Cryptocode

Typesetting Cryptography

Arno Mittelbach
mail@arno-mittelbach.de

November 11, 2018
Abstract

The cryptocode package provides a set of macros to ease the typesetting of pseudocode, algorithms and protocols (such as the one below). In addition it comes with a wide range of tools to typeset cryptographic papers (hence the name). This includes simple predefined commands for concepts such as a security parameter $1^n$ or advantage terms $\text{Adv}_{\text{A},\text{PRF}}(n) = \text{negl}(n)$ but also flexible and powerful environments to layout game-based proofs or black-box reductions.

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x \leftarrow \mathbb{Z}_q$</td>
<td>$y \leftarrow \mathbb{Z}_q$</td>
</tr>
<tr>
<td>$X \leftarrow g^x$</td>
<td>$Y \leftarrow g^y$</td>
</tr>
<tr>
<td>$G, q, g, X$</td>
<td>$Y$</td>
</tr>
<tr>
<td>$k_A \leftarrow Y^x$</td>
<td>$k_B \leftarrow X^y$</td>
</tr>
</tbody>
</table>
## Contents

1 Cryptocode by Example .......................... 1
   1.1 Pseudocode ................................... 1
   1.2 Columns ...................................... 3
   1.3 Protocols ..................................... 3
   1.4 Game-based Proofs .............................. 4
   1.5 Black-box Reductions ........................... 5

2 Cryptographic Notation ............................ 7
   2.1 Security Parameter ............................. 7
   2.2 Advantage Terms ................................. 7
   2.3 Math Operators ................................ 8
   2.4 Adversaries .................................... 8
   2.5 Landau ......................................... 8
   2.6 Probabilities .................................. 9
   2.7 Sets ........................................... 10
   2.8 Crypto Notions ................................ 10
   2.9 Logic .......................................... 11
   2.10 Function Families .............................. 11
   2.11 Machine Model .................................. 11
   2.12 Crypto Primitives ............................... 12
   2.13 Events ........................................ 12
   2.14 Complexity .................................... 12
   2.15 Asymptotics ................................... 13
   2.16 Keys .......................................... 13

3 Pseudocode ........................................ 14
   3.1 Basics ......................................... 14
      3.1.1 Customizing Pseudocode .............. 14
      3.1.2 Customized Pseudocode Commands ... 16
   3.2 Indentation ..................................... 16
   3.3 Textmode ...................................... 19
   3.4 Syntax Highlighting ............................. 19
      3.4.1 Alternative Keywords ............... 22
      3.4.2 Draft Mode ............................... 22
   3.5 Line Numbering ................................ 22
      3.5.1 Manually Inserting Line Numbers ... 23
      3.5.2 Start Values ............................. 23
      3.5.3 Separators ............................... 24
   3.6 Subprocedures ................................. 24
      3.6.1 Numbering in Subprocedures ... 24
   3.7 Stacking Procedures ............................ 25
   3.8 Divisions and Linebreaks ...................... 27
   3.9 Matrices and Math Environments within pseudocode 28
   3.10 Fancy Code with Overlays .................... 28
3.10.1 Example: Explain your Code ........................................... 30

4 Tabbing Mode .................................................................... 31
  4.1 Tabbing in Detail ............................................................. 32
    4.1.1 Overriding The Tabbing Character ....................... 32
    4.1.2 Custom Line Spacing and Horizontal Rules .......... 32

5 Protocols .......................................................................... 33
  5.1 Tabbing .......................................................................... 35
  5.2 Multiline Messages ........................................................ 35
    5.2.1 Multiplayer Protocols ......................................... 35
    5.2.2 Divisions .............................................................. 36
  5.3 Line Numbering in Protocols ....................................... 37
    5.3.1 Separators ............................................................. 38
  5.4 Sub Protocols .............................................................. 38

6 Game Based Proofs ............................................................. 40
  6.1 Basics ........................................................................... 40
    6.1.1 Highlight Changes .................................................. 40
    6.1.2 Boxed games ......................................................... 41
    6.1.3 Reduction Hints ..................................................... 41
    6.1.4 Numbering and Names ......................................... 42
    6.1.5 Default Name and Argument ................................ 43
    6.1.6 Two Directional Games ....................................... 43
    6.1.7 Styling game procedures ...................................... 43
  6.2 Game Descriptions ....................................................... 44

7 Black-box Reductions ......................................................... 45
  7.1 Nesting of Boxes ........................................................... 47
  7.2 Messages and Queries .................................................. 47
    7.2.1 Options .................................................................. 49
    7.2.2 Add Space ............................................................ 50
    7.2.3 Loops .................................................................... 51
    7.2.4 Intertext ............................................................... 52
  7.3 Oracles .......................................................................... 53
    7.3.1 Communicating with Oracles ................................ 54
  7.4 Challengers .................................................................... 55
    7.4.1 Communicating with Challengers ......................... 56
  7.5 Examples ....................................................................... 57

8 Known Issues ..................................................................... 59
  8.1 Pseudocode KeepSpacing within Commands ............ 59
  8.2 AMSFonts ................................................................. 59
  8.3 Hyperref ...................................................................... 59
Chapter 1

Cryptocode by Example

The cryptocode package provides a set of commands to ease the typesetting of pseudocode, protocols, game-based proofs and black-box reductions. In addition it comes with a large number of predefined commands. In this chapter we present the various features of cryptocode by giving small examples. But first, let’s load the package

\usepackage{cryptocode}

Note that all the options refer to a set of commands. That is, without any options cryptocode will provide the mechanisms for writing pseudocode, protocols, game-based proofs and black-box reductions but not define additional commands, such as \texttt{\texttt{pk}} or \texttt{\texttt{sk}} (for typesetting public and private/secret keys) which are part of the keys option. We discuss the various options and associated commands in Chapter 2.

1.1 Pseudocode

The cryptocode package tries to make writing pseudocode easy and enjoyable. The \texttt{\texttt{\texttt{pseudocode}}} command takes a single parameter where you can start writing code in mathmode using \texttt{\texttt{\backslash \\backslash}} as line breaks. Following is an IND-CPA game definition using various commands from cryptocode to ease writing keys (\texttt{\texttt{\texttt{pk,sk}}}), sampling (\texttt{\texttt{\texttt{\texttt{sample}}}}), and more:

\begin{verbatim}
1: \ b \leftarrow \{0,1\} \\
2: \ (pk,sk) \leftarrow KGen(1^n) \\
3: \ (state, m_0, m_1) \leftarrow A(1^n, pk, c) \\
4: \ c \leftarrow Enc(pk, m_0) \\
5: \ b' \leftarrow A(1^n, pk, c, state) \\
6: \ \textbf{return} \ b = b'
\end{verbatim}

The above code is generated by (the code is actually wrapped in an \texttt{\texttt{fbox}}).
The pseudocode command thus takes a single mandatory argument (the code) plus an optional argument which allows you to specify options in a key=value fashion. In the above example we used the linenumbering option (which not surprisingly adds line numbers to the code) as well as the syntaxhighlighting option which highlights certain keywords (in the example it is responsible for setting “return” as return).

It is easy to define a heading for your code. Either specify the header using the option “head” or use the procedure command which takes an additional argument to specify the headline.

```
\procedure [linenumbering,syntaxhighlight=auto] {%
  b \sample \bin \\
  (pk, sk) \sample kgen (secparam) \\
  (state, m_0, m_1) \sample adv (secparam, pk, c) \\
  c \sample enc (pk, m_b) \\
  b' \sample adv (secparam, pk, c, state) \\
  \return b = b' 
%
\}
```

Here in the example we have not turned on the automatic syntax highlighting but used the command \pcreturn to highlight the return statement. Besides \pcreturn there are a variant of predefined “keywords” such as \pcfor, \pcif, etc. (all prefixed with pc).

There is a lot more that we will discuss in detail in Chapter 3. Here, for example is the same code with an overlay explanation and a division of the pseudocode.

```
\begin {pcimage}
\procedure [linenumbering] {%
  \begin {enc \adv} \\
  \sample bin \\
  (pk, sk) \sample kgen (secparam) \\
  (state, m_0, m_1) \sample adv (secparam, pk, c) \\
  c \sample enc (pk, m_b) \\
  b' \sample adv (secparam, pk, c, state) \\
  \pcreturn b = b' 
%
\}
\end {pcimage}
```

IND-CPA\textsubscript{Enc}$^4$

1: $b \leftarrow \{0,1\}$
2: $(pk, sk) \leftarrow \text{KGen}(1^n)$
3: $(state, m_0, m_1) \leftarrow \text{A}(1^n, pk, c)$
4: $c \leftarrow \text{Enc}(pk, m_b)$
5: $b' \leftarrow \text{A}(1^n, pk, c, state)$
6: \return $b = b'$

KGen$(1^n)$ samples a public key $pk$ and a private key $sk$. 

```
\begin {pcimage}
\procedure [linenumbering] {%
  \begin {enc \adv} \\
  \sample bin \\
  (pk, sk) \sample kgen (secparam) \pcnode{kgen} \pcnode{pcnode} \\
  (state, m_0, m_1) \sample adv (secparam, pk, c) \\
  \pcinserttext {dotted} \{ Setup Completed \} \\
  c \sample enc (pk, m_b) \\
  b' \sample adv (secparam, pk, c, state) \\
%
\}
\end {pcimage}
```
1.2 Columns

The \texttt{pseudocode} and \texttt{procedure} commands allow the usage of multiple columns. You switch to a new column by inserting a \texttt{>}. This is similar to using an align environment and placing a tabbing & character.

\begin{center}
\begin{tabular}{cccc}
First & Second & Third & Fourth \\
$ b \leftarrow \{0,1\} $ & $ b \leftarrow \{0,1\} $ & $ b \leftarrow \{0,1\} $ & $ b \leftarrow \{0,1\} $ \\
\end{tabular}
\end{center}

As you can see the first column is left aligned the second right, the third left and so forth. In order to get only left aligned columns you could thus simply always skip a column by using \texttt{\}}. You can also use \texttt{\}<} a shorthand for \texttt{\}\}}.

\begin{center}
\begin{tabular}{cccc}
First & Second & Third & Fourth \\
$ b \leftarrow \{0,1\} $ & $ b \leftarrow \{0,1\} $ & $ b \leftarrow \{0,1\} $ & $ b \leftarrow \{0,1\} $ \\
\end{tabular}
\end{center}

1.3 Protocols

Using columns makes it easy to write even complex protocols. Following is a simple three party protocol

\begin{center}
\begin{tabular}{|c|c|c|c|}
\hline
Alice & Bob & Charlie \\
\hline
\begin{itemize}
\item Work
\end{itemize} & Work result & Work result \\
\hline
\begin{itemize}
\item Work
\end{itemize} & Work result & Bottom message \\
\hline
\begin{itemize}
\item A long message for Alice
\end{itemize} & & \\
\hline
\end{tabular}
\end{center}

In fact, the \texttt{pseudocode} command is based on amsmath’s flalign environment.

\footnote{In fact, the \texttt{pseudocode} command is based on amsmath’s flalign environment.}
The commands `\sendmessage{}` and `\sendmessageleft()` are very flexible and allow to style the sending of messages in various ways. Also note the `\hline` at the end of the first line. Here the first optional argument allows us to specify the lineheight (similarly to the behavior in an align environment). The second optional argument allows us to, for example, draw a horizontal line.

In multi player protocols such as the one above the commands `\sendmessageright{}` and `\sendmessageleft{}` (note the x at the end) allow to send messages over multiple columns. In the example, as we were using `\:` the final message thus spans 8 columns.

For basic protocols you might also utilize the `\sendmessageright{*}` and `\sendmessageleft{*}` commands which simply take a message which is displayed.

![Diagram](image.png)

We will discuss protocols in greater detail in Chapter 5.

