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 typesetting probabilities and “commonly encountered math” as well as for concepts such as a security parameter $1^n$ or advantage terms $\text{Adv}_{\text{PRF}}(n) = \text{negl}(n)$. Furthermore, it includes environments to layout game-based proofs or black-box reductions.

$\begin{array}{c|c}
\text{Alice} & \text{Bob} \\
\hline
x \leftarrow \$ \mathbb{Z}_q & y \leftarrow \$ \mathbb{Z}_q \\
X \leftarrow g^x & Y \leftarrow g^y \\
Y \leftarrow g^\cdot, q, g, X & Y \leftarrow g^\cdot, q, g, X \\
k_A \leftarrow Y^x & k_B \leftarrow X^y
\end{array}$

*If you use cryptocode in your work, consider starring the repository on GitHub and/or rating it on CTAN.
## Contents

### 1 Cryptocode by Example 1

- 1.1 Pseudocode ............................................. 1
- 1.2 Stacking ................................................ 3
- 1.3 Columns ................................................. 4
- 1.4 Protocols ............................................... 5
- 1.5 Game-Based Proofs ..................................... 6
- 1.6 Black-Box Reductions .................................. 6

### 2 Notation Macros 8

- 2.1 Security Parameter .................................... 8
- 2.2 Advantage Terms ...................................... 8
- 2.3 Math Operators ....................................... 9
- 2.4 Adversaries ............................................. 9
- 2.5 Landau .................................................. 10
- 2.6 Probabilities .......................................... 10
- 2.7 Sets ..................................................... 11
- 2.8 Cryptographic Notions ................................ 12
- 2.9 Logic .................................................... 12
- 2.10 Function Families .................................... 13
- 2.11 Machine Model ....................................... 13
- 2.12 Crypto Primitives .................................... 14
- 2.13 Oracles ................................................ 14
- 2.14 Events ................................................ 15
- 2.15 Complexity .......................................... 15
- 2.16 Asymptotics .......................................... 16
- 2.17 Keys ................................................... 16

### 3 Pseudocode 17

- 3.1 Basics ................................................ 17
  - 3.1.1 Customizing Pseudocode .......................... 18
  - 3.1.2 Customized Pseudocode Commands .............. 20
- 3.2 Indentation ........................................... 21
- 3.3 Textmode ............................................. 23
- 3.4 Syntax Highlighting .................................. 24
  - 3.4.1 Automatic Syntax Highlighting (Experimental) 25
- 3.5 Line Numbering ....................................... 28
  - 3.5.1 Skipping Line Numbers ............................ 28
  - 3.5.2 Manually Inserting Line Numbers ............... 28
  - 3.5.3 Start Values ..................................... 29
  - 3.5.4 Separators ....................................... 29
  - 3.5.5 Style ............................................. 29
- 3.6 Subprocedures ........................................ 30
  - 3.6.1 Numbering in Subprocedures .................... 30
- 3.7 Stacking Procedures .................................. 31
  - 3.7.1 Stacking Options ................................ 33
- 3.8 Default Arguments ................................... 33
- 3.9 Divisions and Linebreaks ............................. 34
  - 3.9.1 Optimizing Layout ............................... 35
- 3.10 Matrices and Math Environments within Pseudocode 35
- 3.11 Fancy Code with Overlays ........................... 36
4 Tabbing Mode ............................................. 38
  4.1 Tabbing in Detail ...................................... 38
    4.1.1 Overriding The Tabbing Character ............... 39
    4.1.2 Custom Line Spacing and Horizontal Rules ....... 39

5 Protocols .................................................. 40
  5.1 Tabbing ................................................. 42
  5.2 Multiline Messages .................................... 42
    5.2.1 Multiplayer Protocols ........................... 43
    5.2.2 Divisions ........................................ 43
  5.3 Line Numbering in Protocols ........................ 44
    5.3.1 Separators ........................................ 45
    5.3.2 Spacing .......................................... 45
  5.4 Sub Protocols ......................................... 45

6 Game-Based Proofs ........................................ 47
  6.1 Basics .................................................. 47
    6.1.1 Highlight Changes ................................. 47
    6.1.2 Boxed Games ....................................... 48
    6.1.3 Reduction Hints .................................... 48
    6.1.4 Numbering and Names ............................. 49
    6.1.5 Default Name and Argument ....................... 50
    6.1.6 Bi-Directional Games .............................. 50
    6.1.7 Styling Game Procedures .......................... 51
  6.2 Game Descriptions ...................................... 51

