Semantically Secure Order-Revealing Encryption: Multi-Input Functional Encryption Without Obfuscation

Dan Boneh[1], Kevin Lewi[1], Mariana Raykova[2], Amit Sahai[3], Mark Zhandry[1], and Joe Zimmerman[1]

[1]Stanford University
[2]SRI
[3]Computer Science, UCLA and Center for Encrypted Functionalities

Abstract

Deciding "greater-than" relations among data items just given their encryptions is at the heart of search algorithms on encrypted data, most notably, non-interactive binary search on encrypted data. Order-preserving encryption provides one solution, but provably provides only limited security guarantees. Two-input functional encryption is another approach, but requires the full power of obfuscation machinery and is currently not implementable.

We construct the first implementable encryption system supporting greater-than comparisons on encrypted data that provides the "best-possible" semantic security. In our scheme there is a public algorithm that given two ciphertexts as input, reveals the order of the corresponding plaintexts and nothing else. Our constructions are inspired by obfuscation techniques, but do not use obfuscation. For example, to compare two 16-bit encrypted values (e.g., salaries or age) we only need a 9-way multilinear map. More generally, comparing k-bit values requires only a $(k/2+1)$-way multilinear map. The required degree of multilinearity can be further reduced, but at the cost of increasing ciphertext size.

Beyond comparisons, our results give an implementable secret-key multi-input functional encryption scheme for functionalities that can be expressed as (generalized) branching programs of polynomial length and width. Comparisons are a special case of this class, where for k-bit inputs the branching program is of length $k+1$ and width 4.

1 Introduction

Functional encryption [BSW11] is a public-key encryption system that supports "partial" decryption keys: decrypting a ciphertext $c = E(\mathsf{pk}, m)$ using a key sk_f reveals $f(m)$ and nothing else. Multi-input functional encryption [GGG+14] is a generalization of functional encryption where the key sk_f acts on ℓ ciphertexts $c_1 = E(\mathsf{pk}, m_1), \ldots, c_\ell = E(\mathsf{pk}, m_\ell)$ to reveal $f(m_1, \ldots, m_\ell)$ and nothing else. Existing constructions for general multi-input functional encryption are based on obfuscation and thus are not currently feasible to implement, even for simple functionalities.

In this paper we present a construction for *secret-key* multi-input functional encryption from multilinear maps. By restricting our attention to the secret-key setting, we are able to achieve a much more efficient construction, without the full machinery of obfuscation and NIZK proofs.

For concreteness, in the introduction we present our results as they apply to a specific application called *order-revealing encryption* [AKSX04, BCLO09, BCO11]. The paper body presents the results in their full generality, namely as a *secret-key* multi-input functional encryption scheme.

1.1 Order-revealing encryption

Definition. A secret-key encryption scheme is *order-revealing*[1] [BCO11] if there is a public procedure that takes two *encrypted* plaintexts as input and reports their lexicographic ordering. This procedure, which we call the order-revealing algorithm, requires no secrets and can be evaluated by anyone. More precisely, an order-revealing scheme is a tuple (G, E, D) of algorithms. Algorithm G outputs a pair $(\mathsf{sk}, \mathsf{comp})$ where sk is a secret encryption key and $\mathsf{comp}(\cdot,\cdot)$ is an efficient deterministic algorithm that takes two ciphertexts as input and outputs either '$<$' or '$\geq$'. Algorithms $E(\mathsf{sk}, m)$ and $D(\mathsf{sk}, c)$ are standard encryption/decryption algorithms where $m \in \{0, \ldots, B\}$ for some B. In addition to the standard correctness of decryption we also require that for all $(\mathsf{sk}, \mathsf{comp})$ output by G and for all plaintexts m_0, m_1 we have:

$$\begin{aligned} m_0 < m_1 \quad &\Longrightarrow \quad \Pr[\mathsf{comp}(\ E(\mathsf{sk}, m_0)\ ,\ E(\mathsf{sk}, m_1)\ ) = '<'] = 1 \\ m_0 \geq m_1 \quad &\Longrightarrow \quad \Pr[\mathsf{comp}(\ E(\mathsf{sk}, m_0)\ ,\ E(\mathsf{sk}, m_1)\ ) = '\geq'] = 1 \end{aligned}$$

An order-revealing encryption scheme is secure if a ciphertext reveals nothing about the corresponding plaintext beyond its lexicographic relation relative to other ciphertexts. This is defined using a simple variant of the standard semantic security game [GM82]: the adversary is given algorithm $\mathsf{comp}(\cdot,\cdot)$ and access to a "left-right-oracle" $\mathcal{O}(\cdot,\cdot)$ that on input (m_0, m_1) returns $E(\mathsf{sk}, m_b)$ for some $b \in \{0, 1\}$ chosen at the beginning of the game. After adaptively querying the oracle $\mathcal{O}$ the adversary outputs a guess b' and wins the game if $b = b'$. Let $(m_0^{(0)}, m_1^{(0)}), \ldots, (m_0^{(q)}, m_1^{(q)})$ be the adversary's queries to $\mathcal{O}$. To ensure that the adversary cannot use algorithm $\mathsf{comp}(\cdot,\cdot)$ to trivially win the game we require that the relative ordering of messages on the left is the same as the relative ordering on the right, namely for all $0 \leq i, j \leq q$:

$$m_0^{(i)} < m_0^{(j)} \quad \Longleftrightarrow \quad m_1^{(i)} < m_1^{(j)}$$

The scheme is secure if the adversary cannot win this game with non-negligible advantage. We refer to this notion as *best-possible semantic security*. We give a complete (and more general) definition in Section 4.

Note that a *public-key* order-revealing encryption scheme is impossible: if an adversary has unrestricted access to the encryption algorithm, he can use the encryption algorithm and the order-revealing algorithm $\mathsf{comp}(\cdot,\cdot)$ to decrypt any ciphertext using binary search without the secret key.

Applications. Order-revealing encryption (ORE) is motivated by the problem of answering range queries on a remote encrypted database [AKSX04, BCLO09]. Consider a remote database holding encrypted pairs $(\text{name}, \text{salary})$. The data owner wishes to retrieve all records with a salary greater than t. If salaries are encrypted using an ORE then the database can sort all records on its own from lowest salary to highest. This sorting can be done even when records are inserted sequentially into the database (perhaps by multiple users who share the secret encryption key) and requires no interaction with the data owner(s). To issue the range query the data owner sends the encryption of t under the ORE key. In response, the database first uses binary search on the encrypted salaries to locate the smallest encrypted record R with a salary greater then t and then simply sends all records to the "right" of R back to the user. Thus, for a database of n records, the database's work is $O(\log n)$ and requires only one round of interaction with the client, as in the case of a cleartext database. Security of the ORE ensures that the database learns nothing beyond the relative ordering of records and queries.

[1] In [BCO11] order revealing encryption was called "efficiently-orderable encryption."

Alternate approaches. Before describing our construction we briefly survey a few alternate constructions for answering range queries on a remote encrypted database.

Boldyreva et al. [BCLO09, BCO11] describe an elegant primitive called Order Preserving Encryption (OPE) where encryption preserves the relative ordering of plaintexts. Comparing encrypted data is then done by simply comparing the corresponding ciphertexts. However, OPE leaks information about the relative distances of plaintexts. Recent work of Malkin et al. [MTY13] constructs an OPE scheme with a partial security guarantee, hiding the low-order bits of plaintexts, but still does not achieve best-possible semantic security. Indeed, Boldyreva et al. [BCLO09] prove that no OPE scheme can possibly achieve best-possible semantic security. In ORE, unlike OPE, comparisons are done with a dedicated algorithm $\mathsf{comp}(\cdot,\cdot)$ which is the reason best-possible semantic security can be achieved.

A very different approach to answering range queries on encrypted data uses garbled RAMs [LO13, GHL$^+$14]. With garbled RAMs the database can answer range queries without learning any information about the data, but answering the range queries requires more rounds of interaction per query and the database's work is higher than with ORE.

Other approaches to answering range queries are based on public-key predicate encryption [BW07, SBC$^+$07, KSW08] and require a linear scan through the database. With ORE, range queries can be answered in logarithmic time in the size of the database. We also mention a result of Popa et al. [PLZ13] who describe an interactive protocol for answering range queries. Interaction is used to maintain a sorted data structure at the database by offloading some comparisons to the client. Finally, we note that ORE is a special case of secret-key two-input functional encryption [GGG$^+$14].

1.2 Order revealing encryption: our construction

Our construction begins with a simple automaton for the comparison function on two inputs that we represent as a low-width matrix branching program. We encrypt ciphertexts in a way such that given two independently-created ciphertexts, anyone can run the comparison branching program to reveal the relative ordering of the corresponding plaintexts. While our encryption scheme applies to any multi-input functionality expressed as a matrix branching program (see Section 2.2), for the rest of this section we use the two-input comparison automaton and its branching program as a concrete example to illustrate the construction.

The comparison automaton and branching program. Figure 1 shows a five-state automaton A that computes the ordering of two inputs $x = x_1x_2\cdots x_n$ and $y = y_1y_2\ldots y_n$ in $\{0,1\}^n$ when the input is processed in an interleaved order (of the form $x_1y_1x_2y_2\cdots x_ny_n$). From this automaton we derive four 5×5 matrices $\mathbf{X}_0, \mathbf{X}_1, \mathbf{Y}_0, \mathbf{Y}_1$, where each is the adjacency matrix of a subgraph of A: for $b\in\{0,1\}$, the matrix $\mathbf{X}_b$ is the adjacency matrix of the subgraph consisting only of the b-transitions used by input bits of x, and the matrix $\mathbf{Y}_b$ is the adjacency matrix of the subgraph consisting only of the b-transitions used by input bits of y. Note that these matrices are not invertible because of the sink states in the automaton. This introduces additional challenges in the security proof; however, we are able to handle branching programs with non-invertible matrices using recent results of Sahai and Zhandry [SZ14].

Let $\mathbf{e}_i$ be the 5-vector containing 1 in position i and zero elsewhere. Then the product $\mathbf{e}_1^\intercal\cdot\prod_{i=1}^n(\mathbf{X}_{x_i}\mathbf{Y}_{y_i})$ results in a vector with a single "1" in three possible locations (corresponding to either the "$x>y$", "$x<y$", or "$=$" final states), and the location of the "1" determines the result of the comparison operation on x and y. Hence, the matrices $\mathbf{X}_0, \mathbf{X}_1, \mathbf{Y}_0, \mathbf{Y}_1$ form a matrix branching program for the two-input comparison function. In Section 3 we show that a simple re-ordering of the inputs reduces the matrix program length to only $n+1$ matrices each of dimension 4×4, but for simplicity we ignore this optimization here.

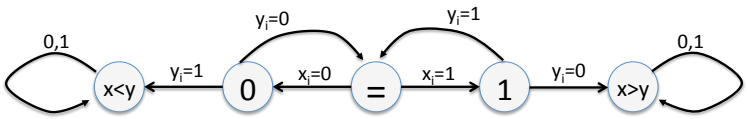

Figure 1: The 5-state comparison automaton on inputs $x, y \in \{0,1\}^n$ where '=' is the start state. Input bits are processed in an interleaved order $x_1\ y_1\ x_2\ y_2 \ldots$

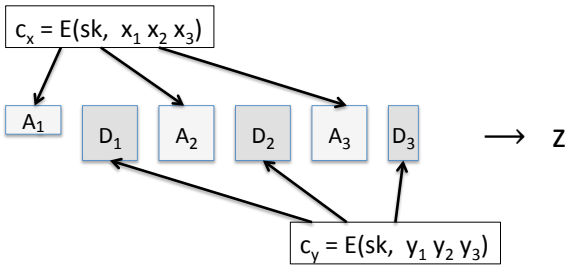

Figure 2: The order-revealing algorithm applied to encryptions of $x_1x_2x_3$ and $y_1y_2y_3$.

The ORE encryption scheme. Fix a prime q. The setup algorithm G uniformly samples $2n-1$ invertible matrices $\mathbf{R}_1, \ldots, \mathbf{R}_{2n-1}$ from $\mathrm{GL}_5(\mathbb{Z}_q)$. These matrices form the secret encryption key sk. During encryption these matrices will be used to randomize the matrices of the comparison branching program using Kilian's randomization technique [Kil88]. We define two additional vectors $\mathbf{R}_0 := \mathbf{e}_1^\intercal$ and $\mathbf{R}_{2n} := \mathbf{e}_5$. The secret key also contains the parameters for an asymmetric multilinear map [GGH13a] with $2n$ indices (i.e., of degree $2n$). We divide the $2n$ indices into two disjoint size-n sets $\mathcal{U}_1$ and $\mathcal{U}_2$.

The encryption algorithm encrypts a plaintext $x = x_1x_2\cdots x_n \in \{0,1\}^n$ as follows. It first samples a partition $(S_1, \ldots, S_n)$ of $\mathcal{U}_1$ and a partition $(T_1, \ldots, T_n)$ of $\mathcal{U}_2$. These partitions are sampled at random from a family of partitions we call an "exclusive partition family." They must satisfy a specific combinatorial property needed to prevent certain "mix-and-match" attacks where the attacker tries to run the comparison algorithm on improperly formed ciphertexts. We define and construct these partition families in Section 2.5. They are a generalization of the "straddling sets" used in Barak et al. [BGK+14].

Next, the encryption algorithm samples random scalars $\alpha_1, \ldots, \alpha_{2n} \in \mathbb{Z}_q^*$ and constructs the 5×5 matrices

$$\hat{\mathbf{X}}_i = \alpha_i \cdot (\mathbf{R}_{2i-2}\ \mathbf{X}_{x_i}\ \mathbf{R}_{2i-1}^{-1}) \qquad \text{and} \qquad \hat{\mathbf{Y}}_i = \alpha_{n+i} \cdot (\mathbf{R}_{2i-1}\ \mathbf{Y}_{x_i}\ \mathbf{R}_{2i}^{-1})$$

for $i \in [n]$ where we define $\mathbf{R}_{2n}^{-1} := \mathbf{e}_5$. Recall that the matrices $\mathbf{R}_i$ are taken from the secret key and the matrices $\mathbf{X}_0, \mathbf{X}_1$ and $\mathbf{Y}_0, \mathbf{Y}_1$ are the matrices in the comparison branching program. Because $\mathbf{R}_0$ and $\mathbf{R}_{2n}$ are vectors, so are $\hat{\mathbf{X}}_0$ and $\hat{\mathbf{Y}}_n$. All other ciphertext components are square matrices.

Finally, for $i \in [n]$ the encryption algorithm encodes the entries of $\hat{\mathbf{X}}_i$ under the index set S_i of the multilinear map, and encodes the entries of $\hat{\mathbf{Y}}_i$ under the index set T_i. The resulting $2n$ encoded 5×5 matrices $(\{\hat{\mathbf{X}}_i\}_{i=1}^n, \{\hat{\mathbf{Y}}_j\}_{j=1}^n)$ are output as the encryption of $x \in \{0,1\}^n$.

The order-revealing algorithm. Given two independently-created ciphertexts c_x and c_y corresponding to plaintexts x and y, the order-revealing algorithm computes the interleaved product of the matrices in the left half of c_x with the matrices in the right half of c_y. In other words, if $c_x = (\{\mathbf{A}_i\}_{i=1}^n, \{\mathbf{B}_j\}_{j=1}^n)$ and $c_y = (\{\mathbf{C}_i\}_{i=1}^n, \{\mathbf{D}_j\}_{j=1}^n)$ then $z = \mathbf{A}_1\mathbf{D}_1\mathbf{A}_2\mathbf{D}_2\cdots\mathbf{A}_n\mathbf{D}_n$, as shown in Figure 2. We compute z using the

multilinear map and the result is a single group element (a scalar) because $\mathbf{A}_1$ and $\mathbf{D}_n$ are vectors. Finally, the algorithm zero-tests z and the outcome reveals the ordering of x and y. Zero-testing this z is possible because it is an encoding of an element under the full $2n$ index set, by the structure of the partitions.