### 1.4 Game-based Proofs

Cryptocode supports authors in visualizing game-based proofs. It defines an environment `gameproof` which allows to wrap a number of game procedures displaying helpful information as to what changes from game to game, and to what each step is reduced.

```plaintext
\begin{gameproof}
\begin{gameprocedure}[linenumbering, mode=text]
% some hint
\begin{align*}
\text{Game}_1(n) & \quad \text{Game}_2(n) \\
1 : & \quad \text{Step 1} \quad \text{Step 1} \\
2 : & \quad \text{Step 2} \quad \text{Step 2 is different} \\
3 : & \quad \text{Step 3} \quad \text{Step 3}
\end{align*}
\end{gameprocedure}
\end{gameproof}
```
Note that we made use of the option “mode=text” in the above example which tells the underlying pseudocode command to not work in math mode but in plain text mode. We’ll discuss how to visualize game-based proofs in Chapter 6.

1.5 Black-box Reductions

Cryptocode provides a structured syntax to visualize black-box reductions. Basically cryptocode provides an environment to draw boxes that may have oracles and that can be communicated with. Cryptocode makes heavy use of TIKZ (https://www.ctan.org/pkg/pgf) for this, which gives you quite some control over how things should look like. Additionally, as you can specify node names (for example the outer box in the next example is called “A”) you can easily extend the pictures by using plain TIKZ commands.
We’ll discuss the details in Chapter 7.
Chapter 2

Cryptographic Notation

In this section we’ll discuss the various commands for notation that can be loaded via package options.

Remark. The commands defined so far are far from complete and are currently mostly targeted at what I needed in my papers (especially once you get to cryptographic notions and primitives). So please if you feel that something should be added drop me an email.

2.1 Security Parameter

In cryptography we make use of a security parameter which is usually written as $1^n$ or $1^\lambda$. The cryptocode package, when loading either option “n” or option “lambda” will define the commands

\begin{verbatim}
\usepackage[
advantage, operators, sets, adversary, landau, probability, notions, logic, ff, mm,
primitives, events, complexity, asymptotics, keys
]{cryptocode}
\end{verbatim}

\begin{verbatim}
\texttt{secpar}
\texttt{secparam}
\end{verbatim}

The first command provides the “letter”, i.e., either $n$ or $\lambda$, whereas \texttt{secparam} points to $1^n$.

2.2 Advantage Terms

Load the package option “advantage” in order to define the command \texttt{advantage} used to specify advantage terms such as:

$$Adv_{\text{PRF}}^\text{prf}(n) = \negl(n)$$

\begin{verbatim}
\advantage{prf}{\adv}{\text{prf}} = \negl
\end{verbatim}

Specify an optional third parameter to replace the $(n)$. 


In order to redefine the styles in which superscript and subscript are set redefine

\snow{\text{pcadvantagesuperstyle}}{|1|}{\text{mathrm}{\text{\MakeLowercase{#1}}}}
\snow{\text{pcadvantagesubstyle}}{|\#1|}

2.3 Math Operators

The “operators” option provides the following list of commands:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Result</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>\sample</td>
<td>Sampling from a distribution, or running a randomized procedure</td>
<td>$\leftarrow$</td>
<td>$b \leftarrow {0,1}$</td>
</tr>
<tr>
<td>\floor{42.5}</td>
<td>Rounding down</td>
<td>$42.5$</td>
<td></td>
</tr>
<tr>
<td>\ceil{41.5}</td>
<td>Rounding up</td>
<td>$41.5$</td>
<td></td>
</tr>
<tr>
<td>\Angle{x,y}</td>
<td>Vector product</td>
<td>$\langle x,y \rangle$</td>
<td></td>
</tr>
<tr>
<td>\abs{42.9}</td>
<td>Absolute number</td>
<td>$42.9$</td>
<td></td>
</tr>
<tr>
<td>\norm{x}</td>
<td>Norm</td>
<td>$|x|$</td>
<td></td>
</tr>
<tr>
<td>\concat</td>
<td>Verbose concatenation (I usually prefer simply |)</td>
<td>$x \leftarrow a|b$</td>
<td></td>
</tr>
<tr>
<td>\emptystring</td>
<td>The empty string</td>
<td>$\epsilon$</td>
<td>$x \leftarrow \epsilon$</td>
</tr>
</tbody>
</table>

2.4 Adversaries

The “adversary” option provides the following list of commands:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>\adv</td>
<td>Adversary</td>
<td>$A$</td>
</tr>
<tr>
<td>\bdv</td>
<td>Adversary</td>
<td>$B$</td>
</tr>
<tr>
<td>\cdv</td>
<td>Adversary</td>
<td>$C$</td>
</tr>
<tr>
<td>\ddv</td>
<td>Adversary</td>
<td>$D$</td>
</tr>
<tr>
<td>\mdv</td>
<td>Adversary</td>
<td>$M$</td>
</tr>
<tr>
<td>\pdv</td>
<td>Adversary</td>
<td>$P$</td>
</tr>
<tr>
<td>\sdv</td>
<td>Adversary</td>
<td>$S$</td>
</tr>
</tbody>
</table>

The style in which an adversary is rendered is controlled via

\snow{\text{pcadvstyle}}{|1|}{\text{mathcal}{#1}}

2.5 Landau

The “landau” option provides the following list of commands:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>\bigO{n^2}</td>
<td>Big O notation</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>\smallO{n^2}</td>
<td>small o notation</td>
<td>$o\left(n^2\right)$</td>
</tr>
<tr>
<td>\bigOmega{n^2}</td>
<td>Big Omega notation</td>
<td>$\Omega\left(n^2\right)$</td>
</tr>
<tr>
<td>\bigsmallO{n^2}</td>
<td>Big and small O notation</td>
<td>$\Theta\left(n^2\right)$</td>
</tr>
</tbody>
</table>
2.6 Probabilities

The “probability” option provides commands for writing probabilities. Use

1. \prob{X=x}
2. \probsub{x}{\binom{n}{x}}{x=5}
3. \condprob{X=x}{A=b}
4. \condprobsub{x}{\binom{n}{x}}{x=5}{A=b}

to write basic probabilities, probabilities with explicit probability spaces and conditional probabilities.

\[ \Pr[X = x] \]
\[ \Pr_x \leftrightarrow (0,1)^n [X = x] \]
\[ \Pr[X = x \mid A = b] \]
\[ \Pr_x \leftrightarrow (0,1)^n [x = 5 \mid A = b] \]

You can control the probability symbol (Pr) by redefining

\[ \renewcommand{\probname}{Pr} \]

For expectations you can use

1. \expect{X}
2. \expsub{x,y}{\set{1,\ldots,6}}{x+y}
3. \condexp{X}{Y}{Y>3}
4. \condexpsub{x,y}{\set{1,\ldots,6}}{x+y}{y>3}

yielding

\[ E[X] \]
\[ E_{x,y \leftrightarrow (1,\ldots,6)} [x + y] \]
\[ E[X + Y \mid Y > 3] \]
\[ E_{x,y \leftrightarrow (1,\ldots,6)} [x + y \mid y > 3] \]

You can control the expectation symbol (E) by redefining

\[ \renewcommand{\expectationname}{\ensuremath{\mathbb{E}}} \]

The support \( \text{Supp}(X) \) of a random variable \( X \) can be written as

\[ \text{supp}(X) \]

where again the name can be controlled via

\[ \renewcommand{\supportname}{\text{Supp}} \]

For denoting entropy and min-entropy use

\[ \entropy{X} \]
\[ \minentropy{X} \]
\[ \condentropy{X}{Y=5} \]
\[ \condminentropy{X}{Y=5} \]
\[ \condavgminentropy{X}{Y=5} \]
This yields

\[
H(X) \\
H_\infty(X) \\
H(X \mid Y = 5) \\
H_\infty(X \mid Y = 5) \\
\tilde{H}_\infty(X \mid Y = 5)
\]

### 2.7 Sets

The “sets” option provides commands for basic mathematical sets. You can write sets and sequences as

\[
\set{1, \ldots, 10} \\
\sequence{1, \ldots, 10}
\]

which is typeset as

\[
\{1, \ldots, 10\} \\
(1, \ldots, 10)
\]

In addition the following commands are provided:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>\bin</td>
<td>The set containing 0 and 1</td>
<td>{0, 1}</td>
</tr>
<tr>
<td>\NN</td>
<td>Natural numbers</td>
<td>\mathbb{N}</td>
</tr>
<tr>
<td>\ZZ</td>
<td>Integers</td>
<td>\mathbb{Z}</td>
</tr>
<tr>
<td>\QQ</td>
<td>Rational numbers</td>
<td>\mathbb{Q}</td>
</tr>
<tr>
<td>\CC</td>
<td>Complex numbers</td>
<td>\mathbb{C}</td>
</tr>
<tr>
<td>\RR</td>
<td>Reals</td>
<td>\mathbb{R}</td>
</tr>
<tr>
<td>\PP</td>
<td></td>
<td>\mathbb{P}</td>
</tr>
<tr>
<td>\FF</td>
<td></td>
<td>\mathbb{F}</td>
</tr>
</tbody>
</table>

### 2.8 Crypto Notions

The “notions” option provides the following list of commands:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>\indcpa</td>
<td>IND-CPA security for encryption schemes</td>
<td>IND-CPA</td>
</tr>
<tr>
<td>\indcca</td>
<td>IND-CCA security for encryption schemes</td>
<td>IND-CCA</td>
</tr>
<tr>
<td>\indccai</td>
<td>IND-CCA1 security for encryption schemes</td>
<td>IND-CCA1</td>
</tr>
<tr>
<td>\indccaii</td>
<td>IND-CCA2 security for encryption schemes</td>
<td>IND-CCA2</td>
</tr>
<tr>
<td>\priv</td>
<td>PRIV security for deterministic public-key encryption schemes</td>
<td>PRIV</td>
</tr>
<tr>
<td>\ind</td>
<td>IND security (for deterministic public-key encryption schemes)</td>
<td>IND</td>
</tr>
<tr>
<td>\prvcda</td>
<td>PRV-CDA security (for deterministic public-key encryption schemes)</td>
<td>PRV-CDA</td>
</tr>
<tr>
<td>\prvrcda</td>
<td>PRV$$-CDA security (for deterministic public-key encryption schemes)</td>
<td>PRV$$-CDA</td>
</tr>
<tr>
<td>\kiae</td>
<td>Key independent authenticated encryption</td>
<td>KIAE</td>
</tr>
<tr>
<td>\kdae</td>
<td>Key dependent authenticated encryption</td>
<td>KDAE</td>
</tr>
<tr>
<td>\mle</td>
<td>Message locked encryption</td>
<td>MLE</td>
</tr>
<tr>
<td>\uce</td>
<td>Universal computational extractors</td>
<td>UCE</td>
</tr>
</tbody>
</table>

The style in which notions are displayed can be controlled via redefining

\[
\renewcommand{\pcnotionstyle}[1]{\ensuremath{\text{\textnormal{#1}}}}
\]
2.9 Logic

The “logic” option provides the following list of commands:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>\AND</td>
<td>Logical AND</td>
<td>AND</td>
</tr>
<tr>
<td>\NAND</td>
<td>Logical NAND</td>
<td>NAND</td>
</tr>
<tr>
<td>\OR</td>
<td>Logical OR</td>
<td>OR</td>
</tr>
<tr>
<td>\NOR</td>
<td>Logical NOR</td>
<td>NOR</td>
</tr>
<tr>
<td>\XOR</td>
<td>Logical XOR</td>
<td>XOR</td>
</tr>
<tr>
<td>\XNOR</td>
<td>Logical XNOR</td>
<td>XNOR</td>
</tr>
<tr>
<td>\NOT</td>
<td>not</td>
<td>NOT</td>
</tr>
<tr>
<td>\xor</td>
<td>exclusive OR</td>
<td>⊕</td>
</tr>
<tr>
<td>\false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>\true</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>