7 Black-Box Reductions ..................................... 53
  7.1 Nesting of Boxes ....................................... 54
  7.2 Messages and Queries .................................. 55
    7.2.1 Options .......................................... 57
    7.2.2 First Message ..................................... 58
    7.2.3 Add Space ......................................... 58
    7.2.4 Loops ............................................ 60
    7.2.5 Intertext ......................................... 62
  7.3 Oracles ................................................ 63
    7.3.1 Communicating with Oracles ....................... 63
  7.4 Challengers ............................................ 64
    7.4.1 Communicating with Challengers .................. 65
  7.5 Examples ............................................... 66

8 Known Issues .............................................. 68
  8.1 Pseudocode KeepSpacing within Commands ............. 68
  8.2 AMSFonts .............................................. 68
  8.3 Hyperref ............................................... 68

9 Implementation ........................................... 69
  9.1 Package Options ....................................... 69
    9.1.1 operators ......................................... 69
    9.1.2 adversary ......................................... 70
    9.1.3 landau ........................................... 70
    9.1.4 probability ....................................... 70
    9.1.5 sets ............................................. 71
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 section we present the various features of cryptocode by giving small examples. But first, let’s load the package

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

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{\pk} or \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 Section 2.

1.1 Pseudocode

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

\begin{verbatim}
\pseudocode[linenumbering]{
\begin{align*}
& 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_b) \\
& 5: \ b' \leftarrow \A(1^n, pk, c, state) \\
& 6: \ \text{return} \ b = b'
\end{align*}
}
\end{verbatim}

\begin{verbatim}
\pseudocode[linenumbering]{
\texttt{b \ sample \ bin } \\
\texttt{(pk, sk) \ sample \ \kgen (secparam) } \\
\texttt{(state, m_0, m_1) \ sample \ \adv (secparam, pk, c) } \\
\texttt{c \ sample \ \enc (pk, m_b) } \\
\texttt{b' \ sample \ \adv (secparam, pk, c, state) } \\
\texttt{\pcreturn \ b = b'}
}\end{verbatim}
In many cases, we want to set pseudocode blocks in-between paragraphs with spacing similar to how we would offset equations. For this, and for laying out multiple code blocks, cryptocode offers “stacking” environments \texttt{pchstack} and \texttt{pcvstack}. For typesetting a code block nicely centered and boxed

\begin{pchstack}
\begin{pseudocode}[linenumbering]
\begin{array}{c}
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: & \text{return } b = b'
\end{array}
\end{pseudocode}
\end{pchstack}

you could thus use:

\begin{lstlisting}
begin{pchstack}[center,boxed]
pseudocode[linenumbering]{
\begin{array}{c}
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: & \text{return } b = b'
\end{array}
}
end{pchstack}
\end{lstlisting}

As this is a common task, cryptocode offers the \texttt{pseudocodeblock} command which is a shorthand for the above (without the frame). In case you want to provide different options or a shorter command (say \texttt{pcb}) you can easily define the command via

\begin{lstlisting}
createpseudocodeblock{pcb}{center,boxed}{}{}{}
\end{lstlisting}

The above could now be written, more succinctly as

\begin{lstlisting}
pcb[linenumbering]{
\begin{array}{c}
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: & \text{return } b = b'
\end{array}
}
\end{lstlisting}

The pseudocode command (and its block variant) 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.

It is easy to define a heading for your code. Either specify the header using the option “head” or use the \texttt{procedure} command (or its block variant \texttt{procedureblock}) which takes an additional argument to specify the headline.

\begin{lstlisting}
IND-CPA^n_{Enc}(n)
begin{array}{c}
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: & \text{return } b = b'
\end{array}
\end{lstlisting}
Similarly to before, we can define a shorthand and boxed variant as

```
\begin{pcimage}
\procedureblock{linenumbering}{{$\text{\textbf{indcpa}}_{\text{\textbf{enc}}}^\text{\textbf{\textbf{adv}}} (\text{\textbf{secpar}})$}}{
  b \sample \bin \\\n  (\pk, \sk) \sample \kgen (\secparam) \\\n  (m_0, m_1) \sample \adv (\secparam, \pk, c) \\\n  c \sample \enc (\pk, m_b) \\\n  b' \sample \adv (\secparam, \pk, c, \state) \\\n  \pcreturn b = b' }
\end{pcimage}
```