To verify that the final zero-test correctly reveals the ordering of x and y, observe that the scalar z expands to the quantity

$$\left(\mathbf{e}_1^\intercal \, \mathbf{X}_{x_1} \, \mathbf{R}_1^{-1}\right) \left(\mathbf{R}_1 \mathbf{Y}_{y_1} \mathbf{R}_2^{-1}\right) \cdots \left(\mathbf{R}_{2n-2} \mathbf{X}_{x_n} \mathbf{R}_{2n-1}^{-1}\right) \left(\mathbf{R}_{2n-1} \mathbf{Y}_{y_n} \mathbf{e}_5\right) \tag{1.1}$$

Hence, z takes on a non-zero value if and only if the comparison automaton terminates in the state "$x < y$". Note that we omitted the scalars α_i in the expansion (1.1) for ease of exposition. Their presence causes z to be either 0 or non-zero, as opposed to 0 or 1.

Security. We prove the security of a generalization of this construction in the generic multilinear map model [GGH$^+$13b, BR14, BGK$^+$14]. The use of Kilian's randomization technique in the encryption key restricts the adversary's ability to manipulate ciphertext components in an elementary manner, such as by computing products of matrices out of order. Also, the use of the random scalars $\alpha_1, \ldots, \alpha_{2n}$ prevents the adversary from correlating multiple encryptions of plaintexts which share the same bit pattern. However, there is still a large domain of attacks that the adversary could potentially take advantage of. For example, an adversary can combine components from multiple ciphertexts to look for relations, or he can compare the results of partial evaluations of the branching program on different inputs.

In order to handle these types of attacks, we use the combinatorial structure provided by our exclusive partition families. Intuitively, the use of a random partition from an exclusive partition family for each ciphertext ensures that if the adversary computes a partial evaluation of the branching program, or tries to mix components from multiple ciphertexts, he will not be able to obtain a group element which is encoded in the index set for the zero-tester, as required by the generic multilinear map model. In fact, it turns out that the use of these exclusive partition families is indeed sufficient to prove security of the construction in the generic model.

Performance. Our basic construction requires a $(2n + 2)$-way multilinear map to evaluate comparisons on n-bit numbers. However, simple optimizations, including re-ordering of the matrices in the branching program, enables us to shrink the total length of the comparison branching program to only $(n + 1)$ matrices each of dimension 4×4 (see Section 2.2 for details). Consequently, we only need an $(n+1)$-way multilinear map to evaluate comparisons on n-bit numbers. The secret encryption key contains $16n$ elements in $\mathbb{Z}_q$, and each ciphertext is $16n - 8$ encoded group elements. We can further reduce the required degree of multilinearity by a factor of $\log_2 B$ by representing messages in base-B (instead of base-2) and modifying the comparison automaton to compare one base-B digit per step. This shortens the length of the branching program (and therefore the degree of multilinearity) by a factor of $\log_2 B$, but at the cost of increasing the number of states in the automaton by a factor of B and consequently increasing the number of group elements in the ciphertext by a factor of approximately $B^2/\log_2 B$. For example, moving to base $B = 4$ gives multilinearity $(n/2 + 1)$, with ciphertexts requiring $18n - 24$ group elements.

Concretely, for $n = 16$ bits, we can use a 9-linear map giving ciphertexts of 264 group elements. While this scheme is still too inefficient for practical use, the construction *can* be implemented and provides an important step towards more realistic ORE schemes. This is in contrast to the immense number of levels of multilinearity required to obtain ORE from obfuscation-based constructions.

Generalizing to multi-input functional encryption. While we used order-revealing encryption (ORE) as an example application, our construction is more general: it gives a secret-key multi-input functional encryption where the degree of multilinearity needed for decrypting with a key sk_f depends on the length of the branching program representing f. In fact, every matrix in the branching program can depend on *all* the bits of one of the inputs to f and this can be used to shrink the length of the branching program. We refer to these as *generalized branching programs* and define them precisely in the next section.

Our base multi-input functional encryption scheme supports a single function f (such as comparison) fixed a-priori during initial key generation. This function f defines the branching program relative to which all encryptions are computed. This apparent single-function limitation is easily removed using universal circuits: the functionality fixed a-priori is a universal circuit U that takes as input the description of a function f and its inputs $x_1, \ldots, x_n$ and outputs $f(x_1, \ldots, x_n)$. Now, a functional encryption "key" sk_f for a function f is simply the encryption of f under our encryption scheme. Given sk_f and the encryptions of $x_1, \ldots, x_n$ the functionality for the universal circuit U can be used to compute $f(x_1, \ldots, x_n)$ in the clear.

1.3 Other related work

Multi-input functional encryption was introduced by Goldwasser et al. [GGG+14], who gave constructions based on indistinguishability obfuscation [BGI+01, GGH+13b] and differing-inputs obfuscation [BGI+01, BCP14, ABG+13].

Our construction of multi-input functional encryption is inspired by obfuscation techniques [GGH+13b, BBC+14, AGIS14], but does not use obfuscation. Instead we build multi-input functional encryption directly from multilinear maps. Several other results use obfuscation techniques to obtain more efficient constructions directly from multilinear maps. Zhandry [Zha14] showed how to construct n-way Diffie-Hellman key exchange without trusted setup, a result that was previously known only using obfuscation [BZ14]. Concurrently with this work, Garg et al. [GGHZ14] showed how to construct single-input functional encryption from multilinear maps; however, their motivation was to obtain security proofs from concrete assumptions, rather than efficiency. The constructions in this paper are considerably more efficient (we make use of a much smaller number of matrices), but our security proof is in the generic multilinear map model.

Single-input functional encryption [BSW11] has been traditionally defined in the public-key settings and studied extensively [O'N10, GVW12, AGVW13, BO13, CIJ+13, GGH+13b, GKP+13, BCP14]. In this paper, however, we focus on secret-key (multi-input) functional encryption, which is sufficient for data processing on a remote encrypted database, including order-revealing encryption. Focusing on the secret-key setting enables us to give a simple construction from multilinear maps. Single-input *secret-key* functional encryption was previously explored for the inner-product functionality by Shen et al. [SSW09] and more generally by Goldwasser et al. [GKP+13]. Brakerski and Segev [BS14] recently showed how to convert any secret-key functional encryption scheme into one where secret keys do not reveal their functionality.

2 Preliminaries

2.1 Conventions

For an integer n, we write $[n]$ to denote the set $\{1, \ldots, n\}$. For a finite set S, we write $\mathsf{Uniform}(S)$ to denote the probability distribution that is uniform over the elements of S. When working with vectors in $\mathbb{Z}^n$ for some integer n, for each $i \in [n]$ we write $\mathbf{e}_i$ to denote the i^{th} unit column vector, i.e., the vector $(x_1, x_2, \ldots, x_n)^\mathsf{T}$ such that $x_i = 1$ and, for all $i' \neq i \in [n]$, we have $x_{i'} = 0$. We write $\mathrm{GL}_w(\mathbb{Z}_q)$ to represent the set of all $w \times w$ invertible matrices over $\mathbb{Z}_q$.

2.2 Matrix Branching Programs (MBPs)

In this section, we define a variant of matrix branching programs for which our main construction applies. These generalized matrix branching programs are a sequence of efficiently computable Boolean circuits that turn a given multi-variate input into a matrix.

Definition 2.1 (Generalized Matrix Branching Program)**.** Let $\mathcal{X} \subset \{0,1\}^*$ be a set of possible input strings, and let $f : \mathcal{X}^m \to \{0,1\}$ be a multi-input function. A *generalized matrix branching program* P of length ℓ and width w, over $\mathbb{Z}_q$ for a prime q, is a tuple of the form

$$P = (\ q,\ m,\ d,\ \mathsf{inp},\ (\mathcal{M}_1,\ \ldots,\ \mathcal{M}_\ell)\ ),$$

where for each $j \in [\ell]$, the function $\mathcal{M}_j : \mathcal{X} \to \mathbb{Z}_q^{w \times w}$ is computable by an efficient deterministic algorithm. The value inp is a lookup table of the form

$$\mathsf{inp} = (\mathsf{inp}(1),\ \ldots,\ \mathsf{inp}(\ell)),$$

where for each $j \in [\ell]$, we have $\mathsf{inp}(j) \in [m]$. The branching program takes m inputs and we say that at step j it *inspects* input number $\mathsf{inp}(j) \in [m]$. To simplify notation, we require the branching program to inspect each of its m input variables exactly d times[2] (so that the length of the program, ℓ, is precisely md). We also introduce the following shorthand notations:

- For a branching program step $j \in [\ell]$, input slot $i \in [m]$, and sub-index $h \in [d]$, we write $j = \mathsf{inp.j}(i,h)$ to signify that j is the step in which the program inspects input slot i for the h^{th} time.
- For a branching program step $j \in [\ell]$ and sub-index $h \in [d]$, we write $h = \mathsf{inp.h}(j)$ to signify that j is the step in which the program inspects the corresponding input slot $\mathsf{inp}(j)$ for the h^{th} time.

We say that P *computes* the function f if, for all inputs $\mathbf{x} = (x^{(1)}, \ldots, x^{(m)}) \in \mathcal{X}^m$,

$$\left(\prod_{j \in [\ell]} \mathcal{M}_j(x^{(\mathsf{inp}(j))})\right)[1,1] = 0 \quad \iff \quad f(\mathbf{x}) = 1.$$

Since every program P computes a unique function f, we also write $P(\mathbf{x})$ to denote $f(\mathbf{x})$.

Following Sahai and Zhandry [SZ14], we also define the notion of a *non-shortcutting* matrix branching program.

Definition 2.2 (Shortcuts in Matrix Branching Programs [SZ14])**.** A branching program has a *shortcut* on input $\mathbf{x} = (x^{(1)}, \ldots, x^{(m)}) \in \mathcal{X}^m$ if either:

$$\left(\prod_{j \in [\ell]} \mathcal{M}_j(x^{(\mathsf{inp}(j))})\right) \cdot \mathbf{e}_1 = \mathbf{0}_{w \times 1} \qquad \text{or} \qquad \mathbf{e}_1^\intercal \cdot \left(\prod_{j \in [\ell]} \mathcal{M}_j(x^{(\mathsf{inp}(j))})\right) = \mathbf{0}_{1 \times w}$$

[2] We note that this assumption is without loss of generality, since given any program of length ℓ that does not satisfy this condition, we can construct a new program whose value of d is the original program's value of ℓ, and pad the program with dummy matrix functions that always return the identity matrix regardless of their input string. (Alternatively, for practical applications, it is also easy to adapt the techniques we describe to the general case, albeit at the expense of cumbersome notation.)

In such a case, it is possible to determine that $f(\mathbf{x}) = 1$ without carrying out the entire matrix product. We say that a branching program is *non-shortcutting* if, for all inputs $\mathbf{x}$, it has no shortcuts on $\mathbf{x}$. We require that every generalized matrix branching program is non-shortcutting.

We note that there are multiple ways to obtain a generalized matrix branching program from a circuit, or from a time-bounded Turing machine or RAM. Barrington's theorem [Bar86] shows how to convert a Boolean circuit of depth d into a matrix branching program of length $O(4^d)$ and width 5. The work of Ananth, Gupta, Ishai, and Sahai [AGIS14] takes a different approach to obtain MBPs for Boolean formulas that avoids the complexity of Barrington's construction. They construct a layered automaton for any Boolean formula which consists of several states including a starting state and an accepting state together with edges denoting the transitions between states based on the input bit values. Given such an automaton representation, a formula can be evaluated by counting the number of paths between the starting and the accepting state. Ananth et al. show that a Boolean formula of size s can be converted into a layered graph-based branching program with $O(s)$ layers with matrices of size $O(s^2)$. Thus, the size of the resulting MBP is $O(s^3)$. Subsequently, Sahai and Zhandry [SZ14] improve the conversion, giving MBP's of length $O(s)$ and size $O(s(\log_2 s)^2)$. Our approach follows the general method of computing automata with generalized MBPs, but we observe that for some problems such as comparing two-bit strings, we can directly construct extremely efficient automata that do not use the general translation from formulas to automata.

For more details, we refer the reader to Section 3.

2.3 Randomized Matrix Branching Programs

In our construction, as in obfuscation constructions that use MBPs [GGH$^+$13b, BGK$^+$14, BR14, AGIS14], we must make sure that the adversary always evaluates the MBP by multiplying together one matrix selection for each step $j \in [\ell]$. In particular, we must ensure that partial matrix products, which omit some steps, will not reveal any information about the program.

The main ingredient we need here is the MBP randomization technique of Kilian [Kil88], in which we pre- and post-multiply each matrix in the MBP by matching, invertible random "blinding" matrices $\mathbf{R}_0, \ldots, \mathbf{R}_\ell$. Intuitively, the resulting randomized MBP fixes the order in which the randomized MBP matrices can be multiplied, i.e., requiring one matrix for each step in the original MBP. Any other product will also contain at least one random "blinding" matrix, rendering the result useless to the adversary.

In addition, we combine Kilian's randomization technique with "bookend vectors" $\hat{\mathbf{s}}, \hat{\mathbf{t}}$, as introduced in [GGH$^+$13b], which further restrict the adversary to projecting a single scalar entry of the matrix product resulting from the MBP evaluation (namely, the entry at position $[1, 1]$). Testing whether this scalar is zero suffices to determine the Boolean output of the program, while preventing the adversary from learning extra information by testing other matrix entries.

We now present the details of the randomized MBP construction.

Definition 2.3 (Randomized MBPs ([Kil88], adapted))**.** We define an efficient randomized procedure $\mathsf{MBPRand}$, such that, for a given generalized matrix branching program

$$P = (\ q,\ m,\ d,\ \mathsf{inp},\ (\mathcal{M}_1,\ \ldots,\ \mathcal{M}_\ell)\ ),$$

the procedure $\mathsf{MBPRand}(P)$ outputs a tuple $\hat{P}$ of the form

$$\hat{P} = \Big(\ q,\ m,\ d,\ \mathsf{inp},\ (\hat{\mathcal{M}}_1,\ \ldots,\ \hat{\mathcal{M}}_\ell),\ \hat{\mathbf{s}},\ \hat{\mathbf{t}}\ \Big),$$

where, for each $j \in [\ell]$, the function $\hat{\mathcal{M}}_j : \mathcal{X} \to \mathbb{Z}_q^{w \times w}$ is represented, like $\mathcal{M}_j$, as a Boolean circuit; and $\hat{\mathbf{s}}$ and $\hat{\mathbf{t}}$ are vectors in $\mathbb{Z}_q^w$.

The procedure $\mathsf{MBPRand}$ operates as follows. It samples $(\ell+1)$ invertible matrices $\mathbf{R}_0, \ldots, \mathbf{R}_\ell$ uniformly at random from $\mathrm{GL}_w(\mathbb{Z}_q)$. It computes the values

$$\hat{\mathbf{s}} = \mathbf{e}_1^\intercal \, \mathbf{R}_0^{-1} \qquad \text{and} \qquad \hat{\mathbf{t}} = \mathbf{R}_\ell \, \mathbf{e}_1,$$

and, for each $j \in [\ell]$, the function $\hat{\mathcal{M}}_j$ defined as

$$\hat{\mathcal{M}}_j(x) = \mathbf{R}_{j-1}^{-1} \, \mathcal{M}_j(x) \, \mathbf{R}_j.$$

Finally, it outputs the tuple

$$\hat{P} = \Big(\, q, \; m, \; d, \; \mathsf{inp}, \; (\hat{\mathcal{M}}_1, \; \ldots, \; \hat{\mathcal{M}}_\ell), \; \hat{\mathbf{s}}, \; \hat{\mathbf{t}} \, \Big).$$

To evaluate a randomized MBP $\hat{P}$ on an input $\mathbf{x} = (x^{(1)}, \ldots, x^{(m)})$, we run each (randomized) matrix function $\hat{\mathbf{M}}_j$ on the indicated input string $x^{(\mathsf{inp}(j))}$, producing a randomized matrix $\mathbf{M}_j$. We write $\mathsf{MBPSelect}(\hat{P}, \mathbf{x})$ to denote the sequence of randomized matrices and bookend vectors $(\hat{\mathbf{s}}, \mathbf{M}_1, \ldots, \mathbf{M}_\ell, \hat{\mathbf{t}})$, and to evaluate the program, we multiply all of these randomized matrices and vectors together. Formally, we define the following procedures.