2.10 Function Families

The “ff” option provides the following list of commands:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>\kgen</td>
<td>Key generation</td>
<td>KGen</td>
</tr>
<tr>
<td>\pgen</td>
<td>Parameter generation</td>
<td>Pgen</td>
</tr>
<tr>
<td>\eval</td>
<td>Evaluation</td>
<td>Eval</td>
</tr>
<tr>
<td>\invert</td>
<td>Inversion</td>
<td>Inv</td>
</tr>
<tr>
<td>\il</td>
<td>Input length</td>
<td>il</td>
</tr>
<tr>
<td>\ol</td>
<td>Output length</td>
<td>ol</td>
</tr>
<tr>
<td>\kl</td>
<td>Key length</td>
<td>kl</td>
</tr>
<tr>
<td>\nl</td>
<td>Nonce length</td>
<td>nl</td>
</tr>
<tr>
<td>\rl</td>
<td>Randomness length</td>
<td>rl</td>
</tr>
</tbody>
</table>

The style in which these are displayed can be controlled via redefining
\[ \text{\texttt{\textbackslash renewcommand\{pcalgestyle\}[1]{\texttt{\textbackslash ensuremath\{\texttt{\textbackslash mathsf\{#1\}}\}}}}} \]

2.11 Machine Model

The “mm” option provides the following list of commands:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>\CRKT</td>
<td>A circuit</td>
<td>C</td>
</tr>
<tr>
<td>\TM</td>
<td>A Turing machine</td>
<td>M</td>
</tr>
<tr>
<td>\PROG</td>
<td>A program</td>
<td>P</td>
</tr>
<tr>
<td>\uTM</td>
<td>A universal Turing machine</td>
<td>UM</td>
</tr>
<tr>
<td>\uC</td>
<td>A universal Circuit</td>
<td>UC</td>
</tr>
<tr>
<td>\uP</td>
<td>A universal Program</td>
<td>UEval</td>
</tr>
<tr>
<td>\tmtime</td>
<td>Time (of a TM)</td>
<td>time</td>
</tr>
<tr>
<td>\ppt</td>
<td>Probabilistic polynomial time</td>
<td>PPT</td>
</tr>
</tbody>
</table>

The style in which these are displayed can be controlled via redefining
\[ \text{\texttt{\textbackslash renewcommand\{pcmachinemodelstyle\}[1]{\texttt{\textbackslash ensuremath\{\texttt{\textbackslash mathsf\{#1\}}\}}}}} \]
2.12 Crypto Primitives

The “primitives” option provides the following list of commands:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>prover</td>
<td>Prover</td>
<td>P</td>
</tr>
<tr>
<td>verifier</td>
<td>Verifier</td>
<td>V</td>
</tr>
<tr>
<td>nizk</td>
<td>Non interactive zero knowledge</td>
<td>NIZK</td>
</tr>
<tr>
<td>hash</td>
<td>A hash function</td>
<td>H</td>
</tr>
<tr>
<td>ghash</td>
<td>A hash function</td>
<td>G</td>
</tr>
<tr>
<td>fash</td>
<td>A hash function</td>
<td>F</td>
</tr>
<tr>
<td>enc</td>
<td>Encryption</td>
<td>Enc</td>
</tr>
<tr>
<td>dec</td>
<td>Decryption</td>
<td>Dec</td>
</tr>
<tr>
<td>sig</td>
<td>Signing</td>
<td>Sig</td>
</tr>
<tr>
<td>verify</td>
<td>Verifying</td>
<td>Vf</td>
</tr>
<tr>
<td>obf</td>
<td>Obfuscation</td>
<td>O</td>
</tr>
<tr>
<td>iO</td>
<td>Indistinguishability obfuscation</td>
<td>iO</td>
</tr>
<tr>
<td>diO</td>
<td>Differing inputs obfuscation</td>
<td>diO</td>
</tr>
<tr>
<td>mac</td>
<td>Message authentication</td>
<td>MAC</td>
</tr>
<tr>
<td>puncture</td>
<td>Puncturing</td>
<td>Puncture</td>
</tr>
<tr>
<td>source</td>
<td>A source</td>
<td>S</td>
</tr>
<tr>
<td>predictor</td>
<td>A predictor</td>
<td>P</td>
</tr>
<tr>
<td>sam</td>
<td>A sampler</td>
<td>Sam</td>
</tr>
<tr>
<td>distinguisher</td>
<td>A distinguisher</td>
<td>Dist</td>
</tr>
<tr>
<td>dist</td>
<td>A distinguisher</td>
<td>D</td>
</tr>
<tr>
<td>simulator</td>
<td>A simulator</td>
<td>Sim</td>
</tr>
<tr>
<td>ext</td>
<td>An extractor</td>
<td>Ext</td>
</tr>
</tbody>
</table>

The style in which these are displayed can be controlled via redefining:

\setcounter{equation}{0}
\renewcommand{\calsymbol}{\ensuremath{\mathsf{#1}}}

2.13 Events

The “events” option provides the following list of commands.

To classify an event use

\begin{itemize}
  \item \texttt{\textbackslash event \{Event\}}
  \item \texttt{\textbackslash nevent \{Event\}}
\end{itemize}

where the second is meant as the negation. These are typset as

\begin{itemize}
  \item Event
  \item \texttt{\textbackslash nevent \{Event\}}
\end{itemize}

For bad events, use \texttt{\textbackslash bad \{bad\}}.

2.14 Complexity

The “complexity” option provides the following list of commands:
The style in which these are displayed can be controlled via redefining
\renewcommand{\pcpolynomialstyle}{\ensuremath{\mathsf{#1}}}

\begin{tabular}{|c|c|}
\hline
Command & Description \\
\hline
\negl & A negligible function \\
\poly & A polynomial \\
\pp & some polynomial p \\
\qq & some polynomial q \\
\hline
\end{tabular}

The style in which these are displayed can be controlled via redefining
\renewcommand{\pckeystyle}{\ensuremath{\mathsf{#1}}}

\begin{tabular}{|c|c|}
\hline
Command & Description \\
\hline
\pk & public key \\
\vk & verification key \\
\sk & secret key \\
\key & a plain key \\
\hk & hash key \\
\gk & gash key \\
\fk & function key \\
\hline
\end{tabular}

The style in which these are displayed can be controlled via redefining
\renewcommand{\pckeystyle}{\ensuremath{\mathsf{#1}}}

2.15 Asymptotics

The “asymptotics” option provides the following list of commands:

\begin{tabular}{|c|c|}
\hline
Command & Description \\
\hline
\negl & A negligible function \\
\poly & A polynomial \\
\pp & some polynomial p \\
\qq & some polynomial q \\
\hline
\end{tabular}

The style in which these are displayed can be controlled via redefining
\renewcommand{\pccomplexitystyle}{\ensuremath{\mathsf{#1}}}
Chapter 3

Pseudocode

In this chapter we discuss how to write pseudocode with the cryptocode library.

3.1 Basics

The cryptocode package provides the command \texttt{pseudocode} for typesetting algorithms. Consider the following definition of an IND-CPA game

\begin{verbatim}
\texttt{pseudocode}\n\{\%
\texttt{b} \texttt{\sample} \texttt{\{0,1\}}
\texttt{(pk, sk)} \texttt{\sample} \texttt{KGen(1^n)}
\texttt{(m_0, m_1)} \texttt{\sample} \texttt{A(1^n, pk, c)}
\texttt{c} \texttt{\leftarrow Enc(pk, m_b)}
\texttt{b'} \texttt{\sample} \texttt{A(1^n, pk, c)}
\texttt{return b = b'}\}
\end{verbatim}

which is generated as

\begin{verbatim}
\begin{verbatim}
\texttt{b} \texttt{\sample} \texttt{\{0,1\}}
\texttt{(pk, sk)} \texttt{\sample} \texttt{KGen(1^n)}
\texttt{(m_0, m_1)} \texttt{\sample} \texttt{A(1^n, pk, c)}
\texttt{c} \texttt{\leftarrow Enc(pk, m_b)}
\texttt{b'} \texttt{\sample} \texttt{A(1^n, pk, c)}
\texttt{return b = b'}
\end{verbatim}
\end{verbatim}

As you can see the pseudocode command provides a math based environment where you can simply start typing your pseudocode separating lines by \texttt{\textbackslash \textbackslash}.

\textbf{Boxed appearance} Although most examples here appear centered and boxed this is not directly part of the pseudocode package but due to the examples being typeset as

\begin{verbatim}
\begin{verbatim}
\verb|\begin{center}| \verb|\fbox| \verb|\{\%
\texttt{Code}
\verb|\end{center}|\}
\end{verbatim}
\end{verbatim}

3.1.1 Customizing Pseudocode

Besides the mandatory argument the \texttt{\textbackslash pseudocode} command can take an optional argument which consists of a list of key=value pairs separated by commas (,).
The following keys are available:

- **head** A header for the code
- **width** An exact width. If no width is specified, cryptocode tries to automatically compute the correct width.
- **lnstart** The starting line number when using line numbering.
- **lnstartright** The starting line number for right aligned line numbers when using line numbering.
- **linenumbering** Enables line numbering.
- **syntaxhighlight** When set to “auto” cryptocode will attempt to automatically highlight keywords such as “for”, “foreach” and “return”.
- **keywords** Provide a comma separated list of keywords for automatic syntax highlighting. To customize the behavior of automatic spacing you can provide keywords as
  - **keywordsindent** After seeing this keyword all following lines will be indented one extra level.
  - **keywordsunindent** After seeing this keyword the current and all following lines will be unindented one extra level.
  - **keywordsuninindent** After seeing this keyword the current line will be unindented one level.
- **addkeywords** Provide additional keywords for automatic syntax highlighting.
- **altkeywords** Provide a second list of keywords for automatic syntax highlighting that are highlighted differently.
- **mode** When set to text pseudocode will not start in math mode but in text mode.
- **space** Allows you to enable automatic spacing mode. If set to “keep” the spaces in the input are preserved. If set to “auto” it will try to detect spacing according to keywords such as “if” and “fi”.
- **codesize** Allows to specify the fontsize for the pseudocode. Set to \scriptsize for a smaller size.
- **colspace** Allows to insert spacing between columns. In particular this allows to also overlap columns by inserting negative space.
- **jot** Allows to specify extra space between each line. Use jot=1mm.
- **beginline** Allows to specify a macro that is placed at the beginning of each line.
- **endline** Allows to specify a macro that is placed at the end of each line.
- **xshift** Allows horizontal shifting
- **yshift** Allows horizontal shifting
- **headlinesep** Specifies the distance between header and the line.
- **bodylinesep** Specifies the distance between body and the line.
- **colsep** Defines the space between columns.
- **addtolength** Is added to the automatically computed width of the pseudocode (which does not take colsep into account).
- **valign** Controls the vertical alignment of the pseudocode. Pseudocode is wrapped in a minipage environment and valign value is passed as orientation for the minipage. By default valign is set to “t”.

