Creating transformations for matrix obfuscation

Stephen Drape and Irina Voiculescu

Oxford University Computing Laboratory,
Wolfosn Building, Parks Road,
Oxford, UK, OX1 3QD
{sjd,irina}@comlab.ox.ac.uk

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*. Obfuscation is one of a number of techniques that can be employed to protect sensitive areas of code. 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 is an improvement on previous methods for matrix obfuscation because we will provide a range of techniques which can be used in concert.
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. We then evaluate our methods by considering how our obfuscations perform under certain relevant program analysis-based attacks.

Key words: Obfuscation, Matrix Operations, Information Hiding, Program Transformations

1 Introduction

An *obfuscation* is a behaviour-preserving program transformation whose aim is to make an input program "harder to understand". The landmark paper by Collberg *et al.* [6] gives a range of transformations which can be used as obfuscating transformations. The purpose of such transformations is to decrease the opportunities for a user to reverse engineer a commercially supplied program [1, 6]. 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 they 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 [10].

The view of obfuscation from Collberg *et al.* [6] concentrates on concrete data structures such as variables and arrays. However, the thesis of Drape [8] 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 [10] and for creating obfuscations which impede the effectiveness of program slicing [18].

The focus of this paper consists of a data-type for finite matrices having four operations: scalar multiplication, addition, transposition and multiplication, specified mathematically. We use data refinement [7] to provide a way of proving the correctness of our obfuscated operations. Thus we are guaranteed that our obfuscations are behaviour-preserving. We will review a previous matrix obfuscation, called matrix splitting [8], and we will discuss problems with this obfuscation. We will then describe a new technique for matrix obfuscation and we also show how matrices can be used to obfuscate another data-type. 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.

The notion of "harder to understand" can be a little vague as it is not easy to measure — the creation of a suitable measure of the quality of an obfuscation is an open problem. When creating obfuscations we will make reference to an attack model including what analysis techniques we expect to perform. In the work of Majumdar *et al.* [18], 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 [8] in which, when defining data-types, we also specify a set of assertions which are true for the operations of that data-type. According to [8], the comparison between the assertions proofs for unobfuscated and obfuscated operations gives a measure of the effectiveness of the obfuscation. In this paper, we do not show such proofs but example proofs for various data-types can be found in [8].

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 [8], data obfuscation was considered as a data refinement [7]. 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

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

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 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.

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 $dim\,(\mathbf{M})$ returns the dimensions of $\mathbf{M}$.

In Figure 1 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. 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 1 we have a possible (but not complete) 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.

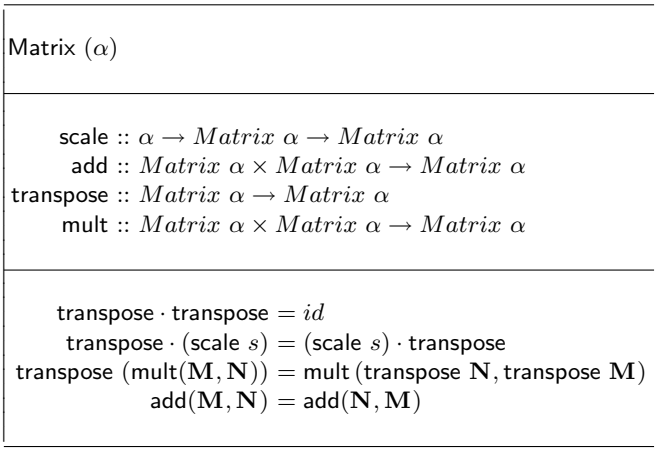

Fig. 1. A data-type for Matrices

Note that for addition the 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 in the usual way. We assume that basic arithmetic operations take constant time and so the computational complexities of add($\mathbf{M}, \mathbf{N}$), scale s $\mathbf{M}$ and transpose $\mathbf{M}$ are all $r \times c$ and the complexity of mult($\mathbf{M}, \mathbf{P}$) is $r \times c \times d$ where $(r, c) = dim(\mathbf{M}) = dim(\mathbf{N})$ and $(c, d) = dim(\mathbf{P})$.

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. One such case is when, in expressing a rigid body transformation by way of a matrix, the matrix has a particular structure. For example, the two-dimensional translation of an object by displacements (d_x, d_y) is usually written in the form

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

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

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.* [6] discuss an obfuscation called an *array*

split. This obfuscation was generalised in Drape [9] and so we can apply the concept of splitting to other data-types such as 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 $i : [0..n)$.