Definition 2.4 (Evaluation for Randomized MBPs ([Kil88], adapted))**.** Fix a generalized matrix branching program P and a vector of inputs $\mathbf{x} = (x^{(1)}, \ldots, x^{(m)}) \in \mathcal{X}^m$, and suppose that

$$\hat{P} = \Big(\, q, \; m, \; d, \; \mathsf{inp}, \; (\hat{\mathcal{M}}_1, \; \ldots, \; \hat{\mathcal{M}}_\ell), \; \hat{\mathbf{s}}, \; \hat{\mathbf{t}} \, \Big) \; \leftarrow \; \mathsf{MBPRand}(P).$$

For each $j \in [\ell]$, we define $\hat{\mathbf{M}}_j = \hat{\mathcal{M}}_j(x^{(\mathsf{inp}(j))})$, and we define

$$\mathsf{MBPSelect}(\hat{P}, \mathbf{x}) = \Big(\hat{\mathbf{s}}^\intercal, \; \hat{\mathbf{M}}_1, \ldots, \hat{\mathbf{M}}_\ell, \; \hat{\mathbf{t}} \Big).$$

Finally, we define

$$\mathsf{MBPEval}\Big(\hat{\mathbf{s}}^\intercal, \; \hat{\mathbf{M}}_1, \ldots, \hat{\mathbf{M}}_\ell, \; \hat{\mathbf{t}} \Big) \;\; = \;\; \hat{\mathbf{s}}^\intercal \left(\prod_{j \in \ell} \hat{\mathbf{M}}_j \right) \hat{\mathbf{t}}.$$

Given the above definitions, the proof of the following lemma follows immediately.

Lemma 2.5 (Correctness for Randomized MBPs)**.** *Fix a generalized matrix branching program P, and a vector of inputs $\mathbf{x} = (x^{(1)}, \ldots, x^{(m)}) \in \mathcal{X}^m$. Then,*

$$\mathsf{MBPEval}(\mathsf{MBPSelect}(\hat{P}, \mathbf{x})) = 0 \quad \iff \quad f(\mathbf{x}) = 1.$$

Ordinarily, for MBPs derived from Barrington's theorem [Bar86], we would also be able to state a simulation theorem, showing that the output distribution $\mathsf{MBPSelect}(\hat{P}, \mathbf{x})$ depends only the output of the original program, $P(\mathbf{x})$. In our construction, however, we obtain much more efficient programs by other techniques (Section 3), and the matrices $\mathcal{M}_j(x)$ in these programs do not always have full rank. Indeed, the kernel of each matrix may depend on the input vector $\mathbf{x}$, and as a result, the output distributions $\mathsf{MBPSelect}(\hat{P}, \cdot)$ may be noticeably different for different inputs $\mathbf{x}_0, \mathbf{x}_1$, even if the outputs of the program, $P(\mathbf{x}_0) = P(\mathbf{x}_1)$, are ultimately identical.

Instead of constructing a simulator, we rely on a weaker property that is still strong enough to prove security of our main construction. Specifically, we show that even though the distributions $\mathsf{MBPSelect}(\hat{P}, \mathbf{x}_0)$ and $\mathsf{MBPSelect}(\hat{P}, \mathbf{x}_1)$ may differ, they cannot be distinguished by a certain weak family of tests; in our construction (Section 5), we will show that these are the only tests an adversary can possibly perform in our security model. To define such a family of tests, we refer to the following definition of Sahai and Zhandry [SZ14].

Definition 2.6 (Allowable Tests [SZ14])**.** Let $p : \mathbb{Z}_q^{2w+w^2\ell} \to \mathbb{Z}_q$ be a multilinear (multivariate) polynomial over the entries of $\hat{\mathbf{s}}, \hat{\mathbf{t}} \in \mathbb{Z}_q^w$ and $\hat{\mathbf{M}}_1, \ldots, \hat{\mathbf{M}}_\ell \in \mathbb{Z}_q^{w \times w}$ (as formal variables). We say p is an *allowable* test polynomial if each monomial in the expansion of p contains at most one entry of each vector $\hat{\mathbf{s}}, \hat{\mathbf{t}}$ and matrix $\hat{\mathbf{M}}_1, \ldots, \hat{\mathbf{M}}_\ell$.

Lemma 2.7 (Security for Randomized MBPs)**.** *Fix a non-shortcutting generalized matrix branching program P (over $\mathbb{Z}_q$, for $q > 2^\lambda$), two input vectors $\mathbf{x}_0, \mathbf{x}_1$ such that $P(\mathbf{x}_0) = P(\mathbf{x}_1)$, and an allowable test polynomial p (Def. 2.6). Then either*

$$\Pr\left[\hat{P} \leftarrow \mathsf{MBPRand}(P) \ ; \ p(\mathsf{MBPSelect}(\hat{P}, \mathbf{x}_b)) = 0\right] = 1$$

for both bits $b \in \{0, 1\}$, or,

$$\Pr\left[\hat{P} \leftarrow \mathsf{MBPRand}(P) \ ; \ p(\mathsf{MBPSelect}(\hat{P}, \mathbf{x}_b)) = 0\right] < \mathrm{negl}(\lambda)$$

for both bits $b \in \{0, 1\}$.

Lemma 2.7 follows immediately from the results of Sahai and Zhandry [SZ14]; we defer the formal treatment to Appendix A.1.

2.4 Multilinear Maps

Multilinear maps [BS03], also known as graded encodings, or graded multilinear maps [GGH13a, CLT13], are a generalization of bilinear maps such as pairings over elliptic curves [Mil04, MOV93, Jou00, BF01]. Roughly speaking, a multilinear map lets us take a scalar $x \in \mathbb{F}_q$ and produce an encoded version, $\hat{x} = [x]_S$, where $S \subseteq \mathcal{U}$ is a finite set, called an *index set*, that indicates the level of the encoding $\hat{x}$ in a given hierarchy (namely, the subsets of $\mathcal{U}$ ordered by inclusion).[3]

By convention, we will say that these index sets are made up of *formal symbols*, denoted by capital letters (A, B, C), which serve the same role as formal variables in polynomials. To be fully precise, we state the following definitions.

Definition 2.8 (Formal Symbol)**.** A *formal symbol* is a bit string in $\{0, 1\}^*$, and distinct variables denote distinct bit strings. A *fresh* formal symbol is any bit string in $\{0, 1\}^*$ that has not already been assigned to another formal symbol.

Definition 2.9 (Index Sets)**.** An *index set* is a set of formal symbols called *indices*. By convention, for index sets we use set notation and product notation interchangeably, so that ABC represents $\{A, B, C\}$, and $ABC \cup D = ABCD$.

Definition 2.10 (Multilinear Map ([BS03, GGH13a, CLT13]))**.** A multilinear map over prime-order finite fields supports the following operations. Each of the operations (MM.Setup, MM.Add, MM.Mult, MM.ZeroTest, MM.Encode) is implemented by an efficient randomized algorithm.

[3] We describe here the case of *asymmetric* multilinear maps, since this is the one relevant to our constructions in this work.

- The setup procedure receives as input an index set $\mathcal{U}$ (Definition 2.9), which we refer to as the "top-level index set", as well as the security parameter λ (in unary). It produces public parameters pp (which include an $O(\lambda)$-bit prime q), and secret evaluation parameters sk:

$$\mathsf{MM.Setup}(\mathcal{U},\ 1^\lambda)\ \rightarrow\ (\mathsf{pp},\ \mathsf{sk})$$

- For each index set $\mathcal{S} \subseteq \mathcal{U}$, and each scalar $x \in \mathbb{Z}_q$, there is a set of strings $[x]_\mathcal{S} \subseteq \{0,1\}^*$, i.e., the set of all valid encodings of x at index set $\mathcal{S}$. [4] From here on, we will abuse notation to write $[x]_\mathcal{S}$ to stand for any element of $[x]_\mathcal{S}$ (i.e., any valid encoding of x at the index set $\mathcal{S}$).

- Elements at the same index set $\mathcal{S} \subseteq \mathcal{U}$ can be added, with the result also encoded at $\mathcal{S}$:

$$\mathsf{MM.Add}(\mathsf{pp},\ [x]_\mathcal{S},\ [y]_\mathcal{S})\ \rightarrow\ [x+y]_\mathcal{S}$$

- Elements at two index sets $\mathcal{S}_1, \mathcal{S}_2$ can be multiplied, with the result encoded at the union of the two sets, as long as their union is still contained in $\mathcal{U}$:

$$\mathsf{MM.Mult}(\mathsf{pp},\ [x]_{\mathcal{S}_1},\ [y]_{\mathcal{S}_2})\ \rightarrow\ \begin{cases} [xy]_{\mathcal{S}_1 \cup \mathcal{S}_2} & \text{if } \mathcal{S}_1 \cup \mathcal{S}_2 \subseteq \mathcal{U} \\ \perp & \text{otherwise} \end{cases}$$

- Elements at the top level $\mathcal{U}$ can be *zero-tested*:

$$\mathsf{MM.ZeroTest}(\mathsf{pp},\ [x]_\mathcal{S})\ \rightarrow\ \begin{cases} \text{“zero”} & \text{if } \mathcal{S} = \mathcal{U} \text{ and } x = 0 \in \mathbb{Z}_q \\ \text{“nonzero”} & \text{otherwise} \end{cases}$$

- Using the secret parameters, one can generate a representation of a given scalar $x \in \mathbb{Z}_q$ at any index set $\mathcal{S} \subseteq \mathcal{U}$:

$$\mathsf{MM.Encode}(\mathsf{sp},\ x,\ \mathcal{S})\ \rightarrow\ [x]_\mathcal{S}$$

- For the trivial index set $\mathcal{S} = \emptyset$, we specify that the only valid encoding of $[x]_\emptyset$ is just the scalar $x \in \mathbb{F}_q$. (So, for instance, we can perform subtraction via $\mathsf{MM.Add}$, by scalar multiplication with -1.)

By convention, we refer to the cardinality of $\mathcal{U}$ as the *degree of multilinearity* of the map.[5] Technically, known instantiations of multilinear maps [GGH13a, CLT13] are only approximate, and have a "noise" term that restricts the degree of multilinearity to a pre-specified polynomial in the security parameter. However, this restriction will not affect our results in this work, and to keep the presentation simple we do not model the restriction formally.

When the context is clear, we also abuse notation to write, for encoded elements $\hat{a}, \hat{b}$, the expression $\hat{a} + \hat{b}$ to mean $\mathsf{MM.Add}(\mathsf{MM.pp}, \hat{a}, \hat{b})$; the expression $\hat{a}\hat{b}$ to mean $\mathsf{MM.Mult}(\mathsf{MM.pp}, \hat{a}, \hat{b})$; and likewise for other arithmetic expressions.

[4] To be more precise, we define $[x]_\mathcal{S} = \{\chi \in \{0,1\}^* : \mathsf{MM.IsEncoding}(\mathsf{pp}, \chi, x, \mathcal{S})\}$, where the predicate $\mathsf{MM.IsEncoding}$ is specified by the concrete instantiation of the multilinear map. In general, the predicate $\mathsf{MM.IsEncoding}$ is not necessarily efficiently decidable—and indeed, for the security of the multilinear map, it should not be.

[5] In some cases, when we optimize a construction that uses multilinear maps, we find that we never need to encode elements of a given singleton index set. Thus in general, for constructions that are optimized in this way, we relax the definition of multilinearity degree to refer to the total number of sequential multiplications that must be performed on any encoded elements in the construction.

The generic multilinear map model. To define security, we will operate in the generic multilinear map model (also known as the *generic graded encoding model* [GGH$^+$13b, BR14, BGK$^+$14]). This model is very similar to the generic group model [Sho97]—intuitively, in this model, the only operations an adversary can use with encoded elements are the operations of the multilinear map. More precisely, we say a scheme that uses multilinear maps is secure in the generic multilinear map model if, for any concrete adversary breaking the real scheme, there is an ideal adversary breaking a modified scheme in which each concrete encoded element is replaced by a "handle" (concretely, a fresh nonce), mapped to the actual encoded scalar in a table unavailable to the adversary. Each multilinear map operation is replaced by an oracle query that takes two handles and returns another fresh handle (creating a new table entry), except for the zero-test oracle query, which, when given a handle, returns "zero" if the corresponding scalar in the table is zero, and "nonzero" otherwise. We defer the formal definitions to Appendix C.

2.5 Exclusive Partition Families

Even though the randomized MBPs of Section 2.3 impose certain restrictions on how their matrices can be multiplied together to remove the randomizing factors, this alone does not prevent an adversary from learning more information than just the outputs of honest evaluations on the MBP. The issue is that the adversary may execute "mix-and-match" attacks, using encoded matrices from multiple ciphertexts in the same evaluation. Our construction will use the multilinear map's index sets to enforce constraints on the adversary's evaluation, ruling out this kind of attack. As with the "straddling sets" technique of Barak et al. [BGK$^+$14], in order to use index sets to enforce this restriction, we need to design these index sets with some combinatorial properties in mind.

In more detail, suppose U is the top-level index set in the multilinear map, and $\mathcal{F}$ is some family of partitions of U. Intuitively, whenever we *intend* terms to be multiplied together (e.g., because they are matrix elements from a consistent choice of ciphertexts), the index sets of those terms will partition U, so that the product of the encoded elements can legally be zero-tested. We will design the partition family $\mathcal{F}$ so that our intended partitions (those in $\mathcal{F}$) are the only partitions of U that the adversary can possibly construct given the index sets of the terms we provide, thereby ruling out "mix-and-match" attacks.

Formally, we define the following:

Definition 2.11 (Partition)**.** The collection of sets $P = \{S_1, \ldots, S_d\}$ is a *partition* of a set U if $S_1 \cup \cdots \cup S_d = U$; each S_i is a nonempty subset of U; and $S_i \cap S_j = \emptyset$ for each $i \neq j$.

Definition 2.12 (Exclusive Partition Family)**.** Fix a set U, and a family $\mathcal{F}$ of partitions of U, where we write the N partitions in the family $\mathcal{F}$ as the rows of the matrix:

$$\begin{pmatrix} S_{1,1} & S_{1,2} & \cdots & S_{1,d} \\ \vdots & \vdots & \ddots & \vdots \\ S_{N,1} & S_{N,2} & \cdots & S_{N,d} \end{pmatrix}$$

We say that $\mathcal{F}$ is an *(N, d)-exclusive partition family* of U if the only partitions of U that can be formed from sets in $\mathcal{F}$ are precisely the rows of the matrix. (Formally: for all $(i_1, j_1), \ldots, (i_m, j_m) \in [N] \times [d]$, the collection $P = \{S_{i_1,j_1}, \ldots, S_{i_m,j_m}\}$ is a partition of U if and only if $i_1 = \ldots = i_m$ and $\{j_1, \ldots, j_m\} = [d]$.)

We say that an exclusive partition family $\mathcal{F}$ is *explicit* if there is an efficient deterministic algorithm which, when given $i \in [N], j \in [d]$, outputs the elements of $S_{i,j}$ (i.e., outputs the index of each element in some canonical ordering of the elements of U). We note that if $\mathcal{F}$ is explicit, then it is also easy to sample a

partition $(S_{i,1}, \ldots, S_{i,d})$ uniformly over all of the partitions in $\mathcal{F}$, simply by choosing uniform $i \leftarrow [N]$. To simplify notation, we write this sampling procedure as $(S_{i,1}, \ldots, S_{i,d}) \xleftarrow{\$} \mathcal{F}$.

Construction 2.13 ($(2^\lambda, d)$-Exclusive Partition Families)**.** Let $d, \lambda > 0$ be integers, and let U be a set of size $(1 + (d-1)(\lambda+1))$. Denote the elements of U as

$$U = \{\, a_1, a_2, \ldots, a_d,\ \ b_{2,1}, \ldots, b_{2,\lambda},\ \ \ldots,\ \ b_{d,1}, \ldots, b_{d,\lambda} \,\},$$

and identify an index $i \in [2^\lambda]$ with the string $\rho(i) \in \{0,1\}^\lambda$ that forms the binary representation of $(i-1)$. Define

$$S_{i,1} = \{a_1\} \cup \bigcup_{j \in \{2,\ldots,d\}} \{b_{j,k} : \rho(i)_k = 1\}$$

and for each $j \in \{2, \ldots, d\}$, define

$$S_{i,j} = \{a_j\} \cup \{b_{j,k} : \rho(i)_k = 0\}.$$

Finally, define the family $\mathcal{F}(d, \lambda) = ((S_{1,1}, \ldots, S_{1,d}), \ldots, (S_{N,1}, \ldots, S_{N,d}))$.