15
Forces syntax highlighting also in draft mode.

The following code

\begin{pseudocode}[linenumbering,syntaxhighlight=auto,head=Header]{return null}
\end{pseudocode}

creates

<table>
<thead>
<tr>
<th>Header</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 : return null</td>
</tr>
</tbody>
</table>

### 3.1.2 Customized Pseudocode Commands

Besides the `pseudocode` command the command `procedure` provides easy access to generate code with a header. It takes the following form

\begin{procedure}[options]{Header}{Body}
\end{procedure}

**Examples**

```
IND-CPA^A
b \leftarrow \{0,1\}
(pk, sk) \leftarrow KGen(1^n)
(m_0, m_1) \leftarrow A(1^n, pk, c)
c \leftarrow Enc(pk, m_b)
b' \leftarrow A(1^n, pk, c)
return b = b'
```

which is generated as

\begin{procedure}[$\indcpa^A_{\text{enc \hat{\text{adv}}}}$]{\%}
\begin{algorithmic}[1]
\State $b \leftarrow \{0,1\}$
\State $(pk, sk) \leftarrow KGen(1^n)$
\State $(m_0, m_1) \leftarrow A(1^n, pk, c)$
\State $c \leftarrow Enc(pk, m_b)$
\State $b' \leftarrow A(1^n, pk, c)$
\State return $b = b'$
\end{algorithmic}
\end{procedure}

You can define customized pseudocode commands which either take one optional argument and two mandatory arguments (as the `procedure` command) or one optional and one mandatory argument (as the `pseudocode` command). The following

\begin{verbatim}
\createprocedurecommand{mypseudocode}{}{}{linenumbering}
\createpseudocodecommand{myheadlesscmd}{}{}{linenumbering}
\end{verbatim}

creates the commands `mypseudocode` and `myheadlesscmd` with line numbering always enabled. The first command has an identical interface as the `pseudocode` command, the second has an interface as the `procedure` command. The second and third argument that we kept empty when generating the commands allows us to specify commands that are executed at the very beginning when the command is called (argument 2) and a prefix for the header.

### 3.2 Indentation

In order to indent code use `\pcindent` or short `\t`. You can also use customized spacing such as `\quad` or `\hspace` when using the pseudocode command in math mode.
for $i = 1 \ldots 10$ do
$T[i] \leftarrow \{0,1\}^n$
for $i = 1 \ldots 10$ do
$T[i] \leftarrow \{0,1\}^n$

which is generated as

```
\texttt{\textbackslash for i = 1 \ldots 10 \textbackslash do}
\texttt{\textbackslash \quad T[i] \leftarrow \{0,1\}^n}
\texttt{\textbackslash for i = 1 \ldots 10 \textbackslash do}
\texttt{\textbackslash \quad T[i] \leftarrow \{0,1\}^n}
```

You can specify multiple levels via the optional first argument

```
\texttt{\textbackslash for i = 1 \ldots 10 \textbackslash do}
\texttt{\textbackslash \quad \textbackslash \quad \texttt{\textbackslash pcind}}
\texttt{\textbackslash \quad \texttt{T[i] \leftarrow \{0,1\}^n}}
```

You can customize the indentation shortcut by redefining

```
\texttt{\renewcommand{\textbackslash pcindentname}{t}}
```

**Automatic Indentation**

The pseudocode command comes with an option “space=auto” which tries to detect the correct indentation from the use of keywords. When it sees one of the following keywords

```
\texttt{\textbackslash pcif, \textbackslash pcfor, \textbackslash pcwhile, \textbackslash pcrepeat, \textbackslash pcforeach}
```

it will increase the indentation starting from the next line. It will again remove the indentation on seeing

```
\texttt{\textbackslash pcfi, \textbackslash pcendif, \textbackslash pcendfor, \textbackslash pcendwhile, \textbackslash pcuntil, \textbackslash pcendforeach}
```

Additionally, on seeing

```
\texttt{\textbackslash pelse, \textbackslash pelseif}
```

it will remove the indentation for that particular line. Thus the following
for $a \in [10]$ do
  for $a \in [10]$ do
    for $a \in [10]$ do
      if $a = b$ then
        some operation
      elseif $a = c$ then
        some operation
      else
        some default operation
      fi
    endfor
  endfor
endfor
return $a$

Note that the manual indentation in the above example is not necessary for the outcome. Further note that the same works when using automatic syntax highlighting (see Section 3.4).

Keep Input Indentation

The pseudocode package comes with an experimental feature that preserves the spacing in the input. This can be enabled with the option “space=keep”. Thus the above can also be written as

```latex
\begin{pseudocode}[space=keep]
  \begin{algorithm}
    \begin{algorithmic}
      \function{main}{$n$}
        \For{$i = 1..10$} 
          \State $T[i] \leftarrow \{0,1\}^n$
          \State $T[i] \leftarrow \{0,1\}^n$
          \State $T[i] \leftarrow \{0,1\}^n$
    \end{algorithmic}
  \end{algorithm}
\end{pseudocode}
```

Note that automatic spacing only works when the `\pseudocode` command is not wrapped within another command. Thus in order to get a frame box `\fbox{\pseudocode[space=keep]{code}}` will not work but you
3.3 Textmode

By default pseudocode enables LaTeX\textsuperscript{\textregistered} math mode. You can change this behavior and tell the pseudocode command to interpret the content in text mode by setting the option “mode=text”.

```latex
\begin{pseudocode}[mode=text]
\%
This is \ just \ simple \ text
\end{pseudocode}
```

3.4 Syntax Highlighting

In the above examples we have used commands `\pcreturn` and `\pcfor` to highlight certain keywords. Besides the `\pcreturn`, `\pcfor` and `\pcdo` (where the pc stands for pseudocode) that were used in the above examples the package defines the following set of constants:

<table>
<thead>
<tr>
<th>name</th>
<th>usage</th>
<th>outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>pcabort</td>
<td><code>\pcabort</code></td>
<td>abort</td>
</tr>
<tr>
<td>pcontinue</td>
<td><code>\pcontinue</code></td>
<td>continue</td>
</tr>
<tr>
<td>pccomment</td>
<td><code>\pccomment{comment}</code></td>
<td>comment</td>
</tr>
<tr>
<td>pcdo</td>
<td><code>\pcdo</code></td>
<td>do</td>
</tr>
<tr>
<td>pcdone</td>
<td><code>\pcdone</code></td>
<td>done</td>
</tr>
<tr>
<td>pcfail</td>
<td><code>\pcfail</code></td>
<td>fail</td>
</tr>
<tr>
<td>pcfalse</td>
<td><code>\pcfalse</code></td>
<td>false</td>
</tr>
<tr>
<td>pcif</td>
<td><code>\pcif</code></td>
<td>if</td>
</tr>
<tr>
<td>pcfi</td>
<td><code>\pcfi</code></td>
<td>fi</td>
</tr>
<tr>
<td>pcendif</td>
<td><code>\pcendif</code></td>
<td>endif</td>
</tr>
<tr>
<td>pcelse</td>
<td><code>\pcelse</code></td>
<td>else</td>
</tr>
<tr>
<td>pcelseif</td>
<td><code>\pcelseif</code></td>
<td>elseif</td>
</tr>
<tr>
<td>pcfor</td>
<td><code>\pcfor</code></td>
<td>for</td>
</tr>
<tr>
<td>pcendforeach</td>
<td><code>\pcendforeach</code></td>
<td>foreach</td>
</tr>
<tr>
<td>pcendfor</td>
<td><code>\pcendfor</code></td>
<td>endfor</td>
</tr>
<tr>
<td>pcforeach</td>
<td><code>\pcforeach</code></td>
<td>foreach</td>
</tr>
<tr>
<td>pcforeground</td>
<td><code>\pcforeground</code></td>
<td>foreground</td>
</tr>
<tr>
<td>pcglobvar</td>
<td><code>\pcglobvar</code></td>
<td>gbl</td>
</tr>
<tr>
<td>pcin</td>
<td><code>\pcin</code></td>
<td>in</td>
</tr>
<tr>
<td>pcnew</td>
<td><code>\pcnew</code></td>
<td>new</td>
</tr>
<tr>
<td>pcnull</td>
<td><code>\pcnull</code></td>
<td>null</td>
</tr>
<tr>
<td>pcparse</td>
<td><code>\pcparse</code></td>
<td>parse</td>
</tr>
<tr>
<td>pcrepeat</td>
<td><code>\pcrepeat{10}</code></td>
<td>repeat 10 times</td>
</tr>
<tr>
<td>pcreturm</td>
<td><code>\pcreturn</code></td>
<td>return</td>
</tr>
<tr>
<td>pcuntil</td>
<td><code>\pcuntil</code></td>
<td>until</td>
</tr>
<tr>
<td>pcthen</td>
<td><code>\pcthen</code></td>
<td>then</td>
</tr>
<tr>
<td>pcttrue</td>
<td><code>\pcttrue</code></td>
<td>true</td>
</tr>
<tr>
<td>pcwhile</td>
<td><code>\pcwhile</code></td>
<td>while</td>
</tr>
<tr>
<td>pcendwhile</td>
<td><code>\pcendwhile</code></td>
<td>endwhile</td>
</tr>
</tbody>
</table>

Note that `\pcdo`, `\pcin` and `\pcthen` have a leading space. This is due to their usual usage scenarios such as
for $i$ in $\{1, \ldots , 10\}$

Furthermore all constants have a trailing space. This can be removed by adding the optional parameter \, such as

for $i$ in $\{1, \ldots , 10\}$

In order to change the font you can overwrite the command \highlightkeyword which is defined as

\newcommand{\highlightkeyword}{\ensuremath{\mathbf{#2}#1}}

**Automatic Syntax Highlighting**

The pseudocode command comes with an experimental feature to automatically highlight keywords. This can be activated via the option “syntaxhighlight=auto”. The preset list of keywords it looks for are

```
for, foreach, return, { do }, in, new, if, null, null, true, true, until, { to }, false, false, { then }, repeat, else, done, done, fi
```

Note that the keywords are matched with spaces and note the grouping for trailing spaces. That is, the “do” keyword won’t match within the string “don’t”. Via the option “keywords” you can provide a custom list of keywords. Thus the following bubblesort variant (taken from http://en.wikipedia.org/wiki/Bubble_sort)

\begin{verbatim}
Bubblesort(A : list of items)
    n ← length(A)
    repeat
        s ← false
        for $i = 1$ to $n - 1$
            // if this pair is out of order
            if $A[i - 1] > A[i]$ then
                // swap them and remember something changed
                swap($A[i - 1], A[i]$)
                s ← true
        until ¬s

can be typeset as

\begin{verbatim}
\procedure{synta\_highlight=auto}{(Bubblesort(A : list of items))}{
    n \gets length(A) \n    repeat \n        \t s \gets false \n        \t for $i = 1$ to $n - 1$
        \t \t \pccomment{if this pair is out of order} \n        \t \t if $A[i - 1] > A[i]$ then \n        \t \t \t \pccomment{swap them and remember something changed} \n        \t \t \t \mathbf{swap}(A[i - 1], A[i]) \n        \t \t s \gets true \n    until \neg s}
\end{verbatim}

You can also define additional keywords using the “addkeywords” option. This would allow us to specify “length” and “swap” in the above example. Combined with automatic spacing we could thus get
Bubblesort(A : list of items)

\[ n \leftarrow \text{length}(A) \]

repeat
    \[ s \leftarrow \text{false} \]
    for \[ i = 1 \text{ to } n-1 \] do
        // if this pair is out of order
        if \[ A[i-1] > A[i] \] then
            // swap them and remember something changed
            swap(A[i-1], A[i])
            \[ s \leftarrow \text{true} \]
    endfor
until \neg s

Simply by writing (note the \(\neg\) in order to not have a space before \(s\)):

\[
\begin{align*}
\text{Bubblesort}(A : \text{list of items}) \\
n & \leftarrow \text{length}(A) \\
\text{repeat} \\
    s & \leftarrow \text{false} \\
    \text{for} \ i = 1 \text{ to } n-1 \text{ do} \\
        \text{// if this pair is out of order} \\
        \text{if} \ A[i-1] > A[i] \text{ then} \\
        \text{// swap them and remember something changed} \\
        \text{swap}(A[i-1], A[i]) \\
        s & \leftarrow \text{true} \\
\end{align*}
\]
until \neg s

Also note that a simple \fbox around the above \procedure command has the effect that the automatic spacing fails. For this also see Section 8.1. As an alternative we could use automatic spacing and insert “group end” keywords such as “\text{\texttt{fi}}”:

\[
\begin{align*}
\text{Bubblesort}(A : \text{list of items}) \\
n & \leftarrow \text{length}(A) \\
\text{repeat} \\
    s & \leftarrow \text{false} \\
    \text{for} \ i = 1 \text{ to } n-1 \text{ do} \\
        \text{// assuming this pair is out of order} \\
        \text{if} \ A[i-1] > A[i] \text{ then} \\
        \text{// swap them and remember something changed} \\
        \text{swap}(A[i-1], A[i]) \\
        s & \leftarrow \text{true} \\
\end{align*}
\]
until \neg s

The last example is generated as (note that here fbox is fine.)
3.4.1 Alternative Keywords

There is a second keyword list that you can add keywords to which are highlighted not via \highlightkeyword command but via \highlightaltkeyword. This allows you to have two different keyword styles which are by default defined as

\begin{verbatim}
\newcommand{\highlightkeyword}{\ensuremath{\mathbf{#2}}#1}
\newcommand{\highlightaltkeyword}{\ensuremath{\mathsf{#1}}}
\end{verbatim}

This allows you to rewrite the above example and highlight the different nature of swap and length.

```
procedure [space=auto, syntaxhighlight=auto, addkeywords={swap, length}]
{ Bubblesort (A : list of items)

n ← length(A)
repeat
   s ← false
   for i = 1 to n - 1 do
      // assuming this pair is out of order
      if A[i-1] > A[i] then
         // swap them and remember something changed
         swap(A[i-1], A[i])
         s ← true
      endif
   endfor
until ¬s
```

3.4.2 Draft Mode

Automatic syntax highlighting is a somewhat expensive operation as it requires several rounds of regular expression matching. In order to speed up compilation the pseudocode command will not attempt automatic highlighting when the document is in draft mode. When in draft mode and you want to force a specific instance of \pseudocode to render the code with automatic syntax highlighting you can use the option nodraft.