For this characterisation, $\mathbf{M}$ 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:

$$\mathbf{M}_t(f_t(i,j)) = \mathbf{M}(i,j) \text{ where } t = ch(i,j) \tag{5}$$

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 (5) can be considered to be the definition of a conversion function and so for a matrix split

$$cf(\mathbf{M}(i,j)) = \mathbf{M}_t(f_t(i,j)) \qquad \text{where } t = ch(i,j) \tag{6}$$

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

$$af(\mathbf{M}_t(i,j)) = \mathbf{M}({f_t}^{-1}(i,j)) \tag{7}$$

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 discussed by Horn and Johnson [15] in which a matrix can be represented by a sequence of smaller submatrices.

3.2 Splitting in squares

We now describe a simple matrix split that splits a square matrix into four matrices — two of which are square. Suppose that we have a square matrix $\mathbf{M}^{r\times r}$ and choose a positive integer k such that $k < r$. The choice function $ch(i,j)$ is defined as

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

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

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

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

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

where the subscript s_k denotes the $(k\times k)$-square split. Using this split, how can we define our matrix operations given in Figure 1?

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

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

The proofs for these definitions can be found in [8]. Also in [8] it was shown 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:

$$\begin{pmatrix} \mathbf{M}_0 & \mathbf{M}_1 \\ \mathbf{M}_2 & \mathbf{M}_3 \end{pmatrix}^T = \begin{pmatrix} {\mathbf{M}_0}^T & {\mathbf{M}_2}^T \\ {\mathbf{M}_1}^T & {\mathbf{M}_3}^T \end{pmatrix}$$

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

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

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

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:

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

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? If we use the definition of split matrices to add up the number of multiplications required by each component then we find that the total number of multiplications is still n^3.

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. 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 1) we considered four matrix operations. Could we define obfuscations for other matrix operations? Computing inverses and determinants for dense matrices which have been split can prove to be difficult. We can, however, define obfuscations of these operations using results for partitioned matrices — we omit the details here.

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 is 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 two-dimensional 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 [17]). One is the power basis $\begin{pmatrix}1 & x & \dots & x^n\end{pmatrix}$ and another is the Bernstein basis $\begin{pmatrix}B_0^n(x) & B_1^n(x) & \dots & B_n^n(x)\end{pmatrix}$ where $B_k^n(x) = \binom{n}{k} x^k (1-x)^{n-k},\ \forall x \in [0, 1],\ k = 0, \dots, n$ are the corresponding Bernstein Polynomials [3].

4.1 Power-form and Bernstein-form polynomials

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

$$p(x, y) = \sum_{i=0}^{m} \sum_{j=0}^{n} a_{ij} x^i y^j, \tag{8}$$

where $a_{ij} \in \mathbb{Q}$. For given $m, n \in \mathbb{N}$ there are $m+1$ univariate degree-m Bernstein polynomials in x, and $n+1$ univariate degree-n Bernstein polynomials in y. Any bivariate power-form polynomial can be represented on the interval $[0,1]$ using its equivalent Bernstein form as

$$p_{\mathcal{B}}(x,y) = \sum_{i=0}^{m}\sum_{j=0}^{n} c_{ij} B_i^m(x) B_j^n(y) \tag{9}$$

where c_{ij} are the Bernstein coefficients corresponding to the degree-n base. The two representations $p(x,y)$ and $p_{\mathcal{B}}(x,y)$ are equivalent and it is possible to convert one into the other. In the case of bivariate polynomials this conversion requires some care and is based on the univariate case shown by Farouki and Rajan [11].