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

IND-CPA

<table>
<thead>
<tr>
<th>\textbf{Enc}(n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: ( b \leftarrow {0, 1} )</td>
</tr>
<tr>
<td>2: ((\pk, \sk) \leftarrow \text{KGen}(1^n))</td>
</tr>
<tr>
<td>\ldots \text{ Setup Completed } \ldots</td>
</tr>
<tr>
<td>3: ((m_0, m_1) \leftarrow \text{A}(1^n, \pk, c))</td>
</tr>
<tr>
<td>4: (c \leftarrow \text{Enc}(\pk, m_b))</td>
</tr>
<tr>
<td>5: (b' \leftarrow \text{A}(1^n, \pk, c, \text{state}))</td>
</tr>
<tr>
<td>6: \pcreturn b = b'</td>
</tr>
</tbody>
</table>

\begin{pcimage}
\begin{procedureblock}{linenumbering}{{$\text{\textbf{indcpa}}_{\text{\textbf{\textbf{enc}}}^\text{\textbf{\textbf{adv}}} (\text{\textbf{secpar}})$}}{
\node[rectangle callout, callout absolute pointer=($(kgen) - (0cm, 0cm)$), fill=orange] at ([shift={(+3, -1)}]kgen) {
\begin{varwidth}{3cm}
\text{KGen}(1^n) \text{ samples a public key } \pk \text{ and a private key } \sk.
\end{varwidth}
};
\end{procedureblock}
\end{pcimage}

1.2 Stacking

To arrange multiple procedures, cryptocode offers horizontal and vertical stacking environments `\pcstack` and `\pcvstack`. In the example below we arrange four code blocks in three columns equispaced with 1cm distance and stack two procedures in the center column.
1.3 Columns

The \texttt{pseudocode} and \texttt{procedure} commands allow the usage of multiple columns. You switch to a new column by inserting a \texttt{\textgreater}. This is similar to using an \texttt{align} environment and placing a tabbing \& character.\footnote{In fact, the \texttt{pseudocode} command is based on amsmath’s \texttt{flalign} environment.}

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

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 always skip a column by using \texttt{\textgreater\textgreater} or you can alternatively use \texttt{\textless} as a shorthand for \texttt{\textgreater\textgreater}.

\begin{tabular}{cccc}
\textbf{First} & \textbf{Second} & \textbf{Third} & \textbf{Fourth}  \\
\hline  
  $b \leftarrow \{0,1\}$ & $b \leftarrow \{0,1\}$ & $b \leftarrow \{0,1\}$ & $b \leftarrow \{0,1\}$ \\
\end{tabular}
1.4 Protocols

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

\begin{pseudocodeblock}
\textbf{Alice} \textbf{Bob} \textbf{Charlie} \\
work & work & work \\
\hline \\
\text{Work result} & \text{Work result} & \text{Bottom message} \\
\hline \\
\end{pseudocodeblock}

The commands \texttt{\sendmessageright} and \texttt{\sendmessageleft} are very flexible and allow to style the sending of messages in various ways. Also note the \texttt{\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) while the second optional argument allows us to, for example, draw a horizontal line.

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

For basic protocols you might also utilize the \texttt{\sendmessengeright*} and \texttt{\sendmessageleft*} commands which simply take a message which is displayed (in math mode) on top.

\begin{pseudocodeblock}
x \leftarrow Z_q \\
X \leftarrow g^x \\
\hline \\
\mathcal{G}, q, g, X \\
\hline \\
y \leftarrow Z_q \\
Y \leftarrow g^y \\
Y \\
\hline \\
k_A \leftarrow Y^x \\
k_B \leftarrow X^y
\end{pseudocodeblock}
We will discuss protocols in greater detail in Section 5.

1.5 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.

