his input to the hidden automaton which produces an encrypted state vector. If Bob wants to learn whether there indeed is a virus present, he needs to send back an encrypted state vector to Alice. She can then decrypt the encrypted state vector and notify Bob accordingly.

Additionally, the analysis of Genise et al. [16] does not consider an *adaptive attack* in the form of multiple queries with an oracle that reports accept/reject for arbitrary inputs. As mentioned, such an oracle can be used to leak parts or all of the finite automaton description, cf. Balcázar et al. [3]. In this adaptive setting, we argue that the number of allowed oracle queries needs to be small enough for arbitrary finite automata, or a specific class needs to be used: We propose the class of evasive finite automata.

Our Contribution. We consider obfuscation for deterministic finite automata and in particular restrict to the class of evasive DFAs in light of Balcázar et al. [3]. DFAs can represent problems such as *regular expressions* and *conjunctions* (also known as *pattern matching with wildcards*).

- We obtain an obfuscator for evasive regular expressions and consequently solve the open problem of *obfuscated substring matching*. Given a plaintext input $s \in \{0,1\}^n$, obfuscated substring matching is the problem of identifying whether s contains a secret substring $x \in \{0,1\}^k$, for $k \leq n \in \mathbb{N}$. We achieve something even more general, as the substring can be given by a regular expression. This gives a complete and non-interactive solution to the virus testing application suggested by Genise et al. [16].
- We obtain an obfuscator for arbitrary evasive conjunctions. A conjunction on Boolean variables $b_1, \ldots, b_k$ is $\chi(b_1, \ldots, b_k) = \bigwedge_{i=1}^k c_i$ where each c_i is of the form b_j or $\neg b_j$ for some $1 \leq j \leq k$. Pattern matching with wildcards is an alternative representation of a conjunction. Consider a vector $x \in \{0, 1, \star\}^k$ of length $k \in \mathbb{N}$ where $\star$ is a special *wildcard* symbol. Such an x then corresponds to a conjunction $\chi : \{0,1\}^k \to \{0,1\}$ which, using Boolean variables $b_1, \ldots, b_k$, can be written as $\chi(b) = \bigwedge_{i=1}^k c_i$ where $c_i = \neg b_i$ if $x_i = 0$, $c_i = b_i$ if $x_i = 1$, and $c_i = 1$ if $x_i = \star$. Additionally, we can consider a set of conjunctions to obtain a boolean formula in *disjunctive normal form* $\bigvee_i \bigwedge_j (\neg) b_{ij}$. In conclusion, our DFA obfuscator allows for yet another solution of this problem.

Our techniques. Consider the matrix FHE scheme by Hiromasa et al. [20] with parameters $q, n \in \mathbb{N}$, and scaling factor $\beta = q/2$. Given a secret matrix $M \in \{0,1\}^{r \times r}$, we encode it to form a matrix C such that $SC = MSG + E$, for another small secret matrix S and small error matrix E. Here G is a so called *gadget matrix* which is used to construct a lattice trapdoor. Given the secret S, we may decode the ciphertext C to recover M. Similarly, we may encrypt a vector $v \in \{0,1\}^r$ to obtain a ciphertext c such that $Sc = \beta v + e$ for a small error vector e. Vector decryption is correct by rounding $\lceil (1/\beta)(Sc \bmod q) \rfloor$ if the error e is bounded by $\|e\|_\infty \leq \beta/2$. The fully homomorphic property then allows us to multiply encoded matrices via $C_1 \odot C_2 := C_1 G^{-1}(C_2)$ which corresponds to an encoding of $M_1 M_2$. We can further apply encoded matrices to encrypted vectors by computing $CG^{-1}(c)$ which corresponds to an encoding of Mv.

Given any DFA with $r \in \mathbb{N}$ states and alphabet Σ, we can obtain *transition matrices* $\{M_\sigma\}_{\sigma \in \Sigma}$ with $M_\sigma \in \{0,1\}^{r \times r}$. These matrices then act on state vectors which for a DFA are simply the canonical basis vectors $e_1, \ldots, e_r$. Without loss of generality, assume that the initial state is given by e_1 and that the accepting state is given by e_r. We will only consider *evasive* DFAs whose shortest accepted input word has a large min-entropy. We will also assume that the DFA matrices are given in a certain *canonical* form which safeguards from leaking states after partial evaluation and intermediate state transitions.

Finally, we encode the DFA matrices $\{M_\sigma\}_{\sigma\in\Sigma}$ to obtain a set of encodings $\{C_\sigma\}_{\sigma\in\Sigma}$, we encrypt the initial state vector e_1 to obtain the ciphertext vector c, and we publish the last row of the HAO15 secret S, call it s_r. The obfuscation of the DFA is then the tuple $(s_r, \{C_\sigma\}_{\sigma\in\Sigma}, c)$. The security of our scheme is based on the security of the HAO15 scheme with the last row of the secret known.

Using the multiplicative property, we may then evaluate the obfuscated DFA on an input word $w \in \Sigma^*$ by first computing an encryption c_w of the state vector of the DFA on input w

$$c_w = \left(\bigodot_{i=|w|}^{1} C_{w_i} \right) G^{-1}(c).$$

This corresponds to evaluating the DFA in the plaintext space by computing $t = (\prod_{i=|w|}^{1} M_{w_i})e_1$. Finally, to check whether c_w is an encryption of the final state (and thus whether the DFA accepts the input word w), we use the following identity

$$t_r = \left\lceil \frac{s_r \cdot c_w \mod q}{\beta} \right\rfloor,$$

where $\lceil . \rfloor$ denotes rounding to the nearest integer. This identity implies that knowing the last row s_r of the secret S is sufficient to check whether c_w is an encryption of e_r, in which case $t_r = 1$. If $t_r = 0$, then c_w is an encryption of any of the other possible state vectors $e_1, \ldots, e_{r-1}$. Note that these encryptions are indistinguishable since s_r can only decrypt the last coordinate. All a user can learn is whether or not the final state is the accepting state, there is no other leakage of the structure of the DFA. We will show that this construction preserves functionality as long as the length of the input word is shorter than a certain maximal length which depends on the individual system parameters.

Outline of This Work. Section 2 recalls basic (obfuscation) definitions. Sections 3 and 4 introduce the notion of a matrix graded encoding scheme and exhibits two candidate constructions of matrix (graded) encoding schemes. The hardness of these schemes is based on lattice problems and new computational assumptions. In Section 5 we explain how to represent finite automata using *transition matrices* and consider possible ways of learning (partial) information from such a representation. Sections 6 and 7 present the DFA obfuscator, security reductions to VBB and perfect circuit-hiding, and consider some parameters. Additionally, in Appendix C we briefly consider obfuscated DFAs from general matrix graded encoding schemes.

2 Obfuscation Definitions

We are interested in obfuscating a special type of programs, namely ones which either accept or reject almost all inputs. The following definition formalises this situation.

Definition 2.1 (Evasive Program Collection). *Let $\mathcal{P} = \{\mathcal{P}_n\}_{n\in\mathbb{N}}$ be a collection of polynomial-size programs such that every $P \in \mathcal{P}_n$ is a program $P : \{0,1\}^n \to \{0,1\}$. The collection $\mathcal{P}$ is called* evasive *if there exists a negligible function ϵ such that for every $n \in \mathbb{N}$ and for every $y \in \{0,1\}^n$:*

$$\Pr_{P \leftarrow \mathcal{P}_n}[P(y) = 1] \le \epsilon(n).$$

In short, Definition 2.1 means that a random program from an evasive collection $\mathcal{P}$ evaluates to 0 with overwhelming probability. Finally, we call a member $P \in \mathcal{P}_n$ for some $n \in \mathbb{N}$ an *evasive program* or an *evasive function.*

Definition 2.2 (Perfect Circuit-Hiding Obfuscation [4]). *An obfuscator $\mathcal{O}$ for a collection of evasive programs $\mathcal{P}$ is* perfect circuit-hiding, *if for every PPT adversary $\mathcal{A}$ there exists a negligible function*

ϵ such that for every $n \in \mathbb{N}$, every balanced predicate $\varphi : \mathcal{P}_n \to \{0,1\}$, and every auxilliary input $\alpha \in \{0,1\}^{poly(n)}$ to $\mathcal{A}$:

$$\Pr_{P \leftarrow \mathcal{P}_n}[\mathcal{A}(\alpha, \mathcal{O}(P)) = \varphi(P)] \leq \frac{1}{2} + \epsilon(n),$$

where the probability is also over the randomness of $\mathcal{O}$.