Lemma 2.14 ($(2^\lambda, d)$-Exclusive Partition Families)**.** *For integers $d, \lambda > 0$, the family $\mathcal{F}(d, \lambda)$ defined by Construction 2.13 is an explicit $(2^\lambda, d)$-exclusive partition family.*

Proof. By construction, each $(S_{i,1}, \ldots, S_{i,d})$ is a partition of U consisting of d sets, and the elements of each set are efficiently computable. Now, suppose that for some choice of sets $(i_1, j_1), \ldots, (i_m, j_m) \in [N] \times [d]$, the collection $P = \{S_{i_1,j_1}, \ldots, S_{i_m,j_m}\}$ is a partition of U. Then there exists some $r^* \in [m]$ such that the set $S_{i_{r^*},j_{r^*}}$ contains a_1. The only such sets are of the form:

$$S_{i_{r^*},j_{r^*}} = S_{i_{r^*},1} = \{a_1\} \cup \bigcup_{j \in \{2,\ldots,d\}} \{b_{j,k} : \rho(i_{r^*})_k = 1\}$$

Assume for sake of contradiction that for some $r \in [m]$, $i_r \neq i_{r^*}$. We cannot have $j_r = 1$, since this would cover the element a_1 twice: once by $S_{i_{r^*},j_{r^*}}$, and once by S_{i_r,j_r}. Thus $j_r \in \{2, \ldots, d\}$, and the set S_{i_r,j_r} is of the form:

$$S_{i_r,j_r} = \{a_{j_r}\} \cup \{b_{j_r,k} : \rho(i_r)_k = 0\}$$

But for each $k \in [\lambda]$, the only sets that contain $b_{j_r,k}$ also contain either a_1 or a_{j_r}, and we already have $a_1 \in S_{i^*,1}$ and $a_{j_r} \in S_{i_r,j_r}$ covered by the putative partition P. Hence the only elements $b_{j_r,k}$ that are covered by P are those of the form $\{b_{j,k} : \rho(i_{r^*})_k = 1\}$ and $\{b_{j,k} : \rho(i_r)_k = 0\}$. Since by assumption $i_r \neq i_{r^*}$, the strings $\rho(i_r), \rho(i_{r^*})$ differ on some bit $k^* \in [\lambda]$. If $\rho(i_{r^*})_{k^*} = 0$ and $\rho(i_r)_k = 1$, then P fails to cover b_{j_r,k^*}, while if $\rho(i_{r^*})_{k^*} = 1$ and $\rho(i_r)_k = 0$, then P covers b_{j_r,k^*} twice. In either case P is not a partition of U, contradicting our assumption. So we conclude that $i_{r^*} = i_1 = \ldots = i_m$, and thus $\mathcal{F}$ is an explicit $(2^\lambda, d)$-exclusive partition family. □

We also observe that our definition of exclusive partition families generalizes the *straddling set systems* of Barak et al. [BGK+14]. Indeed, for any integer $d > 0$, a straddling set system $\mathbb{S}_d$ (as defined in [BGK+14]) is a $(2, d)$-exclusive partition family.

3 Finite Automata as Branching Programs

In this section, we explain how to realize finite automata as matrix branching programs. A finite (non-deterministic) automaton is a directed graph on w nodes, and we call w the *size* of the automaton. There is a single node s marked as the start state, and a subset T of nodes marked as accept states. Every edge (transition) is labeled by a symbol d from some universe U, and every node has at least one outgoing edge labeled by d for each $d \in U$. On an input $x \in U^n$, the automaton starts at the starting state s, and then reads the digits of x one by one. When it reads a digit d, the automaton non-deterministically follows the edge(s) out of the current node labeled by d. We call the current node the *state* of the automaton, and the automaton *accepts* if one of the final states it arrives at after reading the entire input is an accept state.

In the case of multi-input functionalities, we have a choice of how we can order the input digits when fed into the automaton. For example, on input $x^{(1)}, \ldots, x^{(m)}$, we can order the input digits as $x_1^{(1)} \cdots x_n^{(1)} x_1^{(2)} \cdots x_n^{(2)} \cdots$. Alternatively, for some functionalities, it may be beneficial (or even required) to interleave the digits, for example as $x_1^{(1)} x_1^{(2)} \cdots x_n^{(m)} x_2^{(1)} x_2^{(2)} \cdots$.

Layered automata. We also consider the notion of a *layered* automaton. Such an automaton consists of n "layers" labeled "1" through "n" of w states each, plus an additional starting layer labeled "0" which contains the start state. The final layer n also contains a subset T of states marked as accept states. The automaton has transitions only from layer i to layer $i+1$, so that the transitions from layer i only occur when reading the i^{th} input digit. Even though the graph representing a layered automaton contains $nw+1$ nodes, we say that the automaton has size w, the size of each layer. We note that any (non-layered) automaton of size w can be easily converted to a layered automaton of size w by "unrolling" the automaton, and creating n identical layers of size w.

From automata to branching programs. Any automaton can easily be converted into a matrix branching program. Consider a w-state automaton that acts on m inputs, each of length n, and let $\ell = mn$ be the total input length. Suppose the inputs are interleaved as

$$x_{k_1}^{(i_1)} x_{k_2}^{(i_2)}, \ldots, x_{k_\ell}^{(i_\ell)}.$$

For a digit d, let G_d be the directed graph on the w states of the automaton where state u has an edge to state v if the automaton transitions from u to v when reading d. Order the states so that the start state is the first state, and let A_b be the $w \times w$ adjacency matrix of G_b as a directed bipartite graph. For now, assume the start state is also the unique accept state. Then the branching program has length ℓ, where $\mathsf{inp}(j) = i_j$, and

$$\mathcal{M}_j(x^{(i_j)}) = A^T_{x_{k_j}^{(i_j)}}.$$

To see why this works, consider the products

$$(1, 0, \ldots, 0) \cdot \prod_{j \in [r]} \mathcal{M}_j(x^{(\mathsf{inp}(j))}) = (1, 0, \ldots, 0) \cdot \prod_{j \in [r]} A^T_{x_{k_j}^{(i_j)}}$$

for $r = 0, \ldots, \ell$. The result is a row vector of positive integers, where the value at position v is the number of non-deterministic paths the automaton could follow from the start state to state v after reading the first r digits. Thus the product

$$\prod_{j \in [\ell]} \mathcal{M}_j(x^{(\mathsf{inp}(j))})$$

is non-zero in the upper left corner if and only if on reading $\mathbf{x}$, there is some non-deterministic path from the start state to itself. Since we assume the start state is the unique accept state, this is equivalent to the automaton accepting.

For multiple accept states and/or accept states other than the start state, we modify the above slightly. Let $T \subseteq [w]$ be the subset of accept states, and let $\mathbf{v} \in \{0,1\}^w$ be the incidence vector for T. Then, pick an invertible matrix P whose first column is $\mathbf{v}$. The final matrix $\mathcal{M}_\ell$ is set as

$$\mathcal{M}_\ell = A^T_{x_{k_\ell}^{(i_\ell)}} \cdot P,$$

and all other matrices are the same as before. Now, the upper left entry in the matrix $\prod_{j\in[r]} \mathcal{M}_j(x^{(\mathsf{inp}(j))}) \cdot P$ contains the sum of the number of paths from the start state to each of the accept states, and this sum is non-zero if and only if the automaton accepts.

It is straightforward to adapt the conversion above to apply to layered automata as well. For a digit d and position k, let $G_{k,d}$ be the bipartite directed graph consisting of layers $k-1$ and k and edges between them labeled by d. Let $A_{k,d}$ be the $w \times w$ adjacency matrix for the edges going from layer $k-1$ to layer k labeled by d. We note that for layered automata, we can collapse all the accept nodes in the final layer to a single accept node, which will be the first node of the layer. This allows us to avoid the matrix P from above. Then, define

$$\mathcal{M}_j(x^{(i_j)}) = A^T_{j,x_{k_j}^{(i_j)}}.$$

Optimization. Note that if $\mathsf{inp}(k) = \mathsf{inp}(k+1)$, we can merge $\mathcal{M}_k$ and $\mathcal{M}_{k+1}$ into a single function $\mathcal{M}'_k(x) = \mathcal{M}_k(x) \cdot \mathcal{M}_{k+1}(x)$. This allows us to reduce the length of the branching program. We will use this optimization below to shrink the length of our branching program for comparisons by a factor of two, essentially for free.

3.1 Finite Automata for Comparisons

Clearly, it is not possible to compare two integers using a finite automaton if the input bits are processed with all the bits of x occurring before all the bits of y (or vice versa). However, if we interleave the bits as $x_n y_n x_{n-1} y_{n-1} x_{n-2} \ldots$, then a simple 5-state automaton is possible (see Figure 1). The five states are labeled $=, <, >, 0, 1$. The state $=$ represents that the automaton has read $2k$ total bits (k from each input), and the bits from x equaled the bits from y. The state $>$ (resp. $<$) represents that the automaton has read $2k$ or $2k+1$ total bits (k from y, and k or $k+1$ from x), and the integers $x_{[1,k]}, y_{[1,k]}$ represented by the k bits of x and y read so far satisfy $x_{[1,k]} > y_{[1,k]}$ (resp. $x_{[1,k]} < y_{[1,k]}$). The state 0 (resp. 1) represents that $2k+1$ bits have been read, the first k bits of x and y are equal, and the most recent bit (bit $k+1$ of x) is 0 (resp. 1). It is straightforward to determine the state transitions for this automaton. Intuitively, the automaton simply keeps track of the most recent bit read, as well as the result of comparing the integers read so far. Using the conversion above, this gives a width-5 branching program of length $2n$.

3.2 Optimizing the Layered Automata

We introduce two optimizations on the finite automaton implementing the comparison function in order to reduce the total number of steps for its evaluation, which corresponds to the number of matrix multiplications in the decryption algorithm in our encryption scheme. While the above automaton reads one bit from each input in turns, which results in as many transitions as the total bits length of the two inputs, our idea here is to

process more than one bit from each of the two inputs at once. These transformations increase the number of states, which no longer independent of the length of the input, and the number of transition symbols in the automaton alphabet, but preserve the layered structure of the automaton which is required for our construction.

We now make two optimizations. First, notice above that after reading $2k$ bits, the automaton is in one of three possible states ($=$, $>$, or $<$), and that after reading $2k+1$ bits, the automaton is on one of four states ($>$, $<$, 0, or 1). Therefore, it is straightforward to "unroll" the automaton into a layered automaton of total width 4.

Next, in the interest of taking advantage of the optimization from the previous section, we want to collect input bits from each input together into chunks that are as large as possible. One way to do this is to consider representing the integer in a base B other than 2. The number of states will increase to $B+2$, but the length of the branching program will decrease by a factor of $\log_2 B$.

As another independent optimization, we can also interleave the input bits as $x_n y_n y_{n-1} x_{n-1} x_{n-2} \cdots$. Using this ordering, it is still possible to use a $(B+2)$-state layered automaton, intuitively because the automaton still only has to remember at most one digit of the input. The advantage of this order is that input bits are read in pairs from the respective inputs, so the total length of the branching program shrinks from $2\lceil n/\log_2 B \rceil$ to $\lceil n/\log_2 B \rceil + 1$, while the width remains at $B+2$. For simplicity, to implement comparisons for order-revealing encryption in our main construction, we take $B=2$, so that the total length of the branching program is $(n+1)$. We defer the formal description of the optimized branching program to Appendix B.

4 Secret-Key Multi-Input Functional Encryption (SK-MIFE)

We now discuss the definition of secret-key multi-input functional encryption (SK-MIFE), which is a special case of the definition of multi-input functional encryption (MIFE) in [GGG+14].

In fact we will specialize this definition further, to the case of SK-MIFE with a *single* function evaluation key (1SK-MIFE). We note that it is straightforward to construct ordinary SK-MIFE (enabling multiple function keys) from 1SK-MIFE, as follows. We can set the single functionality in 1SK-MIFE to be a universal branching program, $\mathsf{U}(f, x_1, \ldots, x_n)$, which takes as one of its inputs the function f to be evaluated. In this SK-MIFE scheme, the key to evaluate a particular function f will be the 1SK-MIFE encryption $\mathsf{1SK\text{-}MIFE.Enc}(\mathsf{sk}, 1, f)$ (for input slot 1 in the universal program U).

We also note that 1SK-MIFE already covers the application of order-revealing encryption (ORE), since here we only want to enable a single function on MIFE ciphertexts: namely, the comparison function. As we will see below, working with 1SK-MIFE enables us to achieve a much more efficient construction. Thus, we will restrict our attention to 1SK-MIFE here.

4.1 Definitions

A single-key, secret-key multi-input functional encryption (1SK-MIFE) scheme

$$\Pi = (\mathsf{1SK\text{-}MIFE.Setup},\ \mathsf{1SK\text{-}MIFE.Enc},\ \mathsf{1SK\text{-}MIFE.Dec})$$

supports the following operations. Each operation is implemented by a randomized algorithm, which (with all but negligible probability) runs in time polynomial in its input length and the security parameter λ.

- The setup procedure takes as input a security parameter λ and a program P, given as an m-input matrix branching program (Section 2.2) over $\mathbb{Z}_q$ for some prime $q > 2^\lambda$. The setup procedure outputs an evaluation key ek and a secret key sk.

$$\text{1SK-MIFE.Setup}(\lambda, P) \rightarrow (\mathsf{ek}, \mathsf{sk})$$

- The encryption procedure takes as input a secret key sk, an input variable index $i \in [m]$, and an input $x \in \mathcal{X}$, and outputs a ciphertext ct.

$$\text{1SK-MIFE.Enc}(\mathsf{sk}, i, x) \rightarrow \mathsf{ct}$$

- The decryption procedure takes as input an evaluation key ek and ciphertexts $\mathsf{ct}^{(1)}, \ldots, \mathsf{ct}^{(m)}$, and outputs a computation result $b \in \{0, 1\}$.

$$\text{1SK-MIFE.Dec}(\mathsf{ek}, \mathsf{ct}^{(1)}, \ldots, \mathsf{ct}^{(m)}) \rightarrow b$$

Definition 4.1 (1SK-MIFE Correctness)**.** A 1SK-MIFE scheme Π is *correct* if for any uniform multi-input matrix branching program P, and any inputs $x^{(1)}, \ldots, x^{(m)} \in \mathcal{X}$, if $(\mathsf{ek}, \mathsf{sk}) \leftarrow \text{1SK-MIFE.Setup}(\lambda, P)$ and for each $i \in [m]$ it is the case that $\mathsf{ct}^{(i)} \leftarrow \text{1SK-MIFE.Enc}(\mathsf{sk}, i, x^{(i)})$, then,

$$\text{1SK-MIFE.Dec}(\mathsf{ek}, \mathsf{ct}^{(1)}, \ldots, \mathsf{ct}^{(m)}) \rightarrow P(x^{(1)}, \ldots, x^{(m)}).$$

Security. It is clearly impossible to achieve (standard) semantic security for 1SK-MIFE, since by design, our scheme must leak some information about the plaintexts—namely, the result of evaluating the 1SK-MIFE program P on every possible choice of query plaintext tuple. Our goal, then, is best-possible semantic security, in which we leak *only* this information. In this respect, our security definition is similar to IND-OCPA in the special case of order-preserving (or order-revealing) encryption [BCLO09], but of course we must generalize it for 1SK-MIFE. Our definition is also similar to the indistinguishability-based definitions of (general) multi-input functional encryption given by Goldwasser et al. [GGG+14]. We now present the formal details.

Definition 4.2 (1SK-MIFE Security Game)**.** Fix a generalized matrix branching program P (to be used in the 1SK-MIFE scheme). For an adversary $\mathcal{A}$, and for each "world" bit $b \in \{0, 1\}$, we define the experiment $\mathsf{Expt}^{\text{1SK-MIFE}}_{P,Q,b}(\mathcal{A})$, parameterized over a number of queries Q:

Experiment $\mathsf{Expt}^{\text{1SK-MIFE}}_{P,Q,b}(\mathcal{A})$**:**

1. $\mathcal{A}$ receives an evaluation key ek, where $(\mathsf{ek}, \mathsf{sk}) \leftarrow \text{1SK-MIFE.Setup}(\lambda, P)$.
2. $\mathcal{A}$ makes Q adaptive queries to a left-or-right encryption oracle, as follows. For each $t \in [Q]$, the adversary sends $\text{query}_t = (i_t, x_{t,0}, x_{t,1})$, and is given a ciphertext $\mathsf{ct}_t \leftarrow \text{1SK-MIFE.Enc}(\mathsf{sk}, i_t, x_{t,b})$.
3. $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$, which is the output of the experiment.