3.5 Line Numbering

The pseudocode command allows to insert line numbers into pseudocode. You can either manually control line numbering or simply turn on the option “linenumbering”.

22
is generated by

\begin{verbatim}
\procedure {[linenumbering]} {indcpa\_enc \_adv}\{%
  b \sample \bin \%
  (pk, sk) \sample \KGen{\secparam} \%
  (m_0, m_1) \sample \Adv{\secparam, \pk, c} \%
  c \sample \Enc{\pk, m_b} \%
  b' \sample \Adv{\secparam, \pk, c} \%
  \pc\return b = b'
\}
\end{verbatim}

Note how you can use labels such as \label{tmp:line:label2} which now points to line 3.

### 3.5.1 Manually Inserting Line Numbers

In order to manually insert line numbers use the command \pcln.

\begin{verbatim}
\procedure {indcpa\_enc \_adv}\{%
  b \sample \bin \%
  (pk, sk) \sample \KGen{\secparam} \%
  (m_0, m_1) \sample \Adv{\secparam, \pk, c} \%
  c \sample \Enc{\pk, m_b} \%
  b' \sample \Adv{\secparam, \pk, c} \%
  \pc\return b = b'
\}
\end{verbatim}

Note that the label \texttt{tmp:line:label2} now points to line number 3.

### 3.5.2 Start Values

You can specify the start value (-1) of the counter by setting the option “lnstart”.

\begin{verbatim}
\procedure {lnstart=10, linenumbering} {Header} {Body}
\end{verbatim}
3.5.3 Separators

The commands \(\text{\textbackslash prinseparator}\) defines the separator between the pseudocode and the line numbering. By default the left separator is set to (:) colon. Also see Section 5.3.1.

3.6 Subprocedures

The pseudocode package allows the typesetting of sub procedures such as

```
IND-CPA\textsubscript{Enc}

11: \(b \leftarrow \{0,1\}\)
12: \((pk, sk) \leftarrow KGen(1^n)\)
13: \((m_0, m_1) \leftarrow A(1^n, pk)\)
14: \(c \leftarrow \text{Enc}(pk, m_b)\)
15: \(b' \leftarrow A(1^n, pk, c)\)
16: \text{return} \(b = b'\)
```

To create a subprocedure use the \textit{subprocedure} environment. The above example is generated via

```
\procedure{linenumbering}{\textbackslash indcpa\textsubscript{\textbackslash enc}\textbackslash adv}{%}
1 \(b \leftarrow \{0,1\}\) \% 
2 \((pk, sk) \leftarrow KGen(1^n)\) \% 
3 \((m_0, m_1) \leftarrow A(1^n, pk, c)\) \% 
4 \text{return} \(m_0, m_1\) \%
5 \text{return} \(c \leftarrow \text{Enc}(pk, m_b)\) \%
6 \text{return} \(b' \leftarrow A(1^n, pk, c)\) \%
7 \text{return} \(b = b'\) \%
```

Here the \texttt{dbox} command (from the dashbox package) is used to generate a dashed box around the sub procedure.

3.6.1 Numbering in Subprocedures

Subprocedures as normal pseudocode allow you to create line numbers. By default the line numbering starts with 1 in a subprocedure while ensuring that the outer numbering remains intact. Also note that the linenumbering on the outer procedure in the above example is inherited by the subprocedure. For more control, either use manual numbering or set the option “linenumbering=off” on the subprocedure.
IND-CPA\textsuperscript{4}_{\text{Enc}}

1: \( b \leftarrow \{0, 1\} \)
2: \((pk, sk) \leftarrow KGen(1^n)\)
3: \((m_0, m_1) \leftarrow A(1^n, pk, c)\)
   1: Step 1
   2: Step 2
   3: return \( m_0, m_1 \)
4: \( c \leftarrow \text{Enc}(pk, m_b) \)
5: \( b' \leftarrow A(1^n, pk, c) \)
6: return \( b = b' \)

3.7 Stacking Procedures

You can stack procedures horizontally or vertically using the environments “pchstack” and “pcvstack”.

The following example displays two procedures next to one another. As a spacing between two horizontally outlined procedures use \texttt{pchspace} which takes an optional length as a parameter.

IND-CPA\textsuperscript{4}_{\text{Enc}} \hspace{1cm} Oracle \( O \)

1: \( b \leftarrow \{0, 1\} \)
2: \((pk, sk) \leftarrow KGen(1^n)\)
3: \((m_0, m_1) \leftarrow A(1^n, pk)\)
4: \( c \leftarrow \text{Enc}(pk, m_b) \)
5: \( b' \leftarrow A(1^n, pk, c) \)
6: return \( b = b' \)
Similarly you can stack two procedures vertically using the “pcvstack” environment. As a spacing between two vertically stacked procedures use \pcvspace which takes an optional length as a parameter.

\begin{center}
\begin{verbatim}
\begin{pcvstack}
\procedure{$\text{indcpa}^*_{\text{Enc}}$}{\%}
\pcln b \text{sample} \bin \%
\pcln (pk, sk) \text{sample} \KGen(\secparam) \%
\pcln (m_0, m_1) \text{sample} \Adv(O(1^n), pk) \%
\pcln c \text{sample} \Enc(pk, m_b) \%
\pcln b' \text{sample} \Adv(O(1^n), pk, c) \%
\pcreturn b = b'
\end{pcvstack}
\end{verbatim}
\end{center}

Oracle $O$
\begin{center}
\begin{verbatim}
\begin{pcvstack}
\procedure{$\text{Oracle} O$}{\%}
\pcln \text{line 1} \\
\pcln \text{line 2}
\end{pcvstack}
\end{verbatim}
\end{center}

Oracle $H_1$
\begin{center}
\begin{verbatim}
\begin{pcvstack}
\procedure{$\text{Oracle} H_1$}{\%}
\pcln \text{line 1} \\
\pcln \text{line 2}
\end{pcvstack}
\end{verbatim}
\end{center}

Oracle $H_2$
\begin{center}
\begin{verbatim}
\begin{pcvstack}
\procedure{$\text{Oracle} H_2$}{\%}
\pcln \text{line 1} \\
\pcln \text{line 2}
\end{pcvstack}
\end{verbatim}
\end{center}

Horizontal and vertical stacking can be combined

\begin{center}
\begin{verbatim}
\textbf{IND-CPA}^*_{\text{Enc}}
\begin{array}{llllllllll}
1: & b \leftarrow \{0, 1\} \\
2: & (pk, sk) \leftarrow \KGen(1^n) \\
3: & (m_0, m_1) \leftarrow \Adv(O(1^n), pk) \\
4: & c \leftarrow \Enc(pk, m_b) \\
5: & b' \leftarrow \Adv(O(1^n), pk, c) \\
6: & \text{return } b = b'
\end{array}
\end{verbatim}
\end{center}
3.8 Divisions and Linebreaks

Within the pseudocode command you generate linebreaks as \\ . In order to specify the linewidth you can add an optional argument

Furthermore, you can add, for example a horizontal line by using the second optional argument and write

\begin{center}
\begin{tabular}{l}
\textbf{IND-CPA}_{\text{Enc}}$
\end{tabular}
\end{center}

\begin{itemize}
\item 1: \( b \leftarrow \{0, 1\} \)
\item 2: \( (pk, sk) \leftarrow KGen(1^n) \)
\item 3: \( (m_0, m_1) \leftarrow \mathcal{A}^O(1^n, pk) \)
\item 4: \( c \leftarrow \text{Enc}(pk, m_a) \)
\item 5: \( b' \leftarrow \mathcal{A}(1^n, pk, c) \)
\item 6: \( \text{return } b = b' \)
\end{itemize}
3.9 Matrices and Math Environments within pseudocode

In order to work its magic, cryptocode (in particular within the \texttt{pseudocode} command) mingles with a few low level commands such as \texttt{\\ or \halign}. The effect of this is, that when you use certain math environments, for example, to create matrices, within pseudocode the result may be unexpected. Consider the following example

\begin{verbatim}
\texttt{pseudocode}{
A \halign B + C
}\end{verbatim}

which, somewhat unexpectedly, yields

\[
\text{compute } P = \begin{pmatrix} A \\ B + C \end{pmatrix}
\]

That is, the alignment is somewhat off. In order to allow for the \texttt{pmatrix} environment to properly work without interference from \texttt{pseudocode} you can wrap it into a \texttt{pcmbox} environment (where \texttt{pcmbox} is short for pseudocode math box). This ensures that the low-level changes introduced by \texttt{pseudocode} are not active.

\begin{verbatim}
\texttt{pseudocode}{
\texttt{pcmbox}{
A \halign B + C
}\end{verbatim}

3.10 Fancy Code with Overlays

Consider the IND-CPA game. Here we have a single adversary $A$ that is called twice, first to output two messages then given the ciphertext of one of the messages to “guess” which one was encrypted. Often this is not visualized. Sometimes an additional state $\text{state}$ is passed as we have in the following example on the left. On the right, we visualize the same thing in a bit more fancy way.

\begin{verbatim}
\begin{procedure}$\texttt{indcpa}_A^{\text{Enc}}$
\begin{itemize}
1: $b \leftarrow \{0,1\}$
2: $(pk, sk) \leftarrow \text{KGen}(1^n)$
3: $(state, m_0, m_1) \leftarrow A(1^n, pk, c)$
4: $c \leftarrow \text{Enc}(pk, m_0)$
5: $b' \leftarrow A(1^n, pk, c, state)$
6: \text{return } b = b'$
\end{itemize}
\end{procedure}
\end{verbatim}

IND-CPA$_A^{\text{Enc}}$

\begin{verbatim}
\begin{procedure}$\texttt{indcpa}_A^{\text{Enc}}$
\begin{itemize}
1: $b \leftarrow \{0,1\}$
2: $(pk, sk) \leftarrow \text{KGen}(1^n)$
3: $(m_0, m_1) \leftarrow A(1^n, pk, c)$
4: $c \leftarrow \text{Enc}(pk, m_0)$
5: $b' \leftarrow A(1^n, pk, c, state)$
6: \text{return } b = b'$
\end{itemize}
\end{procedure}
\end{verbatim}

