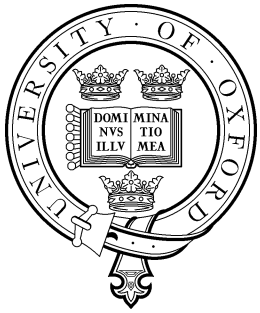
Computer Science Group

THE USE OF MATRICES IN OBFUSCATION

Stephen Drape and Irina Voiculescu

CS-RR-08-12



Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford OX1 3QD

Abstract

There are many programming situations where it would be convenient to conceal the meaning of code, or the meaning of certain variables. This can be achieved through program transformations which are grouped under the term *obfuscation*. This paper presents obfuscation methods for the purpose of concealing the meaning of matrices by changing the pattern of the elements.

We give two separate methods: one which, through splitting a matrix, changes its size and shape, and one which, through a change of basis in a ring of polynomials, changes the values of the matrix and any patterns formed by these. Furthermore, the paper illustrates how matrices can be used in order to obfuscate a scalar value.

This paper considers obfuscations as data refinements. Thus we consider obfuscations at a more abstract level without worrying about implementation issues. For our obfuscations, we can construct proofs of correctness easily. We show how the refinement approach enables us to generalise and combine existing obfuscations.

1 Introduction

An *obfuscation* is a behaviour-preserving transformation whose aim is to make an input program "harder to understand". The landmark paper by Collberg *et al.* [4] gives a range of transformations which can be used as obfuscating transformations. Its purpose is to decrease the opportunities for a user to reverse engineer a commercially supplied program [1, 4]. In this paper, we interpret "harder to understand" as keeping some information secret for as long as possible from some set of adversaries.

After the proof of Barak *et al.* [1], there seems little hope of designing a perfectly-secure software black-box, for any broad class of programs. To date, no one has devised an alternative to Barak's model, in which we would be able to derive proofs of security for systems of practical interest. These theoretical difficulties do not lessen practical interest in obfuscation, nor should it prevent us from placing appropriate levels of reliance on obfuscated systems in cases where the alternative of a hardware black-box is infeasible or uneconomic.

The view of obfuscation from Collberg *et al.* [4] concentrates on concrete data structures such as variables and arrays. However, the thesis of Drape [7] viewed obfuscations at a more abstract level by considering an abstract data-type and defining operations for this data-type — thus we should obfuscate the data-type according to these operations. This work had lead to the development of the specification of obfuscations for imperative programs [9] and for creating obfuscations which impede the effectiveness of program slicing [17].

The focus of the paper consists of a data-type for finite matrices having four operations: scalar multiplication, addition, transposition and multiplication, specified mathematically. We use data refinement [6] to provide a way of proving the correctness of our obfuscated operations. Thus we are guaranteed that our obfuscations are behaviour-preserving. We will demonstrate a number of different obfuscation technique using matrices. Since we consider our operations at a more abstract level than program code, we will be able to discuss how we can generalise our obfuscations.

When creating obfuscations we should make reference to an attack model. In the work of Majumdar *et al.* [17], the obfuscations were created with the intention of trying to protect against an attacker armed with a program slicer. In this paper, we adopt the attack model of Drape [7] in which when defining data-types we also

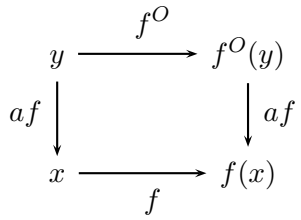


Figure 1: A commuting diagram for data obfuscation

specify a set of assertions which are true for the operations of that data-type. The aim of the obfuscations is to make the proofs of these assertions harder.

2 Preliminaries

In this section we will discuss how we can prove the correctness of our obfuscations and we will define a data-type for matrices.

2.1 Obfuscation as data refinement

In Drape's thesis [7], data obfuscation was considered as a data refinement [6]. Suppose that a data-type D is obfuscated using an obfuscation $\mathcal{O}$ to produce a data-type E. Under the refinement approach, an abstraction function $af :: E \to D$ and a data-type invariant dti are needed such that, for $x :: D$ and $y :: E$:

$$x \rightsquigarrow y \iff (x = af(y)) \wedge dti(y) \tag{1}$$

The term $x \rightsquigarrow y$ is read as "x is obfuscated by y".

For a function $f :: D \to D$, an obfuscated function $f^{\mathcal{O}}$ is correct with respect to f if it satisfies:

$$(\forall x :: D; y :: E)\ x \rightsquigarrow y \Rightarrow f(x) \rightsquigarrow f^{\mathcal{O}}(y)$$

Using Equation (1) we can rewrite this as

$$f \cdot af = af \cdot f^{\mathcal{O}} \tag{2}$$

This equation is illustrated in Figure 1. The abstraction function af is surjective and so we have a function $cf :: D \to E$, called the conversion function, which satisfies $af \cdot cf = id$. Thus we can rewrite Equation (2) to obtain:

$$f = af \cdot f^{\mathcal{O}} \cdot cf \tag{3}$$

and we can use this equation to prove the correctness of $f^{\mathcal{O}}$.

If we also have that $cf \cdot af = id$ then we can rewrite Equation (2) to obtain:

$$f^{\mathcal{O}} = cf \cdot f \cdot af \tag{4}$$

Thus when af is bijective then we can use Equation (4) to give us a way of deriving an obfuscated operation $f^{\mathcal{O}}$ from the original operation f.

2.2 Non-homogeneous operations

Suppose that we have a operation

$$f :: B \to C$$

where B and C are the state spaces of two data-types. Let af_B and af_C be abstraction functions for some obfuscations of B and C. How do we define a correct obfuscation $f^{\mathcal{O}}$ of f? Suppose $x :: B$ and $x \rightsquigarrow y$ and consider:

$$\begin{aligned} & f(x) \rightsquigarrow f^{\mathcal{O}}(y) \\ \equiv \quad & \{\text{Equation (1) using } af_C\} \\ & f(x) = af_C(f^{\mathcal{O}}(y)) \\ \equiv \quad & \{\text{Equation (1) using } af_B\} \\ & f(af_B(y)) = af_C(f^{\mathcal{O}}(y)) \end{aligned}$$

Thus

$$f \cdot af_B = af_C \cdot f^{\mathcal{O}} \tag{5}$$

Some operations, have type:

$$f :: D \times D \to D$$

for some data-type D. If af is an abstraction function for D then the corresponding abstraction for $D \times D$ is

$$cross(af, af)$$

where *cross* is an operation with type

$$cross :: (\alpha \to \gamma, \beta \to \delta) \to (\alpha, \beta) \to (\gamma, \delta)$$

which satisfies

$$cross\ (f, g)\ (a, b) = (f\ a, g\ b) \tag{6}$$

Thus if $f^{\mathcal{O}}$ is an obfuscation of f then using Equation (5) we have that

$$f \cdot cross(af, af) = af \cdot f^{\mathcal{O}} \tag{7}$$

We will be able to use this equation to prove the correctness of binary matrix operations such as addition and multiplication.

2.3 Matrices

A matrix is an array of numbers which are arranged in a meaningful tabular form. It is usually two-dimensional and can have any width and height. It is also possible to use multi-dimensional matrices and, even though these are harder to write down, it is fairly easy to manipulate them in a computer program.

Matrices are used for a wide variety of applications such as solving systems of equations, wavelets, graph theory and graphics. There are applications when it is desirable to hide the meaning of a matrix. Perhaps one such case which is easy to grasp is when, in expressing a rigid body transformation by way of a matrix,

the matrix has a particular structure. For example, the translation of an object by displacements (d_x, d_y, d_z) is usually written if the form

$$\begin{pmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Should somebody wish to hide the fact that a particular matrix is a translation matrix, they would need to design an obfuscation method which changes not only the values, but also the visible pattern of these values.

The matrix $\mathbf{M}$ which has r rows and c columns (for natural numbers r and c) will be denoted by $\mathbf{M}^{r\times c}$. The element of $\mathbf{M}$ that is located at row i and column j will be written as $\mathbf{M}(i,j)$, and, for simplicity, assumed to be rational. The operation $\mathsf{dim}\,(\mathbf{M})$ returns the dimensions of $\mathbf{M}$.

There are well-defined algebraic operations on matrices, and the set of square matrices with elements from a ring form a ring under matrix addition and multiplication.

In Figure 2 we define a data-type for matrices — for the rest of the paper we will suppose that $Matrix\ \alpha$ is $\mathbb{Q}^{r\times c}$ which denotes matrices with r rows and c columns with rational number elements. So, from our data-type, we would like to obfuscate matrices with the following matrix operations: *scalar multiplication*, *addition*, *transposition* and *multiplication*. In the lower part of Figure 2 we have a possible set of assertions. As we stated in Section 1 we should aim to obfuscate our operations with the intention that they make the proofs of correctness for assertions harder. In this paper, we do not show such proofs but example proofs for other data-types can be found in [7] and [9].

Note that for addition the two matrices must have the same size and for multiplication we need the matrices to be *conformable*, *i.e.* the number of columns of the first is equal to the number of rows in the second. We can define the operations element-wise as follows:

$$\begin{array}{rcl} (\mathsf{scale}\ s\ \mathbf{M})(i,j) & = & s \times \mathbf{M}(i,j) \\ (\mathsf{add}\ (\mathbf{M},\mathbf{N}))(i,j) & = & \mathbf{M}(i,j) + \mathbf{N}(i,j) \\ (\mathsf{transpose}\ \mathbf{M})(i,j) & = & \mathbf{M}(j,i) \\ (\mathsf{mult}\ (\mathbf{M},\mathbf{P}))(i,k) & = & \sum_{j=1}^{c} \mathbf{M}(i,j) \times \mathbf{P}(j,k) \end{array}$$

for matrices $\mathbf{M}^{r\times c}$, $\mathbf{N}^{r\times c}$ and $\mathbf{P}^{c\times d}$ with $i : [0..r)$, $j : [0..c)$ and $k : [0..d)$.