Barak et al. [4, Theorem 2.1] showed that for evasive programs perfect circuit-hiding obfuscation is equivalent to *virtual black box* obfuscation.

Definition 2.3 (Distributional Virtual Black-Box Obfuscator with Auxiliary Input). *Let $\mathcal{P} = \{\mathcal{P}_n\}_{n\in\mathbb{N}}$ be a family of polynomial-size programs with input size n and let $\mathcal{O}$ be a PPT algorithm which takes as input a program $P \in \mathcal{P}$, a security parameter $\lambda \in \mathbb{N}$ and outputs a program $\mathcal{O}(P)$ (which itself is not necessarily in $\mathcal{P}$). Let $\mathcal{D}$ be a class of distribution ensembles $D = \{D_\lambda\}_{\lambda\in\mathbb{N}}$ that sample $(P, \alpha) \leftarrow D_\lambda$ with $P \in \mathcal{P}$ and α some auxiliary input. The algorithm $\mathcal{O}$ is a* VBB obfuscator for the distribution class *$\mathcal{D}$ over the program family $\mathcal{P}$ if it is functionality preserving, implies polynomial slowdown, and satisfies the following property:*

- *Virtual black-box: For every (non-uniform) polynomial size adversary $\mathcal{A}$, there exists a (non-uniform) polynomial size simulator $\mathcal{S}$ with oracle access to P, such that for every $D = \{D_\lambda\}_{\lambda\in\mathbb{N}} \in \mathcal{D}$, and every (non-uniform) polynomial size predicate $\varphi : \mathcal{P} \to \{0,1\}$:*

$$\left| \Pr_{P \leftarrow D_\lambda, \mathcal{O}, \mathcal{A}}[\mathcal{A}(\mathcal{O}(P, \alpha)) = \varphi(P)] - \Pr_{P \leftarrow D_\lambda, \mathcal{S}}\left[\mathcal{S}^P(|P|, \alpha) = \varphi(P)\right] \right| \leq \epsilon(\lambda)$$

  where $\epsilon(\lambda)$ is a negligible function.

In simple terms, Definition 2.3 states that a VBB obfuscated program $\mathcal{O}(P)$ does not reveal anything more than would be revealed from having black box access to the program P itself.

A definition that is more convenient to work with for proving security is *distributional indistinguishability*. To make sense of this, we will need the following definition that tells when two distributions are indistinguishable in a computational sense.

Definition 2.4 (Computational Indistinguishability). *We say that two ensembles of random variables $X = \{X_\lambda\}_{\lambda\in\mathbb{N}}$ and $Y = \{Y_\lambda\}_{\lambda\in\mathbb{N}}$ are* computationally indistinguishable *and write $X \overset{c}{\approx} Y$ if for every (non-uniform) PPT distinguisher $\mathcal{A}$ it holds that*

$$|\Pr[\mathcal{A}(X_\lambda) = 1] - \Pr[\mathcal{A}(Y_\lambda) = 1]| \leq \epsilon(\lambda)$$

where $\epsilon(\lambda)$ is some negligible function.

Definition 2.5 (Distributional Indistinguishability [25]). *An obfuscator $\mathcal{O}$ for the distribution class $\mathcal{D}$ over a family of programs $\mathcal{P}$ satisfies* distributional indistinguishability *if there exists a (non-uniform) PPT simulator $\mathcal{S}$ such that for every distribution ensemble $D = \{D_\lambda\}_{\lambda\in\mathbb{N}} \in \mathcal{D}$ the following distributions are computationally indistinguishable*

$$(\mathcal{O}(P), \alpha) \overset{c}{\approx} (\mathcal{S}(|P|), \alpha) \tag{2.1}$$

where $(P, \alpha) \leftarrow D_\lambda$. Here α denotes some auxiliary information.

Note that the sampling procedure for the left and right side of Equation (2.1) in Definition 2.5 is slightly different. For both we sample $(P, \alpha) \leftarrow D_\lambda$ and for the left side we simply output $(\mathcal{O}(P), \alpha)$ immediately. On the other hand, for the right side we record $|P|$, discard P and finally output $(\mathcal{S}(|P|), \alpha)$ instead.

It can be shown that distributional indistinguishability implies VBB security under certain conditions. To see this, we first define the augmentation of a distribution class by a predicate.

Definition 2.6 (Predicate Augmentation [25]). *For a distribution class $\mathcal{D}$, its* augmentation under predicates $\mathrm{aug}(\mathcal{D})$ *is defined as follows: For any (non-uniform) polynomial-time predicate $\varphi : \{0,1\}^* \to \{0,1\}$ and any $D = \{D_\lambda\}_{\lambda\in\mathbb{N}} \in \mathcal{D}$, the class $\mathrm{aug}(\mathcal{D})$ indicates the distribution $D' = \{D'_\lambda\}_{\lambda\in\mathbb{N}}$ where D'_λ samples $(P, \alpha) \leftarrow D_\lambda$, computes $\alpha' = (\alpha, \varphi(P))$ and outputs (P, α'). Here α denotes some auxiliary information.*

The following theorem shows that distributional indistinguishability for the larger augmented class $\mathrm{aug}(\mathcal{D})$ implies distributional VBB security for the class $\mathcal{D}$.

Theorem 2.1 (Distributional Indistinguishability Implies VBB [25]). *For any family of programs $\mathcal{P}$ and a distribution class $\mathcal{D}$ over $\mathcal{P}$, if an obfuscator satisfies distributional indistinguishability (Definition 2.5) for the class of distributions* $\mathrm{aug}(\mathcal{D})$ *then it also satisfies distributional VBB security for the distribution class $\mathcal{D}$ (Definition 2.3).*

Proof. See [10, Lemma 2.2] and [26, Theorem 3.4]. □

3 Matrix (Graded) Encoding Schemes

A matrix (graded) encoding scheme allow us to securely encode matrices. We can add and multiply encoded matrices which corresponds to adding and multiplying the underlying plaintext matrices. Finally, a zero-testing primitive should allow us to test whether an encoded matrix is an encoding of the zero matrix.

We will consider the definition of a *matrix (graded) encoding scheme* (similar to [8, 9]) first and then remind the reader of possible candidates and give one new specialised construction based on the matrix FHE scheme by Hiromasa et al. [20].

3.1 HAO15

The matrix FHE scheme of Hiromasa et al. [20] is somewhat related to the scheme by Gentry et al. [17] we describe in Appendix B. The hardness of both schemes is connected to the hardness of finding *approximate eigenspaces*.

Depending on a security parameter $\lambda \in \mathbb{N}$, fix a modulus q, a lattice dimension n, and a distribution χ over $\mathbb{Z}$. We are working over the ring $R = \mathbb{Z}/q\mathbb{Z}$. Assume the matrices we want to encode are from $\{0,1\}^r$ for some $r \in \mathbb{N}$. Set $\ell = \lceil \log(q) \rceil$, $N = (n+r)\ell$.

Let $g = (2^i)_{i=0,\dots,\ell-1} \in R^\ell$ be the *gadget vector*. Fix $G = g^T \otimes \mathrm{id}_{n+r} \in R^{(n+r)\times N}$, the *gadget matrix*. We may further assume that there exists a randomized algorithm $G^{-1}(v)$ that for an input $v \in R^{n+r}$, samples a vector $v' \leftarrow G^{-1}(v) \in R^N$ such that $Gv' = v$.

Key Generation. For key generation, we sample a secret matrix $S' \leftarrow \chi^{r\times n}$ and set $S = \big(\mathrm{id}_r \,|\, -S'\big) \in R^{r\times(n+r)}$. A priori, this matrix FHE scheme does not support zero-testing and since we are not interested in public encryption, we do not need any public parameters here. We will describe the public key when we discuss our solution for a zero-testing primitive.

Matrix Encoding. Given a matrix $M \in \{0,1\}^{r\times r}$, we sample $A' \leftarrow R^{n\times N}$ uniformly and $E \leftarrow \chi^{r\times N}$ and output the encoding

$$C = \left(\frac{S'A' + E}{A'}\right) + \left(\frac{MS}{0}\right) G \in R^{(n+r)\times N}.$$

It holds that $SC = MSG + E$.

Matrix Decoding. Given an encoding C of a matrix M with respect to the secret S, finding $M_{i,j}$ works as follows: Compute the scalar product $x = S_i \cdot C_{j\ell-1}$ (S_i and $C_{j\ell-1}$ are the i-th and $(j\ell-1)$-th rows of S and C, respectively). It holds that $M_{i,j} = 1$ if x is close to $q/4$ and 0 otherwise.