The image on the right is generated by:

\begin{verbatim}
\begin{pcimage}
\begin{pcode}
\begin{adv}$\texttt{indcpa}_A^{\text{Enc}}$\end{adv}$\end{pcode}
\end{pcimage}
\end{verbatim}
In order to achieve the above effect cryptocode utilizes TIKZ underneath. The `pcnode` command generates TIKZ nodes and additionally we wrapped the pseudocode (or procedure) command in an `\begin{pcimage}\end{pcimage}` environment which allows us to utilize these nodes later, for example using the `pcdraw` command. We can achieve a similar effect without an additional `pcimage` environment as
3.10.1 Example: Explain your Code

As an example of what you can do with this, let us put an explanation to a line of the code.

\[
\text{IND-CPA}^A_{\text{Enc}}
\]

1. \( b \leftarrow \{0, 1\} \)
2. \((pk, sk) \leftarrow \text{KGen}(1^n)\)
3. \((m_0, m_1) \leftarrow \mathcal{A}(1^n, pk, c)\)
4. \(c \leftarrow \text{Enc}(pk, m_b)\)
5. \(b' \leftarrow \mathcal{A}(1^n, pk, c, \text{state})\)
6. \text{return } b = b'\)

\text{KGen}(1^n) \text{ samples a public key } pk \text{ and a private key } sk.\)
Chapter 4

Tabbing Mode

In the following chapter we discuss how to create multiple columns within a pseudocode command. Within a pseudocode command you can switch to a new column by inserting a `\>`. This is similar to using an align environment and placing a tabbing character (&). Also, similarly to using align, you should ensure that the number of `\>` are identical on each line.

\[
\textbf{First} \quad \textbf{Second} \quad \textbf{Third} \quad \textbf{Fourth} \\
b \rightarrow \{0,1\} b \rightarrow \{0,1\} b \rightarrow \{0,1\} b \rightarrow \{0,1\}
\]

As you can see the first column is left aligned the second right, the third left and so forth. In order to get only left aligned columns you could thus simply always skip a column by using `\>`. You can also use `\<` a shorthand for `\>\>`.

\[
\textbf{First} \quad \textbf{Second} \quad \textbf{Third} \quad \textbf{Fourth} \\
b \rightarrow \{0,1\} b \rightarrow \{0,1\} b \rightarrow \{0,1\} b \rightarrow \{0,1\}
\]

Column Spacing  You can control the space between columns using the option “colsep=2em”. Note that when doing so you should additionally use “addtolength=5em” (where 5em depends on the number of columns) in order to avoid having overfull hboxes.

\[
\textbf{First} \quad \textbf{Second} \quad \textbf{Third} \quad \textbf{Fourth} \\
b \rightarrow \{0,1\} b \rightarrow \{0,1\} b \rightarrow \{0,1\} b \rightarrow \{0,1\}
\]

This is basically all you need to know in order to go on to writing protocols with the cryptocode package. So unless you want to know a bit more about tabbing (switching columns) and learn some of the internals, feel free to proceed to Chapter 5.
4.1 Tabbing in Detail

At the heart of the pseudocode package is an align (or rather a flalign*) environment which allows you to use basic math writing. Usually an align (or flalign) environment uses & as tabbing characters. The pseudocode comes in two modes the first of which changes the default align behavior. That is, it automatically adds a tabbing character to the beginning and end of each line and changes the tabbing character to \>. This mode is called mintabmode and is active by default.

In mintabmode in order to make use of extra columns in the align environment (which we will use shortly in order to write protocols) you can use \> as you would use & normally. But, don’t forget that there is an alignment tab already placed at the beginning and end of each line. So the following example

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b \leftarrow {0, 1}$</td>
<td>\text{send over $b$}</td>
</tr>
<tr>
<td></td>
<td>\text{do something}</td>
</tr>
</tbody>
</table>

is generated by

\begin{pseudocode}
\textbf{Alice} & $b \leftarrow \{0, 1\}$ & \textbf{Bob} & \text{send over $b$} & \text{do something}
\end{pseudocode}

In Chapter 5 we’ll discuss how to write protocols in detail. The next two sections are rather technical, so feel free to skip them.

4.1.1 Overriding The Tabbing Character

If you don’t like \> as the tabbing character you can choose a custom command by overwriting \pctabname. For example

\begin{pseudocode}
\textbf{Alice} & $b \leftarrow \{0, 1\}$ & \textbf{Bob} & \text{send over $b$} & \text{do something}
\end{pseudocode}

\begin{pseudocode}
\textbf{Alice} & \textbf{Bob} & \text{send over $b$} & \text{do something}
\end{pseudocode}

4.1.2 Custom Line Spacing and Horizontal Rules

As explained underlying the pseudocode command is an flalign environment. This would allow the use of [\[\text{spacing}\]] to specify the spacing between two lines or of [\[\text{hline}\]] to insert a horizontal rule. In order to achieve the same effect within the pseudocode command you can use [\[\text{spacing}\][\text{hline}]]. You can also use \textbf{pctb} to get a line break which does not insert the additional alignment characters.
Chapter 5

Protocols

The pseudocode package can also be used to write protocols such as

\begin{procedure}{My Protocol}
% Alice & Bob
\begin{align*}
b & \leftarrow \{0,1\} \\
\text{send over } b & \\
\text{do something} & \\
\text{send over sth. else} & \\
\text{finalize} & \\
\end{align*}
\end{procedure}

which uses the tabbing feature of align and is generated as

In order to get nicer message arrows use the commands \texttt{sendmessage[message]} and \texttt{sendmessageleft[message]}. Both take an additional optional argument specifying the length of the arrow and both are run in math mode.

\begin{align*}
\texttt{sendmessage[3.5cm][message]} \\
\texttt{sendmessageleft[3.5cm][message]}
\end{align*}
Besides the starred version there is also the unstarred version which allows more flexibility. Note that a crucial difference between the starred and unstarred versions are that `\sendmessageleft{message}` wraps an aligned environment around the message.

My Protocol

\[
\begin{array}{c|c}
\textbf{Alice} & \textbf{Bob} \\
\hline
b \leftarrow \{0,1\} & \\
\hline
\end{array}
\]

\begin{tikzpicture}
\node (a) at (0,0) {send over $b$};
\node (b) at (1,0) {do something};
\draw[thick,dashed] (a) -- (b);
\node [below] at (a.south) {Text below};
\node [below] at (b.south) {send over sth. else};
\end{tikzpicture}

finalize

The unstarred commands take key-value pairs. The following keys are available:

- **top** The content to display on top of the arrow.
- **bottom** The content to display below the arrow.
- **left** The content to display on the left of the arrow.
- **right** The content to display on the right of the arrow.
- **topstyle** The TIKZ style to be used for the top node.
- **bottomstyle** The TIKZ style to be used for the bottom node.
- **rightstyle** The TIKZ style to be used for the right node.
- **leftstyle** The TIKZ style to be used for the left node.
- **length** The length of the arrow.
- **style** The style of the arrow.
- **width** The width of the column

`centercol` Can be used to ensure that the message is displayed in the center. This should be set to the column index. In the above example, the message column is the third column (note that there is a column left of Alice that is automatically inserted.).
5.1 Tabbing

When typesetting protocols you might find that using two tabs instead of a single tab usually provides a better result as this ensures that all columns are left aligned. For this you can use `<` instead of `>` (see Chapter 4).

Following is once more the example from before but now with double tapping.

<table>
<thead>
<tr>
<th>My Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
</tr>
<tr>
<td>$b \leftarrow {0, 1}$</td>
</tr>
</tbody>
</table>
|              | $\begin{array}{|c|c|}
|              | $send over b$
|              | \hline
|              | $do something$
|              | \hline
|              | $send over sth. else$
|              | \hline
|              | finalize    |

5.2 Multiline Messages

Using the send message commands you can easily generate multiline messages as the command wraps an \textit{aligned} environment around the message.

<table>
<thead>
<tr>
<th>My Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
</tr>
<tr>
<td>$b \leftarrow {0, 1}$</td>
</tr>
</tbody>
</table>
|              | $\begin{array}{|c|c|}
|              | $send over b$
|              | \hline
|              | $second line$
|              | \hline
|              | $do something$
|              | \hline
|              | left aligned
|              | \hline
|              | multiline message
|              | \hline
|              | finalize    |

5.2.1 Multiplayer Protocols

You are not limited to two players. In order to send messages skipping players use `<sendmessagerightx>` and `<sendmessageleftx>`.

1 \procedure{My Protocol}{% 
2 $\textbf{Alice}$ \textbf{Bob} \\ 
3 $b \leftarrow \{0, 1\}$ & $\begin{array}{|c|c|}
4 | $send over b$
5 | \hline
6 | $do something$
7 | \hline
8 | $send over sth. else$
9 | \hline
10 | finalize    |
11

35
Note that for the last message from Charlie to Alice we needed to specify the number of passed over columns (\text{sendmessageleftx[7cm]{message}}). As we were passing 4 \textless where each creates 2 columns, the total was 8 columns.

### 5.2.2 Divisions
You can use \texttt{\textbackslash pcintertext} in order to divide protocols (or other pseudocode for that matter).

Note that in order to use \texttt{\textbackslash pcintertext} you need to use \texttt{\textbackslash pclb} as the line break for the line before. Also see Chapter 4.
5.3 Line Numbering in Protocols

Protocols can be numbered similarly to plain pseudocode. Additionally to the \pcln there are the commands \pcnr and \pcrln. The first allows you to right align line numbers but uses the same counter as \pcln. The second uses a different counter.

Which is generated as

And using \pcrln:

Which is generated as
5.3.1 Separators

The commands `\pclnseparator` and `\pcrlnseparator` define the separators between the pseudocode and line numbering. By default the left separator is set to (:) colon and the right separator is set to a space of 3 pt.

5.4 Sub Protocols

Use the “subprocedure” function also to create sub protocols.

```
\procedure{My Protocol}{%}
\textbf{Alice} \textbf{Bob}

\begin{subprocedure}{Subprotocol}
\begin{itemize}
  \item do something
  \item \bf{Charlie}
    \begin{itemize}
      \item something more
      \item message
      \item some processing
      \item more processing
      \item message
      \item message
      \item message
    \end{itemize}
  \end{itemize}
\\
\end{subprocedure}

send over sth. else
```

```
\begin{procedure}{My Protocol}{%}
\begin{itemize}
  \item Alice \textbf{Bob}
  \item b \leftrightarrow \{0, 1\}
  \item send over b
  \item do something
  \item Subprotocol
    \begin{itemize}
      \item Charlie
        \begin{itemize}
          \item something more
          \item message
          \item some processing
          \item more processing
          \item message
          \item message
          \item message
        \end{itemize}
    \end{itemize}
  \end{itemize}
\\
\end{procedure}
```
Chapter 6

Game Based Proofs

6.1 Basics

Besides displaying pseudocode the package also comes with commands to display game based proofs. A proof is wrapped in the \texttt{gameproof} environment.

\begin{verbatim}
\begin{gameproof}
proof goes here
\end{gameproof}
\end{verbatim}

Within the proof environment you can use the command \texttt{gameprocedure} which works similarly to the pseudocode command and produces a heading of the form Game\textsubscript{counter} where counter is a consecutive counter. Thus, we can create the following setup

\[
\begin{array}{c@{}c}
\text{Game}_1(n) & \text{Game}_2(n) \\
1: & 1: \\
2: & 2:
\end{array}
\]

by using

\begin{verbatim}
\begin{gameproof}
\gameprocedure[linenumbering,mode=text]{% 
Step 1 \ \
Step 2
} 
\gameprocedure[mode=text]{% 
Step 1 \ \
Step 2
} 
\end{gameproof}
\end{verbatim}

6.1.1 Highlight Changes

In order to highlight changes from one game to the next use \texttt{gamechange}.

\[
\begin{array}{c@{}c}
\text{Game}_1(n) & \text{Game}_2(n) \\
1: & 1: \\
2: & 2:
\end{array}
\]

\begin{verbatim}
\begin{gameproof}
\gameprocedure[linenumbering,mode=text]{% 
Step 1 \ \
Step 2
} 
\end{gameproof}
\end{verbatim}
6.1.2 Boxed games

Use `\tbxgameprocedure` in order to create two consecutive games where the second game is boxed. Use `\pcbox` to create boxed statements.