Definition 4.3 (Input-Consistent Queries)**.** In an execution trace of the experiment $\mathsf{Expt}_{P,Q,b}^{\text{1SK-MIFE}}(\mathcal{A})$ (Definition 4.2), let $t_1, \ldots, t_m \in [Q]$ be time steps in the adversary's query sequence, such that for every input slot index $i \in [m]$, we have $\mathsf{query}_{t_i}[1] = i$ (i.e., at time t_1, the adversary queried for input slot 1; at time t_2, the adversary queried for input slot 2; and so on). Then we say the query time sequence $\boldsymbol{\tau} = (t_1, \ldots, t_m)$ is *input-consistent*. Furthermore, for each world bit $b \in \{0, 1\}$, we say that such an input-consistent sequence $\boldsymbol{\tau}$ *selects* the vector of inputs $\mathbf{x}_{\tau,b} = (x_{t_1,b}, \ldots, x_{t_m,b})$.

Definition 4.4 (Execution Trace)**.** Fix an adversary $\mathcal{A}$ in the generic multilinear map model (Definition C.1). We define the *execution trace* of the experiment $\mathsf{Expt}_{P,Q,b}^{\text{1SK-MIFE}}(\mathcal{A})$ to be the sequence of all oracle query-response pairs, both between $\mathcal{A}$ and the challenger and between $\mathcal{A}$ and the multilinear map oracle.

Definition 4.5 (Admissibility of Execution Traces)**.** An execution trace of the experiment $\mathsf{Expt}_{P,Q,b}^{\text{1SK-MIFE}}(\mathcal{A})$ is *admissible* if the Q adaptive queries made by the adversary satisfy the following condition: for every input-consistent query time sequence $\boldsymbol{\tau} \in [Q]^m$, letting $\mathbf{x}_{\tau,b}$ denote the vector of inputs selected by $\boldsymbol{\tau}$ in world b, we have $P(\mathbf{x}_{\tau,0}) = P(\mathbf{x}_{\tau,1})$.

We note that admissibility can be checked, for any given execution trace, in time $O([Q]^m) \cdot \mathrm{poly}(\lambda, |P|))$, simply by testing the condition every possible sequence $\boldsymbol{\tau}$. Thus, if m is a constant—e.g., for order-revealing encryption, where the arity of the comparison program is $m = 2$—then admissibility can be checked in polynomial time. For general programs P, the arity m may be $\omega(1)$, in which case admissibility may not be efficiently checkable. Nevertheless, we can still define IND-security the same way.

Definition 4.6 (IND-security for 1SK-MIFE)**.** A 1SK-MIFE scheme Π is Q-IND-secure if, for all generalized matrix branching programs P, and all efficient adversaries $\mathcal{A}$, the quantity

$$\mathsf{Adv}_{P,Q}^{\text{1SK-MIFE}}(\mathcal{A}) = |W_0 - W_1|$$

is negligible, where for each world bit $b \in \{0, 1\}$ we define

$$W_b = \Pr\left[\mathsf{Expt}_{P,Q,b}^{\text{1SK-MIFE}}(\mathcal{A}) \text{ outputs } 1 \text{ and yields an admissible execution trace}\right].$$

Application to order-revealing encryption. Our motivating application of 1SK-MIFE is order-revealing encryption (ORE). In this case, the program P is a matrix branching program for the comparison function (Section 3), which takes two bit strings $x, y \in \{0, 1\}^n$ representing numbers in binary, and returns 1 if $x \leq y$. The 1SK-MIFE evaluation key ek then fills the role of the comparison algorithm in ORE.

Strictly speaking, in addition to the comparison algorithm, ORE requires that someone who holds the secret key can also decrypt each ciphertext, revealing the original string $x \in \{0, 1\}^n$. We can accomplish this by including, along with the 1SK-MIFE ciphertext, another encryption of x under an ordinary (semantically-secure) symmetric encryption scheme, and including this scheme's secret key as part of the key in ORE.

5 Our 1SK-MIFE Construction

Consider an m-input generalized matrix branching program (MBP) of length ℓ. We construct a 1SK-MIFE for the function computed by this MBP. To encrypt an input $x \in \mathcal{X}$ we construct the set of matrices obtained by considering the input $(x, \ldots, x) \in \mathcal{X}^m$ to the MBP. We randomize each matrix in the branching program as in Section 2.3 by using a randomizing matrix taken from the secret key. These randomizing matrices $\mathbf{R}_i$ are fixed at key generation time and used for all encryptions. The encryption procedure then chooses random

scalars $\alpha_1, \ldots, \alpha_\ell$ and, more importantly, chooses random index sets for a multilinear map with which to encode each of the matrices (these index sets are chosen from an *exclusive partition family* which, as we will see, have the properties needed for correctness and security). The encryptor encodes each randomized matrix using its assigned index set and outputs the set of encoded matrices as the encryption of x. Now, to compute the MBP function given m independently-created ciphertexts we can select appropriate encoded matrices from each ciphertext and compute their product using the multilinear map, as done in the ORE example in Section 1. We then zero-test the result to learn the output of the function in the clear.

The challenge with this approach is to guarantee that any meaningful evaluation has to use all of the matrices in a set of m ciphertexts and no other elements. In other words, the difficulty in the security proof lies in preventing attacks that evaluate the decryption function by mixing matrices from different encryptions for the same input position. We resolve this issue by relying on exclusive partition families from Definition 2.12. For each input position i that determines d matrices in the generalized MBP, we construct a $(2^\lambda, d)$-exclusive partition family $\mathcal{F}^{(i)}$. To encrypt a message for that position, we sample at random a partition $(S_1^{(i)}, \ldots, S_d^{(i)})$ from the family $\mathcal{F}^{(i)}$ and use the sets from the partition as the index sets for encoded matrices included in the encryption. The properties of the exclusive partition families guarantee that MBP evaluations using matrices from ciphertexts generated by sampling different partitions from $\mathcal{F}^{(i)}$ will fail because the result will not be encoded with respect to the index set for the zero-tester.

We now describe the formal construction for the above intuition.

Construction 5.1 (1SK-MIFE)**.** The 1SK-MIFE construction consists of the following procedures:

- $\underline{\mathsf{1SK\text{-}MIFE.Setup}}(\lambda, P)$:

  The setup procedure receives as input a security parameter λ and a generalized matrix branching program $P : \mathcal{X}^m \to \{0,1\}$ of the form

  $$P = (\ q,\ m,\ d,\ \mathsf{inp},\ (\mathcal{M}_1,\ \ldots,\ \mathcal{M}_\ell)\ ),$$

  as described in Section 2.2, where $\mathcal{X} \subset \{0,1\}^*$ is a space of possible input strings, each $\mathcal{M}_j : \mathcal{X} \to \mathrm{GL}_w(\mathbb{Z}_q)$ is expressed as a Boolean circuit, and $\ell = md$.

  For each input variable index $i \in [m]$, let $\mathcal{F}^{(i)}$ be a $(2^\lambda, d)$-exclusive partition family (Lemma 2.14) over a set $\mathcal{U}_i$ of $O(d\lambda)$ fresh formal indices (Section 2.4), and let A_s, A_t also be fresh formal indices. The setup procedure forms a top-level universe of indices

  $$\mathcal{U} = A_s A_t \prod_{i \in [m]} \mathcal{U}_i,$$

  and generates corresponding parameters for a multilinear map

  $$(\mathsf{MM.pp},\ \mathsf{MM.sp}) \leftarrow \mathsf{MM.Setup}(\mathcal{U}, q).$$

  Then, it randomizes P via the method of Definition 2.3, producing a randomized program $\hat{P}$ as

  $$\hat{P} = \mathsf{MBPRand}(P) = \Big(\ q,\ m,\ n,\ \mathsf{inp},\ (\hat{\mathcal{M}}_1,\ \ldots,\ \hat{\mathcal{M}}_\ell),\ \hat{\mathbf{s}},\ \hat{\mathbf{t}}\ \Big).$$

  Finally, it outputs the evaluation key ek and the secret key sk:

  $$\mathsf{ek} = \Big(\mathsf{MM.pp},\ P,\ [\hat{\mathbf{s}}]_{A_s}\ ,\ \big[\hat{\mathbf{t}}\big]_{A_t}\Big) \qquad \mathsf{sk} = (\mathsf{MM.sp}, \hat{P})$$

  (using $\mathsf{MM.Encode}(\mathsf{MM.sp}, \cdot, \cdot)$ to generate fresh encoded elements $[\hat{\mathbf{s}}]_{A_s}$, $\big[\hat{\mathbf{t}}\big]_{A_t}$).

- $\underline{\mathsf{1SK\text{-}MIFE.Enc}(\mathsf{sk}, i, x)}$:

  The encryption procedure receives as input the secret key $\mathsf{sk} = (\mathsf{MM.sp}, \hat{P})$, an input variable index $i \in [m]$, and a plaintext $x \in \mathcal{X}$ (to be encrypted to the i^{th} input slot of the branching program).

  Let $\mathcal{F}^{(i)}$ be a $(2^\lambda, d)$-exclusive partition family (Lemma 2.14) over $\mathcal{U}_i$, as defined in $\mathsf{1SK\text{-}MIFE.Setup}$ above. The encryption procedure samples a partition uniformly at random from the family $\mathcal{F}^{(i)}$ of the form

  $$\left(S_1^{(i)}, \ldots, S_d^{(i)}\right) \xleftarrow{\$} \mathcal{F}^{(i)}.$$

  The procedure also chooses scalars $\alpha_1, \ldots, \alpha_d \leftarrow \mathbb{Z}_q^*$ uniformly at random. Finally, for each $h \in [d]$, the procedure generates the following fresh encoded elements (using $\mathsf{MM.Encode}(\mathsf{MM.sp}, \cdot, \cdot)$):

  $$\mathsf{ct}_h := \left[\alpha_h \hat{\mathcal{M}}_{\mathsf{inp.j}(i,h)}(x)\right]_{S_h^{(i)}},$$

  and outputs the ciphertext $\mathsf{ct} = (\mathsf{ct}_1, \ldots, \mathsf{ct}_d)$.

- $\underline{\mathsf{1SK\text{-}MIFE.Dec}(\mathsf{ek}, \mathsf{ct}^{(1)}, \ldots, \mathsf{ct}^{(m)})}$: The decryption procedure receives as input the public parameters $\mathsf{ek} = (\mathsf{MM.pp}, P, [\hat{\mathbf{s}}]_{A_s}, [\hat{\mathbf{t}}]_{A_t})$, along with m ciphertexts $\mathsf{ct}^{(1)}, \ldots, \mathsf{ct}^{(m)}$. Each ciphertext is parsed as

  $$\mathsf{ct}^{(i)} = (\mathsf{ct}_1^{(i)}, \ldots, \mathsf{ct}_d^{(i)}) = \left(\hat{\mathbf{C}}_1^{(i)}, \ldots \hat{\mathbf{C}}_d^{(i)}\right),$$

  where the entries of the matrices $\hat{\mathbf{C}}_h^{(i)}$ are encoded elements in the multilinear map. Then, using the multilinear map operations ($\mathsf{MM.Add}$, $\mathsf{MM.Mult}$), it computes

  $$z = [\hat{\mathbf{s}}]_{A_s} \cdot \left(\prod_{j \in [\ell]} \hat{\mathbf{C}}_{\mathsf{inp.h}(j)}^{(\mathsf{inp}(j))}\right) \cdot [\hat{\mathbf{t}}]_{A_t}.$$

  Using the operation $\mathsf{MM.ZeroTest}$, the procedure tests whether z encodes zero in $\mathbb{F}_q$, and outputs 1 if so, and 0 otherwise.

Functional correctness follows from the definition of Construction 5.1, along with the correctness of the multilinear map procedures. Formally, we state the following theorem.

Theorem 5.2. *Construction 5.1 is correct (Definition 4.1).*

To prove Theorem 5.2, we first show that for a given evaluation on m honestly-generated ciphertexts, all of the index sets "match up" for each $i \in [m]$, so that the result z is a valid zero-test query; this follows from the properties of exclusive partition families (Definition 2.12). Then, we show that the actual value of z corresponds to the execution of the original program; this follows by correctness of the randomization procedure $\mathsf{MBPRand}$ (Definition 2.3).

Proof. Fix a multi-input matrix branching program P, a security parameter λ, and a tuple of plaintext inputs $\mathbf{x} = (x^{(1)}, \ldots, x^{(m)})$. Let $(\mathsf{ek}, \mathsf{sk}) \leftarrow \mathsf{1SK\text{-}MIFE.Setup}(\lambda, P)$, and suppose that for each $i \in [m]$, we have $\mathsf{ct}^{(i)} \leftarrow \mathsf{1SK\text{-}MIFE.Enc}(\mathsf{sk}, i, x^{(i)})$. Then we claim that $\mathsf{1SK\text{-}MIFE.Dec}(\mathsf{sk}, x^{(1)}, \ldots, x^{(m)}) \to P(\mathbf{x})$.

To begin, we write:

$$\mathsf{ct}^{(i)} = (\hat{\mathbf{M}}_1^{(i)}, \ldots, \hat{\mathbf{M}}_d^{(i)}),$$

where the entries of the each matrix $\hat{\mathbf{M}}_h^{(i)}$ are encoded elements in the multilinear map. Note that 1SK-MIFE.Dec outputs the result of zero-testing the following encoded element:

$$z = [\hat{\mathbf{s}}]_{A_s} \cdot \left(\prod_{j \in [\ell]} \hat{\mathbf{M}}_{\mathsf{inp.h}(j)}^{(\mathsf{inp}(j))} \right) \cdot [\hat{\mathbf{t}}]_{A_t}$$

Hence, for correctness, it suffices to show that for honestly constructed ciphertexts, z is a valid (non-$\perp$) encoded element at the top-level index set $\mathcal{U}$, and that z's value in $\mathbb{Z}_q$ is zero precisely when P evaluates to 1 on $\mathbf{x}$.

By definition, for any $j \in [\ell]$, we have $j = \mathsf{inp.j}(\mathsf{inp}(j), \mathsf{inp.h}(j))$ (Section 2.2). Thus by construction of 1SK-MIFE.Enc:

$$z = [\hat{\mathbf{s}}]_{A_s} \cdot \left(\prod_{j \in [\ell]} \left[\alpha'_j \, \hat{\mathcal{M}}_j(x^{(\mathsf{inp}(j))}) \right]_{S_{\mathsf{inp.h}(j)}^{(\mathsf{inp}(j))}} \right) \cdot [\hat{\mathbf{t}}]_{A_t} ,$$

for some $\alpha'_j \in \mathbb{Z}_q^*$ chosen by 1SK-MIFE.Enc on some input. Now, note that

$$\left\{ S_{\mathsf{inp.h}(j)}^{(\mathsf{inp}(j))} : j \in [\ell] \right\} \;=\; \left\{ S_h^{(i)} : i \in [m], h \in [d] \right\}. \tag{5.1}$$

Since for each $i \in [m]$, the tuple $(S_h^{(\mathsf{inp}(j))})_{h \in [d]}$ is a partition of $\mathcal{U}_i$ (Lemma 2.14), we conclude that the right-hand side of (5.1) is a partition of $(\mathcal{U} \setminus (A_s \cup A_t))$, and thus so is the left-hand side. Hence each MM.Mult operation performed by the functional decryption procedure is valid, and the result z is an element of $\mathbb{Z}_q$ encoded at the top-level universe $\mathcal{U}$.