We assume that basic arithmetic operations take constant time and so the computational complexities of $\mathsf{add}\ \mathbf{M}\ \mathbf{N}$, $\mathsf{scale}\ s\ \mathbf{M}$ and $\mathsf{transpose}\ \mathbf{M}$ are all $r{\times}c$ and the complexity of $\mathsf{mult}\ \mathbf{M}\ \mathbf{P}$ is $r{\times}c{\times}d$.

3 Splitting method

Now that we have defined our data-type for matrices and given equations for proving the correctness of matrix obfuscations we are ready to discuss our first obfuscation technique. Collberg *et al.* [4] discuss an obfuscation called an *array split*. This obfuscation was generalised in Drape [8] and so we can apply the concept of splitting to other data-types such as matrices.

Matrix (α)
$\mathsf{scale} :: \alpha \to Matrix\ \alpha \to Matrix\ \alpha$ $\mathsf{add} :: Matrix\ \alpha \times Matrix\ \alpha \to Matrix\ \alpha$ $\mathsf{transpose} :: Matrix\ \alpha \to Matrix\ \alpha$ $\mathsf{mult} :: Matrix\ \alpha \times Matrix\ \alpha \to Matrix\ \alpha$
$\mathsf{transpose} \cdot \mathsf{transpose} = id$ $\mathsf{transpose} \cdot \mathsf{add} = \mathsf{add} \cdot \mathsf{cross}\,(\mathsf{transpose}, \mathsf{transpose})$ $\mathsf{transpose} \cdot (\mathsf{scale}\ s) = (\mathsf{scale}\ s) \cdot \mathsf{transpose}$ $\mathsf{transpose}\ (\mathsf{mult}\ (\mathbf{M}, \mathbf{N})) = \mathsf{mult}\ (\mathsf{transpose}\ \mathbf{N}, \mathsf{transpose}\ \mathbf{M})$ $\mathsf{add}\ (\mathbf{M}, \mathbf{N}) = \mathsf{add}\ (\mathbf{N}, \mathbf{M})$

Figure 2: data-type for Matrices

3.1 Defining a matrix split

Suppose that we want to split a matrix $\mathbf{M}^{r \times c}$ into n matrices, called the *split components*,

$$\mathbf{M} \rightsquigarrow \langle \mathbf{M}_0, \ldots, \mathbf{M}_{n-1} \rangle_{sp}$$

where $\mathbf{M}_i$ has size $r_i \times c_i$ for $i : [0..n)$.

In that data representation, $\mathbf{M}$ is represented by n matrices using a split, called sp, which consists of a *choice function*:

$$ch :: [0..r) \times [0..c) \to [0..n)$$

and a family $\mathcal{F}$ of injective functions where $\mathcal{F} = \{f_t\}_{t:[0..n)}$ such that for each t:

$$f_t :: ch^{-1}\{t\} \rightarrowtail [0..r_t) \times [0..c_t)$$

We define the relationship between $\mathbf{M}$ and the split components element-wise by using the choice function and the appropriate function from $\mathcal{F}$ to decide where an element is mapped to:

$$\mathbf{M}_t(f_t(i,j)) = \mathbf{M}(i,j) \text{ where } t = ch(i,j) \tag{8}$$

The requirement that we have a family of injective functions ensures that we can recover a matrix (and thus its properties) from the split components.

Equation (8) can be considered to be a conversion function and so for a matrix split

$$cf(\mathbf{M}(i,j)) = \mathbf{M}_t(f_t(i,j)) \qquad \text{where } t = ch(i,j) \tag{9}$$

The corresponding abstraction function for some split component $\mathbf{M}_t$ is

$$af(\mathbf{M}_t(i,j)) = \mathbf{M}({f_t}^{-1}(i,j)) \tag{10}$$

where ${f_t}^{-1} \cdot f_t = id$ (which is valid as f_t is injective). Using these definitions we can check that $af \cdot cf = id$.

As an example, consider how we could define a split in which a matrix $\mathbf{M}^{r\times 2c}$ is split vertically into two matrices $\mathbf{M}_0^{r\times c}$ and $\mathbf{M}_1^{r\times c}$. The choice function is defined to be

$$ch(i,j) = j \text{ div } c$$

and the family of functions is:

$$\mathcal{F} = \{f_t = (\lambda\,(i,j)\,.\,(i, j \bmod c)) \,|\, t = 0 \vee t = 1\}$$

The process of splitting a matrix is analogous to the concept of a *partitioned* (or *block*) matrix [14] in which a matrix can be represented by a sequence of smaller submatrices.

3.2 Splitting in squares

A simple matrix split is one which splits a square matrix into four matrices — two of which are square. Using this split we can give definitions of our four operations for split matrices. Suppose that we have a square matrix $\mathbf{M}^{r\times r}$ and choose a positive integer k such that $k < n$. The choice function $ch(i,j)$ is defined as

$$ch(i,j) = \begin{cases} 0 & (0 \le i < k) \wedge (0 \le j < k) \\ 1 & (0 \le i < k) \wedge (k \le j < r) \\ 2 & (k \le i < n) \wedge (0 \le j < k) \\ 3 & (k \le i < n) \wedge (k \le j < r) \end{cases}$$

which can be written as a single formula

$$ch(i,j) = 2\,sgn\,(i \text{ div } k) + sgn\,(j \text{ div } k)$$

where sgn is the signum function. The family of functions $\mathcal{F}$ is defined to be the set

$$\mathcal{F} = \left\{ \begin{array}{lcl} f_0 & = & (\lambda\,(i,j)\,.\,(i,j)) \\ f_1 & = & (\lambda\,(i,j)\,.\,(i,j-k)) \\ f_2 & = & (\lambda\,(i,j)\,.\,(i-k,j)) \\ f_3 & = & (\lambda\,(i,j)\,.\,(i-k,j-k)) \end{array} \right\}$$

Again, we can write this in a single formula:

$$\mathcal{F} = \{f_p = (\lambda\,(i,j)\,.\,(i - k\ (p \text{ div } 2),\ j - k\ (p \bmod 2))) \mid p \in [0..3]\}$$

We call this split the $(k\times k)$-*square split* since the first component of the split is a $k\times k$ square matrix.

So if

$$\mathbf{M}(i,j) = \mathbf{M}_t(f_t(i,j)) \text{ where } t = ch(i,j)$$

then we can write

$$\mathbf{M}^{n\times n} \quad \rightsquigarrow \quad \langle \mathbf{M}_0^{k\times k}, \mathbf{M}_1^{k\times(n-k)}, \mathbf{M}_2^{(n-k)\times k}, \mathbf{M}_3^{(n-k)\times(n-k)} \rangle_{s_k}$$

where the subscript s_k denotes the $(k\times k)$-square split. Pictorially, we can see the $(k\times k)$-square split as follows:

$$\left(\begin{array}{ccc|ccc} a_{(0,0)} & \cdots & a_{(0,k-1)} & a_{(0,k)} & \cdots & a_{(0,n-1)} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{(k-1,0)} & \cdots & a_{(k-1,k-1)} & a_{(k-1,k)} & \cdots & a_{(k-1,n-1)} \\ \hline a_{(k,0)} & \cdots & a_{(k,k-1)} & a_{(k,k)} & \cdots & a_{(k,n-1)} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{(n-1,0)} & \cdots & a_{(n-1,k-1)} & a_{(n-1,k)} & \cdots & a_{(n-1,n-1)} \end{array}\right) \rightsquigarrow \left\langle \begin{array}{cc} \mathbf{M}_0, & \mathbf{M}_1, \\ \mathbf{M}_2, & \mathbf{M}_3 \end{array} \right\rangle_{s_k}$$

Using this split, how can we define our matrix operations given in Figure 4?

The operations for scale and add are fairly straightforward. If

$$\mathbf{M} \rightsquigarrow \langle \mathbf{M}_0, \ldots, \mathbf{M}_3 \rangle_{s_k} \quad \text{and} \quad \mathbf{N} \rightsquigarrow \langle \mathbf{N}_0, \ldots, \mathbf{N}_3 \rangle_{s_k}$$

then

$$\begin{array}{rcl} \mathsf{scale}\ s\ \mathbf{M} & \rightsquigarrow & \langle \mathsf{scale}\ s\ \mathbf{M}_0, \ldots, \mathsf{scale}\ s\ \mathbf{M}_3 \rangle_{s_k} \\ \mathsf{add}\ (\mathbf{M}, \mathbf{N}) & \rightsquigarrow & \langle \mathsf{add}\ (\mathbf{M}_0, \mathbf{N}_0), \ldots, \mathsf{add}\ (\mathbf{M}_3, \mathbf{N}_3) \rangle_{s_k} \end{array}$$

The proofs for these definitions can be found in [7].

How can we define $\mathbf{M}^T$? We propose that

$$\mathbf{M}^T \rightsquigarrow \langle {\mathbf{M}_0}^T, {\mathbf{M}_2}^T, {\mathbf{M}_1}^T, {\mathbf{M}_3}^T \rangle_{s_k}$$

which corresponds to the following property for partitioned matrices:

$$\left(\begin{array}{cc} \mathbf{M}_0 & \mathbf{M}_1 \\ \mathbf{M}_2 & \mathbf{M}_3 \end{array}\right)^T = \left(\begin{array}{cc} {\mathbf{M}_0}^T & {\mathbf{M}_2}^T \\ {\mathbf{M}_1}^T & {\mathbf{M}_3}^T \end{array}\right)$$

Since we defined af, cf and transposition (in Section 2.3) element-wise then we have to proof the correctness of our obfuscation element-wise too. Let denote f denote the element-wise transposition function, then we can use Equation (3) to prove

$$f(\mathbf{M}(i,j)) = af(f^{\mathcal{O}}(cf(\mathbf{M}(i,j))))$$

where cf and af were given in Equations (9) and (10). We have four cases — one for each value of ch. So, suppose that $ch(i,j) = 1$ and then using Equation (3):

$$\begin{array}{ll} & af(f^{\mathcal{O}}(cf(\mathbf{M}(i,j)))) \\ = & \quad \{cf \text{ with } ch(i,j) = 1\} \\ & af(f^{\mathcal{O}}(\mathbf{M}_1(f_1(i,j)))) \\ = & \quad \{f_1 = \lambda\ (i,j).(i, j-k)\} \\ & af(f^{\mathcal{O}}(\mathbf{M}_1(i, j-k))) \\ = & \quad \{\text{assume that } f^{\mathcal{O}}(\mathbf{M}_1(i,j)) = {\mathbf{M}_2}^T(i,j)\} \\ & af({\mathbf{M}_2}^T(i, j-k)) \\ = & \quad \{\text{definition of transpose}\} \\ & af(\mathbf{M}_2(j-k, i)) \end{array}$$

$$
\begin{array}{ll}
= & \{\text{definition of } af\} \\
& \mathbf{M}({f_2}^{-1}(j-k,i)) \\
= & \{{f_2}^{-1}(a,b) = (a+k,b)\} \\
& \mathbf{M}(j,i) \\
= & \{\text{definition of transpose}\} \\
& f(\mathbf{M}(i,j))
\end{array}
$$

The proofs for the other three components are similar and so we have that

$$\mathbf{M}^T \rightsquigarrow \langle {\mathbf{M}_0}^T, {\mathbf{M}_2}^T, {\mathbf{M}_1}^T, {\mathbf{M}_3}^T \rangle_{s_k}$$

The obfuscated operation has complexity $n \times n$.