Vector Encoding. Similarly, given a vector $v \in R^r$, we sample $a \leftarrow R^n$ uniformly and $e \leftarrow \chi^r$ and output the encoding

$$c = \left(\frac{S'a + e}{a}\right) + \left(\frac{v}{0}\right) \in R^{n+r}.$$

It holds that $Sc = v + e$.

Vector Encryption. The scheme also supports encryption and decryption of vectors. For this, fix an upper bound b on the $\|\cdot\|_\infty$-norm of vectors that should be possible to encrypt and decrypt and set

$$\beta = \lfloor q/b \rfloor.$$

For example, to encrypt binary secrets we can set $b = 2$. To encrypt a vector v, we will scale it by β such that the $\|\cdot\|_\infty$-norm of the error is bounded by β with high probability. Formally, to encrypt $v \in \{0, \dots, b-1\}^n$, output the encoding c of βv such that $Sc = \beta v + e$. To decrypt c, given the secret S, we compute

$$v = \left\lceil \frac{Sc \mod q}{\beta} \right\rfloor, \tag{3.1}$$

i.e. we round the entries of $(1/\beta)Sc$ to the closest integer.

Homomorphic Operations. Given two encodings C_1, C_2, addition is simply computing $C_1 + C_2$. The encodings can by multiplied by computing $C_1 G^{-1}(C_2)$, denote this by $C_1 \odot C_2$. Applying an encoded matrix C to an encoded vector c is computing $CG^{-1}(c)$.

Zero-testing. Testing whether a given encoding is an encoding of zero is slightly more complicated because we cannot publish the secret matrix S. For our application, the following construction is sufficient. Let M_f be the $r \times r$ matrix which is zero everywhere except for a single 1 in its lower right corner, i.e. $(M_f)_{r,r} = 1$. Let C_f be the encoding of M_f. Let c be an encrypted vector. We need to test whether $C_f c$ is an encryption of e_r, the r-th canonical basis vector. To test for this, we publish the last row of the secret matrix S, call it $s_r \in R^{n+r}$. Assuming we only ever encrypt canonical basis vectors, the problem is then equivalent to checking whether $\lceil (1/\beta)(s_r \cdot c \mod q) \rfloor$ equals 1, see Equation (3.1). Equality holds if and only if $C_f c$ is an encryption of a vector that has a 1 in coordinate r, see the proof of Lemma 6.1 for details. This limited construction allows us to use the HAO15 matrix FHE scheme as a matrix encoding scheme.

Error Bounds and Correctness In the plain HAO15 matrix FHE scheme, to decode an encoded matrix, the error needs to be bounded by $\|E\|_\infty \le q/8$. In our application, we do not need to decode matrices, but decrypt vectors. To correctly decrypt encrypted vectors, we see from Equation (3.1) that the error needs to be bounded by $\|e\|_\infty \le \beta/2 = q/4$. Hiromasa et al. [20] showed that the noise growth is asymmetric and hence computing a polynomial length chain of homomorphic multiplications leads to a noise growth by a multiplicative polynomial factor. Denote with $|\chi|$ the standard deviation of the distribution χ. Genise et al. [16] showed that error produced by the application of κ matrices $(M_i)_{i=1,\dots,\kappa}$ on a vector is bounded by

$$\|e_\kappa\|_\infty \le |\chi| N \left(1 + \kappa \max_{1 \le i \le \kappa} \left\| \prod_{j=\kappa}^{i} M_j \right\|_\infty \right).$$

For our application, we will consider matrices $(M_\sigma)_{\sigma \in \Sigma}$ that describe a DFA. We argue that for such matrices we obtain a large maximal grading $\kappa \sim q/\log(q)$. Genise et al. [16] introduced an *ambiguity measure* that better restricts the error bound for finite automata, depending on their *ambiguity type*. They considered more general NFAs whereas we shall restrict to DFAs only. They showed that DFAs are what they call *unambiguous* and that the error then can be bounded as $\|e_\kappa\|_\infty \le |\chi|(N\kappa + 1)$. We find that for $\|e_\kappa\|_\infty$ to be bounded by $\beta/2 = q/4$, we require that

$$\kappa \le \frac{q}{4\sqrt{n}(n+r)\lceil \log(q) \rceil}. \tag{3.2}$$

4 HAO15 Zero-Testing and Computational Assumptions

The original matrix FHE scheme by Hiromasa et al. [20] enjoys CPA security and does not allow for zero-testing. In Section 3.1 we constructed a zero-testing primitive. This requires us to introduce an additional hardness assumption if we want to speak about security when using our extended HAO15 scheme as a matrix encoding scheme.

We stress that we do not construct an *absolute* zero-testing primitive which would allow to test whether an arbitrary matrix or vector entry is zero. A zero-testing primitive like this, while very powerful, would potentially reveal much more information about the private key. Instead we construct a primitive that only allows to test entries in the last row of a matrix or the last entry of a vector, respectively. Our construction only reveals partial information about the secret key. Finally, note that we do not consider iO but instead VBB and perfect circuit-hiding obfuscation instead for evasive finite automata. See Ananth et al. [1, 2, Appendix A] for a summary of (zeroising) attacks on iO.

Definition 4.1 (DFA Security). *Consider the HAO15 matrix encoding scheme with security parameter $\lambda \in \mathbb{N}$. Fix a secret key S. Let $(M_\sigma)_{\sigma\in\Sigma}$ be a sequence of matrices in $\{0,1\}^{r\times r}$, α some auxiliary information. We say that HAO15 satisfies* DFA security *for $((M_\sigma)_{\sigma\in\Sigma}, \alpha)$ if the following two distributions are computationally indistinguishable:*

$$(s_r, (C_\sigma)_{\sigma\in\Sigma}, c, \alpha) \overset{c}{\approx} (s_r, (C'_\sigma)_{\sigma\in\Sigma}, c, \alpha),$$

where s_r is the last row of the secret key S, and where for all $\sigma \in \Sigma$ we have encodings such that

$$SC_\sigma = M_\sigma SG + E,\ Sc = \beta e_1 + e,$$
$$SC'_\sigma = M'_\sigma SG + E,\ Sc' = \beta e_1 + e,$$

for a sequence of matrices $(M'_\sigma)_{\sigma\in\Sigma}$ in $\{0,1\}^{r\times r}$ such that $|M_\sigma - M'_\sigma|$ is all zeroes apart from a single 1 in some row but not the last row.

Note that Definition 4.1 is closely related to the definition of IND-CPA security for an asymmetric cipher: The adversary is given a number of encryptions of known messages and needs to distinguish them. In our case, we additionally require that the messages are related and there is some partial knowledge of the secret key revealed.

The sequences of matrices $(M_\sigma)_{\sigma\in\Sigma}$ that we will consider are matrices that encode DFAs such that the min-entropy of the shortest accepting input word conditioned on the auxiliary information α is at least $\lambda(r)$. See Definitions 6.1 and 6.3 for a formal definition of such a distribution.

Hiromasa et al. [20, Theorem 4] states that the plain HAO15 scheme is semantically secure based on a circular security assumption and the hardness of the decisional learning with error problem (DLWE) for parameters n, q, χ. If we did not publish s_r, then Definition 4.1 would certainly hold for appropriate parameters.

Given s_r we may test whether the last coordinate of an encoded vector is 0 or 1. Hence we need to consider certain safeguards, which are described in detail in Section 5.1. We want that for every additional encoded *state vector*, the last coordinate is 0 with overwhelming probability. This is true for the distributions of evasive DFAs that we will consider. Using s_r we can also learn the entries of the last row of the DFA matrices. Hence, we assume that the last row of the encoded matrices always follows a certain structure. This ensures indistinguishability as required by Definition 4.1.

Finally, we conjecture that the knowledge of the last row of the secret does not weaken the security of the HAO15 matrix encoding scheme. The hardness of (D)LWE with *leaky secrets* was studied by Goldwasser et al. [18].

5 Finite Automata and Transition Matrices

Fix a number of states $r \in \mathbb{N}$. Fix an alphabet Σ and for each symbol $\sigma \in \Sigma$, let $M_\sigma \in \{0,1\}^{r\times r}$ be the *transition matrix* corresponding to σ.