The polynomials in Equations (8) and (9) can also be written as matrix multiplications:

$$p(x,y) = \begin{pmatrix} 1 & x & \dots & x^m \end{pmatrix} \begin{pmatrix} a_{00} & \dots & a_{0n} \\ \vdots & \ddots & \vdots \\ a_{m0} & \dots & a_{mn} \end{pmatrix} \begin{pmatrix} 1 \\ y \\ \vdots \\ y^n \end{pmatrix} = \mathbf{X}\mathbf{A}\mathbf{Y}$$

$$p_{\mathcal{B}}(x,y) = \begin{pmatrix} B_0^m(x) & B_1^m(x) & \dots & B_m^m(x) \end{pmatrix} \begin{pmatrix} c_{00} & \dots & c_{0n} \\ \vdots & \ddots & \vdots \\ c_{m0} & \dots & c_{mn} \end{pmatrix} \begin{pmatrix} B_0^n(y) \\ B_1^n(y) \\ \vdots \\ B_n^n(y) \end{pmatrix} = \mathbf{B}_m^X \, \mathbf{C} \mathbf{B}_n^Y$$

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

$$\begin{aligned}
\mathbf{B}_m^X &= \begin{pmatrix} B_0^m(x) & B_1^m(x) & \dots & B_m^m(x) \end{pmatrix} \\
&= \begin{pmatrix} \binom{m}{0}(1-x)^m & \dots & \binom{m}{m}x^m \end{pmatrix} \\
&= \begin{pmatrix} \binom{m}{0} & \binom{m}{1}(-x) + \dots + \binom{m}{m}(-x)^m & \dots & \binom{m}{m}x^m \end{pmatrix} \\
&= \underbrace{\begin{pmatrix} 1 & x & \dots & x^m \end{pmatrix}}_{X} \underbrace{\begin{pmatrix} 1 & & & \mathrm{O} \\ \binom{m}{0}\binom{m}{1}(-1)^1 & \binom{m}{1}\binom{m-1}{0}(-1)^0 & & \\ \vdots & & \ddots & \\ \binom{m}{0}\binom{m}{m}(-1)^m & \binom{m}{1}\binom{m-1}{m-1}(-1)^{m-1} & \dots & \binom{m}{m}\binom{m-m}{0}(-1)^0 \end{pmatrix}}_{U_m} \\
&= \mathbf{X}\,\mathbf{U}_m, \qquad \forall x \in [0,1].
\end{aligned}$$

So $\mathbf{B}_m^X = \mathbf{X}\,\mathbf{U}_m$

Similarly $\mathbf{B}_n^Y = \mathbf{V}_n\,\mathbf{Y}$

and $p_{\mathcal{B}}(x,y) = \mathbf{B}_m^X\,\mathbf{C}\,\mathbf{B}_n^Y = \mathbf{X}\,\mathbf{U}_m\,\mathbf{C}\,\mathbf{V}_n\,\mathbf{Y}$

Now we can compute the Bernstein coefficients matrix $\mathbf{C}$:

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

4.2 Bernstein coefficients and obfuscation of matrices

The correspondence shown in Section 4.1 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.

$(\forall \mathbf{A} \in \mathbb{Q}^{\alpha\times\beta})\ (\exists!\ p \in \mathcal{P}[x,y]$ of degree $\alpha-1$ in x and $\beta-1$ in $y)\quad p = \mathbf{X}\,\mathbf{A}\,\mathbf{Y}$
Furthermore $\quad(\exists!\ \mathbf{C} \in \mathbb{Q}^{\alpha\times\beta})\quad p = p_{\mathcal{B}} = \mathbf{X}\,\mathbf{U}_{\alpha-1}\,\mathbf{C}\,\mathbf{V}_{\beta-1}\,\mathbf{Y}$

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

$$af(\mathbf{S}) = \mathbf{U}_a\,\mathbf{S}\,\mathbf{V}_b \qquad \text{where } (a+1, b+1) = dim(\mathbf{S}) \tag{10}$$

We can also define a conversion function cf as follows:

$$cf(\mathbf{S}) = {\mathbf{U}_a}^{-1}\,\mathbf{S}\,{\mathbf{V}_b}^{-1} \qquad \text{where } (a+1, b+1) = dim(\mathbf{S}) \tag{11}$$

It is straightforward to show that these functions are bijections.