<table>
<thead>
<tr>
<th>Game₁(n)</th>
<th>Game₂(n)</th>
<th>Game₃(n)</th>
<th>Game₄(n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 1</td>
<td>Step 1;</td>
<td>Alternative step 1</td>
<td>Step 1</td>
</tr>
<tr>
<td>Step 2</td>
<td>Step 2</td>
<td>Step 2 is different</td>
<td>Step 2</td>
</tr>
</tbody>
</table>

6.1.3 Reduction Hints

In a game based proof in order to go from one game to the next we usually give a reduction, for example, we show that the difference between two games is bound by the security of some pseudorandom generator PRG. To give a hint within the pseudocode that the difference between two games is down to “something” you can use the `\addgamehop` command.

Here options allows you to specify the hint as well as the style. The following options are available:

- **hint** The hint text
- **nodestyle** A TIKZ style to be used for the node.
- **pathstyle** A TIKZ style to be used for the path.
- **edgestyle** A TIKZ style to be used for the edge. This defaults to “bend left”.

```latex
\addgamehop{startgame}{endgame}{options}
```
The edgestyle allows you to specify how the hint is displayed. If you, for example want a straight line, rather than the curved arrow simply use

```latex
\addgamehop{1}{2}{hint=\footnotesize some hint, edgestyle=}
```

If game proofs do not fit into a single picture you can specify start and end hints using the commands

```latex
\addstartgamehop{first game}{options}
\addendgamehop{last game}{options}
```

![Diagram](image)

### 6.1.4 Numbering and Names

By default the `gameproof` environment starts to count from 1 onwards. Its optional parameters allow you to specify a custom name for your game and the starting number.

```latex
\begin{gameproof}\[nr=5,name=$\textstyle{\mathsf{MyGame}}_{\mathcal{N}}$(1^n)$, arg=(1^n)\]
```
6.1.5 Default Name and Argument

The default name and argument are controlled via the commands \pgame\text{\texttt{\textbackslash name}} and \pgame\text{\texttt{\textbackslash arg}}.

<table>
<thead>
<tr>
<th>Command</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>\pgame</td>
<td>\texttt{\textbackslash name}</td>
</tr>
<tr>
<td>\pgame</td>
<td>\texttt{\textbackslash arg}</td>
</tr>
</tbody>
</table>

6.1.6 Two Directional Games

You can use the \pgame\text{\texttt{\textbackslash bxgame\textbackslash procedure}} to generate games for going in two directions. Use the \pgame\text{\texttt{\textbackslash addloop\textbackslash game\textbackslash hop}} to add the gamehop in the middle.

\begin{gameproof}
    \setgameproceduredefaultstyle{\begininline=vphantom{\bin^\prg_\prg}}
    \begin{proof}
        \bxgameprocedure[4]{\texttt{\begininline}}; \pcm{\texttt{Alternative}}; \pcm{\texttt{Step 2}}
        \bxgameprocedure[3]{\texttt{\textbackslash text{\textbackslash Step 1}}}; \pcm{\texttt{Alternative}}; \texttt{\textbackslash gamechange{\texttt{\textbackslash text{\textbackslash Step 2}}}}
        \addstartgamehop{\texttt{\footnotesize some hint}}; \addgamehop{\texttt{\footnotesize some hint}}; \addloopgamehop{\texttt{\footnotesize some loop hint}}; \addgamehop{\texttt{\footnotesize some hint}}
    \end{proof}
\end{gameproof}

6.1.7 Styling game procedures

It may come in handy to define default style arguments for the underlying pseudocode command used by \pgame\text{\texttt{\textbackslash game\textbackslash procedure}}. For this you can define the default arguments by calling \pgame\text{\texttt{\setgameproceduredefaultstyle}} to for example:

\begin{gameproof}
    \setgameproceduredefaultstyle{\begininline=vphantom{\bin^\prg_\prg}}
    \begin{proof}
        \bxgameprocedure[4]{\texttt{\textbackslash text{\textbackslash Step 1}}}; \pcm{\texttt{Alternative}}; \pcm{\texttt{Step 2}}
        \bxgameprocedure[3]{\texttt{\textbackslash text{\textbackslash Step 1}}}; \pcm{\texttt{Alternative}}; \texttt{\textbackslash gamechange{\texttt{\textbackslash text{\textbackslash Step 2}}}}
        \addstartgamehop{\texttt{\footnotesize some hint}}; \addgamehop{\texttt{\footnotesize some hint}}; \addloopgamehop{\texttt{\footnotesize some loop hint}}; \addgamehop{\texttt{\footnotesize some hint}}
    \end{proof}
\end{gameproof}

The default is to not set any options.
6.2 Game Descriptions

Cryptocode also comes with an environment to provide textual descriptions of games such as

\[ \text{MyGame}_3(n) \]: This is the third game. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis condimentum velit et orci volutpat, sed ultrices lorem lobortis. Nam vehicula, justo eu varius interdum, felis mi consectetur dolor, ac posuere nulla lacus varius diam. Etiam dapibus blandit leo, et porttitor augue lacinia auctor.

\[ \text{MyGame}_4(n) \]: This is the second game. The arrow at the side indicates the reduction target.

The above example is generated as

\begin{verbatim}
\begin{gamedescription}
\name{MyGame}, \nr=2
\end{gamedescription}
\describegame
This is the third game. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis condimentum velit et orci volutpat, sed ultrices lorem lobortis. Nam vehicula, justo eu varius interdum, felis mi consectetur dolor, ac posuere nulla lacus varius diam. Etiam dapibus blandit leo, et porttitor augue lacinia auctor.
\describegame[\inhint=reduction target]
This is the second game. The arrow at the side indicates the reduction target.
\end{gamedescription}
\end{verbatim}

The \texttt{gamedescription} environment takes an optional argument to specify name and counter (defaults to \texttt{Game} and 0). The command \texttt{\describegame} starts a new game description and can allows you to provide a reduction hint using the option parameter \texttt{inhint}.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\inhint</td>
<td>Displays an ingoing arrow to denote the reduction target for this game hop.</td>
</tr>
<tr>
<td>\length</td>
<td>Allows to control the length of the arrow.</td>
</tr>
<tr>
<td>\nodestyle</td>
<td>Allows to control the style of the node.</td>
</tr>
<tr>
<td>\hint</td>
<td>Instead of having an ingoing arrow, this adds an outgoing arrow.</td>
</tr>
</tbody>
</table>
Chapter 7

Black-box Reductions

The cryptocode package comes with support for drawing basic black box reductions. A reduction is always of the following form.

\begin{bbrenv}{A}
\begin{bbrbox}[name=Box Name]
% The Box’s content
\end{bbrbox}
% Commands to display communication, input output etc
\end{bbrenv}

That is, a “brenv” (where bbr is short for black-box reduction) environment which takes a single “bbrbox” environment and some additional commands.

The following is a simple example drawing one (black)box with some code and input output:

input

\begin{tikzpicture}
\node[draw,align=center,inner sep=3pt] (box) at (0,0) {Box Name};
\node[align=center,inner sep=3pt] at (box.north east) {step 1};
\node[align=center,inner sep=3pt] at (box.north west) {step 2};
\node[align=center,inner sep=3pt] at (box.south east) {step 3};
\node[align=center,inner sep=3pt] at (box.south west) {\textbf{for some condition \textbf{do}}};
\draw[->] (box.west) -- ++(-2,0) node[align=center,inner sep=3pt] {\textbf{input}};
\draw[->] (box.east) -- ++(2,0) node[align=center,inner sep=3pt] {\textbf{output}};
\end{tikzpicture}

This box is generated as

\begin{verbatim}
\begin{bbrenv}[1cm]{A}[1cm]
\begin{bbrbox}[name=Box Name]
\pseudocode
\text{step 1} \\
\text{step 2} \\
\text{for some condition \textbf{do}} \\
\text{step 3}
\end{bbrbox}
\bbrinput{input}
\bbroutput{output}
\end{bbrenv}
\end{verbatim}

The commands \texttt{bbrinput} and \texttt{bbroutput} allow to specify input and output for the latest “bbrenv” environment. The optional parameters for the \texttt{bbrenv} environment allow to specify leading and trailing space (this may become necessary when using inputs and outputs). The single argument to the \texttt{bbrenv}
environment needs to specify a unique identifier (unique for the current reduction). This id is used as an internal TIKZ node name \texttt{https://www.ctan.org/pkg/pgf}.

As we are drawing a TIKZ image, note that we can easily later customize the image using the labels that we have specified on the way.

The “bbrbox” takes as single argument a comma separated list of key value pairs. In the example we have used \texttt{name=Box Name} to specify the label. The following options are available

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Specifies the box’s label</td>
</tr>
<tr>
<td>namepos</td>
<td>Specifies the position (left, center, right, top left, top center, top right, middle)</td>
</tr>
<tr>
<td>namestyle</td>
<td>Specifies the style of the name</td>
</tr>
<tr>
<td>abovesep</td>
<td>Space above box (defaults to \baselineskip)</td>
</tr>
<tr>
<td>minheight</td>
<td>The minimal height</td>
</tr>
<tr>
<td>addheight</td>
<td>Additional height at the end of the box</td>
</tr>
<tr>
<td>xshift</td>
<td>Allows horizontal positioning</td>
</tr>
<tr>
<td>yshift</td>
<td>Allows horizontal positioning</td>
</tr>
<tr>
<td>style</td>
<td>allows to customize the node</td>
</tr>
</tbody>
</table>
7.1 Nesting of Boxes

Boxes can be nested. For this simply insert a `bbrenv` (together with a single `bbrbox`) environment into an existing `bbrbox`.

```
\begin{bbrenv} \{A\}
  \begin{bbrbox} \{name=Box Name\}
    \text{step 1} \\
    \text{step 2} \\
    \textbf{for some condition do} \\
    \text{step 3}
  \end{bbrbox}
\end{bbrenv}
```

```
\begin{bbrenv} \{B\}
  \begin{bbrbox} \{name=Inner Box\}
    \text{inner step 1} \\
    \text{inner step 2}
  \end{bbrbox}
\end{bbrenv}
```

```
\begin{bbrenv} \{\}
  \begin{pseudocode}
    \text{step 4} \\
    \text{step 5}
  \end{pseudocode}
\end{bbrenv}
```

7.2 Messages and Queries

You can send messages and queries to boxes. For this use the commands

```
\bbrmgsto\{options\}
\bbrmgfrom\{options\}
\bbrqueryto\{options\}
\bbrqueryfrom\{options\}
```
By convention messages are on the left of boxes and queries on the right. Commands ending on to make an arrow to the right while commands ending on from make an arrow to the left. The options define how the message is drawn and consists of a key-value pairs separated by ",".