Finally let us consider how we can multiply split matrices. Let

$$
\begin{array}{rcl}
\mathbf{M}^{n\times n} & \rightsquigarrow & \langle \mathbf{M}_0,\, \mathbf{M}_1,\, \mathbf{M}_2,\, \mathbf{M}_2 \rangle_{s_k} \\
\mathbf{N}^{n\times n} & \rightsquigarrow & \langle \mathbf{N}_0,\ \mathbf{N}_1,\ \mathbf{N}_2,\ \mathbf{N}_3 \rangle_{s_k}
\end{array}
$$

By considering the partitioned matrix product

$$\begin{pmatrix} \mathbf{M}_0 & \mathbf{M}_1 \\ \mathbf{M}_2 & \mathbf{M}_3 \end{pmatrix} \times \begin{pmatrix} \mathbf{N}_0 & \mathbf{N}_1 \\ \mathbf{N}_2 & \mathbf{N}_3 \end{pmatrix}$$

we obtain the following result:

$$
\begin{array}{rcl}
\mathbf{M} \times \mathbf{N} & \rightsquigarrow & \langle (\mathbf{M}_0 \times \mathbf{N}_0) + (\mathbf{M}_1 \times \mathbf{N}_2),\ (\mathbf{M}_0 \times \mathbf{N}_1) + (\mathbf{M}_1 \times \mathbf{N}_3), \\
& & (\mathbf{M}_2 \times \mathbf{N}_0) + (\mathbf{M}_3 \times \mathbf{N}_2),\ (\mathbf{M}_2 \times \mathbf{N}_1) + (\mathbf{M}_3 \times \mathbf{N}_3) \rangle_{s_k}
\end{array}
$$

The computation of $\mathbf{M} \times \mathbf{N}$ using normal matrix multiplication requires n^3 element multiplications. If we multiply the split matrices, does this calculation require more multiplications? From the definition of split matrices, the number of multiplications required to compute each component is:

$$
\begin{array}{l}
\langle k^3 + k^2(n-k),\ k^2(n-k) + k(n-k)^2, \\
\ k^2(n-k) + k(n-k)^2,\ k(n-k)^2 + (n-k)^3 \ \rangle
\end{array}
$$

Adding these up gives

$$
\begin{array}{ll}
& k^3 + 3k^2(n-k) + 3k(n-k)^2 + (n-k)^3 \\
= & (k + (n-k))^3 \\
= & n^3
\end{array}
$$

which is exactly the same number of multiplications as the unsplit version.

To reduce the number of multiplications, we could use *Strassen's algorithm* [5] which performs matrix multiplication by using seven multiplications instead of eight. However the algorithm requires starting with a $2n \times 2n$ matrix and splitting it into four $n \times n$ matrices and so would not be directly applicable for a general matrix.

3.3 Review of matrix splitting

Using our matrix split, we have seen that we can easily define obfuscated operations for our matrix data-type. All of the obfuscated operations have a similar complexity to the original versions. We can also define more complicated splits such as splitting into rectangles rather than squares or by using more split components — see Section 6.1 for more details. Since the matrix split is a generalisation of an array split then we could use matrix splits as obfuscation for arrays. We could do this by folding an array into a matrix, splitting the matrix and then flattening the components back into arrays.

For our matrix data-type (defined in Figure 2) we considered four matrix operations. Could we define obfuscations for other matrix operations? We may wish to compute the inverse of a square matrix — *i.e.*, given a matrix $\mathbf{M}$ we would like a matrix $\mathbf{N}$ such that $\mathbf{M}\times\mathbf{N} = \mathbf{I} = \mathbf{N}\times\mathbf{M}$. We can derive an inverse for matrices which have been split by the $(k\times k)$-square split by using the equation for multiplying two split matrices. This then leads to solving four simultaneous equations. In fact, if

$$\mathbf{M} = \begin{pmatrix} \mathbf{M_0} & \mathbf{M_1} \\ \mathbf{M_2} & \mathbf{M_3} \end{pmatrix}$$

then $\mathbf{M^{-1}}$ is

$$\begin{pmatrix} (\mathbf{M_0} - (\mathbf{M_1}\ \mathbf{M_3}^{-1}\ \mathbf{M_2}))^{-1} & ((\mathbf{M_2}\ \mathbf{M_0}^{-1}\ \mathbf{M_1}) - \mathbf{M_3})^{-1}\,\mathbf{M_0}^{-1}\,\mathbf{M_1} \\ \mathbf{M_3}^{-1}\,\mathbf{M_2}(\mathbf{M_1}\,\mathbf{M_3}^{-1}\,\mathbf{M_2} - \mathbf{M_0})^{-1} & (\mathbf{M_3} - \mathbf{M_2}\,\mathbf{M_0}^{-1}\,\mathbf{M_1})^{-1} \end{pmatrix}$$

(using the expression for inverting partitioned matrices [14]).

If we wish to extend the data-type of matrices to support the *determinant* operation then splitting this data-type proves to very difficult for dense matrices as the computation of a determinant usually requires knowledge of the entire matrix. We can however change a matrix into Jordan form (which preserves the value of the determinant) in which case the determinant is just the product of the diagonal values. However, this method is likely to be more computationally expensive. Since our splitting method is analogous to the concept of partitioned matrices [14] then we can use properties of determinant for partitioned matrices to help use define an obfuscation for det. Consider the following partitioned matrix $\mathbf{M}$:

$$\mathbf{M} = \begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{pmatrix}$$

If $\mathbf{B}$ or $\mathbf{C}$ is the zero matrix $\mathbf{0}$ then we can easily work out the determinant of $\mathbf{M}$:

$$\mathsf{det}\begin{pmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{C} & \mathbf{D} \end{pmatrix} = \mathsf{det}\begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{0} & \mathbf{D} \end{pmatrix} = \mathsf{det}(\mathbf{A})\,\mathsf{det}(\mathbf{D})$$

Then using the identity:

$$\begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{pmatrix} = \begin{pmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{C} & \mathbf{I} \end{pmatrix} \times \begin{pmatrix} \mathbf{I} & \mathbf{A}^{-1}\mathbf{B} \\ \mathbf{0} & \mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B} \end{pmatrix}$$

we have that:

$$\mathsf{det}(\mathbf{M}) = \mathsf{det}(\mathbf{A}) \times \mathsf{det}(\mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B})$$

However this formula is only valid if $\mathbf{A}$ is invertible.

4 Using the Bernstein basis

We have seen that we can obfuscate a matrix by splitting it into many matrices. We can easily define obfuscations for simple operations but it was harder to define obfuscations for calculating inverses and determinants. We will now define an obfuscation that is based on the fact that the elements of a matrix can be used to define the coefficients of a bivariate polynomial.

We denote by $\mathcal{P}[x, y]$ the set of polynomials of variables x and y, with rational coefficients. For a given $n \in \mathbb{N}$, there are several ways to define bases for the ring of degree-n polynomials (see, for example, Lorentz [16]). One is the power basis, denoted by

$$\begin{pmatrix} 1 & x & \dots & x^n \end{pmatrix}$$

and another is the Bernstein basis

$$\begin{pmatrix} B_0^n(x) & B_1^n(x) & \dots & B_n^n(x) \end{pmatrix}$$

where

$$B_k^n(x) = \binom{n}{k} x^k (1-x)^{n-k}, \qquad \forall x \in [0,1], \qquad k = 0, 1, \dots, n \tag{11}$$

4.1 Power-form and Bernstein-form polynomials

Now we consider the Bernstein basis for univariate, bivariate and multivariate polynomials.

4.1.1 Univariate

A power-form polynomial $p \in \mathcal{P}[X]$ of degree $n \in \mathbb{N}$ is given by:

$$p(x) = \sum_{k=0}^{n} a_k x^k, \tag{12}$$

where $a_k \in \mathbb{Q}$. For a given $n \in \mathbb{N}$ there are $n+1$ univariate degree-n Bernstein polynomials, as defined above.

Any univariate power-form polynomial can be represented on the interval $[0, 1]$ using its equivalent Bernstein form as follows:

$$p(x) = \underbrace{\sum_{k=0}^{n} a_k x^k}_{power\ form} = \underbrace{\sum_{k=0}^{n} m_k^n B_k^n(x)}_{Bernstein\ form}$$

where m_k^n are the Bernstein coefficients corresponding to the degree-n base. The two univariate representations $p(x)$ and $p_{\mathcal{B}}(x)$ are equivalent and conversion between them is fairly straightforward. On the unit interval $[0, 1]$ for example, the univariate Bernstein coefficients are easily computed in terms of the power coefficients (also shown by Farouki and Rajan [10]):

$$m_k^n = \sum_{j=0}^{k} \frac{\binom{k}{j}}{\binom{n}{j}} a_j \tag{13}$$

4.1.2 Univariate polynomials written in matrix form

Consider the same polynomial as the one in Equation (12). The calculation in Equation (13) can also be written as a matrix multiplication. The following formulae determine the Bernstein coefficients matrix $\mathbf{M}$ in terms of the power-form coefficients matrix $\mathbf{A}$.