4.3 Bernstein example

Using the formulae in Section 4.1, it is possible to work out the Bernstein form of a polynomial given in power form. Let us take the two-dimensional translation matrix defined in Section 2.2

$$\mathbf{A} = \begin{pmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{pmatrix}$$

The corresponding Bernstein-form matrix is

$$\mathbf{C} = {\mathbf{U}_2}^{-1}\mathbf{A}{\mathbf{V}_2}^{-1} = \begin{pmatrix} 1 & 1 & 1+d_x \\ 1 & \frac{5}{4} & \frac{3}{2}+d_x+\frac{1}{2}d_y \\ 1 & \frac{3}{2} & 3+d_x+d_y \end{pmatrix}$$

It is easy to verify that the polynomials corresponding to $\mathbf{A}$ and $\mathbf{C}$ are the same, that is $\mathbf{X}\,\mathbf{A}\,\mathbf{Y} = \mathbf{X}\,\mathbf{U}_2\,\mathbf{C}\,\mathbf{V}_2\,\mathbf{Y}$. We can see that this obfuscation conceals the fact that $\mathbf{C}$ represents a translation.

4.4 Operations for the Bernstein Obfuscation

Now that we have an obfuscation for matrices we can define obfuscations for our matrix operations (given in Figure 1). If op denotes a matrix operation then $\mathsf{op}_{\mathcal{B}}$ will denote the Bernstein obfuscated operation. In the following definitions, for matrix $\mathbf{S}$ we assume $dim(\mathbf{S}) = (a+1, b+1)$.

We can use Equation (4) to derive the Bernstein obfuscated scalar multiplication (we omit the details). We find that $\mathsf{scale}_{\mathcal{B}}\ \mathbf{S} = \mathsf{scale}\ \mathbf{S}$ and so the operation is unchanged by the obfuscation.

In Appendix A.2 we prove that for a matrix $\mathbf{S}$

$$\mathsf{transpose}_{\mathcal{B}}(\mathbf{S}) = {\mathbf{U}_b}^{-1}\,{\mathbf{V}_b}^{T}\,\mathbf{S}^{T}\,{\mathbf{U}_a}^{T}\,{\mathbf{V}_a}^{-1}$$

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. For some square obfuscated matrix $\mathbf{S}$:

$$\mathsf{inverse}_{\mathcal{B}}(\mathbf{S}) = {\mathbf{U}_a}^{-1}\,{\mathbf{V}_a}^{-1}\,\mathbf{S}^{-1}\,{\mathbf{U}_a}^{-1}\,{\mathbf{V}_a}^{-1}$$

We omit the details of the proof.

We also found it difficult, for split matrices, to define a determinant operation. However, in Appendix A.2, we derive the following obfuscation of the determinant operation under the Bernstein operation:

$$\mathsf{det}_{\mathcal{B}}(\mathbf{S}) = \mathsf{det}(\mathbf{U}_a) \times \mathsf{det}(\mathbf{S}) \times \mathsf{det}(\mathbf{V}_b)$$

As with scalar multiplication, matrix addition is unchanged under the Bernstein obfuscation:

$$\mathsf{add}_{\mathcal{B}}(\mathbf{S}, \mathbf{T}) = \mathbf{S} + \mathbf{T}$$

We omit the details of the proof.

Finally, we can derive an obfuscation for matrix multiplication

$$\mathsf{mult}_{\mathcal{B}}(\mathbf{S}, \mathbf{T}) = \mathbf{S}\,\mathbf{V}_b\,\mathbf{U}_b\,\mathbf{T}$$

This derivation can be found in Appendix A.3.

4.5 Review of the Bernstein obfuscation

In Section 4.4 we stated that determinants and inverses of matrices can be computed easily when matrices have been obfuscated using the Bernstein method — this is an immediate advantage of this method over the matrix splitting method.

One drawback of obfuscating matrices with the Bernstein method is, as shown in Section 4.4, 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 an attacker to

guess their original meaning. This obfuscation technique would work with any change of basis transformation, which would help to strengthen this technique by allowing us to create a set of different obfuscations.

We have explained how the bivariate case works because most programs use two-dimensional matrices. However, conversion between the power form and the Bernstein representation is possible regardless of the number of variables (see Geisow [13] and Garloff [12, 19]). Berchtold's thesis [2] and the book [14] give formulae and algorithms for the computation of the Bernstein form of bivariate and trivariate polynomials. Thus we could adapt the method to more (or, indeed, fewer) variables for use in programs with matrices of higher dimensions (or with arrays).

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 $\mathbf{S}$ such that $n \rightsquigarrow \mathbf{S}$ for some abstraction function af. We need matrix operations plus, times and recip such that, if $n \rightsquigarrow \mathbf{S}$ and $p \rightsquigarrow \mathbf{T}$ then

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

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 $cf \cdot af = id$ does not hold). 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 it is not a good obfuscation. (Referring back to the assertion definition of obfuscation, any assertions about plus, such as commutativity, can be proved easily for this matrix version.) Instead we would like a function that can be applied to more general matrices and so we need a different conversion function.