It only remains to establish that z encodes zero precisely when the program evaluates to 1 on the corresponding inputs. We have that

$$z \;=\; \left[\left(\prod_{j \in [\ell]} \alpha'_j \right) \hat{\mathbf{s}} \cdot \left(\prod_{j \in [\ell]} \hat{\mathcal{M}}_j(x^{(\mathsf{inp}(j))}) \right) \cdot \hat{\mathbf{t}} \right]_{\mathcal{U}},$$

and hence, by the correctness of the randomized encoding (Lemma 2.5),

$$z \;=\; \left[\left(\prod_{j \in [\ell]} \alpha'_j \right) \cdot \left(\prod_{j \in [\ell]} \mathcal{M}_j(x^{(\mathsf{inp}(j))}) \right) [1,1] \right]_{\mathcal{U}}.$$

Since each $\alpha'_j \in \mathbb{Z}_q^*$ is invertible, z encodes zero if and only if

$$\left(\prod_{j \in [\ell]} \mathcal{M}_j(x^{(\mathsf{inp}(j))}) \right) [1,1] = 0,$$

so by the definition of the branching program, we conclude that

$$z = 0 \iff P(\mathbf{x}) = 1.$$

□

Remark 5.3. As written, Construction 5.1 requires an $(\ell+2)$-way multilinear map to support the computation of z in 1SK-MIFE.Dec. However, we note that we could optimize the construction so that 1SK-MIFE.Enc pre-multiplies the vectors $\hat{\mathbf{s}}$ and $\hat{\mathbf{t}}$ with the first and last matrices, respectively, $\hat{\mathbf{C}}^{(\mathsf{inp}(1))}_{\mathsf{inp.h}(1)}, \hat{\mathbf{C}}^{(\mathsf{inp}(\ell))}_{\mathsf{inp.h}(\ell)}, \cdots$. This would enable us to reduce the degree of the computation from $(\ell+2)$ to ℓ (and hence obtain better parameters for the multilinear map); in the special case of order-revealing encryption, we have $\ell = k+1$, and thus we reduce the degree required from $(k+3)$ to $(k+1)$. For simplicity, however, we present the construction without this optimization.

Remark 5.4 (Multi-Bit Output)**.** For simplicity, we present our SK-MIFE construction only for functions that output a single bit. However, the construction can easily be extended to functions with multi-bit output in a number of ways. First, if a given generalized branching program already outputs k bits[6], then we can output the same k bits via the techniques of Sahai and Zhandry, replacing the bookend vectors $\hat{\mathbf{s}}, \hat{\mathbf{t}}$ by randomized diagonal matrices as described in [SZ14]. This transformation yields multi-bit output at essentially no additional performance cost. Alternatively, for arbitrary programs (not represented efficiently as multi-bit branching programs *a priori*), we can also simply run k copies of our scheme in parallel, supporting multi-bit output at the cost of a factor k loss in efficiency.

5.1 Security proof

Our main theorem states that the construction above indeed yields a secure 1SK-MIFE scheme.

Theorem 5.5 (1SK-MIFE Security)**.** *The 1SK-MIFE construction of Section 5 is* $\mathrm{poly}(\lambda)$*-IND-secure in the generic multilinear map model.*

Before proving Theorem 5.5, we first give a few relevant definitions and lemmas. Our proof techniques in this section are similar to those in related works that use the generic multilinear map model [BR14, BGK+14].

Remark 5.6 (Queries Referring to Formal Polynomials)**.** Formally, the generic multilinear map model is defined in terms of oracle queries on "handles" (nonces). In any particular security game, however, it is usually more intuitive to regard each oracle query as a formal polynomial. The formal variables are specified in terms of the expressions initially supplied to the MM.Encode procedure (as appropriate to the security game), and the adversary can construct new polynomials by making oracle queries for the generic-model ring operations MM.Add, MM.Mult. Rather than operating on a handle, then, we can think of each valid MM.ZeroTest query as *referring* to a formal polynomial encoded at the top-level universe $\mathcal{U}$. The result of the query is "zero" precisely if the given polynomial evaluates to zero, when its variables are instantiated with the *real* joint distribution over their values in $\mathbb{Z}_q$, generated as in the actual security game. For precise definitions, we refer the reader to Appendix C.1.

Structure lemmas. Our 1SK-MIFE construction uses index sets to enforce constraints on the adversary's evaluation (as depicted in Figure 3). The purpose of these constraints is to prevent the adversary from constructing zero-test queries that are inconsistent—i.e., use encodings that "mix and match" elements of different ciphertexts. To show that our design indeed prevents these undesired queries, we first state and prove a few simple definitions and "structure lemmas", showing that all valid query polynomials have a certain form.

[6] In such a branching program, the output is determined by the upper left $k_1 \times k_2$ submatrix ($k = k_1 k_2$) of the final matrix product, as opposed to just the upper left entry. The output of the program is the $k_1 \times k_2$ Boolean matrix indicating which entries in the submatrix are 0.

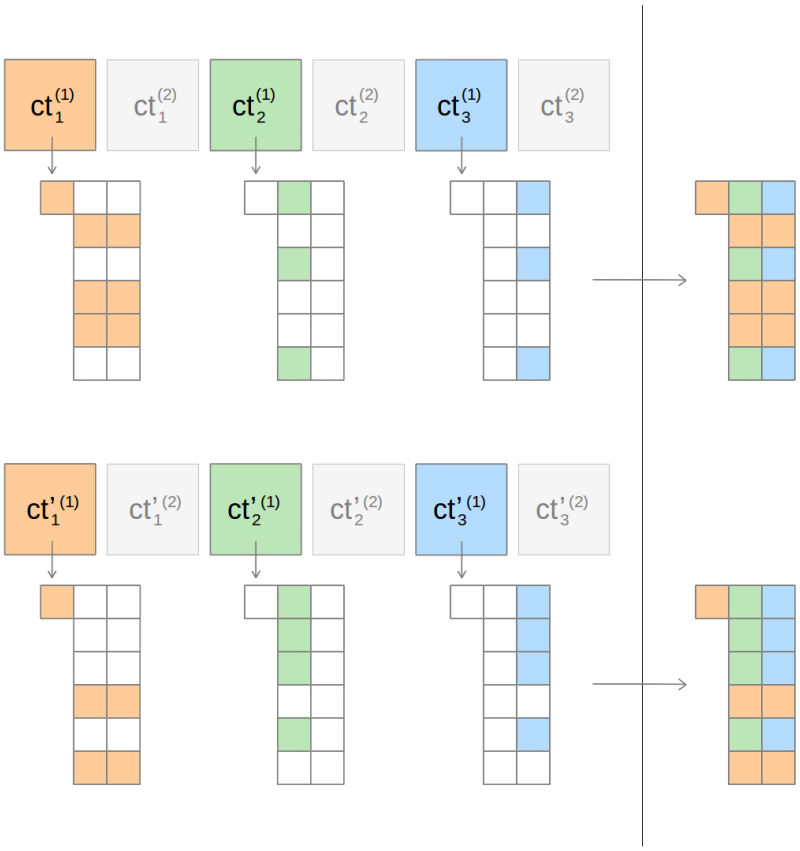

Figure 3: The matrices of two 1SK-MIFE ciphertexts, $\mathsf{ct} = (\mathsf{ct}_1^{(1)}, \mathsf{ct}_2^{(1)}, \mathsf{ct}_3^{(1)})$ and $\mathsf{ct}' = (\mathsf{ct}_1{}'^{(1)}, \mathsf{ct}_2{}'^{(1)}, \mathsf{ct}_3{}'^{(1)})$ (both encrypted to slot 1), with the index set of each matrix depicted below it. Since the index sets are defined by two different elements of the same exclusive partition family, the adversary cannot "mix and match" elements from the two ciphertexts.

Definition 5.7 (Query-Consistent Polynomials)**.** For an execution trace of the experiment $\mathsf{Expt}^{\mathsf{1SK\text{-}MIFE}}_{P,Q,b}(\mathcal{A})$ in the generic multilinear map model, consider any input-consistent sequence $\boldsymbol{\tau} = (t_1, \ldots, t_m)$ of query times (Definition 4.3). By definition of the encryption procedure, the corresponding ciphertexts for those query times are encoded elements that refer to formal polynomials (Remark 5.6) of the form $\mathsf{ct}_{t_i,h} = \alpha_{t_i,h}\hat{\mathbf{M}}_{t_i,h}$, where $\alpha_{t_i,h}$ is a scalar and $\hat{\mathbf{M}}_{t_i,h}$ is a $w \times w$ matrix. We now define the formal polynomial

$$\alpha_{\boldsymbol{\tau}} = \prod_{i \in [m],\ h \in [d]} \alpha_{t_i,h}$$

(intuitively, the α coefficient that would be present, for a given query sequence $\boldsymbol{\tau}$, in an honest evaluation of the program), as well as the tuple of formal polynomials

$$\hat{\mathbf{M}}|_{\boldsymbol{\tau}} = \left(\hat{\mathbf{M}}_{t_{\mathsf{inp}(1)},\mathsf{inp.h}(1)}, \ldots, \hat{\mathbf{M}}_{t_{\mathsf{inp}(\ell)},\mathsf{inp.h}(\ell)}\right)$$

(intuitively, the matrices whose entries would be involved in an honest evaluation of the program). Finally, we say that a formal polynomial $z_{\boldsymbol{\tau},b}$ is *consistent* with the query sequence $\boldsymbol{\tau}$ if it can be expressed as a polynomial in the entries of the correct vectors and matrices ($\hat{\mathbf{s}}$, $\hat{\mathbf{M}}|_{\boldsymbol{\tau}}$, and $\hat{\mathbf{t}}$), scaled by the correct blinding coefficient, $\alpha_{\boldsymbol{\tau}}$. More precisely, $z_{\boldsymbol{\tau}}$ is consistent with $\boldsymbol{\tau}$ if it is identically equal to a formal polynomial of the form

$$z_{\boldsymbol{\tau}} = \alpha_{\boldsymbol{\tau}} \cdot p_{\boldsymbol{\tau}}(\hat{\mathbf{s}},\ \hat{\mathbf{M}}|_{\boldsymbol{\tau}},\ \hat{\mathbf{t}})$$

for some polynomial $p_{\boldsymbol{\tau}}$ of degree $\mathrm{poly}(\lambda)$.

Lemma 5.8 (Decomposition of Zero-Test Queries)**.** *Fix any efficient adversary $\mathcal{A}$. In the experiment $\mathsf{Expt}^{\mathsf{1SK\text{-}MIFE}}_{P,Q,b}(\mathcal{A})$, with all but negligible probability, every* MM.ZeroTest *query made by $\mathcal{A}$ that is valid (i.e., whose handle is at the top-level universe $\mathcal{U}$), refers to a polynomial (Remark 5.6) formally equal to a sum of (potentially exponentially many) query-consistent polynomials of the form*

$$z = \sum_{\boldsymbol{\tau}} \alpha_{\boldsymbol{\tau}} \cdot p_{\boldsymbol{\tau}}(\hat{\mathbf{s}},\ \hat{\mathbf{M}}|_{\boldsymbol{\tau}},\ \hat{\mathbf{t}}),$$

and each polynomial $p_{\boldsymbol{\tau}}$ is allowable (Definition 2.6) and consistent with a query sequence $\boldsymbol{\tau}$ (Definition 5.7).

Proof. Consider any valid formal polynomial z submitted to MM.ZeroTest. First, we expand the polynomial z into a sum of monomials (for purposes of analysis, not by the scheme), and collect like terms with respect to the α variables. Each term in the resulting expression must be encoded at the top-level universe $\mathcal{U}$, since some valid zero-testing handle refers to their sum. This means, in particular, that the index set of each term must contain a partition of every $\mathcal{U}_i$.

The only variables available to the adversary whose index sets contain elements of $\mathcal{U}_i$ are the ciphertexts $\mathsf{ct}_{t,h}$ generated during time steps $t \in \mathcal{T}^{(i)}$, where $\mathcal{T}^{(i)}$ is the set of all times at which the adversary made chosen-plaintext queries for input slot i. For these time steps, we will assume that the partitions selected by the challenger:

$$\left(P_t = (S^{(i)}_{t,1}, \ldots, S^{(i)}_{t,d}) : t \in \mathcal{T}^{(i)}\right)$$

are distinct, since each is drawn independently uniform from a family of size 2^λ, regardless of the adversary's queries, and thus by the birthday bound a collision occurs with negligible probability.

This implies that the index sets $S^{(i)}_{t,h}$ are distinct elements of the exclusive partition family $\mathcal{F}_{i_t}$, and thus by Lemma 2.14, for each $i \in [m]$, the only monomials whose index sets can cover each $\mathcal{U}_i$ all share the

same value of the partition P_t (and hence of t), and thus are precisely products of one element from each component of the same query ciphertext, ct_{t_i}. Finally, for each $h \in [d]$, we note that the h^{th} term of each such ciphertext contains precisely the factors $\alpha_{t_i,h}$ and $\mathbf{M}_{t_i,h}$. Thus, letting $\boldsymbol{\tau} = (t_1, \ldots, t_m) \subseteq [Q]$, we conclude that such monomials have precisely a leading factor of $\alpha_{\boldsymbol{\tau}}$, while the remaining factors are drawn from $\mathbf{M}|_{\boldsymbol{\tau}}$, as desired. We observe that each such monomial (and hence their sum, $p_{\boldsymbol{\tau}}$) must be allowable (Definition 2.6), since all entries of each vector and matrix $\hat{\mathbf{s}}, \hat{\mathbf{M}}|_{\boldsymbol{\tau}}, \hat{\mathbf{t}}$ are encoded at the same index set, and thus the monomial can only include one factor from each. Finally, the degree of the polynomial $p_{\boldsymbol{\tau}}$ must be at most $\mathrm{poly}(\lambda)$, since the index set of any formal polynomial grows with its degree, and the size of any valid index set is bounded by the size of the top-level universe $\mathcal{U}$. □

We are now ready to present the main proof of Theorem 5.5.

5.2 Proof of Theorem 5.5

Proof. Fix an efficient adversary $\mathcal{A}$ for the experiment $\mathsf{Expt}^{\mathsf{1SK\text{-}MIFE}}_{P,\mathrm{poly}(\lambda),b}(\mathcal{A})$ in the generic graded encoding model. We will show that for every admissible trace π in the experiment (Definition 4.5), except for failure events of negligible probability, the probability that the experiment yields the trace π when $b = 0$ differs by a negligible amount from the probability that it yields the trace π when $b = 1$. It then follows immediately that $\mathsf{Adv}^{\mathsf{1SK\text{-}MIFE}}_{P,Q}(\mathcal{A}) = |W_0 - W_1|$ is negligible, as desired.

First, we note that in any trace π, the only responses sent to $\mathcal{A}$ are either (a) handles in the multilinear map, via $\mathsf{MM.Encode}$, from the public parameters and from ciphertexts generated for chosen-plaintext queries; (b) handles in the multilinear map, via $\mathsf{MM.Add}$, $\mathsf{MM.Mult}$, from queries to the generic map oracle $\mathcal{M}$; or else (c) answers to $\mathsf{MM.ZeroTest}$ queries on handles in the multilinear map. Since in the generic model the handles for (a) and (b) are uniform independent nonces, their distribution clearly does not depend on b. Thus, our task reduces to showing that for each $\mathsf{MM.ZeroTest}$ query, the probability of each response ("zero", "nonzero") differs by a negligible amount between the cases $b = 0$ and $b = 1$. The claim will then follow by a union bound, since $\mathcal{A}$ (being efficient) can make only polynomially many oracle queries.[7]

Fix a valid $\mathsf{MM.ZeroTest}$ query, which refers to a formal multivariate polynomial z (Remark 5.6, Definition C.2. By Lemma 5.8, z is identically equal to a polynomial of the form

$$\sum_{\boldsymbol{\tau}} \alpha_{\boldsymbol{\tau}} \cdot p_{\boldsymbol{\tau}}(\hat{\mathbf{s}},\ \hat{\mathbf{M}}|_{\boldsymbol{\tau}},\ \hat{\mathbf{t}}) \ ,$$

where each polynomial $p_{\boldsymbol{\tau}}$ is allowable (Definition 2.6) and consistent with the query sequence $\boldsymbol{\tau}$ (Definition 5.7). For each bit $b \in \{0,1\}$, let $\mathbf{x}_{\boldsymbol{\tau},b} = (x_{t_1,b}, \ldots, x_{t_m,b})$ be the chosen-plaintext queries corresponding to $\boldsymbol{\tau}$ in the adversary's execution trace up to the point of query z. Since by assumption the execution trace is admissible (Definition 4.5), we have $P(\mathbf{x}_{\boldsymbol{\tau},0}) = P(\mathbf{x}_{\boldsymbol{\tau},1})$. By Lemma 2.7, we now conclude that each formal polynomial $p_{\boldsymbol{\tau}}$, when evaluated on the real distribution of values in $\mathbb{Z}_q$ from the oracle's table, is either zero with probability 1 for both values of $b \in \{0,1\}$, or else is nonzero with all but negligible probability for both values of $b \in \{0,1\}$. We consider the following cases:

- Suppose that for all $\boldsymbol{\tau}$ in the formal sum for z, the polynomial $p_{\boldsymbol{\tau}}$ evaluates to zero on its argument's entire support. In this case, the entire query z will evaluate to zero always, regardless of the value of b.