Two other ways of writing the polynomial $p(x)$ are:

$$p(x) = \sum_{k=0}^{n} a_k x^k = \begin{pmatrix} 1 & x & \dots & x^n \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \mathbf{X}\,\mathbf{A}$$

$$p(x) = \sum_{k=0}^{n} m_k^n B_k^n(x) = \begin{pmatrix} B_0^n(x) & B_1^n(x) & \dots & B_n^n(x) \end{pmatrix} \begin{pmatrix} m_0 \\ m_1 \\ \vdots \\ m_n \end{pmatrix} = \mathbf{B}_X\,\mathbf{M}.$$

Rewriting the vector $\mathbf{B}_X$ of Bernstein polynomials in terms of matrix multiplication gives:

$$\begin{aligned}
\mathbf{B}_X &= \begin{pmatrix} B_0^n(x) & B_1^n(x) & \dots & B_n^n(x) \end{pmatrix} \\
&= \begin{pmatrix} \binom{n}{0}(1-x)^n & \dots & \binom{n}{n}x^n \end{pmatrix} \\
&= \begin{pmatrix} \binom{n}{0}\left(1 + \binom{n}{1}(-x) + \dots + \binom{n}{n}(-x)^n\right) & \dots & \binom{n}{n}x^n \end{pmatrix} \\
&= \underbrace{\begin{pmatrix} 1 & x & \dots & x^n \end{pmatrix}}_{X} \underbrace{\begin{pmatrix} 1 & & & \mathrm{O} \\ \binom{n}{0}\binom{n}{1}(-1)^1 & \binom{n}{1}\binom{n-1}{0}(-1)^0 & & \\ \vdots & & \ddots & \\ \binom{n}{0}\binom{n}{n}(-1)^n & \binom{n}{1}\binom{n-1}{n-1}(-1)^{n-1} & \dots & \binom{n}{n}\binom{n-n}{0}(-1)^0 \end{pmatrix}}_{U_X} \\
&= \mathbf{X}\,\mathbf{U}_X, \qquad \forall x \in [0,1].
\end{aligned}$$

So

$$\mathbf{B}_X = \mathbf{X}\,\mathbf{U}_X \tag{14}$$

$$\text{and} \quad p(x) = \mathbf{B}_X\,\mathbf{M} = \mathbf{X}\,\mathbf{U}_X\,\mathbf{M} \tag{15}$$

Now compute the Bernstein coefficients matrix $\mathbf{M}$.

$$\begin{aligned}
\mathbf{X}\,\mathbf{A} &= \mathbf{X}\,\mathbf{U}_X\mathbf{M} \\
\mathbf{M} &= (\mathbf{U}_X)^{-1}\mathbf{A}
\end{aligned}$$

where $(\mathbf{U}_X)^{-1}$ is the inverse matrix of $\mathbf{U}_X$.

4.1.3 The bivariate case in matrix form

The implicit expression of a bivariate polynomial in the power basis can also be rewritten in terms of matrix multiplication:

$$p(x,y) = a_{00} + a_{10}x + a_{01}y + a_{11}xy + \ldots + a_{mn}x^m y^n = \mathbf{X}\mathbf{A}\,\mathbf{Y},$$

where

$$\mathbf{X} = \begin{pmatrix} 1 & x & \ldots & x^m \end{pmatrix} \qquad \mathbf{Y} = \begin{pmatrix} 1 \\ y \\ \vdots \\ y^n \end{pmatrix} \qquad \mathbf{A} = \begin{pmatrix} a_{00} & \ldots & a_{0n} \\ \vdots & & \vdots \\ a_{m0} & \ldots & a_{mn} \end{pmatrix}$$

By analogy with the univariate case,

$$p(x,y) = \mathbf{X}\,\mathbf{A}\,\mathbf{Y} = \mathbf{B}_X\,\mathbf{M}\,\mathbf{B}_Y$$

where $\mathbf{B}_X$ and $\mathbf{B}_Y$ are Bernstein vectors in the variables $x \in [0,1]$ and $y \in [0,1]$. These vectors can be decomposed as shown in Section 4.1.2.

In the case of the Bernstein vector corresponding to the variable $\mathbf{Y}$ and the factor $\mathbf{U}$ in Equation (14) will appear in reverse order. This happens because $\mathbf{B}_Y$ is a column vector (as opposed to $\mathbf{B}_X$ which is a row vector).

Hence

$$\begin{aligned} \mathbf{X}\,\mathbf{A}\,\mathbf{Y} &= \mathbf{X}\,\mathbf{U}_X\,\mathbf{M}\,\mathbf{U}_Y\,\mathbf{Y} \\ \mathbf{M} &= (\mathbf{U}_X)^{-1}\,\mathbf{A}\,(\mathbf{U}_Y)^{-1} \qquad \forall x, y \in [0,1] \end{aligned}$$

4.1.4 Multivariate

The generalisation of Bernstein bases to multivariate polynomials is not immediate.

The power form of a polynomial in d variables is written in terms of $x_1, \ldots, x_d$ like this:

$$p(x_1, \ldots, x_d) = \sum_{0 \le k_1 + \cdots + k_d \le n} a_{(k_1,\ldots,k_d)} x_1^{k_1} \cdots x_d^{k_d}$$

where the coefficients $a_{(k_1,\ldots,k_d)} \in \mathbb{Q}$. By convention, the degree of each term is $k_1 + \cdots + k_d$, and the degree of the polynomial is the maximum of all degrees of its terms.

The multivariate Bernstein form is defined recursively as a polynomial whose main variable is x_d and whose coefficients are multivariate Bernstein-form polynomials in $x_1, \ldots, x_{d-1}$.

Formula (13) can be generalised to more variables. Conversion between the power form and the Bernstein representation is possible regardless of the number of variables (see Geisow [12] and Garloff [11, 19]). Voiculescu's thesis [18] and the book [13] give formulae and algorithms for the computation of the Bernstein form of bivariate and trivariate polynomials.

4.2 Bernstein coefficients and obfuscation of matrices

The correspondence shown in Section 4.1.3 between the matrix of a polynomial's power-form coefficients and that of the Bernstein-form coefficients of the same polynomial is unique, because they represent the same element in the ring of polynomials. We have also shown that the transformation between the matrix representations is well-defined.

$$(\forall \mathbf{M} \in \mathbb{Q}^{a\times b})\ (\exists!\ p \in \mathcal{P}[x,y] \text{ of degree } a \text{ in } x \text{ and } b \text{ in } y) \quad p = \mathbf{X}\,\mathbf{M}\,\mathbf{Y}$$
$$\text{Furthermore} \quad (\exists!\ \mathbf{S} \in \mathbb{Q}^{a\times b}) \quad p = p_{\mathcal{B}} = \mathbf{X}\,\mathbf{U}_a\,\mathbf{S}\,\mathbf{V}_b\,\mathbf{Y}$$

Thus $\mathbf{S}$ is the matrix of coefficients of the Bernstein-form polynomial $p_{\mathcal{B}}$. We will call the operation that transforms $\mathbf{M}$ into $\mathbf{S}$ the *Bernstein Obfuscation* of $\mathbf{M}$, thus $\mathbf{M} \rightsquigarrow \mathbf{S}$. The abstraction function af for this obfuscation is:

$$af(\mathbf{A}) = \mathbf{U}_a\,\mathbf{A}\,\mathbf{V}_b \qquad \text{where } (a,b) = dim(\mathbf{A}) \tag{16}$$

We can also define a conversion function cf as follows:

$$cf(\mathbf{A}) = {\mathbf{U}_a}^{-1}\,\mathbf{A}\,{\mathbf{V}_b}^{-1} \qquad \text{where } (a,b) = dim(\mathbf{A}) \tag{17}$$

These functions are bijections

$$\begin{aligned}
& af(cf(\mathbf{A})) \\
= \quad & \{\text{definition of } cf \text{ with } (a,b) = dim(\mathbf{A})\} \\
& af({\mathbf{U}_a}^{-1}\,\mathbf{A}\,{\mathbf{V}_b}^{-1}) \\
= \quad & \{\text{definition of } af \text{ with } (a,b) = dim({\mathbf{U}_a}^{-1}\,\mathbf{A}\,{\mathbf{V}_b}^{-1})\} \\
& \mathbf{U}_a({\mathbf{U}_a}^{-1}\,\mathbf{A}\,{\mathbf{V}_b}^{-1})\mathbf{V}_b \\
= \quad & \{\text{associativity of matrix multiplication}\} \\
& (\mathbf{U}_a{\mathbf{U}_a}^{-1})\,\mathbf{A}\,({\mathbf{V}_b}^{-1}\mathbf{V}_b) \\
= \quad & \{\text{matrix inverses}\} \\
& \mathbf{A}
\end{aligned}$$

Thus $af \cdot cf = id$. Similarly we can prove that $cf \cdot af = id$.

Since the abstraction and conversion functions for the Bernstein matrix obfuscation are bijective, we can use Equation (3) to prove the correctness of an obfuscated operation. Additionally, we can use Equation (4) to derive obfuscated operations. If op denotes a matrix operation then $\mathsf{op}_{\mathcal{B}}$ will denote the Bernstein obfuscated operation.

4.3 Bernstein example

Using the formulae in Section 4.1.3, it is possible to work out the Bernstein form of a polynomial given in power form. Let us take the power form of the

unit circle $p = x^2 + y^2 - 1$. Preserving the notation from Section 4.1.3, its coefficient matrix is

$$A = \begin{pmatrix} -1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

The corresponding Bernstein-form matrix is

$$M = U_3^{-1} A V_3^{-1} = \begin{pmatrix} -1 & -1 & 0 \\ -1 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

and so the equivalent Bernstein-form polynomial is

$$\begin{aligned} p_B &= X_3 M Y_3 \\ &= \quad \left(-(1-x)^2 - 2x(1-x)\right)(1-y)^2 \\ &\quad + 2\left(-(1-x)^2 - 2x(1-x)\right) y(1-y) \\ &\quad + x^2 y^2 \end{aligned}$$

Ignoring the actual meaning of the polynomials, matrix M can therefore be used as an obfuscation of matrix A.