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

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

This conversion function allows some freedom in choosing the elements of the matrix that represents n. Suppose that we choose values of a and non-zero b. We propose the following conversion function:

$$cf(n) = \begin{pmatrix} a & b \\ \frac{(a-1)(n-a)}{b} & n+1-a \end{pmatrix} \text{ where } b \neq 0 \tag{12}$$

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

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

5.2 Arithmetic operations

Now let us define 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 the conversion function. We need to find definitions for plus, times and recip.

First we want an operation that adds together n and p. We propose

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

We can check that the trace of the resulting matrix is

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

as required. We can also check that the determinant is $n + p$.

For a multiplication operation, we propose

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

Finally, for a reciprocal operation, we propose:

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

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

More details of the development of the definitions for these operations can be found in Appendix A.4. Note that we are free to create many different definitions for each of these operations since we have some degree of flexibility in our conversion function.

5.3 Review of number obfuscation

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 discussed in Collberg *et al.* [6]). If we choose this matrix transformation 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.

6 Evaluation of techniques

As stated in the Introduction, when creating obfuscations we should make reference to an attack model and any analyses we expect to run. For a human reader, our obfuscated operations are harder to understand because the obfuscated operations are not the expected matrix operations. To understand the Bernstein obfuscated operations, an attacker needs to have familiarity with change of basis transformations and, more importantly, needs to realise the connection between matrices and polynomial bases.

Following the assertion attack model of Drape [8], when defining the matrix data-type (as seen in Figure 1) 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 — example assertion proofs can be found in Drape [8]. One way of at least checking whether 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 [5], which is based precisely on sets of otherwise difficult to prove assertions.

Majumdar *et al.* [18] describe another attack model for obfuscation in which obfuscations were created with the aim of protecting against an adversary armed with a static program slicer. Majumdar *et al.* found that adding arrays to code

fragments reduces the effectiveness of program slicing. Thus a particularly effective obfuscation against a slicing attack should be the determinant obfuscation described in Section 5.1 as it replaces numbers by array-like objects.

The data refinement approach means that we create obfuscations for a set of defined operations. If we want to obfuscate other operations or data-types then we may have to use different obfuscations. For instance, the determinant operation (discussed in Section 5.1) would not be suitable to obfuscate the individual elements of a matrix as the complexity of the matrix operations would drastically increase. Future work would be to see whether these obfuscations would be suitable if we allowed an update operation so that we could change individual elements of a matrix (rather than by using algebraic matrix operations).

One advantage of specifying obfuscations as data refinements is that we can easily produce equations which help us to prove the correctness of our obfuscations. In the Appendix we give some examples of correctness proofs using equations given in Section 2.1. Another advantage of using data refinement is we can compose our obfuscation functions to help us create more complicated obfuscations. For example, we can create an inverse operation for split matrices by using the Bernstein obfuscation:

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

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 and we could also combine different change of basis transformations.

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, we can create an obfuscation that changes both the structure and the elements. Obviously, if efficiency is a concern then we have to restrict how complicated we make our obfuscations — there is usually a trade-off between how complicated the obfuscations are and efficiency. One way to alleviate the slow-down of a program is, as discussed in Section 4.5, 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 ra-

tional operations nowadays are either supported by programming languages (e.g. Java) or through integrated packages (see, for example, MP [4] or LiDIA [16]).

In this paper we have used a variety of methods, both from number theory and from previous work in obfuscation. Our methods bring improvements on previous methods for matrix (and array) obfuscation because, as discussed in Section 6, we have provided a range of techniques that can be used to create transformations which provide greater obscurity. We do not give concrete programming details of our matrix operations, since we considered obfuscation at an appropriate level of abstraction, such that implementing these operations (and their obfuscations) is a straightforward exercise.