<table>
<thead>
<tr>
<th>Game 1(n)</th>
<th>Game 2(n)</th>
<th>Game 3(n)</th>
<th>Game 4(n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: Step 1</td>
<td>Step 1</td>
<td></td>
<td>Step 1</td>
</tr>
<tr>
<td>2:</td>
<td>From game 3 on</td>
<td></td>
<td>From game 3 on</td>
</tr>
<tr>
<td>3: Step 2</td>
<td>Step 3 is different</td>
<td></td>
<td>Step 3 adapted again</td>
</tr>
</tbody>
</table>

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 will discuss how to visualize game-based proofs in Section 6.

1.6 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/or challengers 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

\begin{gameproof}
\begin{pchstack}[center, space=1em]
\gameprocedure{linenumbering, minlineheight=1.5em}{%
\text{Step 1} \\\n\text{Step 3}}%
\text{Step 1} \\\n\pcbox{\text{From game 3 on}} \\\n\gamechange{\text{Step 3 is different}}
\} %
\\gameprocedure{linenheight=1.5em}{%
\text{Step 1} \\\n\text{From game 3 on}\} \\\n\text{Step 3 adapted again}}
\} %
\end{pchstack}
\end{gameproof}
plain TIKZ commands. Following is an example reduction. We discuss the details in Section 7.

\begin{verbatim}
\begin{bbrenv}{A}
\begin{bbrbox}[name=Reduction]
\pseudocode{
\text{Do something} \\
\text{Step 2}
}
\end{bbrbox}
\begin{bbrenv}{B}
\begin{bbrbox}[name=Adversary, minheight=2.25cm]
\bbormsgto{top=$m$}
\bbormsgfrom{top=$\sigma$}
\bbormsgtxt{\pseudocode{\text{more work}}}
\bbormsgto{top=$m$}
\bbormsgfrom{top=$\sigma$}
\bbraqryto{side=$m$}
\bbraqryfrom{side=$b$}
\end{bbrbox}
\pseudocode{\text{finalize}}
\end{bbrenv}
\bbrinput{input}
\bbroutput{output}
\begin{bbroracle}{OraA}
\begin{bbrbox}[name=Oracle 1, minheight=1cm]
\end{bbrbox}
\end{bbroracle}
\begin{bbroracle}{OraB}
\begin{bbrbox}[name=Oracle 2, minheight=1cm]
\end{bbrbox}
\end{bbroracle}
\end{verbatim}
2 Notation Macros

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

\begin{verbatim}
\usepackage[
  n, % or lambda
  advantage,
  operators,
  sets,
  adversary,
  landau,
  probability,
  notions,
  logic,
  if,
  mm,
  primitives,
  events,
  complexity,
  oracles,
  asymptotics,
  keys
]{cryptocode}
\end{verbatim}

Remark. Note that the available command sets are far from complete and reflect my own work (especially once you get to cryptographic notions and primitives). In case you feel that something should be added feel free to drop me an email, or better yet, open an issue and pull request on github (https://github.com/arnomi/cryptocode).

2.1 Security Parameter

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

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

The first command provides the “letter”, i.e., either \(n\) or \(\lambda\), whereas \(\secparam\) prints \(1^{\secpar}\) (i.e., \(1^n\) for option “n”). Finally, \(\SECPAR\) yields \(N_0\) (resp. \(\Lambda\)) and is meant to be used in sentences such as, “there exists \(N_0 \in \mathbb{N}\) such that for all \(n \geq N_0\), ...”

2.2 Advantage Terms

Load the package option “advantage” in order to define the command \(\advantage\) used to specify advantage terms such as:

\[
\Adv_{A,\text{PRF}}^\prf(n)
\]

\begin{verbatim}
\advantage\{\prf\}\{\adv,\prf\}
\advantage\{\prf\}\{\adv,\prf\}\{\text{arg}\}
\end{verbatim}

