1 Introduction

CRYPS is a language designed to help regular users as well as programmers perform cryptographic operations. The language has support for generation, storage and processing of numbers in singular and collection forms, which is essential to implementation of common cryptography mechanisms and research on newer ones.

CRYPS is intended to be used by diverse backgrounds of people. Internet users can use it to encipher messages before exchanging confidential information over mail or instant messengers. Also, it could be used for E-commerce and banking transactions requiring web authentication. Enterprises with intranet systems can encrypt employee credentials with CRYPS. Most importantly, it can be used as a tool to develop new cryptographic algorithms.

2 Lexical Conventions

2.1 Identifiers

An identifier can consist of one or more letters, digits and underscore. The first character should be a letter followed by any sequence of letters, digits and underscore character. First ten characters are significant. Uppercase and lowercase letters are different.

2.2 Comments

Single line comments start with a # character and are terminated at the end of a line.

Multiple line comments start with ‘(#’ and are terminated with ‘#)’.

2.3 Whitespace

The only way to represent whitespace is by binding it with double quotes (” “) on either side, all other forms of whitespace are not considered.

2.4 Reserved Keywords

if
else
while
for
1.5 Data Types

Fundamental data types are \textbf{int} and \textbf{void}. Apart from the fundamental data types, there is a class of derived types constructed from fundamental data types in the following ways:

(1) Arrays of objects of \textbf{int} - The syntax convention for an array is an identifier followed by ‘[’ and ‘]’.

(2) Matrix of objects of int

(3) Functions which may or may not return objects of a given type.

1.6 Constants

CRYPS has only integer constants.

1.7 Separators

The ‘,’ symbol is used to represent a comma separated list. The symbol ‘;’ is used to indicate the end of a statement.

1.8 Scope Rules

CRYPS is a block-structured language, and the scope of names declared in a block is within the body of the block. That is the language using static scoping.
## 3 Expressions and Operators

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Operations</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
<td>Left associative</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
<td>Left associative</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>Left associative</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
<td>Left associative</td>
</tr>
<tr>
<td>^</td>
<td>Exponential</td>
<td>Left associative</td>
</tr>
<tr>
<td>%</td>
<td>Modulus</td>
<td>Left associative</td>
</tr>
<tr>
<td>=</td>
<td>Equal to</td>
<td>Left associative</td>
</tr>
<tr>
<td>!=</td>
<td>Not equal to</td>
<td>Left associative</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
<td>Left associative</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or Equal to</td>
<td>Left associative</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
<td>Left associative</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or Equal to</td>
<td>Left associative</td>
</tr>
<tr>
<td>&lt;&lt;</td>
<td>Left Shift</td>
<td>Left associative</td>
</tr>
<tr>
<td>&gt;&gt;</td>
<td>Right Shift</td>
<td>Left associative</td>
</tr>
<tr>
<td>@</td>
<td>EXOR</td>
<td>Left associative</td>
</tr>
<tr>
<td>,</td>
<td>Sequence</td>
<td>Left associative</td>
</tr>
<tr>
<td>AND</td>
<td>Logical And</td>
<td>Left associative</td>
</tr>
<tr>
<td>OR</td>
<td>Logical Or</td>
<td>Left associative</td>
</tr>
<tr>
<td>&lt;-</td>
<td>Assignment</td>
<td>Right associative</td>
</tr>
</tbody>
</table>
3 **Control Structures**

Conditional Statement

\[
\text{if (expression) statement}
\]

\[
\text{if (expression) statement else statement}
\]

While statement

\[
\text{While (expression) statement}
\]

For statement

\[
\text{for identifier <- expression to expression statements inc val statements}
\]

Switch statement

\[
\text{switch expression \{case constant statements break; default statements\}}
\]

4 **Declaration**

Declarations are used within function definitions to specify the interpretation for each identifier; Declarations have the form:

\[
\text{declaration:}
\]

\[
\text{decl-specifiers declarator-list;}
\]

\[
\text{decl-specifiers:}
\]

\[
\text{type-specifier}
\]

\[
\text{type-specifier:}
\]

\[
\text{int}
\]

\[
\text{void}
\]

\[
\text{declarator-list:}
\]

\[
\text{declarator}
\]

\[
\text{declarator, declarator-list}
\]

Declarators have the syntax:

\[
\text{declarator:}
\]

\[
\text{identifier}
\]

\[
\text{declarator ( )}
\]
5 Namespace

We have only one namespace and have no support for features like class definitions or typedefs that require additional namespaces.

6 Grammar

\[
\text{prg} \rightarrow \text{statements} \\
\text{statements} \rightarrow '{'(\text{stmt};')^+ '} \\
\text{stmt} \rightarrow \text{if-stmt} | \text{while-stmt} | \text{for-stmt} | \text{switch-stmt} | \text{function-defn} \\
| \text{var-decl} | \text{expn} | \text{return-stmt} \\
\text{expn} \rightarrow \text{expn \ binop \ expn} | \text{unary-op \ expn} | '('\text{expn})' \\
| \text{function-call} | \text{array-val} | \text{val} \\
\text{binop} \rightarrow '+'|'-'|'*'|'/'|'^'|'%'|'<'|'='|'>'|'<='|'='|'>='|'^<'|'^>' | '@' \\
| 'AND'| 'OR'| '!=' | '=' \\
\text{digit} \rightarrow ['0' - '9'] \\
\text{assignment-op} \rightarrow '<-' \\
\text{unary-op} \rightarrow '!'|'~'|'-' \\
\text{type} \rightarrow \text{int} | \text{void} \\
\text{array} \rightarrow '['\text{E} | \text{expn} ']' \\
\text{array-val} \rightarrow \text{id} '['\text{expn} ']' \\
\text{matrix} \rightarrow '['\text{E} | \text{expn} ']' '['\text{E} | \text{expn} ']' \\
\text{matrix-val} \rightarrow \text{id} '['\text{expn} ']' '['\text{expn} ']' \\
\text{if-stmt} \rightarrow \text{if \ expn \ statements} \\
| \text{if \ expn \ statements} (\text{elseif \ expn \ statements})* \ \text{else \ statements} \\
\text{while-stmt} \rightarrow \text{while \ expn \ statements} \\
\text{for-stmt} \rightarrow \text{for \ id \ assignment-op \ val \ 'to' \ val \ ('inc' \ val)? \ statements} \\
\text{switch-stmt} \rightarrow \text{switch \ expn} '{'(\text{case \ digit+ \ statements \ break';')*+'default \ statements'}' \\
\text{function-defn} \rightarrow \text{type \ id} '('\text{E} \ | \ \text{parameters}')' \ \text{statements}
parameters -> type parameter ',', parameters | type parameter
parameter -> id | array | matrix

assignment-expn -> id assignment-op expn | id array assignment-op
digit-set
    | id matrix assignment-op digit-mat-set
digit-list -> digit ',', digit-list
digit-set -> '{' digit-list '}'
digit-mat-set -> '{' digit-mat-list '}'
digit-mat-list -> digit-list | digit-list ';' digit-mat-list
function-call -> id '(' E | argument-list ')'argument-list -> argument | argument ',', argument-list
argument -> val|expn
val -> id | digit+
var-decl -> type var-decl-list
var-decl-list -> id | id',,' var-decl-list
    | assignment-expn | assignment-expn',,' var-decl-list
    | array | array ',,' var-decl-list
    | matrix | matrix ',,' var-decl-list
return-stmt -> return | return expn | return val