4.4 Unary operations

The operation scale performs scalar multiplication and so scale λ $\mathbf{M}$ represents $\lambda\mathbf{M}$ for matrix $\mathbf{M}$ and constant λ. Let us use Equation (4) to derive the Bernstein obfuscated scalar multiplication for some obfuscated matrix $\mathbf{S}$ with $(a, b) = dim(\mathbf{S})$ and $f =$ scale λ

$$\begin{aligned} & cf(f(af(\mathbf{S}))) \\ = \quad & \{\text{definition of } af\} \\ & cf(f(\mathbf{U}_a\, \mathbf{S}\, \mathbf{V}_b)) \\ = \quad & \{\text{definition of } f\} \\ & cf(\lambda(\mathbf{U}_a\, \mathbf{S}\, \mathbf{V}_b)) \\ = \quad & \{\text{commutativity of scalar multiplication: } \mu\mathbf{B}\,\mathbf{C} = \mathbf{B}\,\mu\mathbf{C}\} \\ & cf(\mathbf{U}_a\, (\lambda\mathbf{S})\, \mathbf{V}_b) \\ = \quad & \{\text{definition of } cf \text{ with } (a, b) = dim(\mathbf{U}_a\, (\lambda\mathbf{S})\, \mathbf{V}_b)\} \\ & {\mathbf{U}_a}^{-1}(\mathbf{U}_a\, (\lambda\mathbf{S})\, \mathbf{V}_b){\mathbf{V}_b}^{-1} \\ = \quad & \{\text{associativity of matrix multiplication and inverses}\} \\ & \lambda\mathbf{S} \end{aligned}$$

Thus scale$_{\mathcal{B}}$ = scale and so the operation is unchanged by the obfuscation.

Now let us consider transpose. As a shorthand, we will use the usual ^{T} notation. For a matrix $\mathbf{S}$ we propose that if $f = {}^T$ then

$$f_{\mathcal{B}}(\mathbf{S}) = \mathbf{U}_b{}^{-1}\,\mathbf{V}_b{}^T\,\mathbf{S}^T\,\mathbf{U}_a{}^T\,\mathbf{V}_a{}^{-1} \text{ where } (a,b) = dim(\mathbf{S})$$

We prove the correctness of this using Equation (3) for some unobfuscated matrix $\mathbf{M}$:

$$\begin{aligned}
& af(f^{\mathcal{O}}(cf(\mathbf{M}))) \\
= & \quad \{\text{definition of } cf\} \\
& af(f^{\mathcal{O}}(\mathbf{U}_a{}^{-1}\,\mathbf{M}\,\mathbf{V}_b{}^{-1})) \\
= & \quad \{\text{definition of } f^{\mathcal{O}} \text{ with } (a,b) = dim(\mathbf{U}_a{}^{-1}\,\mathbf{M}\,\mathbf{V}_b{}^{-1})\} \\
& af(\mathbf{U}_b{}^{-1}\,\mathbf{V}_b{}^T\,(\mathbf{U}_a{}^{-1}\,\mathbf{M}\,\mathbf{V}_b{}^{-1})^T\,\mathbf{U}_a{}^T\,\mathbf{V}_a{}^{-1}) \\
= & \quad \{(\mathbf{B}\,\mathbf{C})^T = \mathbf{C}^T\,\mathbf{B}^T\} \\
& af(\mathbf{U}_b{}^{-1}\,\mathbf{V}_b{}^T\,\mathbf{V}_b{}^{-1^T}\,\mathbf{M}^T\,\mathbf{U}_a{}^{-1^T}\,\mathbf{U}_a{}^T\,\mathbf{V}_a{}^{-1}) \\
= & \quad \{\mathbf{C}^T\,(\mathbf{C}^{-1})^T = (\mathbf{C}^{-1}\,\mathbf{C})^T = \mathbf{I}^T = \mathbf{I}\} \\
& af(\mathbf{U}_b{}^{-1}\,\mathbf{M}^T\,\mathbf{V}_a{}^{-1}) \\
= & \quad \{\text{definition of } af \text{ with } (b,a) = dim(\mathbf{U}_b{}^{-1}\,\mathbf{M}^T\,\mathbf{V}_a{}^{-1})\} \\
& \mathbf{U}_b(\mathbf{U}_b{}^{-1}\,\mathbf{M}^T\,\mathbf{V}_a{}^{-1})\,\mathbf{V}_a \\
= & \quad \{\text{associativity of matrix multiplication and inverses}\} \\
& \mathbf{M}^T \\
= & \quad \{\text{definition of } f\} \\
& f(\mathbf{M})
\end{aligned}$$

When performing matrix splits, it was hard to write an obfuscation for matrix inversion. However using the Bernstein obfuscation we are able to write such an obfuscation. Suppose that $f = {}^{-1}$ then using Equation 4 for some square obfuscated matrix $\mathbf{S}$:

$$\begin{aligned}
& cf(f(af(\mathbf{S}))) \\
= & \quad \{\text{definition of } af \text{ with } (a,a) = dim(\mathbf{S})\} \\
& cf(f(\mathbf{U}_a\,\mathbf{S}\,\mathbf{V}_a)) \\
= & \quad \{\text{definition of } f\} \\
& cf((\mathbf{U}_a\,\mathbf{S}\,\mathbf{V}_a)^{-1}) \\
= & \quad \{(\mathbf{B}\,\mathbf{C})^{-1} = \mathbf{C}^{-1}\,\mathbf{B}^{-1}\} \\
& cf(\mathbf{V}_a{}^{-1}\,\mathbf{S}^{-1}\,\mathbf{U}_a{}^{-1}) \\
= & \quad \{\text{definition of } cf \text{ with } (a,a) = dim(\mathbf{V}_a{}^{-1}\,\mathbf{S}^{-1}\,\mathbf{U}_a{}^{-1})\} \\
& \mathbf{U}_a{}^{-1}\,\mathbf{V}_a{}^{-1}\,\mathbf{S}^{-1}\,\mathbf{U}_a{}^{-1}\,\mathbf{V}_a{}^{-1}
\end{aligned}$$

$$
\begin{array}{ll}
= & \{\text{obfuscation}\} \\
& f_{\mathcal{B}}(\mathbf{S})
\end{array}
$$

In Section 3.3 we discussed how it is very difficult to define an obfuscation for the determinants of dense, split matrices. We can, however, define determinants under the Bernstein obfuscation. The determinant operation det is different to the operation considered so far as the output from this operation is a number rather than another matrix. Thus to derive a determinant operation for Bernstein obfuscated matrices we consider $\mathsf{det} \cdot af$ (since numbers are not obfuscated there is no conversion function)

$$
\begin{array}{ll}
& \mathsf{det}_{\mathcal{B}}(\mathbf{S}) \\
= & \quad \{\text{deriving equation}\} \\
& \mathsf{det}(af(\mathbf{S})) \\
= & \quad \{\text{definition of } af \text{ with } (a,b) = dim(\mathbf{S})\} \\
& \mathsf{det}(\mathbf{U}_a\, \mathbf{S}\, \mathbf{V}_b) \\
= & \quad \{\mathsf{det}(\mathbf{B}\, \mathbf{C}) = \mathsf{det}(\mathbf{B}) \times \mathsf{det}\mathbf{C}\} \\
& \mathsf{det}(\mathbf{U}_a) \times \mathsf{det}(\mathbf{S}) \times \mathsf{det}(\mathbf{V}_b)
\end{array}
$$

4.5 Binary operations

For a binary matrix operation $\otimes$, we use the non-homogeneous equations defined in Section 2.2. If $cross(af, af)$ is the abstraction function for $Matrix\ \alpha \times Matrix\ \alpha$ then the corresponding conversion function is $cross(cf, cf)$ (this follows from the definition of $cross$). So, for example, to obfuscate an operation $\otimes$ we use Equation (7) and multiply by cf to get $cf \cdot (\otimes) \cdot cross(af, af)$.

Now we will use this equation to derive a definition for multiplication. Suppose that we have two matrices $\mathbf{S}$ and $\mathbf{T}$ with $(a,b) = dim(\mathbf{S})$ and $(b,c) = dim(\mathbf{T})$ (thus the matrices are conformable). Then, writing mult as the prefix matrix multiplication operator in the equation above (but using normal matrix multiplication elsewhere), we can use this equation to derive an obfuscation:

$$
\begin{array}{ll}
& cf(\mathsf{mult}(cross(af, af)\,(\mathbf{S}, \mathbf{T}))) \\
= & \quad \{\text{definition of } cross\} \\
& cf(\mathsf{mult}(af(\mathbf{S}), af(\mathbf{T}))) \\
= & \quad \{\text{definition of } af \text{ with appropriate dimensions}\} \\
& cf(\mathsf{mult}(\mathbf{U}_a\, \mathbf{S}\, \mathbf{V}_b, \mathbf{U}_b\, \mathbf{T}\, \mathbf{V}_c)) \\
= & \quad \{\text{definition of } \mathsf{mult}\} \\
& cf(\mathbf{U}_a\, \mathbf{S}\, \mathbf{V}_b\, \mathbf{U}_b\, \mathbf{T}\, \mathbf{V}_c)
\end{array}
$$

$$
\begin{aligned}
&= \quad \{\text{definition of } cf \text{ with } (a,c) = dim(\mathbf{U}_a\,\mathbf{S}\,\mathbf{V}_b\,\mathbf{U}_b\,\mathbf{T}\,\mathbf{V}_c)\}\\
&\quad {\mathbf{U}_a}^{-1}\,(\mathbf{U}_a\,\mathbf{S}\,\mathbf{V}_b\,\mathbf{U}_b\,\mathbf{T}\,\mathbf{V}_c)\,{\mathbf{V}_c}^{-1}\\
&= \quad \{\text{associativity of matrix multiplication and inverses}\}\\
&\quad \mathbf{S}\,\mathbf{V}_b\,\mathbf{U}_b\,\mathbf{T}
\end{aligned}
$$

Thus, $\mathsf{mult}_{\mathcal{B}}(\mathbf{S},\mathbf{T}) = \mathbf{S}\,\mathbf{V}_b\,\mathbf{U}_b\,\mathbf{T}$ where $(a,b) = dim(A)$.