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

In order to redefine the styles in which superscript and subscript are set, or in case you want to replace the term \(\Adv\), redefine:
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>$b \leftarrow {0,1}$</td>
<td></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{\frac{a}{b}}</td>
<td>Absolute number</td>
<td>$\frac{a}{b}$</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>$x \leftrightarrow a|b$</td>
<td></td>
</tr>
<tr>
<td>\emptystring</td>
<td>The empty string</td>
<td>$\epsilon$</td>
<td>$x \leftrightarrow \epsilon$</td>
</tr>
<tr>
<td>\argmax</td>
<td>arg max</td>
<td>$\arg \max$</td>
<td>$\arg \max_{x \in S} f(x)$</td>
</tr>
<tr>
<td>\argmin</td>
<td>arg min</td>
<td>$\arg \min$</td>
<td>$\arg \min_{x \in S} f(x)$</td>
</tr>
<tr>
<td>\pindist</td>
<td>Perfect indistinguishability</td>
<td>$\equiv$</td>
<td>$X \equiv Y$</td>
</tr>
<tr>
<td>\sindist</td>
<td>Statistical indistinguishability</td>
<td>$\approx$</td>
<td>$X \approx Y$</td>
</tr>
<tr>
<td>\cindist</td>
<td>Computational indistinguishability</td>
<td>$\approx$</td>
<td>$X \approx Y$</td>
</tr>
</tbody>
</table>

The paired operators \floor, \ceil, \Angle, \norm, and \abs also come in a form for flow text which does not scale the outer delimiter. These are \tfloor, \tceil, \tAngle, \tnorm, and \tabs.

Note that \argmax and \argmin in block formulas will set their subscripts as limits, i.e.:

$$\arg \max_{x \in S} f(x)$$

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>\edv</td>
<td>Adversary</td>
<td>$E$</td>
</tr>
<tr>
<td>\adv</td>
<td>Adversary</td>
<td>$M$</td>
</tr>
<tr>
<td>\pdv</td>
<td>Adversary</td>
<td>$P$</td>
</tr>
<tr>
<td>\rdv</td>
<td>Adversary</td>
<td>$R$</td>
</tr>
<tr>
<td>\adv</td>
<td>Adversary</td>
<td>$S$</td>
</tr>
</tbody>
</table>

The style in which an adversary is rendered is controlled via

$$\renewcommand{\padvstyle}{\ensuremath{\mathcal{#1}}\mathcal{#1}}$$
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(micron) notation</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>\smallO{n^2}</td>
<td>small o(micron) notation</td>
<td>$o(n^2)$</td>
</tr>
<tr>
<td>\bigOmega{n^2}</td>
<td>Big Omega notation</td>
<td>$\Omega(n^2)$</td>
</tr>
<tr>
<td>\bigTheta{n^2}</td>
<td>Big Theta</td>
<td>$\Theta(n^2)$</td>
</tr>
<tr>
<td>\orderOf</td>
<td>On the order of</td>
<td>$f(n) \sim g(n)$</td>
</tr>
</tbody>
</table>

2.6 Probabilities

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

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

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

\begin{align*}
\Pr[X = x] \\
\Pr_{x \leftarrow \{0,1\}^n}[X = x] \\
\Pr[X = x | A = b] \\
\Pr_{x \leftarrow \{0,1\}^n}[x = 5 | A = b]
\end{align*}

You can control the probability symbol (Pr) by redefining

1. \renewcommand{\probname}{Pr}

The probability commands have a flowtext version \tprob{X=X} or \tcondprob{X=x}{Y=y} which does not scale the delimiters. In case the probability space is more complex, you can use

1. \probsublong{x,y\sample\set{1,2,3,4,5,6}, z = x + y}{z = 7}

which yields

\[ \Pr[z = 7 : x, y \leftarrow \{1,2,3,4,5,6\}, z = x + y]. \]

For specifying expectations the following commands are defined

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

yielding

\[ \begin{align*}
\mathbb{E}[X] \\
\mathbb{E}_{x,y \leftarrow \{1,\ldots,6\}}[x + y] \\
\mathbb{E}[X + Y | Y > 3] \\
\mathbb{E}_{x,y \leftarrow \{1,\ldots,6\}}[x + y | y > 3]
\end{align*} \]

Again flowtext versions such as \texttt{texpect{X}} are available. To control the expectation symbol (E), redefine

1. \renewcommand{\expname}{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

$$\entropy{X} \quad \minentropy{X} \quad \condentropy{X}{Y=5} \quad \condminentropy{X}{Y=5} \quad \condavgminentropy{X}{Y=5}$$

This yields

$$H(X) \quad H_{\infty}(X) \quad H(X \mid Y = 5) \quad H_{\infty}(X \mid Y = 5) \quad \tilde{H}_{\infty}(X \mid Y = 5)$$