In case of a finite automata M, Σ represents the different input symbols which induce transitions between the r different states, i.e. $(M_\sigma)_{j,i} = 1$ if and only if there is a transition from state i to state j for an input σ. Hence, such an M_σ acts on the i-th canonical basis vector e_i such that $e_j = M_\sigma e_i$. Without loss of generality, let 1 be the initial state (represented by e_1) and let r be the final state (represented by e_r). There is a distinction between *deterministic* and *non-deterministic* finite automata (DFA and NFA, respectively). On the one hand, a DFA has a unique state transition for each state and input. On the other hand, a NFA may transition into multiple states on each input or transition without any input at all. In general, a NFA will not have a unique accepting state.

5.1 General Safeguards

We will now introduce two general safeguards to avoid partial evaluation and leaking intermediate states and state transitions. These safeguards are important for our specific construction based on the HAO15 matrix encoding scheme of Section 3.1 as well as the general construction from arbitrary matrix (graded) encoding schemes we will introduce in Appendix C.

State Transitions. Without loss of generality, let $\Sigma = \{\sigma_1, \ldots, \sigma_m\}$ be the set of symbols, for some $m \in \mathbb{N}$. Consider a DFA with $r \in \mathbb{N}$ states and let r be the accepting state. To avoid leaking state transitions, we need to ensure that the matrices representing the DFA have the following structure:

$$M_{\sigma_1} = \begin{pmatrix} * & * \\ 0 & 0 \end{pmatrix}, \ldots, M_{\sigma_{m-1}} = \begin{pmatrix} * & * \\ 0 & 0 \end{pmatrix}, M_{\sigma_m} = \begin{pmatrix} * & \cdots & * & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ * & \cdots & * & 0 & 0 \\ 0 & \cdots & 0 & 1 & 1 \end{pmatrix}.$$

We set the last row of the matrices $M_{\sigma_1}, \ldots, M_{\sigma_{m-1}}$ to zero. This means that none of the input symbols $\sigma_1, \ldots, \sigma_{m-1}$ can transition the DFA into the accepting state r. The structure of the matrix M_{σ_m} is chosen such that σ_m is the unique input symbol which can transition the DFA into the accepting state r. We also allow for an arbitrary number of additional inputs of the symbol σ_m sending the state r to itself.

This ensures that an attacker does not learn anything that is not already public knowledge in the system. As mentioned in Section 4, this is important for the validity of the security assumption in our application. We require that the last rows of all transition matrices follow the same structure.

Partial Evaluation. We need to make sure that no adversary can distinguish between states after merely partially evaluating the DFA. To see why, consider the following attack strategy.

We can evaluate the obfuscated DFA on progressively longer input words and each time record the encoded state vector. Although we do not learn the state vector itself, using a zero-testing primitive, we can decide when two states are the same for different inputs. Even if we force a fixed input word length (for example by restricting the zero-test to only be possible after evaluating a certain number of input symbols), we can simply prepend each different word by a fixed prefix. Using statistical analysis on the number of encountered states, we can then try to either construct an accepted input directly or at least (partially) learn the *structure* of the underlying DFA.

To remedy this, we need to make sure that nothing can be learned about individual states after (partial) DFA evaluation, apart from whether or not they are accepting states. The key idea is to *erase* all non-accepting states before zero-testing is possible. For this, consider the following matrix

$$M_f = \begin{pmatrix} 0 & \cdots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & 0 \\ 0 & \cdots & 0 & 1 \end{pmatrix} \in \{0,1\}^{r \times r}.$$

It holds that for all canonical basis vectors e_i for $i = 1, \ldots, r-1$ we have $M_f e_i = 0$, whereas $M_f e_r = e_r$. Another way to express this is that the matrix M_f maps all state vectors to the zero vector if they are not equal to the final state vector but leaves the final state vector invariant.

6 Obfuscated Finite Automata

We would like to construct an obfuscator for finite automata. Every finite automaton induces a program $P : \Sigma^* \to \{0,1\}$ that outputs 1 for an accepted input sequence and 0 for a rejected one.

Definition 6.1 (Evasive Finite Automata Collection). *Let $\{\mathcal{M}_r\}_{r\in\mathbb{N}}$ be a collection of finite automata such that every automaton in $\mathcal{M}_r$ has r states. The collection is called* evasive *if there exists a negligible function ϵ such that for every $r \in \mathbb{N}$ and for every polynomial-size input $y \in \Sigma^*$:*

$$\Pr_{M \leftarrow \mathcal{M}_r}[M(y) = 1] \leq \epsilon(r).$$

It is important to limit to polynomial size inputs $y \in \Sigma^*$ in Definition 6.1 since otherwise we could let y be the string that contains all possible substrings of a certain length. We need to consider evasive finite automata since the transition matrices of a non-evasive one can be learned from its input/accept/reject behaviour [3]. It is then natural to use Definition 2.2 – perfect circuit-hiding obfuscation – as the security notion for evasive automata. An adversary finds an accepted input with negligible probability and so cannot recover the description of the automata.

We will also restrict to deterministic finite automata since, as we mentioned, they have a unique accepting state.

Definition 6.2 (Min-Entropy). *The* min-entropy *of a random variable X is defined as*

$$\mathrm{H}_\infty(X) = -\log\left(\max_x \Pr[X = x]\right).$$

The (average) conditional min-entropy *of a random variable X conditioned on a correlated variable Y is defined as*

$$\mathrm{H}_\infty(X|Y) = -\log\left(\mathop{\mathrm{E}}_{y \leftarrow Y}\left[\max_x \Pr[X = x|Y = y]\right]\right).$$

Definition 6.3 (DFA Min-Entropy). *We say that a collection $\{\mathcal{M}_r\}_{r\in\mathbb{N}}$ of finite automata has* min-entropy *at least $\lambda(r)$ if for every $(M, \alpha) \leftarrow \mathcal{M}_r$ the min-entropy $\mathrm{H}_\infty(w|\alpha)$ of the shortest accepted word $w \in \Sigma^*$ of M conditioned on the auxiliary information α is at least $\lambda(r)$.*

Example of an Evasive DFA Distribution. We will give two examples for evasive DFA distributions. See Section 1 for definitions of substring matching and conjunctions.

- *String Matching.* Consider an alphabet of three symbols $\Sigma = \{0, 1, \perp\}$, where $\perp$ is the unique symbol that may transition the DFA into the accepting state as Section 5.1 demands. Consider U_k, the uniform distribution over $\{0, 1\}^k$. Then any string sampled from U_k has min-entropy at least k. Define now $\{\mathcal{M}_r\}_{r\in\mathbb{N}}$ to be the collection of evasive DFAs with $r = k + 2$ states such that a DFA sampled from $\mathcal{M}_r$ matches some string sampled from U_k. Hence $\{\mathcal{M}_r\}_{r\in\mathbb{N}}$ is an evasive DFA collection which has min-entropy at least $\lambda(r) = r - 2 = k$. This collection is efficiently samplable by sampling a random string x from U_k and outputting the DFA matching the word $x \,\|\, \perp$ (i.e. x concatenated with $\perp$).
- *Conjunctions.* Another example of an evasive DFA collection are conjunctions. We will need to define what it means for a conjunction to be evasive.

  Definition 6.4 (Conjunction Evasive Distribution). *Consider an ensemble $\mathcal{D} = \{D_\mu\}_{\mu\in\mathbb{N}}$ of distributions D_μ over $\{0, 1, \star\}^{n(\mu)}$ for some function $n(\mu)$. We say that $\mathcal{D}$ is* conjunction evasive *if the min-entropy of D_μ is at least μ.*

  Consider again the alphabet $\Sigma = \{0, 1, \perp\}$ as above. Given a conjunction evasive distribution D_μ for conjunctions of length $n = n(\mu)$, we can define an evasive DFA distribution $\mathcal{M}_r$ with $r = n + 2$ states which has min-entropy at least $\lambda(r) = \mu$: Every DFA from this distribution accepts a string y that satisfies the corresponding conjunction from D_μ.

6.1 Obfuscator and Obfuscated Program

For every evasive DFA M with maximal input length κ, there exists a program $P_M : \Sigma^* \to \{0, 1\}$ that computes whether M accepts an input word $w \in \Sigma^*$ (with $|w| \leq \kappa$) and evaluates to 1 in this case, otherwise to 0. Denote by $\mathcal{P}$ the family of all such programs P_M. The obfuscator $\mathcal{O} : \mathcal{P} \to \mathcal{P}'$ takes one such program $P_M \in \mathcal{P}$ and uses Algorithm 6.1 to output another program in a different family denoted by $\mathcal{P}'$.