As with scalar multiplication, addition is unchanged by the Bernstein obfuscation. To show this, we will write $\mathsf{add}_{\mathcal{B}}$ to denote the Bernstein obfuscated definition of addition. So, for two matrices $\mathbf{M}$ and $\mathbf{N}$ with dimensions (a,b), we have that:

$$
\begin{aligned}
&\quad af(\mathsf{add}_{\mathcal{B}}(cross(cf,cf)(\mathbf{M},\mathbf{N})))\\
&= \quad \{\text{definition of } cross\}\\
&\quad af(\mathsf{add}_{\mathcal{B}}(cf(\mathbf{M}),cf(\mathbf{N})))\\
&= \quad \{\text{definition of } cf \text{ with } dim(\mathbf{M}) = (a,b) = dim(\mathbf{N})\}\\
&\quad af(\mathsf{add}_{\mathcal{B}}({\mathbf{U}_a}^{-1}\,\mathbf{M}\,{\mathbf{V}_b}^{-1}, {\mathbf{U}_a}^{-1}\,\mathbf{N}\,{\mathbf{V}_b}^{-1}))\\
&= \quad \{\text{definition of } \mathsf{add}_{\mathcal{B}}\}\\
&\quad af({\mathbf{U}_a}^{-1}\,\mathbf{M}\,{\mathbf{V}_b}^{-1} + {\mathbf{U}_a}^{-1}\,\mathbf{N}\,{\mathbf{V}_b}^{-1})\\
&= \quad \{\text{distributivity of matrix multiplication over addition}\}\\
&\quad af({\mathbf{U}_a}^{-1}\,(\mathbf{M}+\mathbf{N})\,{\mathbf{V}_b}^{-1})\\
&= \quad \{\text{definition of } af \text{ with } (a,b) = dim({\mathbf{U}_a}^{-1}\,(\mathbf{M}+\mathbf{N})\,{\mathbf{V}_b}^{-1})\}\\
&\quad \mathbf{U}_a\,({\mathbf{U}_a}^{-1}\,(\mathbf{M}+\mathbf{N})\,{\mathbf{V}_b}^{-1})\,\mathbf{V}_b\\
&= \quad \{\text{associativity of matrix multiplication and inverses}\}\\
&\quad \mathbf{M}+\mathbf{N}\\
&= \quad \{\text{definition}\}\\
&\quad \mathsf{add}(\mathbf{M},\mathbf{N})
\end{aligned}
$$

4.6 Review of the Bernstein obfuscation

In Figure 3 we summarise the definitions for the matrix operations using the Bernstein obfuscation.

We proved in Section 4.4 that determinants and inverses of matrices can be computed when matrices have been obfuscated using the Bernstein method. The immediate advantage of the Bernstein method over the matrix splitting method is that the former always allows for determinants to be calculated, whereas the latter only allows determinant calculations if the top left corner matrix is invertible. Computing the determinant of a matrix obfuscated with the Bernstein method is also simpler, as it merely requires the computation (and multiplication) of three determinants.

$$
\begin{aligned}
\mathsf{scale}_{\mathcal{B}}\ \lambda\ \mathbf{S} &= \lambda\mathbf{S} \\
\mathsf{add}_{\mathcal{B}}(\mathbf{S},\mathbf{T}) &= \mathbf{S}+\mathbf{T} \\
\mathsf{transpose}_{\mathcal{B}}(\mathbf{S}) &= {\mathbf{U}_b}^{-1}\,{\mathbf{V}_b}^{T}\,\mathbf{S}^{T}\,{\mathbf{U}_a}^{T}\,{\mathbf{V}_a}^{-1} \\
\mathsf{mult}_{\mathcal{B}}(\mathbf{S},\mathbf{T}) &= \mathbf{S}\,\mathbf{V}_b\,\mathbf{U}_b\,\mathbf{T} \\
\mathsf{det}_{\mathcal{B}}(\mathbf{S}) &= \mathsf{det}(\mathbf{U}_a)\times\mathsf{det}(\mathbf{S})\times\mathsf{det}(\mathbf{V}_b) \\
\mathsf{inv}_{\mathcal{B}}(\mathbf{S}) &= {\mathbf{U}_a}^{-1}\,{\mathbf{V}_a}^{-1}\,\mathbf{S}^{-1}\,{\mathbf{U}_a}^{-1}\,{\mathbf{V}_a}^{-1}
\end{aligned}
$$

Note that $(a,b) = dim(\mathbf{S})$

Figure 3: Bernstein obfuscated operations

One drawback of obfuscating matrices with the Bernstein method is, as shown in Section 4.5, that when scaling and adding matrices, the operations themselves are not obfuscated.

This slight disadvantage is clearly outweighed by the method's major advantage, namely that the obfuscated matrices have an entirely different structure from the original entities. Any symmetry or other patterns are shuffled in the transformation, thus making it difficult for a perpetrator to guess their original meaning.

Furthermore, the Bernstein method can be generalised to matrices of any number of dimensions. This is due to the correlation with multivariate Bernstein polynomials and to the easy conversion between the power-form and the Bernstein-form polynomial representations. The conversion is possible regardless of the number of variables, as mentioned in Section 4.1.4.

The important advantages of this method are obtained at the cost of its complexity. For each obfuscated matrix there are several matrices to compute, invert and multiply together. One way in which these computations can be kept low is by way of storing (rather than calculating) a table of the $\binom{n}{k}$ combinations (such as in the form of Pascal's triangle). If the matrices to be obfuscated are of similar sizes, then it should be possible to store, for significant values of a, the matrices U_a and U_a^{-1}.

5 Using matrices to obfuscate a number

Up to now we have discussed creating obfuscation for a matrix data-type but we can use matrices to obfuscate other data-types. As an example, let us see how we could use matrices to obfuscate rational numbers with three rational operations: $+$, $\times$ and $\_^{-1}$. So, for a number n we want a matrix $\mathbf{A}$ such that $n \rightsquigarrow \mathbf{A}$ for some abstraction function af. We need matrix operations plus,

times and recip such that, if $n \rightsquigarrow \mathbf{A}$ and $p \rightsquigarrow \mathbf{C}$ then

$$n + p \rightsquigarrow \mathsf{plus}(\mathbf{A}, \mathbf{C}) \qquad n \times p \rightsquigarrow \mathsf{times}(\mathbf{A}, \mathbf{C}) \qquad n^{-1} \rightsquigarrow \mathsf{recip}(\mathbf{A})$$

5.1 Using determinants

We can define the abstraction function to be the determinant of the matrix. So, for example,

$$af \begin{pmatrix} a & b \\ c & d \end{pmatrix} = \mathsf{det} \begin{pmatrix} a & b \\ c & d \end{pmatrix} = a \times d - b \times c$$

We now need to define a suitable conversion function — remember that we are free to choose any conversion function cf such that $af \cdot cf = id$. We could choose the conversion function to be:

$$cf(n) = \begin{pmatrix} n & 0 \\ 0 & 1 \end{pmatrix}$$

We can immediately see that $af(cf(n)) = n$ (but it is not the case that $cf \cdot af = id$). We can define plus to be

$$\mathsf{plus}(\begin{pmatrix} m & 0 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} n & 0 \\ 0 & 1 \end{pmatrix}) = \begin{pmatrix} m+n & 0 \\ 0 & 1 \end{pmatrix}$$

However we can only use this definition of plus for matrices that are in a very specific form — it is fairly easy to understand what the function is doing and so the number is not very well obfuscated. Instead we would like a function that can be applied to a more general matrices and so we need a different conversion function.

Let us suppose that to obfuscate a number n we pick a matrix $\mathbf{A}$ that has n as an eigenvalue. If $\mathbf{A}$ is a 2×2 matrix then $\mathbf{A}$ has two eigenvalues (which may be the same). So that we can recover n from $\mathbf{A}$ then we could fix the other eigenvalue of A and we will suppose that $\mathbf{A}$ had two eigenvalues namely 1 and n.

Here are some properties of eigenvalues that will be useful. Let suppose that $\mathbf{M}$ is a $n \times n$ matrix and let $\lambda_1, \ldots, \lambda_n$ be the eigenvalues of $\mathbf{M}$.

Sum $\mathsf{trace}(\mathbf{M}) = \sum_i \lambda_i$ (the *trace* of a matrix is the sum of the elements on the leading diagonal)

Product $\mathsf{det}(\mathbf{M}) = \prod_i \lambda_i$

Using these properties, we can define

$$cf(n) = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \text{ where } ad - bc = n \wedge a + d = n + 1$$

and so the eigenvalues of $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ are 1 and n. This conversion function allows some freedom in choosing the elements of the matrix that represents n. Let us suppose that we choose values of a and non-zero b . Using the definition of the conversion function we have that $d = n + 1 - a$ and

$$\begin{aligned} c &= \frac{ad - n}{b} \\ &= \frac{ad - (a + d - 1)}{b} \\ &= \frac{(a-1)(d-1)}{b} \\ &= \frac{(a-1)(n-a)}{b} \end{aligned}$$

So now we can write our conversion function as follows:

$$cf(n) = \begin{pmatrix} a & b \\ \frac{(a-1)(n-a)}{b} & n + 1 - a \end{pmatrix} \text{ where } b \neq 0 \tag{18}$$

We can check that $\mathsf{trace}(cf(n)) = n + 1$ and $\mathsf{det}(cf(n)) = af(cf(n)) = n$. Thus, we can define

$$n \rightsquigarrow \begin{pmatrix} a & b \\ c & d \end{pmatrix} \iff n = af(\begin{pmatrix} a & b \\ c & d \end{pmatrix}) \ \wedge \ a + c = n - 1$$

Note that it is not necessarily the case that $cf \cdot af = id$ and so we cannot use Equation (4) to derive obfuscated functions but we can use Equation (3) to check the correctness of our operations.