### 2.7 Sets

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

$$\set{1, \ldots, 10} \quad \sequence{1, \ldots, 10}$$

which are typeset as

$$\{1, \ldots, 10\} \quad (1, \ldots, 10)$$

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>\WN</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>P</td>
<td>\mathbb{P}</td>
</tr>
<tr>
<td>\FF</td>
<td>F</td>
<td>\mathbb{F}</td>
</tr>
<tr>
<td>\GG</td>
<td>G</td>
<td>\mathbb{G}</td>
</tr>
</tbody>
</table>

The style in which sets are being set can be adapted by redefining
### 2.8 Cryptographic Notions

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

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{\textbackslash\texttt{indcpa}}</td>
<td>IND-CPA security for encryption schemes</td>
<td>IND-CPA</td>
</tr>
<tr>
<td>\texttt{\textbackslash\texttt{indcca}}</td>
<td>IND-CCA security for encryption schemes</td>
<td>IND-CCA</td>
</tr>
<tr>
<td>\texttt{\textbackslash\texttt{indccai}}</td>
<td>IND-CCA1 security for encryption schemes</td>
<td>IND-CCA1</td>
</tr>
<tr>
<td>\texttt{\textbackslash\texttt{indccaii}}</td>
<td>IND-CCA2 security for encryption schemes</td>
<td>IND-CCA2</td>
</tr>
<tr>
<td>\texttt{\textbackslash\texttt{ind}}</td>
<td>IND security</td>
<td>IND</td>
</tr>
<tr>
<td>\texttt{\textbackslash\texttt{priv}}</td>
<td>PRIV security for deterministic public-key encryption schemes</td>
<td>PRIV</td>
</tr>
<tr>
<td>\texttt{\textbackslash\texttt{prvda}}</td>
<td>PRV-CDA security (for deterministic public-key encryption schemes)</td>
<td>PRV-CDA</td>
</tr>
<tr>
<td>\texttt{\textbackslash\texttt{prvrcda}}</td>
<td>PRV$-$CDA security (for deterministic public-key encryption schemes)</td>
<td>PRV$-$CDA</td>
</tr>
<tr>
<td>\texttt{\textbackslash\texttt{kiae}}</td>
<td>Key independent authenticated encryption</td>
<td>KIAE</td>
</tr>
<tr>
<td>\texttt{\textbackslash\texttt{kdae}}</td>
<td>Key dependent authenticated encryption</td>
<td>KDAE</td>
</tr>
<tr>
<td>\texttt{\textbackslash\texttt{mle}}</td>
<td>Message locked encryption</td>
<td>MLE</td>
</tr>
<tr>
<td>\texttt{\textbackslash\texttt{uce}}</td>
<td>Universal computational extractors</td>
<td>UCE</td>
</tr>
<tr>
<td>\texttt{\textbackslash\texttt{eufcma}}</td>
<td>Existential unforgeability under chosen message attack</td>
<td>EUF-CMA</td>
</tr>
<tr>
<td>\texttt{\textbackslash\texttt{eufnacma}}</td>
<td>Non-adaptive existential unforgeability under chosen message attack</td>
<td>EUF-naCMA</td>
</tr>
<tr>
<td>\texttt{\textbackslash\texttt{seufcma}}</td>
<td>Strong existential unforgeability under chosen message attack</td>
<td>SUF-CMA</td>
</tr>
<tr>
<td>\texttt{\textbackslash\texttt{eufko}}</td>
<td>Existential unforgeability under key only attack</td>
<td>EUF-KO</td>
</tr>
</tbody>
</table>