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. Jakob Berchtold. *The Bernstein basis in set-theoretic geometric modelling.* PhD thesis, University of Bath, 2000.
3. Serge Bernstein. Démonstration du théorème de Weierstrass fondée sur le calcul des probabilités. *Comm. Kharkov Math. Soc.*, 13(1–2):49–194, 1912.
4. Richard P. Brent. A FORTRAN multiple–precision arithmetic package. *ACM Transactions on Mathematical Software*, 4(1):57–70, 1978.
5. Koen Claessen and John Hughes. QuickCheck: a lightweight tool for random testing of Haskell programs. In *ACM SIGPLAN Notices*, 2000.
6. Christian Collberg, Clark Thomborson, and Douglas Low. A taxonomy of obfuscating transformations. Technical Report 148, Department of Computer Science, University of Auckland, July 1997.
7. 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.
8. Stephen Drape. *Obfuscation of Abstract Data-Types.* DPhil thesis, Oxford University Computing Laboratory, 2004.
9. Stephen Drape. Generalising the array split obfuscation. *Information Sciences*, 177(1):202–219, January 2007.
10. 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.
11. R. T. Farouki and V. T. Rajan. Algorithms for polynomials in Bernstein form. *Computer Aided Geometric Design*, 5:1–26, 1988.
12. J. Garloff. Convergent bounds for the range of multivariate polynomials. *Interval Mathematics 1985, Lecture Notes in Computer Science*, 212:37–56, 1985.
13. Adrian Geisow. *Surface Interrogations.* PhD thesis, University of East Anglia, 1983.
14. 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.

15. Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Cambridge University Press, 1985.
16. LiDIA Group, Darmstadt University of Technology. www.cdc.informatik.tu-darmstadt.de/TI/LiDIA.
17. George G. Lorentz. *Bernstein Polynomials*. Chelsea Publishing Company, New York, 1986.
18. 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.
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.

A Correctness Proofs

In this appendix we outline some proofs of correctness for the various obfuscations given in the main body of the paper. For our proofs we will use the results of Section 2.1 along with the results of the following section.

A.1 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

$$f \cdot af_B = af_C \cdot f^{\mathcal{O}} \tag{13}$$

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

$$cross\ (f, g)\ (a, b) = (f\ a, g\ b) \tag{14}$$

Thus if $f^{\mathcal{O}}$ is an obfuscation of f then using Equation (13) we have that

$$f \cdot cross(af, af) = af \cdot f^{\mathcal{O}} \tag{15}$$

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

A.2 Unary operations under the Bernstein obfuscation

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

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

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

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

The determinant operation det is different to the other matrix operations we have considered 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, the conversion function is id) as follows:

$$
\begin{aligned}
& \mathsf{det}_{\mathcal{B}}(\mathbf{S}) \\
= \;& \{\text{deriving equation}\} \\
& \mathsf{det}(af(\mathbf{S})) \\
= \;& \{\text{definition of } af \text{ with } (a+1, b+1) = dim(\mathbf{S})\} \\
& \mathsf{det}(\mathbf{U}_a \, \mathbf{S} \, \mathbf{V}_b) \\
= \;& \{\mathsf{det}(\mathbf{B}\,\mathbf{C}) = \mathsf{det}(\mathbf{B}) \times \mathsf{det}\mathbf{C}\} \\
& \mathsf{det}(\mathbf{U}_a) \times \mathsf{det}(\mathbf{S}) \times \mathsf{det}(\mathbf{V}_b)
\end{aligned}
$$

A.3 Binary operations under the Bernstein obfuscation

For a binary matrix operation $\otimes$, we use the non-homogeneous equations defined in Section A.1. 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 (15) 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 $(a+1, b+1) = dim(\mathbf{S})$ and $(b+1, c+1) = dim(\mathbf{T})$ (thus the matrices are conformable). Then, writing mult as the prefix matrix multiplication operator in the place of $\otimes$ in the equation above (but using normal matrix multiplication elsewhere), we can use this equation to derive an obfuscation:

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

Thus, $\mathsf{mult}_{\mathcal{B}}(\mathbf{S}, \mathbf{T}) = \mathbf{S} \, \mathbf{V}_b \, \mathbf{U}_b \, \mathbf{T}$ where $(a+1, b+1) = dim(\mathbf{S})$.

A.4 Arithmetic operations for the number obfuscation

In Section 5.1 we define an obfuscation for numbers by representing a number as the determinant of a matrix. In this section, we discuss the definitions of the arithmetic operations in more details. 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 the conversion function defined in Section 5.1. 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 $n = 0$.

We can easily prove that these operations are correct by using Equations (2) or (15) 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.