For example, to draw a message with a label on top and on the side use

\begin{bbrmsgto}{top=Top Label, side=Side Label}

If your label contains a "," (comma), then group the label in \{\} (curly brackets).

\begin{bbrmsgto}{top=Top Label, side={Side, Label}}

Following is a complete example. Notice that cryptocode takes care of the vertical positioning.

\begin{verbatim}
\begin{bbrenv}{A}
\begin{bbrbox}[name=Box Name]
pseudocode{
\text{step 1} \\
\text{step 2} \\
\textbf{for} some condition \textbf{do} \\
\text{step 3} \\
\quad m_0, m_1 \\
inner step 1 \\
inner step 2 \\
q \quad q \\
\text{step 4} \\
\text{step 5}
}
\end{bbrbox}
\bbrmsgto{top={$m_0, m_1$}}
\bbrmsgfrom{top=$q$}
\end{bbrenv}

\begin{bbrenv}{B}
\begin{bbrbox}[name=Inner Box]
pseudocode{
\text{inner step 1} \\
\text{inner step 2}
}
\end{bbrbox}
\bbrmsgto{top={$m_0, m_1$}}
\bbrmsgfrom{top=$q$}
\end{bbrenv}
\end{verbatim}

\end{verbatim}
7.2.1 Options

Besides specifying labels for top, side and bottom you can further specify how cryptocode renders the message. Remember that underneath the reduction commands is a TIKZ image (https://www.ctan.org/pkg/pgf). For each label position (top, side, bottom) a node is generated. You can provide additional properties for this node using the options:

- topstyle
- sidestyle
- bottomstyle

You can additionally provide custom names for the nodes for later reference using:

- topname
- sidename
- osidename
- bottomname

The “osidename” allows you to provide a name for the “other side”. Via the option “length” you can specify the length of the arrow.

\begin{bbenv}{A}
\begin{bbrbox} [name=Box Name]
\pscode{
    \text{step 1} \\
    \text{step 2} \\
    \textbf{for some condition} \textbf{do} \\
    \text{step 3} \\
    \textbf{inner step 1} \\
    \textbf{inner step 2}
}

\end{bbrbox}
\end{bbenv}
7.2.2 Add Space

If the spacing between messages is not sufficient you can use the `bbrmsgspace` and `bbrqryspsace` commands to add additional space.

```
\begin{bbrenv}{B}
\begin{bbrbox}{name=Inner Box}
\begin{pseudocode}
\text{inner step 1} \ \ \ \ \\
\text{inner step 2} \ \ \\
\end{pseudocode}
\end{bbrbox}
\bbrmsgto{top=\{m_0, m_1\}, side=Side Label, bottom=\$b\$, length=2cm, tostyle={draw, solid}, sidestyle={red}, bottomstyle={draw, dashed}}
\end{bbrenv}
```

```latex
\begin{pseudocode}
\text{step 4} \\
\text{step 5} \\
\end{pseudocode}
```

```
\end{bbrenv}
```

```
\begin{bbrenv}{A}
\begin{bbrbox}{name=Reduction}
\begin{pseudocode}
\text{Do something}
\end{pseudocode}
\end{bbrbox}
\begin{bbrbox}{name=Adversary, minheight=3cm, xshift=4cm}
\bbrmsgto{top=\$m\$}
\bbrmsgspace{1.5cm}
\bbrmsgfrom{top=\$\sigma\$}
\bbrqryspsace{1cm}
\bbrqryfrom{top=\$\sigma\$}
\end{bbrbox}
\end{bbrenv}
```
7.2.3 Loops

Often an adversary may send poly many queries to an oracle, or a reduction sends many queries to an adversary. Consider the following setting:

```
\begin{bbrenv}{A}
  \begin{bbrbox}[name=Reduction]
    \pseudocode{
      \text{Do something}
    }
  \end{bbrbox}
  \bbrloop{BeginLoop}{EndLoop}{center=$q$}
\end{bbrenv}
```

First note that by specifying the minheight and xshift option we shifted the adversary box a bit to the right and enlarged its box. Further we specified custom names for the node on the side of the two messages. We can now use the \texttt{bbrloop} command to visualize that these two messages are exchanged \( q \) many times.

```
\begin{bbrenv}{B}
  \begin{bbrbox}[name=Adversary, minheight=3cm, xshift=4cm]
    \bbrmsgto{top=$m$}
    \bbrmsgfrom{top=$\sigma$}
  \end{bbrbox}
\end{bbrenv}
```

The \texttt{bbrloop} command takes two node names and a config which allows you to specify if the label is to be shown on the left, center or right. Here is the result.

```
7.2.4 Intertext

If your reduction needs to do some extra work between queries use the \bbrmsgtxt and \bbrqrytxt commands.
7.3 Oracles

Each box can have one or more oracles which are drawn on the right hand side of the box. An oracle is created similarly to a \texttt{bbrenv} environment using the \texttt{bbroracle} environment. Oracles go behind the single \texttt{bbbox} environment within an \texttt{bbrenv} environment.
Via the option “\texttt{hdistance=length}” and “\texttt{vdistance=length}” you can control the horizontal and vertical position of the oracle. By default this value is set to 1.5cm and \texttt{baselineskip}.

### 7.3.1 Communicating with Oracles

As oracles use the \texttt{bbrbox} environment we can directly use the established ways to send messages and queries to oracles. In addition you can use the \texttt{bbroracleqryfrom} and \texttt{bbroracleqyroto}.

Here options allow you to specify where the label goes (top, bottom). In addition you can use \texttt{bbroracleqrysae} to generate extra space between oracle messages. Note that oracle messages need to be added after the closing \texttt{end\{bbroracle\}} command.
7.4 Challengers

Each box can have one or more challengers which are drawn on the left hand side of the box. Challengers behave identically to oracles with the exception that they are to the left of the box. A challenger is created similarly to a \textit{bbrenv} environment using the \textit{bbrchallenger} environment. Challengers go behind the single \textit{bbrbox} environment within an \textit{bbrenv} environment.
Via the option “hdistance=length” and “vdistance=length” you can control the horizontal and vertical position of the challenger. By default this value is set to 1.5cm and \baselineskip.

### 7.4.1 Communicating with Challengers

As challengers use the \bbrbox environment we can directly use the established ways to send messages and queries to oracles. In addition you can use the \bbrchallengerqryfrom and \bbrchallengerqryto.

Here options allow you to specify where the label goes (top, bottom). In addition you can use \bbrchallengerqryspace to generate extra space between oracle messages. Note that challenger messages need to be added after the closing \end{bbrchallenger} command.
7.5 Examples

A reduction for full domain hash.

$$j \leftarrow [q]$$

/* begin simulation */

$$A_{f_k}^\sigma m_1$$  

\[ ... \]

$$m_j$$  

\[ ... \]

$$m_{j-1}$$  

$$m_j$$  

\[ ... \]

$$y$$  

$$m_{j+1}$$  

\[ ... \]

$$m_q$$

/* end simulation */

$$y \leftarrow \sigma$$

\[ y \in F^{-1}(f_k, x) \]

\begin{verbatim}
\begin{bbrenv}{Red}
  \begin{bbrbox}[name=\textsc{Reduction }$B$\textsc{dv}]
    \pseudocode{
      \begin{array}{c}
        j \leftarrow [q] \\
        \text{/* begin simulation */} \\
        m_1 \\
        \vdots \\
        m_{j-1} \\
        \vdots \\
        m_j \\
        \vdots \\
        m_q \\
        \text{/* end simulation */} \\
        y \leftarrow \sigma
      \end{array}
    \end{bbrbox}
  \end{bbrenv}
\end{verbatim}
Chapter 8

Known Issues

8.1 Pseudocode KeepSpacing within Commands

The “space=keep” option of pseudocode which should output spacing identical to that of the input will fail, if the pseudocode command is called from within another command. An example is to wrap the \pseudocode command with an \fbox. As a workaround for generating frame boxes you should hence use a package such as mdframed [https://www.ctan.org/pkg/mdframed] which provides a frame environment.

\begin{tabular}{|l|}
\hline
Pseudocode & with & - spaces - \\
\hline
\end{tabular}

\begin{verbatim}
\pseudocode [space=keep, mode=text]{
Pseudocode with - spaces -
}
\end{verbatim}

As an alternative you could use a \savebox (in combination with the \lrbox environment):

\begin{verbatim}
\newsavebox{\mybox}
\begin{lrbox}{\mybox}%
\pseudocode [space=keep, mode=text]{
Pseudocode with - spaces -}%
\end{lrbox}
\fbox{\usebox{\mybox}}
\end{verbatim}

8.2 AMSFonts

Some packages are not happy with the “amsfonts” package. Cryptocode will attempt to load amsfonts if it is loaded with either the “sets” or the “probability” option. In order to not load amsfonts you can additionally add the “noamsfonts” at the very end. Note that in this case you should ensure that the command \mathbb is defined as this is used by most of the commands in “sets” and some of the commands in “probability”.

8.3 Hyperref

The hyperref package [https://www.ctan.org/pkg/hyperref] should be loaded before cryptocode. If this is not possible call the \pcfixhyperref after \begin{document}.
Index

\&. 32
\<. 32
\>. 32
\addgamehop. 42
\addloopgamehop. 44
\bbrchallengerqryfrom. 57
\bbrchallengerqryto. 57
\bbrinput. 46
\bbroop. 52
\bbrmsgfrom. 48
\bbrmsgspace. 51
\bbrmsgto. 48
\bbrmsgtxt. 53
\bbroracleqryfrom. 55
\bbroracleqryto. 55
\bbroutput. 46
\bbrqueryfrom. 48
\bbrqueryspace. 51
\bbrqueryto. 48
\bbrquerytxt. 53
\bxgameprocedure. 44
\createprocedurecommand. 17
\createpseudocodecommand. 17
\fbox. 60
\gamechange. 41
\highlightkeyword. 21
\hline. 28
\pccomment. 20
\pccontinue. 20
\pcdo. 20
\pcdone. 20
\pcelse. 20
\pcelseif. 20
\pcendif. 20
\pcendwhile. 20
\pcf. 20
\pcfforeach. 20
\pcglobvar. 20
\pcif. 20
\pcin. 20
\pcind. 17
\pcindentname. 18
\pclr. 33
\pcln. 23
\pcnr. 23
\pclnseparator. 25
\pcnew. 20
\pcnull. 20
\pcparse. 20
\pcrepeat. 20
\pcreturn. 20
\pcrln. 23
\pcstmtname. 33
\pcthen. 20
\pctrue. 20
\pcuntil. 20
\pcwhile. 20
\procedure. 17
\pseudocode. 15
\sendmessageleft. 34
\sendmessageleft*. 34
\sendmessageright. 34
\sendmessageright*. 34
\text. 17
\tbxgameprocedure. 42
addkeywords. 15, 21
adttolength. 15, 32
adversary. see package options
altkeywords. 15, 23
amsfonts. 60
asymptotics. see package options
bbrbox. 46
bbrchallenger. 56
bbrenv. 46
bbroracle. 53
beginline. 19
bodylinesep. 15
codesize. 15
colsep. 15, 32
complexity. see package options
Crypto notions. see package options
Crypto primitives. see package options
draft mode. 23
emphasize. see highlight keywords 21
endline. 15
events. see package options
ff. see package options
framebox. 60
function families, see package options

gameprocedure, 41
gameproof, 41
hdistance, 54, 56
head, 15
headlinesep, 15
highlight game change, 41
highlight keywords, 21
indentation, 17
jot, 15
keys, see package options
keywords, 15
keywordsindent, 15
keywordsunindent, 15
keywordsuninindent, 15
Landau, see package options
line numbering, 23
linebreaks, 28
linenumbering, 15
lnstart, 15
lnstartright, 15
logic, see package options
machine model, see package options
math operators, see package options
minheight, 47
mm, see package options
mode, 15
namepos, 47
namestyle, 47
noamsfonts, see package options
nodraft, 15, 23
notions, see package options
operators, see package options
package options, 8
adversary, 9
asymptotics, 14
complexity, 13
events, 13
if, 12
keys, 14
Landau, 9
logic, 11
mm, 12
noamsfonts, 60
notions, 11
operators, 8
primitives, 12
probabilities, 10
security parameter, 8
sets, 11
performance, 29
primitives, see package options
probability, see package options
security parameter, see package options
sets, see package options
space, 15
subprocedure, 25
syntaxhighlight, 15
Tabbing Mode, 32
text mode, 20
vdistance, 54, 56
xshift, 15, 47
yshift, 15, 47