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

```latex
\renewcommand{\pcnotionstyle}[1]{\ensuremath{\texttt{\textbackslash\texttt{#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>\texttt{\textbackslash\texttt{\textbackslash\texttt{AND}}}</td>
<td>Logical AND</td>
<td>AND</td>
</tr>
<tr>
<td>\texttt{\textbackslash\texttt{\textbackslash\texttt{NAND}}}</td>
<td>Logical NAND</td>
<td>NAND</td>
</tr>
<tr>
<td>\texttt{\textbackslash\texttt{\textbackslash\texttt{OR}}}</td>
<td>Logical OR</td>
<td>OR</td>
</tr>
<tr>
<td>\texttt{\textbackslash\texttt{\textbackslash\texttt{NOR}}}</td>
<td>Logical NOR</td>
<td>NOR</td>
</tr>
<tr>
<td>\texttt{\textbackslash\texttt{\textbackslash\texttt{XOR}}}</td>
<td>Logical XOR</td>
<td>XOR</td>
</tr>
<tr>
<td>\texttt{\textbackslash\texttt{\textbackslash\texttt{XNOR}}}</td>
<td>Logical XNOR</td>
<td>XNOR</td>
</tr>
<tr>
<td>\texttt{\textbackslash\texttt{\textbackslash\texttt{notimplies}}}</td>
<td>Negated implication</td>
<td>$\implies$</td>
</tr>
<tr>
<td>\texttt{\texttt{\textbackslash\texttt{NOT}}}</td>
<td>not</td>
<td>NOT</td>
</tr>
<tr>
<td>\texttt{\texttt{\textbackslash\texttt{xor}}}</td>
<td>exclusive or</td>
<td>$\oplus$</td>
</tr>
<tr>
<td>\texttt{\texttt{\textbackslash\texttt{false}}}</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>\texttt{\texttt{\textbackslash\texttt{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

\renewcommand{\pcalgoirstyle}[1]{\ensuremath{\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

\renewcommand{\pmachinemodelstyle}[1]{\ensuremath{\mathsf{#1}}}

13
## 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>Proover</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>\gash</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>\pad</td>
<td>A padding function</td>
<td>pad</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>\sign</td>
<td>Signing</td>
<td>Sign</td>
</tr>
<tr>
<td>\verify</td>
<td>Verifying</td>
<td>Vf</td>
</tr>
<tr>
<td>\owf</td>
<td>One-way function</td>
<td>OWF</td>
</tr>
<tr>
<td>\prf</td>
<td>Pseudorandom function</td>
<td>PRF</td>
</tr>
<tr>
<td>\prp</td>
<td>Pseudorandom permutation</td>
<td>PRP</td>
</tr>
<tr>
<td>\prg</td>
<td>Pseudorandom generator</td>
<td>PRG</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>Punctur</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>\extractor</td>
<td>An extractor</td>
<td>Ext</td>
</tr>
<tr>
<td>\ext</td>
<td>Shorthand for \extractor</td>
<td>Ext</td>
</tr>
</tbody>
</table>

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

\renewcommand{\pcalgostyle}{\ensuremath{\mathsf{#1}}} \renewcommand{\pcoraclestyle}{\ensuremath{\mathsf{#1}}}

## 2.13 Oracles

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

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>\oracle</td>
<td>Generic oracle</td>
<td>O</td>
</tr>
<tr>
<td>\oracle[LoR]</td>
<td>Custom oracle</td>
<td>LoR</td>
</tr>
<tr>
<td>\ro</td>
<td>Random oracle</td>
<td>RO</td>
</tr>
<tr>
<td>\Oracle{\sign}</td>
<td>Oracle version of procedure</td>
<td>O\sign</td>
</tr>
</tbody>
</table>

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

\renewcommand{\pcoraclestyle}{\ensuremath{\mathsf{#1}}}
2.14 Events

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

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>\event{E}</td>
<td>Event E</td>
<td>E</td>
</tr>
<tr>
<td>\nevent{E}</td>
<td>Negated event E</td>
<td>E</td>
</tr>
<tr>
<td>\bad</td>
<td>Bad event</td>
<td>bad</td>
</tr>
<tr>
<td>\nbad</td>