[7] Technically, we must also show that the distribution of the values in the oracle's table, conditioned on each possible subsequence of past oracle query-response pairs (assuming no failure events), has negligible statistical distance from its prior distribution from $\mathsf{MM.Setup}$; this follows by a standard conditional probability argument, given that the probability of each failure event is negligible.

- Suppose that for some $\boldsymbol{\tau}^*$ in the formal sum for z, the polynomial $p_{\boldsymbol{\tau}^*}$ evaluates to zero negligibly often, regardless of the value of b (and consider the lexicographically first such $\boldsymbol{\tau}^*$, without loss of generality). Then for both values of b, when the query z is instantiated with the real distribution of all values except the α variables, $p_{\boldsymbol{\tau}^*}$ evaluates to a polynomial function of the α variables which, with all but negligible probability, is not identically zero. Since the distribution over the α variables is statistically close to independently uniform over $\mathbb{Z}_q$, the Schwartz-Zippel lemma implies that the entire query z will evaluate to a nonzero value regardless of the value of b, except for failure events with negligible probability.

Thus, for each MM.ZeroTest query, the probability that the answer is "zero" differs by a negligible amount between the cases $b = 0$ and $b = 1$, as desired. □

6 Extensions

Stateful Encryption. In the construction of Section 5, since encryption is required to be stateless, we need to generate a fresh partition for each encryption (and rely on the birthday bound to prevent collisions). However, in many applications of SK-MIFE, it may be reasonable to modify the encryption procedure to be stateful. For instance, suppose a client is encrypting an entire database to be stored on a remote server (and later queried according to the functions for which we reveal MIFE evaluation keys). Here the client may know the contents of the entire database in advance, or may be able to retain local state between interactions with the server. In either case, if the maximum number of database elements N is known in advance, then we can simply replace the $(2^\lambda, d)$-exclusive partition families in the construction (Section 5) with $(2^{\lceil \log N \rceil}, d)$-exclusive partition families, and instead of sampling a partition index uniformly at random for each encryption, use the partitions in order: the i^{th} partition for the i^{th} encryption operation, for each $i \in [N]$.

7 Conclusions

We presented a secret-key multi-input functional encryption scheme for functionalities that can be captured by a *generalized* branching programs of polynomial length and width. An interesting functionality in this family is *comparison* which enables comparisons of symmetrically encrypted data. We refer to this specific functionality as order-revealing encryption (ORE). ORE can be used to answer range queries on symmetrically encrypted data in one round and in logarithmic time in the size of the database.

Our construction is inspired by obfuscation techniques, but does not use obfuscation. Instead it is built directly from multilinear maps and is substantially simpler than current obfuscation-based schemes. While the resulting order-revealing encryption (ORE) scheme is still too inefficient for practical use, it provides a first step towards building usable ORE systems. We hope that future work will further improve the efficiency of ORE and, more generally, the efficiency of secret-key multi-input functional encryption.

Acknowledgments

This work was supported by NSF, the DARPA PROCEED program, a grant from ONR, and by a Google faculty scholarship. Opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA.

Research supported in part from a DARPA/ONR PROCEED award, NSF Frontier Award 1413955, NSF grants 1228984, 1136174, 1118096, and 1065276, a Xerox Faculty Research Award, a Google Faculty Research Award, an equipment grant from Intel, and an Okawa Foundation Research Grant. This material is based upon work supported by the Defense Advanced Research Projects Agency through the U.S. Office of Naval Research under Contract N00014-11-1-0389. The views expressed are those of the author(s) and do not reflect the official policy or position of the Department of Defense, the National Science Foundation, or the U.S. Government.

References

[ABG+13] Prabhanjan Ananth, Dan Boneh, Sanjam Garg, Amit Sahai, and Mark Zhandry. Differing-inputs obfuscation and applications. *IACR Cryptology ePrint Archive*, 2013:689, 2013.

[AGIS14] Prabhanjan Ananth, Divya Gupta, Yuval Ishai, and Amit Sahai. Optimizing obfuscation: Avoiding barrington's theorem. *IACR Cryptology ePrint Archive*, 2014:222, 2014.

[AGVW13] Shweta Agrawal, Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption: New perspectives and lower bounds. In *CRYPTO*, pages 500–518, 2013.

[AKSX04] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Order preserving encryption for numeric data, 2004.

[Bar86] David A. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC1. In *Proceedings, 18th ACM STOC*, pages 1–5, 1986.

[BBC+14] Boaz Barak, Nir Bitansky, Ran Canetti, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Obfuscation for evasive functions. In *TCC*, pages 26–51, 2014.

[BCLO09] Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam O'Neill. Order-preserving symmetric encryption. In *EUROCRYPT*, pages 224–241, 2009.

[BCO11] Alexandra Boldyreva, Nathan Chenette, and Adam O'Neill. Order-preserving encryption revisited: Improved security analysis and alternative solutions. In *CRYPTO*, 2011.

[BCP14] Elette Boyle, Kai-Min Chung, and Rafael Pass. On extractability obfuscation. In *TCC*, pages 52–73, 2014.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In *CRYPTO*, pages 213–229, 2001.

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *CRYPTO*, pages 1–18, 2001.

[BGK+14] Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Protecting obfuscation against algebraic attacks. In *EUROCRYPT*, pages 221–238, 2014.

[BO13] Mihir Bellare and Adam O'Neill. Semantically-secure functional encryption: Possibility results, impossibility results and the quest for a general definition. In *CANS*, pages 218–234, 2013.

[BR14] Zvika Brakerski and Guy N. Rothblum. Virtual black-box obfuscation for all circuits via generic graded encoding. In *TCC*, pages 1–25, 2014.

[BS03] Dan Boneh and Alice Silverberg. Applications of multilinear forms to cryptography. *Contemporary Mathematics*, 324(1):71–90, 2003.

[BS14] Zvika Brakerski and Gil Segev. Function-private functional encryption in the private-key setting.

Cryptology ePrint Archive, Report 2014/550, 2014. http://eprint.iacr.org/.

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In *Proc. of TCC*, pages 253–273, 2011.

[BW07] Dan Boneh and Brent Waters. Conjunctive, subset, and range queries on encrypted data. In *TCC*, pages 535–554, 2007.

[BZ14] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In *CRYPTO*, pages 480–499, 2014.

[CIJ+13] Angelo De Caro, Vincenzo Iovino, Abhishek Jain, Adam O'Neill, Omer Paneth, and Giuseppe Persiano. On the achievability of simulation-based security for functional encryption. In *CRYPTO*, pages 519–535, 2013.

[CLT13] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Practical multilinear maps over the integers. In *CRYPTO*, pages 476–493, 2013.

[GGG+14] Shafi Goldwasser, S. Dov Gordon, Vipul Goyal, Abhishek Jain, Jonathan Katz, Feng-Hao Liu, Amit Sahai, Elaine Shi, and Hong-Sheng Zhou. Multi-input functional encryption. In *EUROCRYPT*, pages 578–602, 2014.

[GGH13a] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In *EUROCRYPT 2013*. Springer Berlin Heidelberg, 2013.

[GGH+13b] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. Cryptology ePrint Archive, Report 2013/451, 2013. http://eprint.iacr.org/.

[GGHZ14] Sanjam Garg, Craig Gentry, Shai Halevi, and Mark Zhandry. Fully secure functional encryption without obfuscation. Cryptology ePrint Archive, Report 2014/666, 2014. http://eprint.iacr.org/.

[GHL+14] Craig Gentry, Shai Halevi, Steve Lu, Rafail Ostrovsky, Mariana Raykova, and Daniel Wichs. Garbled RAM revisited. In *EUROCRYPT*, pages 405–422, 2014.

[GKP+13] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In *STOC*, 2013.

[GM82] Shafi Goldwasser and Silvio Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *STOC*, pages 365–377, 1982.

[GVW12] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption with bounded collusions via multi-party computation. In *CRYPTO*, pages 162–179, 2012.

[Jou00] Antoine Joux. A one round protocol for tripartite Diffie-Hellman. In *Proceedings of the 4th International Symposium on Algorithmic Number Theory*, pages 385–394, 2000.

[Kil88] Joe Kilian. Founding cryptography on oblivious transfer. In *STOC*, 1988.

[KSW08] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *EUROCRYPT*, pages 146–162, 2008.

[LO13] Steve Lu and Rafail Ostrovsky. How to garble RAM programs. In *EUROCRYPT*, 2013.

[Mil04] Victor S Miller. The Weil pairing, and its efficient calculation. *Journal of Cryptology*, 2004.

[MOV93] Alfred Menezes, Tatsuaki Okamoto, and Scott A. Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. *IEEE Transactions on Information Theory*, 39(5):1639–1646, 1993.

[MTY13] Tal Malkin, Isamu Teranishi, and Moti Yung. Order-preserving encryption secure beyond one-wayness. *IACR Cryptology ePrint Archive*, 2013:409, 2013.

[O'N10] Adam O'Neill. Definitional issues in functional encryption. Cryptology ePrint Archive, Report 2010/556, 2010. http://eprint.iacr.org/.

[PLZ13] Raluca A. Popa, Frank H. Li, and Nickolai Zeldovich. An ideal-security protocol for order-preserving encoding. In *S&P*, pages 463–477, 2013.

[SBC+07] Elaine Shi, John Bethencourt, Hubert T.-H. Chan, Dawn Xiaodong Song, and Adrian Perrig. Multi-dimensional range query over encrypted data. In *S&P*, pages 350–364, 2007.

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In *EUROCRYPT*, pages 256–266, 1997.

[SSW09] Emily Shen, Elaine Shi, and Brent Waters. Predicate privacy in encryption systems. In *TCC*, pages 457–473, 2009.

[SZ14] Amit Sahai and Mark Zhandry. Obfuscating low-rank matrix branching programs. Cryptology ePrint Archive, Report 2014/773, 2014. http://eprint.iacr.org/.

[Zha14] Mark Zhandry. How to avoid obfuscation using witness PRFs. Cryptology ePrint Archive, Report 2014/301, 2014. http://eprint.iacr.org/.

A Proofs

A.1 Proof of Lemma 2.7

To prove the lemma, we need some additional facts about formal polynomials over entries of matrices, for which we refer to the work of Sahai and Zhandry [SZ14].

Let $w \in \mathbb{Z}^+$. Let $\widehat{\mathbf{M}}_1$ be a $1 \times w$ matrix (that is, a row vector), $\widehat{\mathbf{M}}_k$ for $k \in [2, n-1]$ be $w \times w$ matrices, and $\widehat{\mathbf{M}}_n$ be a $w \times 1$ matrix (that is, a column vector).

Definition A.1 ([SZ14])**.** Let $w, \widehat{\mathbf{M}}_k$ be as above. Consider a multilinear polynomial p on the variables in $\{\widehat{\mathbf{M}}_k\}_{k\in[n]}$. We call p *allowable* if each monomial in the expansion of p contains at most one variable from each of the $\widehat{\mathbf{M}}_k$.

An example of an allowable polynomial is the *matrix product polynomial* $\widehat{\mathbf{M}}_1\widehat{\mathbf{M}}_2\cdots\widehat{\mathbf{M}}_n$. Now, fix a (large) field $\mathbb{F}$, and let $\mathbf{M}_k$ be matrices over $\mathbb{F}$ of the same shape as $\widehat{\mathbf{M}}_k$ for $k = 1, \ldots, n$. Let $\mathbf{R}_k$ be $w \times w$ matrices of variables for $k \in [n]$, and let $\mathbf{R}_k^{-1}$ be the inverse matrix of $\mathbf{R}_k$. Let $\mathbf{R}_0 = \mathbf{R}_{n+1} = 1$. Now suppose we set

$$\widehat{\mathbf{M}}_k = \mathbf{R}_{k-1} \cdot \mathbf{M}_k \cdot \mathbf{R}_k^{-1}$$

Now given a polynomial p over the $\widehat{\mathbf{M}}_k$, there are two ways of looking at p: as a polynomial over its formal variables (the $\widehat{\mathbf{M}}_k$), and as a rational function over the matrices $\mathbf{R}_k$. We now give conditions for which we can relate the two:

Theorem A.2 ([SZ14])**.** *Let $\mathbb{F}, w, \mathbf{M}_k, \mathbf{R}_k, \widehat{\mathbf{M}}_k$ be as above. Consider an allowable polynomial p in the $\widehat{\mathbf{M}}_k$, and suppose p, after making the substitution $\widehat{\mathbf{M}}_k = \mathbf{R}_{k-1} \cdot \mathbf{M}_k \cdot \mathbf{R}_k^{-1}$, is identically 0 as a rational function over the matrices $\mathbf{R}_k$. Then the following is true:*

- *If* $\mathbf{M}_1\mathbf{M}_2\cdots\mathbf{M}_n \neq 0$*, then* p *is identically zero as a polynomial over its formal variables, namely the* $\widehat{\mathbf{M}}_k$.
- *If* $\mathbf{M}_1\mathbf{M}_2\cdots\mathbf{M}_n = 0$ *but*

$$\mathbf{M}_1\mathbf{M}_2\cdots\mathbf{M}_{n-1} \neq 0^{1\times d_n}$$
$$\mathbf{M}_2\cdots\mathbf{M}_{n-1}\mathbf{M}_n \neq 0^{d_2\times 1}$$

then p*, as a polynomial over the* $\widehat{\mathbf{M}}_k$*, is a constant multiple of the matrix product polynomial* $\widehat{\mathbf{M}}_1\widehat{\mathbf{M}}_2\cdots\widehat{\mathbf{M}}_n$.

We note that Sahai and Zhandry [SZ14] used the adjugate matrix $\mathbf{R}_k^{\mathsf{adj}}$ instead of the inverse matrix $\mathbf{R}_k^{-1}$ in Theorem A.2. However, the only difference between these two matrices is a multiplicative factor equal to the determinant $\det(\mathbf{R}_k)$, which with overwhelming probability is non-zero. It is therefore straightforward to adapt their proof to use the matrices $\mathbf{R}_k^{-1}$ in the case of large fields $\mathbb{F}$, which is the only setting we consider.

Now we can prove Lemma 2.7. We have matrices

$$\mathsf{MBPSelect}(\hat{P}, \mathbf{x}_b) = \left(\hat{\mathbf{s}}^\intercal,\ \hat{\mathbf{M}}_1^b, \ldots, \hat{\mathbf{M}}_\ell^b,\ \hat{\mathbf{t}}\right)$$

for some b, and an allowable polynomial p over them. We are given that

$$\mathsf{MBPEval}(\mathsf{MBPSelect}(\hat{P}, \mathbf{x}_0)) = 0 \quad\iff\quad \mathsf{MBPEval}(\mathsf{MBPSelect}(\hat{P}, \mathbf{x}_1)) = 0.$$

We also know that P is a non-shortcutting branching program. Our goal is to show that

$$p(\mathsf{MBPSelect}(\hat{P}, \mathbf{x}_0)) = 0 \quad\iff\quad p(\mathsf{MBPSelect}(\hat{P}, \mathbf{x}_0)) = 0$$

with overwhelming probability.

There are two cases. If p is a multiple of the matrix product polynomial $\mathsf{MBPEval}$, then the lemma follows trivially. In the case where p is not a multiple of $\mathsf{MBPEval}$, we can invoke Theorem A.2 and see that with overwhelming probability p evaluates to non-zero for both $b = 0$ and $b = 1$.

In particular, assume p is not a multiple of the matrix product polynomial. If p evaluates to zero with non-negligible probability, by the Schwartz-Zippel lemma, it must be identically zero as a polynomial over the randomization matrices $\mathbf{R}$. If $\mathsf{MBPEval}(\mathsf{MBPSelect}(\hat{P}, \mathbf{x}_b)) \neq 0$ for both b, then the matrices $\hat{\mathbf{s}}^\intercal, \hat{\mathbf{M}}_1^b, \ldots, \hat{\mathbf{M}}_\ell^b, \hat{\mathbf{t}}$ satisfy the first set of requirements in Theorem A.2, and it follows that p is identically zero as a polynomial over the variables $\mathsf{MBPSelect}(\hat{P}, \mathbf{x}_b)$, which means it is also a (zero) multiple of the matrix product polynomial, a contradiction.