Algorithm 6.1 uses the HAO15 matrix encoding scheme (assume the maximal grading is κ) to encode the required matrices and vectors. The output is given by the tuple $(s_r, (C_\sigma)_{\sigma\in\Sigma}, c)$. In this tuple, s_r is the last row of the HAO15 secret S, $(C_\sigma)_{\sigma\in\Sigma}$ is the sequence of encodings of the state transition matrices $(M_\sigma)_{\sigma\in\Sigma}$, and c is an encoding of the first canonical basis vector e_1.

We assume that the initial and accepting state of the finite automaton are given by the state 1 and state r, respectively. We further assume that the DFA matrices $(M_\sigma)_{\sigma\in\Sigma}$ satisfy the safeguard requirements described in Section 5.1. Erasing partial information from the final state using M_f is equivalent to only being able to test whether the last coordinate of the state vector is 0 or 1. Recall, in Section 3.1, we assumed that our state vectors are always canonical basis vectors. This is certainly true for any DFA. Hence publishing only s_r is equivalent to erasing partial state information using M_f.

As the decoding algorithm is a universal algorithm, we will simply denote the *obfuscated program* $\mathcal{O}(P_M)$ with the tuple $(s_r, (C_\sigma)_{\sigma\in\Sigma}, c)$. During the execution of the obfuscated program, Algorithm 6.2 is used to determine whether an input word $w \in \Sigma^*$ is accepted by the DFA or not.

Algorithm 6.1 Encoding (Obfuscating the finite automaton)

procedure ENCODE($(M_\sigma)_{\sigma\in\Sigma}$)
  Run HAO15 matrix encoding scheme key generation and obtain secret key S.
  Compute $(C_\sigma)_{\sigma\in\Sigma}$ by encoding $(M_\sigma)_{\sigma\in\Sigma}$ such that $SC_\sigma = M_\sigma SG + E$ for all $\sigma \in \Sigma$.
  Compute state vector c by encoding e_1 such that $Sc = \beta e_1 + e$.
  Let s_r be the last row of S.
  return $(s_r, (C_\sigma)_{\sigma\in\Sigma}, c)$
end procedure

6.2 Obfuscated Program Evaluation

We may evaluate the obfuscated automaton on a word $w \in \Sigma^*$ with $|w| \leq \kappa$ as follows:

1. Compute the encoded vector c_w corresponding to $(\prod_{i=|w|}^{1} M_{w_i})e_1$ using the sequence $(C_\sigma)_{\sigma\in\Sigma}$ and the encoded initial state c.
2. The input word w is accepted if c_w is an encryption of the r-th canonical basis vector and thus we simply output $\lceil (1/\beta)(s_r \cdot c_w \mod q) \rfloor$.

Again, Algorithm 6.2 presents an algorithmic description.

Algorithm 6.2 Evaluation (Executing the obfuscated program)

procedure EVALUATE($s_r, (C_\sigma)_{\sigma\in\Sigma}, c; w \in \Sigma^*$)
  for all $i = 1, \ldots, |w|$ **do**
    Update the state vector $c = C_{w_i} G^{-1}(c)$.
  end for
  return $\lceil (1/\beta)(s_r \cdot c \mod q) \rfloor$
end procedure

Lemma 6.1 (Correctness). *Consider the algorithms* ENCODE *(Algorithm 6.1) and* EVALUATE *(Algorithm 6.2) (based on the modified HAO15 matrix FHE scheme with maximal grading κ determined by Equation (3.2)). For every DFA $\mathcal{M}$ represented by $(M_\sigma)_{\sigma\in\Sigma}$, for every*

$$(s_r, (C_\sigma)_{\sigma\in\Sigma}, c) \leftarrow \text{ENCODE}((M_\sigma)_{\sigma\in\Sigma})$$

and for every input $w \in \Sigma^*$ with $|w| < \kappa$ it holds that

$$\text{EVALUATE}(s_r, (C_\sigma)_{\sigma\in\Sigma}, c; w) = P_{\mathcal{M}}(w).$$

Proof. Recall the modified HAO15 matrix FHE scheme from Section 3.1. Given a sequence of transition matrices $(M_\sigma)_{\sigma\in\Sigma}$, the obfuscator produces the tuple $(s_r, (C_\sigma)_{\sigma\in\Sigma}, c)$ such that C_σ is an encoding of M_σ for all $\sigma \in \Sigma$. This means that $SC_\sigma = M_\sigma SG + E$, where S is the HAO15 secret. Further, c is an encoding such that $Sc = \beta e_1 + e$, where e_1 is the first canonical basis vector. Finally, s_r is the last row of the secret S.

The evaluation algorithm computes the final state vector

$$c_w = \left(\bigodot_{i=|w|}^{1} C_{w_i} \right) G^{-1}(c).$$

This corresponds to the following calculation with plaintext information

$$t = \left(\prod_{i=|w|}^{1} M_{w_i} \right) e_1.$$

The automaton accepts the input if $t = e_r$. We see that c_w is an encoding of t such that $Sc_w = \beta t + e$ for some error e. Given only s_r, we have the following equation

$$\left(\frac{0_{(r-1)\times(n+r)}}{s_r} \right) c_w = \beta \left(\frac{0_{r-1}}{t_r} \right) + \left(\frac{0_{r-1}}{e'} \right),$$

where e' is the last coordinate of e. By Equation (3.2) the error is bounded by $\|e_\kappa\|_\infty \le \beta/2$ if we choose the maximal grading κ such that

$$\kappa = \frac{q}{4\sqrt{n}(n+r)\lceil\log(q)\rceil}.$$

If $|w| < \kappa$, then $|e'| \le \|e\|_\infty < \|e_\kappa\|_\infty \le \beta/2$. Hence, computing $\lceil (1/\beta)(s_r \cdot c_w \mod q) \rfloor$ correctly determines whether c_w is an encryption of the accepting state e_r or not. Correctness follows as required. □

6.3 Security

In this section we analyse the security of our DFA obfuscator using the HAO15 matrix encoding scheme which we introduced in Section 3.1.

Note that we make no claim of security once an accepting input is known to an adversary. First and foremost, there is a classical result by Balcázar et al. [3] that shows that the description of a finite automaton can be learned from oracle access when given accepted and rejected inputs. Second, we need to keep in mind that an actual matrix graded encoding scheme could exhibit non-modelled (and thus unwanted) behaviour, cf. Ananth et al. [1, 2, Appendix A].

Theorem 6.1. *Let $\{\mathcal{M}_r\}_{r\in\mathbb{N}}$ be an efficiently samplable evasive finite automata collection which has min-entropy at least $\lambda(r)$. Assume HAO15 with security parameter $\lambda(r)$ is DFA secure (Definition 4.1) for the matrices in $\{\mathcal{M}_r\}_{r\in\mathbb{N}}$. Then the obfuscator $\mathcal{O}$ is a VBB obfuscator for $\{\mathcal{M}_r\}_{r\in\mathbb{N}}$.*

Proof. The obfuscator is functionality preserving by Lemma 6.1. It is also clear that the obfuscator causes only a polynomial slowdown when compared to an unobfuscated DFA since the evaluation Algorithm 6.2 runs in time polynomial in all the involved parameters.

By Theorem 2.1 it suffices to show that there exists a (non-uniform) PPT simulator $\mathcal{S}$ such that, for the distribution ensemble $\{\mathcal{M}_r\}_{r\in\mathbb{N}}$, it holds that

$$(\mathcal{O}(P), \alpha) \overset{c}{\approx} (\mathcal{S}(|P|), \alpha),$$

where $(P, \alpha) \leftarrow \mathcal{M}_r$.

We will construct the simulator $\mathcal{S}$: It takes as input $|P|$ and determines the parameter $n \in \mathbb{N}$ and runs Algorithm 6.3.

Denote now with $(s_r, (C_\sigma)_{\sigma\in\Sigma}, c)$ a real instance obtained from obfuscating an evasive DFA given by the transition matrices $(M_\sigma)_{\sigma\in\Sigma}$ sampled from the distribution $\mathcal{M}_r$ such that the DFA has min-entropy $\lambda(r)$. Similarly, let $(s'_r, (C'_\sigma)_{\sigma\in\Sigma}, c')$ be the output from the simulator $\mathcal{S}$ called on r. This is essentially an obfuscation of a random evasive DFA given by the transition matrices $(M'_\sigma)_{\sigma\in\Sigma}$, again with min-entropy

Algorithm 6.3 Encoding Simulator

procedure SIMULATEENCODE($r \in \mathbb{N}$)
  Sample random DFA $(M'_\sigma)_{\sigma\in\Sigma}$ from $\mathcal{M}_r$, this DFA has min-entropy $\lambda(r)$.
  Run HAO15 matrix encoding scheme key generation and obtain secret key S'.
  Compute $(C'_\sigma)_{\sigma\in\Sigma}$ by encoding $(M'_\sigma)_{\sigma\in\Sigma}$ such that $S'C'_\sigma = M'_\sigma S'G + E$ for all $\sigma \in \Sigma$.
  Compute state vector c by encoding e_1 such that $S'c' = \beta e_1 + e$.
  Let s'_r be the last row of S'.
  return $(s'_r, (C'_\sigma)_{\sigma\in\Sigma}, c')$
end procedure

$\lambda(r)$. The last rows of M_σ and M'_σ are the same for all $\sigma \in \Sigma$. This follows from our assumption of Section 6.1: The input $(M_\sigma)_{\sigma\in\Sigma}$ to Algorithm 6.1 satisfies the safeguards of Section 5.1.