5.2 Arithmetic operations

Now let us define our arithmetic operations using our obfuscation. We suppose that $n \rightsquigarrow \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ and $p \rightsquigarrow \begin{pmatrix} e & f \\ g & h \end{pmatrix}$ using our conversion function. We need to find definitions for plus, times and recip.

First we want an operation that adds together n and p. We need to find a matrix that satisfies

$$\begin{pmatrix} j & k \\ l & m \end{pmatrix} = \mathsf{plus}(\begin{pmatrix} a & b \\ c & d \end{pmatrix}, \begin{pmatrix} e & f \\ g & h \end{pmatrix})$$

Under our obfuscation we know that $a + d = n + 1$ and $e + h = p + 1$ and so the resulting matrix must obfuscate $n + p = a + d + e + h - 2$. Thus we need

$j+m = a+d+e+h-1$ so let us take $j = a+e-1$ and $m = d+h$. We are free to choose any non-zero value for k so we take $k = b \times f$. Finally, from the definition of cf we need

$$\begin{aligned} l &= \frac{(j-1)(n+p-j)}{k} \\ &= \frac{((a+e-1)-1)((a+d+e+h-2)-(a+e-1))}{bf} \\ &= \frac{(a+e-2)(d+h-1)}{bf} \end{aligned}$$

Thus,

$$\mathsf{plus}(\begin{pmatrix} a & b \\ c & d \end{pmatrix}, \begin{pmatrix} e & f \\ g & h \end{pmatrix}) = \begin{pmatrix} a+e-1 & bf \\ \frac{(a+e-2)(d+h-1)}{bf} & d+h \end{pmatrix}$$

Note that this operation is commutative (as with $+$) but, as we free to choose any non-zero value of k, we could easily make this operation non-commutative.

Next, we need to a find a matrix that satisfies

$$\begin{pmatrix} j & k \\ l & m \end{pmatrix} = \mathsf{times}(\begin{pmatrix} a & b \\ c & d \end{pmatrix}, \begin{pmatrix} e & f \\ g & h \end{pmatrix})$$

We know that $n = a+d-1$ and $p = e+h-1$ and so we need our resulting matrix to obfuscate $n \times p = (a+d-1)(e+h-1)$. Expanding this expression we obtain:

$$n \times p = (a+d)(e+h) - (e+h) - (a+d) + 1$$

So we take $j = (a+d)(e+h)+1$ and $m = 1-(a+d)-(e+h)$. We can choose any non-zero value for k so (as before) let us take $k = b \times f$. Using the definition of cf, we have that

$$\begin{aligned} l &= \frac{(j-1)(n \times p - j)}{k} \\ &= \frac{(((a+d)(e+h)+1)-1)((a+d-1)(e+h-1)-((a+d)(e+h)+1))}{bf} \\ &= \frac{-(a+d)(e+h)(a+d+e+h)}{bf} \end{aligned}$$

Thus

$$\mathsf{times}(\begin{pmatrix} a & b \\ c & d \end{pmatrix}, \begin{pmatrix} e & f \\ g & h \end{pmatrix}) = \begin{pmatrix} (a+d)(e+h)+1 & bf \\ \frac{-(a+d)(e+h)(a+d+e+h)}{bf} & 1-a-d-e-h \end{pmatrix}$$

Finally, we would like to find a matrix that satisfies

$$\begin{pmatrix} j & k \\ l & m \end{pmatrix} = \mathsf{recip}(\begin{pmatrix} a & b \\ c & d \end{pmatrix})$$

Under our obfuscation, we know that $n = a + d - 1$ and so we need the result of the operation to obfuscate $\frac{1}{n} = \frac{1}{a+d-1}$. We need the trace of the result matrix to be $1 + \frac{1}{n}$ and so:

$$j + m = 1 + \frac{1}{n} = 1 + \frac{1}{a+d-1} = \frac{a+d}{a+d-1}$$

So let us take $j = \frac{d}{a+d-1}$ and $m = \frac{a}{a+d-1}$. Again, we have a free choice for non-zero k so let's take $k = b$. From the definition of cf we need

$$\begin{aligned} l &= \frac{(j-1)(\frac{1}{n} - j)}{k} \\ &= \left(\frac{1}{b}\right)\left(\frac{d}{a+d-1} - 1\right)\left(\frac{1}{a+d-1} - \frac{d}{a+d-1}\right) \\ &= \left(\frac{1}{b}\right)\left(\frac{1-a}{a+d-1}\right)\left(\frac{1-d}{a+d-1}\right) \\ &= \frac{(1-a)(1-d)}{b(a+d-1)^2} \end{aligned}$$

Hence,

$$\mathsf{recip}(\begin{pmatrix} a & b \\ c & d \end{pmatrix}) = \begin{pmatrix} \frac{d}{a+d-1} & b \\ \frac{(a-1)(d-1)}{b(a+d-1)^2} & \frac{a}{a+d-1} \end{pmatrix}$$

Note that this operation is undefined if $a + d - 1 = 0$ *i.e.* if $n = 0$.

5.3 Review of number obfuscation

We summarise the definitions of our obfuscations of the three arithmetic operations in Figure 4. We can easily prove that the operations defined in Figure 4 are correct by using Equations (2) or (7) as appropriate. The proofs of correctness are fairly straightforward (as we used our conversion function to define our matrices); they essentially check that the determinants of the matrices are correct.

Under this obfuscation, several arithmetic operations (on four numbers) are required, hence the complexity of each operation is increased. Thus this obfuscation should not be used where an increase of complexity is a concern.

We could use this obfuscation to obfuscate certain constants in a program or to obfuscate a variable (in a similar way to a variable split that was

$$\mathsf{plus}(\begin{pmatrix} a & b \\ c & d \end{pmatrix}, \begin{pmatrix} e & f \\ g & h \end{pmatrix}) = \begin{pmatrix} a+e-1 & bf \\ \frac{(a+e-2)(d+h-1)}{bf} & d+h \end{pmatrix}$$

$$\mathsf{times}(\begin{pmatrix} a & b \\ c & d \end{pmatrix}, \begin{pmatrix} e & f \\ g & h \end{pmatrix}) = \begin{pmatrix} (a+d)(e+h)+1 & bf \\ \frac{-(a+d)(e+h)(a+d+e+h)}{bf} & 1-a-d-e-h \end{pmatrix}$$

$$\mathsf{recip}(\begin{pmatrix} a & b \\ c & d \end{pmatrix}) = \begin{pmatrix} \frac{d}{a+d-1} & b \\ \frac{(a-1)(d-1)}{b(a+d-1)^2} & \frac{a}{a+d-1} \end{pmatrix}$$

Figure 4: Obfuscations of arithmetic operations

discussed in Collberg *et al.* [4]). If we choose to this matrix obfuscation to obfuscate a rational variable then we risk adversely affecting the efficiency of a program. If the variable that we choose is used extensively then the obfuscation will add many arithmetic operations whenever the variable is used.

5.4 An example

Let us now consider some simple examples. Suppose

$$3 \rightsquigarrow \begin{pmatrix} 5 & 8 \\ -1 & -1 \end{pmatrix} \quad \text{and} \quad 7 \rightsquigarrow \begin{pmatrix} 4 & 3 \\ 3 & 4 \end{pmatrix}$$

$$\begin{aligned} \mathsf{plus}(\begin{pmatrix} 5 & 8 \\ -1 & -1 \end{pmatrix}, \begin{pmatrix} 4 & 3 \\ 3 & 4 \end{pmatrix}) &= \begin{pmatrix} 5+4-1 & 8\times 3 \\ \frac{(5+4-2)(-1+4-1)}{8\times 3} & 4-1 \end{pmatrix} \\ &= \begin{pmatrix} 8 & 24 \\ \frac{7}{12} & 3 \end{pmatrix} \end{aligned}$$

We can check that $\mathsf{det}(\begin{pmatrix} 8 & 24 \\ \frac{7}{12} & 3 \end{pmatrix}) = 10$ as required. Now,

$$\begin{aligned} \mathsf{times}(\begin{pmatrix} 5 & 8 \\ -1 & -1 \end{pmatrix}, \begin{pmatrix} 4 & 3 \\ 3 & 4 \end{pmatrix}) &= \begin{pmatrix} (5-1)(4+4)+1 & 8\times 3 \\ \frac{-(5-1)(4+4)(5-1+4+4)}{8\times 3} & 1-4-8 \end{pmatrix} \\ &= \begin{pmatrix} 33 & 24 \\ -16 & -11 \end{pmatrix} \end{aligned}$$

and $\mathsf{det}(\begin{pmatrix} 33 & 24 \\ -16 & -11 \end{pmatrix}) = -363 + 384 = 21$ as required. Finally,

$$\mathsf{recip}(\begin{pmatrix} 5 & 8 \\ -1 & -1 \end{pmatrix}) = \begin{pmatrix} -\frac{1}{3} & 8 \\ \frac{4\times -2}{8\times 9} & \frac{5}{3} \end{pmatrix} = \begin{pmatrix} -\frac{1}{3} & 8 \\ -\frac{1}{9} & \frac{5}{3} \end{pmatrix}$$

with $\mathsf{det}(\begin{pmatrix} -\frac{1}{3} & 8 \\ -\frac{1}{9} & \frac{5}{3} \end{pmatrix}) = -\frac{5}{9} + \frac{8}{9} = \frac{1}{3}$.

6 Extensions of the obfuscation techniques

In the previous sections we have given various examples to show how matrix properties can be used to help create obfuscations. In this section we will briefly discuss some extensions of our obfuscation methods.

6.1 Other Splits

In Section 3.1 we discussed how we can define a matrix split and in Section 3.2 we gave an example of a matrix split in which we split up a square matrix into four pieces.