Alternatively, if $\mathsf{MBPEval}(\mathsf{MBPSelect}(\hat{P}, \mathbf{x}_b)) = 0$ for both b, then because P is non-shortcutting, it follows that $\hat{\mathbf{s}}^\intercal, \hat{\mathbf{M}}_1^b, \ldots, \hat{\mathbf{M}}_\ell^b, \hat{\mathbf{t}}$ satisfy the second set of requirements in Theorem A.2. This then shows that p is a multiple of the matrix product polynomial, a contradiction.

B Optimized Branching Program for Comparisons

In Section 3, we outline an approach to optimizations on layered automata. For completeness, we now give the full description of a layered automaton for the comparisons problem.

Let $x, y \in [B^t - 1]$ be represented in base B: $x = x_{t-1}\ldots x_1x_0, y = y_{t-1}\ldots y_1y_0$. We interleave x, y as $x_{t-1}y_{t-1}y_{t-2}x_{t-2}x_{t-3}\ldots$. Our four state layered automaton is as follows:

- **Layer $4i$:** Every fourth layer, starting with layer 0, has $B+2$ states with labels $>, =, <, \emptyset_1, \ldots, \emptyset_{B-1}$. The state $>, =, <$ states represent the result of comparing the first $2i$ digits of x with the first $2i$ digits of y. In other words, starting at the start state, we will reach state $=$ (resp. $>$, $<$) if the integer represented by the first $2i$ digits of x is equal to (resp. greater than, less than) the integer represented by the first $2i$ digits of y. For layer 0, we mark $=$ as the start state. The $\emptyset_i$ states are not used. Upon reading digit $4i$ as d, which is digit $2i$ of x, the automaton will make the following transitions from layer $4(i-1)+3$ to $4i$ (layer $4(i-1)+3$ is described below):
  - $> \to >$
  - $< \to <$
  - $d \to =$
  - $d' \to >$ for $d' < d$
  - $d' \to <$ for $d' > d$
- **Layer $4i+1$:** Every fourth layer, starting with layer 1, has the states $>, <, 0, \ldots, B-1$. $>$ (resp. $<$) are carried over from layer $4i$: the integer represented by the first $2i$ bits of x is greater than (resp. less than) the integer represented by the first $2i$ bits of y. The state d for a digit $d \in \{0, \ldots, B-1\}$ represents that the first $2i$ bits of x are identical to the first $2i$ bits of y, and digit $2i+1$ of x is equal to d. Upon reading digit $4i+1$ as d, which is digit $2i+1$ of x, the automaton will make the following transitions from layer $4i$ to $4i+1$:
  - $> \to >$
  - $< \to <$
  - $= \to d$
  - $\emptyset_i \to d$
- **Layer $4i+2$:** Every fourth layer, starting with layer 2, has the states $>, =, <, \emptyset_1, \ldots, \emptyset_{B-1}$. The states are identical to layer $4i$, in the sense that they represent the comparison of the first $2i+1$ digits of x to the first $2i+1$ digits of y. The only difference is the transitions, since the transition into layer $4i+2$ is determined by most recent digit of y rather than x. The transition on reading digit d are as follows:
  - $> \to >$
  - $< \to <$
  - $d \to =$
  - $d' \to <$ for $d' < d$
  - $d' \to >$ for $d' > d$
- **Layer $4i+3$:** Every fourth layer, starting with layer 1, has the states $>, <, 0, \ldots, B-1$. $<, >$ are carried over from layer $4i+2$, and represent the comparison of the first $2i+1$ digits of each integer, in the case where they are not equal. In the case where they are equal, and the most recent digit (which is the digit $2i+1$ of y) is d, the automaton will be in state d. This gives the following transitions, which are identical to layer $4i+1$:
  - $> \to >$

- $<\rightarrow<$
- $=\rightarrow d$
- $\emptyset_i \rightarrow d$

C The Generic Multilinear Map Model

To define security for multilinear maps, we now define a generic model, represented by a stateful oracle $\mathcal{M}$, that captures the multilinear map functionality. We say a scheme that uses multilinear maps is "secure in the generic multilinear map model" if, for any concrete adversary breaking the real scheme, there is an ideal adversary breaking a modified scheme in which every access to the multilinear map operations (both by the construction and by the adversary) is replaced by access to a stateful oracle $\mathcal{M}$ which performs the corresponding arithmetic operations internally. Our definition of the oracle $\mathcal{M}$ is essentially the same as in other works which use the generic multilinear map model [BR14, BGK$^+$14]). We define the oracle formally as follows.

Definition C.1 (Ideal Multilinear Map Oracle ([GGH$^+$13b, BR14, BGK$^+$14]))**.** An *ideal multilinear map oracle* is a stateful oracle $\mathcal{M}$ that responds to queries as follows.

- On a query $\mathsf{MM.Setup}(\mathcal{U}, 1^\lambda)$, the oracle will generate a value sp as a fresh nonce (i.e., distinct from any previous choices) uniformly at random from $\{0,1\}^\lambda$, generate a prime q as in the real setup procedure, set $\mathsf{pp} = q$, and return $(\mathsf{pp}, \mathsf{sp})$. It will also store the values generated, initialize an internal table $T \leftarrow \{\}$ (to store "handles", as described below), and set internal state so that subsequent $\mathsf{MM.Setup}$ queries fail.

- On a query $\mathsf{MM.Encode}(k,\ x,\ \mathcal{S})$, where $k \in \{0,1\}^\lambda$ and $x \in \mathbb{Z}_q$, the oracle will check that $k = \mathsf{sp}$ and $\mathcal{S} \subseteq \mathcal{U}$ (returning $\perp$ if the check fails). If the check passes, the oracle will generate a fresh nonce ("handle") h uniformly at random from $\{0,1\}^\lambda$, add the entry $h \mapsto (x, \mathcal{S})$ to the table T, and return h.

- On a query $\mathsf{MM.Add}(k, h_1, h_2)$, where $k, h_1, h_2 \in \{0,1\}^\lambda$, the oracle will check that $k = \mathsf{pp}$, and that the handles h_1, h_2 are present in its internal table T, and are mapped to values, resp., $(x_1, \mathcal{S}_1)$ and $(x_2, \mathcal{S}_2)$ such that $\mathcal{S}_1 = \mathcal{S}_2 = \mathcal{S} \subseteq \mathcal{U}$ (returning $\perp$ if the check fails). If the check passes, the oracle will generate a fresh handle h uniformly at random from $\{0,1\}^\lambda$, set $x \leftarrow x_1 + x_2 \in \mathbb{Z}_q$, add the entry $h \mapsto (x, \mathcal{S})$ to the table T, and return h.

- On a query $\mathsf{MM.Mult}(k, h_1, h_2)$, where $k, h_1, h_2 \in \{0,1\}^\lambda$, the oracle will check that $k = \mathsf{pp}$, and that the handles h_1, h_2 are present in its internal table T, and are mapped to values, resp., $(x_1, \mathcal{S}_1)$ and $(x_2, \mathcal{S}_2)$ such that $\mathcal{S}_1 \cup \mathcal{S}_2 \subseteq \mathcal{U}$ (returning $\perp$ if the check fails). If the check passes, the oracle will set $x \leftarrow x_1 \cdot x_2 \in \mathbb{Z}_q$, generate a fresh handle h uniformly at random from $\{0,1\}^\lambda$, add the entry $h \mapsto (x, \mathcal{S}_1 \cup \mathcal{S}_2)$ to the table T, and return h.

- On a query $\mathsf{MM.ZeroTest}(k, h)$, where $k, h \in \{0,1\}^\lambda$, the oracle will check that $k = \mathsf{pp}$, and that the table T contains an entry $h \mapsto (x, \mathcal{U})$ (immediately returning $\perp$ if the check fails). If the check passes, the oracle will return "zero" if $x = 0 \in \mathbb{Z}_q$, and "nonzero" otherwise.

C.1 Queries Referring to Formal Polynomials

As mentioned above in Remark 5.6, to establish security of our construction (Theorem 5.5), we use a more intuitive characterization of the generic multilinear map model: rather than queries in terms of "handles" (nonces), as defined formally in Appendix C, we refer to the model in terms of formal polynomials whose variables are instantiated with real distributions. The following definitions make this language precise.

Definition C.2 (Formal Polynomials for Handles)**.** During the game $\mathsf{Expt}_{P,Q,b}^{\mathsf{1SK\text{-}MIFE}}(\mathcal{A})$ (Section 4.1) in the generic multilinear map model (Definition C.1), we say that an oracle handle h *refers to* a formal polynomial p (at index set S) if either:

- The handle h is the result of running $\mathsf{MM.Encode}(\mathsf{sp}, \hat{\mathbf{s}}, A_s)$ (resp., $\mathsf{MM.Encode}(\mathsf{sp}, \hat{\mathbf{t}}, A_t)$) during $\mathsf{1SK\text{-}MIFE.Setup}$, where $S = A_s$ (resp., A_t), and p is the formal variable $\hat{\mathbf{s}}$ (resp., $\hat{\mathbf{t}}$).[8]
- The handle h is the result of running $\mathsf{MM.Encode}(\mathsf{sp}, \hat{\mathbf{C}}, S)$ during $\mathsf{1SK\text{-}MIFE.Enc}$ (for some index set S), where the matrix $\hat{\mathbf{C}}$ was computed as:

$$\begin{aligned} \hat{\mathbf{C}}_{t_i,h} &= \alpha_{t_i,h}\, \hat{\mathcal{M}}_{\mathsf{inp.j}(i,h)}(x_{t_i,b}) \\ &:= \alpha_{t_i,h}\, \hat{\mathbf{M}}_{t_i,h} \end{aligned}$$

  and p is the product of the formal variables $\alpha_{t_i,h}$ and $\hat{\mathbf{M}}_{t_i,h}$.
- The handle h is the result of a query $\mathsf{MM.Add}(h_1, h_2)$ and p is the polynomial $p_1 + p_2$, where h_1 refers to p_1 (at S) and h_2 refers to p_2 (at S) at the time of the query.
- The handle h is the result of a query $\mathsf{MM.Mult}(h_1, h_2)$ and p is the polynomial $p_1 p_2$, where h_1 refers to p_1 (at S_1) and h_2 refers to p_2 (at S_2) at the time of the query; $S_1 \cap S_2 = \emptyset$; and $S = S_1 \cup S_2$.
- The handle h is the result of a query $\mathsf{MM.Encode}(\perp, c, \emptyset)$ for a scalar $c \in \mathbb{Z}_q$; p is the constant polynomial c; and $S = \emptyset$.

Definition C.3 (Real Values for Formal Polynomials)**.** During the game $\mathsf{Expt}_{P,Q,b}^{\mathsf{1SK\text{-}MIFE}}(\mathcal{A})$ (Section 4.1) in the generic multilinear map model (Definition C.1), for a formal polynomial p, we say that the *real value* of p is $r \in \mathbb{Z}_q$ if either:

- The polynomial p is the formal variable $\hat{\mathbf{s}}$ (resp., $\hat{\mathbf{t}}$), and r is the value of the variable at the time of instantiation in $\mathsf{1SK\text{-}MIFE.Setup}$.
- The polynomial p is the formal variable $\alpha_{t_i,h}$ (resp., $\hat{\mathbf{M}}_{t_i,h}$), and r is the value of the variable at the time of instantiation in $\mathsf{1SK\text{-}MIFE.Enc}$.
- The polynomial p is formally written as $p_1 + p_2$ for some p_1, p_2; the real value of p_1 is r_1; the real value of p_2 is r_2; and $r = r_1 + r_2$.
- The polynomial p is formally written as $p_1 p_2$ for some p_1, p_2; the real value of p_1 is r_1; the real value of p_2 is r_2; and $r = r_1 r_2$.
- The polynomial p is formally written as c for a constant $c \in \mathbb{Z}_q$, and $r = c$.

[8] Note that in Definition C.2, as in the constructions above (Section 5), we interpret all multilinear map operations componentwise for vectors and matrices. Thus, for instance, $\mathsf{MM.Encode}\,(\mathsf{sp}, \hat{\mathbf{s}}, A_s)$ means that $\mathsf{MM.Encode}$ is invoked w times, once on each component of the vector $\hat{\mathbf{s}}$, and Definition C.2 should be interpreted to hold for each of w corresponding formal variables.

Definition C.4 (Index Sets for Formal Polynomials)**.** During the game $\mathsf{Expt}^{\mathsf{1SK\text{-}MIFE}}_{P,Q,b}(\mathcal{A})$ (Section 4.1) in the generic multilinear map model (Definition C.1), for a formal polynomial p, we say that the *index set* of p is $S \subset \mathcal{U}$ if either:

- The polynomial p is the formal variable $\hat{\mathbf{s}}$ (resp., $\hat{\mathbf{t}}$), and S is the value of the index set chosen at the time of instantiation in $\mathsf{1SK\text{-}MIFE.Setup}$.
- The polynomial p is the formal variable $\alpha_{t_i,h}$ (resp., $\hat{\mathbf{M}}_{t_i,h}$), and S is the value of the index set chosen at the time of instantiation in $\mathsf{1SK\text{-}MIFE.Enc}$.
- The polynomial p is formally written as $p_1 + p_2$ for some p_1, p_2; the index set of p_1 is S; and the index set of p_2 is S.
- The polynomial p is formally written as $p_1 p_2$ for some p_1, p_2; the index set of p_1 is S_1; the index set of p_2 is S_2; and $S = S_1 \cup S_2$.
- The polynomial p is formally written as c for a constant $c \in \mathbb{Z}_q$, and $S = \emptyset$.

As an immediate consequence of these definitions, we note the following intuitive properties:

Lemma C.5 (Uniqueness of Real Values)**.** *During the game $\mathsf{Expt}^{\mathsf{1SK\text{-}MIFE}}_{P,Q,b}(\mathcal{A})$ (Section 4.1) if the polynomials p_1 and p_2 are identically equal and the real value (Def. C.3) of p_1 is r, then the real value of p_2 is r.*

Proof. Since $p_1 \equiv p_2$, they are identical when expanded into a sum of monomials. Thus it suffices to prove the claim for a single distributive step. If the real value of $(p_1 + p_2)p_3$ is r, then by case analysis we conclude that that the real values of p_1, p_2, p_3 are, resp., some r_1, r_2, r_3 such that $(r_1 + r_2)r_3 = r$. Hence the real value of $p_1 p_3 + p_2 p_3$ is $r_1 r_3 + r_2 r_3$ as desired. □

Lemma C.6 (Uniqueness of Index Sets)**.** *During the game $\mathsf{Expt}^{\mathsf{1SK\text{-}MIFE}}_{P,Q,b}(\mathcal{A})$ (Section 4.1) if the polynomials p_1 and p_2 are identically equal and the index set (Def. C.4) of p_1 is r, then the index set of p_2 is r.*

Proof. Since $p_1 \equiv p_2$, they are identical when expanded into a sum of monomials. Thus it suffices to prove the claim for a single distributive step. If the index set of $(p_1 + p_2)p_3$ is S, then by case analysis we conclude that that the index sets of p_1, p_2, p_3 are, resp., S_1, S_1, and S_3, for some sets S_1, S_3 such that $S_1 \cup S_3 = S$. Hence the index set of both $p_1 p_3$ and $p_2 p_3$ is $S_1 \cup S_3 = S$, and thus so is that of their formal sum $p_1 p_3 + p_2 p_3$, as desired. □

Lemma C.7 (Evaluation Commutes With Substutition)**.** *During the game $\mathsf{Expt}^{\mathsf{1SK\text{-}MIFE}}_{P,Q,b}(\mathcal{A})$ (Section 4.1) in the generic multilinear map model (Definition C.1), suppose a handle h is mapped to $r \in \mathbb{Z}_q$ in the oracle's table at index set S. Then h refers (Def. C.2) to a formal polynomial p whose index set (Def. C.4) is S and whose real value (Def. C.3) is r.*

Proof. By structural induction on the definition of the mapping. □

In the proofs of Theorem 5.5 and associated lemmas, we make use of Lemmas C.5, C.6, C.7 implicitly.