We will now show, using a sequence of distributions, that $(s_r, (C_\sigma)_{\sigma\in\Sigma}, c, \alpha)$ and $(s'_r, (C'_\sigma)_{\sigma\in\Sigma}, c', \alpha)$ are computationally indistinguishable. The strategy is to start from the real and simulated distributions and remove state transitions one by one from both until we meet in the middle where both encoded DFAs are the same. Hence we need to consider the matrices $M^\Delta_\sigma = M_\sigma - M'_\sigma$. If an entry of M^Δ_σ is 1, we remove a state transition from M_σ; if an entry is -1, we remove a state transition from M'_σ. This ensures that the min-entropy of the intermediate DFAs can only stay the same or grow, but never shrink. Note that removing state transitions may result in a system that does not accept any inputs, or may not even be an encoding of a DFA.

- Game $(0,0,0)$: Here we consider $(s_r, (C_\sigma)_{\sigma\in\Sigma}, c)$, a real instance obtained from the DFA obfuscator $\mathcal{O}$.
- Game $(0,0,1)$: Here we consider $(s'_r, (C'_\sigma)_{\sigma\in\Sigma}, c')$, the output of the simulator $\mathcal{S}$.
- Game $(i,j,0)$ (for $1 \le i < r, 1 \le j \le r$): Start from the real DFA matrices $(M_\sigma)_{\sigma\in\Sigma}$. For all $\sigma \in \Sigma$, do the following:

  Step 1: Replace full columns.
  for $1 \le t < j$ **do**
    for $1 \le s < r$ **do**
      if $(M^\Delta_\sigma)_{s,t} = 1$ **then**
        Replace $(M_\sigma)_{s,t}$ with 0.
      end if
    end for
  end for

  Step 2: Replace partial columns.
  for $1 \le s \le i$ **do**
    if $(M^\Delta_\sigma)_{s,j} = 1$ **then**
      Replace $(M_\sigma)_{s,j}$ with 0.
    end if
  end for

  This yields the distribution $(s_r, (C^{(i,j,0)}_\sigma)_{\sigma\in\Sigma}, c)$, where $(C^{(i,j,0)}_\sigma)_{\sigma\in\Sigma}$ is a randomly chosen encoding of the resulting transition matrices with respect to the fixed secret key S.
- Game $(i,j,1)$ (for $1 \le i < r, 1 \le j \le r$): Start from the simulated DFA matrices $(M'_\sigma)_{\sigma\in\Sigma}$. For all $\sigma \in \Sigma$, do the following:

  Step 1: Replace full columns.
  for $1 \le t < j$ **do**
    for $1 \le s < r$ **do**
      if $(M^\Delta_\sigma)_{s,t} = -1$ **then**
        Replace $(M'_\sigma)_{s,t}$ with 0.
      end if
    end for
  end for

  Step 2: Replace partial columns.
  for $1 \le s \le i$ **do**
    if $(M^\Delta_\sigma)_{s,j} = -1$ **then**
      Replace $(M'_\sigma)_{s,j}$ with 0.
    end if
  end for

  This yields the distribution $(s'_r, (C^{(i,j,1)}_\sigma)_{\sigma\in\Sigma}, c')$, where $(C^{(i,j,1)}_\sigma)_{\sigma\in\Sigma}$ is a randomly chosen encoding of the resulting transition matrices with respect to the fixed secret key S'.

For $1 \le i < r$, $1 \le j \le r$, in Game $(i,j,\{0,1\})$, the min-entropy of the encoded DFA is at least $\lambda(r)$ since we only ever remove state transitions. We have that Game $(i,j,\{0,1\})$ and Game $(i+1,j,\{0,1\})$ for $0 \le i < r-1$, $0 \le j \le r$ are indistinguishable by the DFA security assumption. We also have that Game $(r-1,j,\{0,1\})$ and Game $(1,j+1,\{0,1\})$ for $1 \le j < r$ are indistinguishable by the DFA security assumption. Finally, the two Games $(r-1,r,0)$ and $(r-1,r,1)$ encode the same DFA under different secret keys S and S' and again are indistinguishable. Hence, by a hybrid argument, it follows that

$$(s_r, (C_\sigma)_{\sigma\in\Sigma}, c, \alpha) \overset{c}{\approx} (s'_r, (C'_\sigma)_{\sigma\in\Sigma}, c', \alpha).$$

We showed that a real obfuscation is computationally indistinguishable from a simulated instance. This completes the proof. □

As mentioned in Section 1, there is an equivalence of VBB obfuscation and perfect circuit-hiding obfuscation for evasive programs.

Theorem 6.2. *Let $\{\mathcal{M}_r\}_{r\in\mathbb{N}}$ be an evasive finite automata collection which has min-entropy at least $\lambda(r)$. Assume HAO15 with security parameter $\lambda(r)$ is DFA secure (Definition 4.1) for the matrices in $\{\mathcal{M}_r\}_{r\in\mathbb{N}}$. Then the obfuscator $\mathcal{O}$ is a perfect circuit-hiding obfuscator for $\{\mathcal{M}_r\}_{r\in\mathbb{N}}$.*

Proof. This follows from Theorem 6.1 and [4, Theorem 2.1]. □

7 Parameters

Genise et al. [16] gave example parameters and runtime analysis for both the matrix FHE schemes of Hiromasa et al. [20] (see Section 3.1) and Genise et al. [16]. For a finite automaton with 1024 states, they chose a 42-bit modulus q. Note that such an overstretched modulus is potentially dangerous in the GGHLM19 setting and does not satisfy the claimed security level as was shown by Lee and Wallet [22]. Nevertheless, the HAO15 scheme is assumed to be secure for these parameters and allows for input words of length up to roughly 140000 symbols.

In our case, we achieve obfuscated evaluation of any evasive DFA with sufficient min-entropy. Since we require zero-testing, we obtain a slightly smaller maximal grading. For HAO15, recall Equation (3.2) which we can use to compute the maximal grading if we encode DFA matrices. With a zero-testing primitive, the same parameters as above ($n = 1024, r = 1024, q \approx 2^{42}$) yield a maximal input word length of roughly 10^5 symbols. This is already more than enough for the applications that we described in the introduction, such as substring matching or virus testing.

Desmoulins et al. [13] consider pattern matching on encrypted streams. For this, they construct a searchable encryption scheme based on public key encryption and bilinear pairings. This approach is sensible when the original data needs to be protected by encryption. This is different to the situation we consider since we only wish to protect the substring pattern.

8 Conclusion

We have introduced a new special purpose obfuscator for deterministic finite automata that in particular solves the problem of obfuscated substring matching. We have shown that the obfuscator is VBB secure and perfect circuit-hiding based on a new computational assumption involving the HAO15 FHE matrix scheme.

Open problems include generalisations of substring matching such as securely matching biometric information (for example DNA). This problem seems to be related to obfuscating fuzzy matching with respect to edit distance.

References

[1] Prabhanjan Ananth, Aayush Jain, Moni Naor, Amit Sahai, and Eylon Yogev. "Universal Constructions and Robust Combiners for Indistinguishability Obfuscation and Witness Encryption". In: *CRYPTO 2016*. Springer, 2016, pp. 491–520.

[2] Prabhanjan Ananth, Aayush Jain, Moni Naor, Amit Sahai, and Eylon Yogev. *Universal Obfuscation and Witness Encryption: Boosting Correctness and Combining Security.* Cryptology ePrint Archive, Report 2016/281. https://eprint.iacr.org/2016/281. 2016.

[3] José L. Balcázar, Josep Díaz, Ricard Gavaldà, and Osamu Watanabe. "Algorithms for Learning Finite Automata from Queries: A Unified View". In: *Advances in Algorithms, Languages, and Complexity.* Springer, 1997, pp. 53–72.