Let us now consider splitting non-square matrices. We can define a (k, l) split which will produce four split components so that the first split component is a matrix of size (k, l). The choice function for this split is

$$ch(i, j) = 2\, sgn\,(i \text{ div } k) + sgn\,(j \text{ div } l)$$

and the family of functions is

$$\mathcal{F} = \{f_p = (\lambda\,(i, j)\,.\,(i - k\ (p \text{ div } 2),\ j - l\ (p \bmod 2))) \mid p \in [0..3]\}$$

The representation

$$mss \rightsquigarrow \langle as, bs, cd, ds\rangle_{s_{(k,l)}}$$

with $\mathsf{dim}\ mss = (r, c)$ satisfies the invariant

$$\begin{array}{c} \mathsf{dim}\ as = (k, l) \wedge \mathsf{dim}\ bs = (k, c - l) \wedge \\ \mathsf{dim}\ cs = (r - k, l) \wedge \mathsf{dim}\ ds = (r - k, c - l) \end{array} \tag{19}$$

for some $k : (0..r)$ and $l : (0..c)$.

We will now consider a more general split than the $(k \times k)$ split. Suppose that we have a matrix $\mathbf{M}^{k\times k}$ which we want to split into n^2 blocks. We want to ensure that the blocks down the main diagonal are square. We will call this the *n-square matrix split*, denoted by *nsq*. For this, we will need a set of numbers $S_0, S_1, \ldots, S_m$ such that

$$0 = S_0 < S_1 < S_2 < \ldots < S_{n-1} < S_n = k - 1$$

We require strict inequality so that we have exactly n^2 blocks with both dimensions of each block at least 1.

The n-square matrix split is defined as follows: $nsq = (ch, \mathcal{F})$ such that

$$\begin{array}{rcl} ch & :: & [0..k) \times [0..k) \to [0..n) \\ ch(i,j) & = & p\,n + q \text{ where } S_p \le i < S_{p+1} \wedge S_q \le j < S_{q+1} \end{array}$$

and if $f_r \in \mathcal{F}$ then

$$\begin{array}{rcl} f_r & :: & [0..k) \times [0..k) \to [0..S_p - S_{p-1}) \times [0..S_q - S_{q-1}) \\ f_r(i,j) & = & (i - S_p, j - S_q) \text{ where } r = ch(i,j) = pn + q \end{array}$$

An alternative form for the choice function is

$$ch(i,j) = \sum_{t=1}^{n} \Big(n \times sgn(i \text{ div } S_t) + (j \text{ div } S_t) \Big)$$

Note that if $ch(i,j) = pn + q$ then $ch(j,i) = qn + p$.

The matrices $\mathbf{M}$ and $\mathbf{M_r}$ are related by the formula

$$\mathbf{M_r}(f_r(i,j)) = \mathbf{M}(i,j) \quad \text{where } r = ch(i,j)$$

6.2 Further Arithmetic Operations

In Section 5.1 we saw that we can obfuscate a number by representing the number as the determinant of a matrix and in Section 5.2 we defined obfuscations for three arithmetic operations. Using our matrix functions we can define obfuscations for other arithmetic operations, for example,

$$\mathsf{sub}(\mathbf{A}, \mathbf{C}) = \mathsf{plus}(\mathbf{A}, \mathsf{times}(cf(-1), \mathbf{C})) \quad \text{and} \quad \mathsf{div}(\mathbf{A}, \mathbf{C}) = \mathsf{times}(\mathbf{A}, \mathsf{recip}(\mathbf{C}))$$

There are many other possible definitions for the three arithmetic operations that are correct for our conversion and abstraction functions. This is due to the fact that we have flexibility in defining some of the matrix elements — we just have to ensure that the trace equals $n + 1$ and the determinant is n. In fact, if we wished, we could have a number of different definitions for each operation. For example, if

$$n \rightsquigarrow \mathbf{A} \;\wedge\; p \rightsquigarrow \mathbf{C} \;\wedge\; r \rightsquigarrow \mathbf{E}$$

then

$$n + p + r \rightsquigarrow \mathsf{plus}_1(\mathbf{A}, \mathsf{plus}_2(\mathbf{C}, \mathbf{E}))$$

where plus_1 and plus_2 are different obfuscations of $+$.

This obfuscation used 2×2 matrices and, of course, we could represent a number by a much larger matrix. However this will lead to a major impact of the efficiency of the operations as we would be require to calculate determinants of these larger matrices.

6.3 Combining obfuscations

In Section 3.3 we discussed that we have problems defining operations for inversion and determinants of split matrices. Also, in Sections 4.4 and 4.5 we saw that scalar multiplication and addition were not obfuscated using the Bernstein method. However, since we followed the data refinement approach to obfuscation (as given in Drape's thesis [7]), we have abstraction functions and conversion functions for each of our obfuscations and we can easily compose our obfuscations. Thus to create, for example, a determinant function for split matrices we can obfuscate using Bernstein, find the determinant and re-obfuscate back to split matrices. So

$$\mathsf{det}_{sp} = cf_{sp} \cdot af_{\mathcal{B}} \cdot \mathsf{det}_{\mathcal{B}} \cdot cf_{\mathcal{B}} \cdot af_{sp}$$

where sp and $\mathcal{B}$ denote the split and Bernstein obfuscations respectively. In a similar way we can combine our number obfuscation (from Section 5.1) with our other matrix obfuscations so that we can build a more complicated obfuscation for numbers.

7 Conclusions

An obfuscation should make a program (or a method within a program) harder to understand. When obfuscating matrices one ideally aims to change the structure or the elements within the matrix. Our splitting obfuscation (Section 3.2) changes the size and shape of the matrix (but not the individual elements), whereas the Bernstein obfuscation (Section 4.2) does not alter the size and shape of the matrix, but changes its elements (thus changing their pattern). An advantage of considering obfuscations as data refinements is that obfuscations can then be written as functions, which gives us the ability to compose different obfuscations together. Thus, as we discussed in Section 6.3, we can create an obfuscation that changes both the structure and the elements. Obviously, if efficiency is a concern that we have to restrict how complicated we make our obfuscations — there is usually a trade-off between how complicated the obfuscations are and efficiency.

As we mentioned in the Introduction, when creating obfuscations we should make reference to an attack model. Following the assertion attack model of Drape [7], when defining the matrix data-type in Figure 2 we stated a number of assertions that we expect our operations to satisfy. In most cases (except for the Bernstein obfuscations of add and scale), the proofs of the assertions (which we omit) are more complicated. One way of at least checking the assertions of the obfuscated operations hold is to generate a large set of random examples. In the case of functional languages (e.g. Haskell), such a checking exists in the form of QuickCheck [3], which is based precisely on sets of otherwise difficult to prove assertions.

Unsurprisingly, it is often the case that effective obfuscations affect the efficiency of the program being obfuscated. One way to alleviate the slow-down of a program is, as discussed in Section 4.6, is to pre-compute and store some of the data used frequently. The trade-off between space and time complexity will depend on the individual applications for which the obfuscation method is used.

Evidently, these operations rely on exact arithmetic being available for rational numbers. This is not a major inconvenience, though, since multi-precision rational operations nowadays are either supported by programming languages (e.g. Java) or through integrated packages (see, for example, MP [2] or LiDIA [15]).

In Section 6 we saw that our approach to obfuscation gave us flexibility to extend our obfuscations. To create other matrix splits, matrix forms such as triangular matrices (so that at least one of the components is a zero matrix) or Jordan matrices could be used to create splits. Instead of using the Bernstein basis, other change of basis formulae could be used to produce matrix obfuscations.

References

[1] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, pages 1–18. Springer-Verlag, 2001.

[2] Richard P. Brent. A fortran multiple–precision arithmetic package. *ACM Transactions on Mathematical Software*, 4(1):57–70, 1978.

[3] Koen Claessen and John Hughes. QuickCheck: a lightweight tool for random testing of Haskell programs. In *ACM SIGPLAN Notices*, 2000.

[4] Christian Collberg, Clark Thomborson, and Douglas Low. A taxonomy of obfuscating transformations. Technical Report 148, Department of Computer Science, University of Auckland, July 1997.

[5] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms (Second Edition)*. The MIT Press, 2001.

[6] Willem-Paul de Roever and Kai Engelhardt. *Data Refinement: Model-Oriented Proof Methods and their Comparison*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1998.

[7] Stephen Drape. *Obfuscation of Abstract Data-Types*. DPhil thesis, Oxford University Computing Laboratory, 2004.

[8] Stephen Drape. Generalising the array split obfuscation. *Information Sciences*, 177(1):202–219, January 2007.

[9] Stephen Drape, Clark Thomborson, and Anirban Majumdar. Specifying imperative data obfuscations. In *Proceedings of the 10th Information Security Conference (ISC '07)*, volume 4779 of *Lecture Notes in Computer Science*, pages 299–314. Springer, October 2007.

[10] R. T. Farouki and Rajan V. T. Algorithms for polynomials in Bernstein form. *Computer Aided Geometric Design*, 5:1–26, 1988.

[11] J. Garloff. Convergent bounds for the range of multivariate polynomials. *Interval Mathematics 1985, Lecture Notes in Computer Science*, 212:37–56, 1985.

[12] Adrian Geisow. *Surface Interrogations*. PhD thesis, University of East Anglia, 1983.

[13] Abel Gomes, Irina Voiculescu, Joaquim Jorge, Bryan Wyvill, and Callum Galbraith. *Implicit Curves and Surfaces: Mathematics, Data Structures and Algori thms*. Springer Verlag, to appear 2009.

[14] Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Cambridge University Press, 1985.

[15] LiDIA Group, Darmstadt University of Technology. `www.cdc.informatik.tu-darmstadt.de/TI/LiDIA`.

[16] George G. Lorentz. *Bernstein Polynomials*. Chelsea Publishing Company, New York, 1986.

[17] Anirban Majumdar, Stephen J. Drape, and Clark D. Thomborson. Slicing obfuscations: design, correctness, and evaluation. In *DRM '07: Proceedings of the 2007 ACM workshop on Digital Rights Management*, pages 70–81, New York, NY, USA, 2007. ACM.

[18] Irina Voiculescu. *Implicit function algebra in set-theoretic geometric modelling*. PhD thesis, University of Bath, 2001.

[19] M. Zettler and J. Garloff. Robustness analysis of polynomials with polynomial parameter dependency using Bernstein expansion. *IEEE Transactions on Automatic Control*, 43(3):425–431, 1998.