[4] Boaz Barak, Nir Bitansky, Ran Canetti, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. "Obfuscation for Evasive Functions". In: *TCC 2014*. Springer, 2014, pp. 26–51.

[5] James Bartusek, Tancrède Lepoint, Fermi Ma, and Mark Zhandry. "New Techniques for Obfuscating Conjunctions". In: *EUROCRYPT 2019*. Springer, 2019, pp. 636–666.

[6] Allison Bishop, Lucas Kowalczyk, Tal Malkin, Valerio Pastro, Mariana Raykova, and Kevin Shi. "A Simple Obfuscation Scheme for Pattern-Matching with Wildcards". In: *CRYPTO 2018*. Springer, 2018, pp. 731–752.

[7] Zvika Brakerski and Guy N Rothblum. "Obfuscating Conjunctions". In: *Journal of Cryptology* 30.1 (2017), pp. 289–320.

[8] Zvika Brakerski and Guy N. Rothblum. "Obfuscating Conjunctions". In: *CRYPTO 2013*. Springer, 2013, pp. 416–434.

[9] Zvika Brakerski and Guy N. Rothblum. "Virtual Black-Box Obfuscation for All Circuits via Generic Graded Encoding". In: *TCC 2014*. Springer, 2014, pp. 1–25.

[10] Zvika Brakerski, Vinod Vaikuntanathan, Hoeteck Wee, and Daniel Wichs. "Obfuscating Conjunctions under Entropic Ring LWE". In: *ITCS 2016*. ACM. 2016, pp. 147–156.

[11] Ran Canetti, Guy N Rothblum, and Mayank Varia. "Obfuscation of Hyperplane Membership". In: *TCC 2010*. Springer. 2010, pp. 72–89.

[12] Yilei Chen, Vinod Vaikuntanathan, and Hoeteck Wee. "GGH15 Beyond Permutation Branching Programs: Proofs, Attacks, and Candidates". In: *CRYPTO 2018*. Springer, 2018, pp. 577–607.

[13] Nicolas Desmoulins, Pierre-Alain Fouque, Cristina Onete, and Olivier Sanders. "Pattern Matching on Encrypted Streams". In: *ASIACRYPT 2018*. Springer, 2018, pp. 121–148.

[14] Steven D. Galbraith and Lukas Zobernig. "Obfuscated Fuzzy Hamming Distance and Conjunctions from Subset Product Problems". In: *TCC 2019*. Springer, 2019, pp. 81–110.

[15] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. "Candidate Indistinguishability Obfuscation and Functional Encryption for all Circuits". In: *SIAM Journal on Computing* 45.3 (2016), pp. 882–929.

[16] Nicholas Genise, Craig Gentry, Shai Halevi, Baiyu Li, and Daniele Micciancio. "Homomorphic Encryption for Finite Automata". In: *ASIACRYPT 2019*. Springer, 2019, pp. 473–502.

[17] Craig Gentry, Sergey Gorbunov, and Shai Halevi. "Graph-Induced Multilinear Maps from Lattices". In: *TCC 2015*. Springer, 2015, pp. 498–527.

[18] Shafi Goldwasser, Yael Kalai, Chris Peikert, and Vinod Vaikuntanathan. "Robustness of the Learning with Errors Assumption". In: *Innovations in Computer Science*. 2010, pp. 230–240.

[19] Rishab Goyal, Venkata Koppula, and Brent Waters. "Lockable Obfuscation". In: *FOCS 2017*. IEEE, 2017, pp. 612–621.

[20] Ryo Hiromasa, Masayuki Abe, and Tatsuaki Okamoto. "Packing Messages and Optimizing Bootstrapping in GSW-FHE". In: *PKC 2015*. Springer, 2015, pp. 699–715.

[21] Nikolay Kuzurin, Alexander Shokurov, Nikolay Varnovsky, and Vladimir Zakharov. "On the Concept of Software Obfuscation in Computer Security". In: *ISC 2007*. Springer, 2007, pp. 281–298.

[22] Changmin Lee and Alexandre Wallet. *Lattice analysis on MiNTRU problem*. Cryptology ePrint Archive, Report 2020/230. https://eprint.iacr.org/2020/230. 2020.

[23] Ben Lynn, Manoj Prabhakaran, and Amit Sahai. "Positive Results and Techniques for Obfuscation". In: *EUROCRYPT 2004*. Springer, 2004, pp. 20–39.

[24] Daniele Micciancio and Chris Peikert. "Trapdoors for Lattices: Simpler, Tighter, Faster, Smaller". In: *EUROCRYPT 2012*. Springer, 2012, pp. 700–718.

[25] Daniel Wichs and Giorgos Zirdelis. "Obfuscating Compute-and-Compare Programs under LWE". In: *FOCS 2017*. IEEE. 2017, pp. 600–611.

[26] Daniel Wichs and Giorgos Zirdelis. *Obfuscating Compute-and-Compare Programs under LWE*. Cryptology ePrint Archive, Report 2017/276. https://eprint.iacr.org/2017/276. 2017.

Appendix

A Matrix (Graded) Encoding Scheme

In this section we will give a condensed version of a matrix (graded) encoding scheme which describes the essential parts as introduced by Brakerski and Rothblum [8, 9]. A matrix graded encoding scheme consists of the following algorithms:

- *Key Generation.* Given a matrix dimension $n \in \mathbb{N}$, a compatible security parameter $\lambda \in \mathbb{N}$, and a maximal grading $\kappa \in \mathbb{N}$, the key generation algorithm outputs a secret key $\mathbf{sk}$ and public key $\mathbf{pk}$.
- *Matrix Encoding.* Given a matrix $M \in \{0,1\}^{n \times n}$, the encoding algorithm uses the secret key $\mathbf{sk}$ to output a (possibly *randomised*) encoding C of M.
- *Vector Encoding.* Given a vector $v \in \{0,1\}^n$, the encoding algorithm uses the secret key $\mathbf{sk}$ to output a (possibly *randomised*) encoding c of v.
- *Zero-testing.* Given an encoded matrix C or vector c, the zero-testing algorithm uses the public key $\mathbf{pk}$ to decide whether C or c is an encoding of the zero matrix or zero vector, respectively.

There are algorithms that compute the *sum* and *product* operations of two encodings, we will abbreviate them with standard mathematical notation. The homomorphic properties of the encoded matrices should be as follows:

- *Additive.* Given two encodings C_1, C_2 of M_1, M_2, it holds that the encoding of $M_1 + M_2$ equals $C_1 + C_2$ (up to randomisation).
- *Multiplicative.* Given encodings $C_1, C_2, \ldots, C_i$ of $M_1, M_2, \ldots, M_i$, where $i \leq \kappa$, it holds that the encoding of $M_1 M_2 \cdots M_i$ equals $C_1 C_2 \cdots C_i$ (up to randomisation).
- *Applying Matrix to Vector.* Given encodings $C_1, C_2, \ldots, C_i$ of $M_1, M_2, \ldots, M_i$, where $i \leq \kappa - 1$, and an encoded vector c of v, it holds that the encoding of $M_1 M_2 \cdots M_i v$ equals $C_1 C_2 \cdots C_i c$ (up to randomisation).

In the more general definition of Brakerski and Rothblum [9], there is a grading that we can attach to each encoding. Then it is only possible to add encodings at the same *level* to produce another encoding of the same level. When multiplying elements of different levels, say ℓ_1 and ℓ_2, we produce an encoding of a higher level, for example $\ell_1 + \ell_2$. We should think of the public key $\mathbf{pk}$ as a collection of individual zero-testing keys $\mathbf{pk} = \{\mathbf{pk}_\ell\}_{\ell \in L}$. Concretely, for a fixed level ℓ, if the public key contains $\mathbf{pk}_\ell \in \mathbf{pk}$ then we may zero-test encodings of level ℓ. The downside is that in all instantiations such a grading implies much larger public keys. We will avoid this at the cost of an additional *circular security* assumption.

B GGH15

In this section we remind the reader of the matrix graded encoding scheme by Gentry et al. [17]. We are working over the ring $R = \mathbb{Z}/q\mathbb{Z}$ for some modulus q. Let $m, n \in \mathbb{N}$ be matrix dimensions.

Matrix Encoding. The key idea is the following: Choose a matrix $A \in R^{n \times m}$, a secret matrix $S \in R^{n \times n}$ with small entries is encoded as a matrix $C \in R^{m \times m}$ with small entries such that

$$AC = SA + E \tag{B.1}$$

for some small error matrix $E \in R^{n \times m}$.

Key Generation. Sampling an encoding C as in Equation (B.1) generally requires a lattice trapdoor, such as given by Micciancio and Peikert [24] for example. In practice, depending on a security parameter $\lambda \in \mathbb{N}$, we fix a modulus q, and matrix dimensions $n, m \in \mathbb{N}$. The private key is then the trapdoor and the public key is the matrix A. The small matrices are sampled from a β-bounded distribution χ. Fix a maximal grading $\kappa \in \mathbb{N}$, then the modulus should satisfy $q > (4m\beta)^\kappa \lambda^{\omega(1)}$ which we require for security and additionally for a κ grading.

Vector Encoding. Similarly, given a secret vector $s \in R^n$ with small entries, we can encode it by sampling a short vector $c \in R^m$ according to $Ac = sA + e$ for some short error vector $e \in R^m$.

Homomorphic Operations. This construction is additively and multiplicatively homomorphic. Take two encodings such that $AC_1 = S_1A + E_1$ and $AC_2 = S_2A + E_2$, then we have

$$A(C_1 + C_2) = (S_1 + S_2)A + E' \tag{B.2}$$

for some small E'. Obviously we can only add a finite number of such encodings before the error grows too big. Similarly, for the multiplication of two encodings we have

$$AC_1C_2 = (S_1A + E_1)C_2 = S_1(S_2A + E_2) + E_1C_2 = S_1S_2A + E' \tag{B.3}$$

for some small E'. Finally, applying an encoded matrix C to a vector c works via the identity

$$ACc = (SA + E)c = S(sA + e) + Ec = SsA + e'. \tag{B.4}$$

Zero-testing. Given an encoding C of a secret S at *multiplicative level* ℓ such that the error E is bounded by $\|E\|_\infty \leq \beta(2m\beta)^{\ell-1}$, zero-testing is possible. Compute AC and test whether $\|AC\|_\infty \leq \beta(2m\beta)^{\ell-1}$. If $S = 0$ then this test succeeds and if $S \neq 0$ then $\|AC\|_\infty > \beta(2m\beta)^{\ell-1}$ with high probability.

Error Bounds and Correctness. Assuming $\|C\|_\infty, \|S\|_\infty, \|E\|_\infty \leq \beta$ for some threshold β, it is immediately clear from Equation (B.2) that after adding two encodings, the resulting error is bounded by $\|E'\|_\infty \leq 2\beta$. Similarly, from Equation (B.3) we see that after multiplying two encodings, the resulting error is bounded by $\|E'\|_\infty \leq 2m\beta^2$ (and also for the secret S_1S_2 and encoding C_1C_2).

We said that the maximal grading of the encoding scheme with our choice of parameters is κ. By induction we find that after multiplying κ encodings, the error is bounded by $\beta(2m\beta)^{\kappa-1}$. Now assume $\|c\|_\infty, \|s\|_\infty, \|e\|_\infty \leq \beta$ for an encoding c of a vector s. Finally, by induction, from Equation (B.4) we find that after applying a sequence of $\kappa - 1$ matrices to an encoded vector, the resulting error is bounded by $\|e'\|_\infty \leq \beta(2m\beta)^{\kappa-1}$, see also [25].

Security. Chen et al. [12] considered the GGH15 encoding scheme from the viewpoint of obfuscation for *matrix branching programs*. They give rules about the form of the secret matrices S such that security can be reduced to the LWE assumption for lattices. To encode arbitrary matrices M, they give an embedding of M into a larger matrix S that is still compatible with matrix-multiplication. Chen et al. [12] showed that their generalised GGH15 encodings for branching programs are secure under LWE. They are using the stronger graded encoding scheme model which we mentioned in Section A that restricts homomorphic interaction between levels. Specifically, in the general GGH15 scheme, they encode a secret S along a *path* (i, j) such that $A_jC = SA_i + E$ for different random matrices A_i, A_j. In the end we only publish the very first A_1 that is required for the final zero-test.

In our setting, we set all those matrices A_i equal to a single matrix A, except for a special *final* matrix M_f which we encode with respect to a different matrix B such that $BC_f = M_fA + E$. We do this because unlike circuits, which can be translated into matrix branching programs of a fixed depth, DFAs usually have loops that connect states to themselves under input of certain symbols. We will keep the matrix A secret and only publish B such that we are forced to apply the final matrix M_f before zero-testing. Hence, we need to assume circular security for the encodings. This also shrinks the size of the public parameters and allows for a much larger number of DFA inputs in our application.

C General Encoding Schemes

In Section 6 we gave a specialised construction based on our extension of the HAO15 matrix FHE scheme (recall Section 3.1). In doing so, we were able to give a security reduction from VBB and perfect circuit-hiding to the decisional assumption of Section 4. In this section we want to sketch a generic construction for obfuscated evasive DFAs from arbitrary matrix graded encoding schemes. We will refrain from giving a security reduction to a generic assumption. This should rather be investigated on a case-by-case basis.

We will assume that we are given a matrix graded encoding scheme (with maximal grading κ) such as described in Section 3. The obfuscator runs the key generation algorithm such that the secret key **sk** allows to encode matrices at and between two subsequent levels. We require that the public key **pk** allows for zero-testing only at the second level.

The obfuscator takes as an input an evasive DFA represented by the transition matrices $(M_\sigma)_{\sigma\in\Sigma}$ and outputs the tuple $(s_r, (C_\sigma)_{\sigma\in\Sigma}, c)$. In the output tuple, c is an encoding of the first canonical basis vector e_1 at the first level and z is an encoding of the r-th canonical basis vector e_r at the second level, $(C_\sigma)_{\sigma\in\Sigma}$ is the sequence of encodings of the state transition matrices $(M_\sigma)_{\sigma\in\Sigma}$ at the first level and C_f is an encoding of the final matrix M_f between the first and second level. We assume that the initial and accepting state of the finite automaton are given by the state 1 and state r, respectively. If necessary, transform the DFA matrices $(M_\sigma)_{\sigma\in\Sigma}$ to satisfy the safeguard requirements described in Section 5.1. See Algorithm C.1 for an algorithmic description.

Algorithm C.1 Encoding (Obfuscating the finite automaton)

---

procedure ENCODE($(M_\sigma)_{\sigma\in\Sigma}$)
  Run matrix graded encoding scheme key generation and obtain **sk**, **pk**.
  Compute $(C_\sigma)_{\sigma\in\Sigma}$ by encoding $(M_\sigma)_{\sigma\in\Sigma}$ at the first level using **sk** (no zero-testing possible).
  Compute C_f by encoding M_f at the second level using **sk** (zero-testing possible).
  Compute state vectors c and z by encoding e_1 at the first and e_r at the second level using **sk**, respectively.
  return (**pk**, $(C_\sigma)_{\sigma\in\Sigma}, c, C_f, z$)
end procedure

We may evaluate the obfuscated automaton on a word $w \in \Sigma^*$ with $|w| \leq \kappa$ as follows:

1. Compute the encoded vector c_w corresponding to $(\prod_{i=1}^{|w|} M_{w_i})e_1$ using the sequence $(C_\sigma)_{\sigma\in\Sigma}$ and the encoded initial state c.
2. Evaluate the zero-test using **pk** on $C_f c_w - z$. The word w is accepted by the automaton represented by $(M_\sigma)_{\sigma\in\Sigma}$ if the zero-test succeeds and we output 1 in this case, 0 otherwise.

See Algorithm C.2 for an algorithmic description.

Algorithm C.2 Evaluation (Executing the obfuscated program)

---

procedure EVALUATE($s_r, (C_\sigma)_{\sigma\in\Sigma}, c; w \in \Sigma^*$)
  Initialize the state vector $s = c$.
  for all $i = 1, \ldots, |w|$ **do**
    Update the state vector $s = C_{w_i} s$.
  end for
  Evaluate the zero-test using **pk** on $C_f s - z$.
  return 1 **if** the zero-test failed **else** 0
end procedure

We argue that one should consider VBB or perfect circuit-hiding obfuscation instead of iO for evasive finite automata. One important reason is the possibility of *zeroising attacks*. This class of attacks affects several obfuscation constructions based on graded encoding schemes. The idea is that given an encoding of zero, we get a system of equations over $\mathbb{Z}$ instead of $\mathbb{Z}/q\mathbb{Z}$ that depend not only on small error terms but also on the secret matrices themselves. This seems to be especially problematic for iO schemes which are the prevalent constructions using graded encoding schemes.