<td>Bad event</td>
<td>bad</td>
</tr>
</tbody>
</table>

2.15 Complexity

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

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>\complclass{myClass}</td>
<td>myClass</td>
</tr>
<tr>
<td>\cocomplclass{myClass}</td>
<td>co-myClass</td>
</tr>
<tr>
<td>\npol</td>
<td>NP</td>
</tr>
<tr>
<td>\conpol</td>
<td>co-NP</td>
</tr>
<tr>
<td>\pol</td>
<td>P</td>
</tr>
<tr>
<td>\bpp</td>
<td>BPP</td>
</tr>
<tr>
<td>\ppoly</td>
<td>P/poly</td>
</tr>
<tr>
<td>\NC{1}</td>
<td>NC¹</td>
</tr>
<tr>
<td>\AC{1}</td>
<td>AC¹</td>
</tr>
<tr>
<td>\TC{1}</td>
<td>TC¹</td>
</tr>
<tr>
<td>\AM</td>
<td>AM</td>
</tr>
<tr>
<td>\coAM</td>
<td>co-AM</td>
</tr>
<tr>
<td>\PH</td>
<td>PH</td>
</tr>
<tr>
<td>\csigma{1}</td>
<td>Σ¹p</td>
</tr>
<tr>
<td>\cpi{1}</td>
<td>Π¹p</td>
</tr>
<tr>
<td>\cosigma{1}</td>
<td>co-Σ¹p</td>
</tr>
<tr>
<td>\copi{1}</td>
<td>co-Π¹p</td>
</tr>
</tbody>
</table>

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

\renewcommand{\pccomplexitystyle}{\ensuremath{\mathsf{#1}}}

\[ \text{\renewcommand{\pccomplexitystyle}{\ensuremath{\mathsf{#1}}}} \]
2.16  Asymptotics

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

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>\negl</td>
<td>A negligible function</td>
<td>negl(n) (n is \secpar)</td>
</tr>
<tr>
<td>\negl[x]</td>
<td>A negligible function</td>
<td>negl(x)</td>
</tr>
<tr>
<td>\negl[]</td>
<td>A negligible function</td>
<td>negl</td>
</tr>
<tr>
<td>\poly</td>
<td>A polynomial</td>
<td>poly(n) (n is \secpar)</td>
</tr>
<tr>
<td>\poly[x]</td>
<td>A polynomial</td>
<td>poly(x)</td>
</tr>
<tr>
<td>\poly[]</td>
<td>A polynomial</td>
<td>poly</td>
</tr>
<tr>
<td>\pp</td>
<td>some polynomial p</td>
<td>p</td>
</tr>
<tr>
<td>\pp[t]</td>
<td>some custom polynomial t</td>
<td>t</td>
</tr>
<tr>
<td>\cc</td>
<td>some polynomial c</td>
<td>c</td>
</tr>
<tr>
<td>\ee</td>
<td>some polynomial e</td>
<td>e</td>
</tr>
<tr>
<td>\kk</td>
<td>some polynomial k</td>
<td>k</td>
</tr>
<tr>
<td>\mm</td>
<td>some polynomial m</td>
<td>m</td>
</tr>
<tr>
<td>\nn</td>
<td>some polynomial n</td>
<td>n</td>
</tr>
<tr>
<td>\qq</td>
<td>some polynomial q</td>
<td>q</td>
</tr>
<tr>
<td>\rr</td>
<td>some polynomial r</td>
<td>r</td>
</tr>
</tbody>
</table>

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

1 \renewcommand{\pcpolynomialstyle}[1]{\ensuremath{\mathrm{#1}}}

2.17  Keys

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

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>pk</td>
<td>public key</td>
<td>pk</td>
</tr>
<tr>
<td>vk</td>
<td>verification key</td>
<td>vk</td>
</tr>
<tr>
<td>sk</td>
<td>secret key</td>
<td>sk</td>
</tr>
<tr>
<td>key</td>
<td>a plain key</td>
<td>k</td>
</tr>
<tr>
<td>key[xk]</td>
<td>custom key</td>
<td>xk</td>
</tr>
<tr>
<td>hk</td>
<td>hash key</td>
<td>hk</td>
</tr>
<tr>
<td>gk</td>
<td>gash key</td>
<td>gk</td>
</tr>
<tr>
<td>fk</td>
<td>function key</td>
<td>fk</td>
</tr>
<tr>
<td>st</td>
<td>state</td>
<td>st</td>
</tr>
<tr>
<td>state</td>
<td>state</td>
<td>state</td>
</tr>
<tr>
<td>state{myState}</td>
<td>custom state</td>
<td>state{myState}</td>
</tr>
</tbody>
</table>

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

1 \renewcommand{\pckeystyle}[1]{\ensuremath{\mathsf{#1}}}
3 Pseudocode

3.1 Basics

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

\vspace{1em}
\begin{pchstack} \center
\texttt{\textbackslash pseudocode}
\begin{algorithm}
\texttt{b \sample \{0, 1\}}
\texttt{(pk, sk) \sample \texttt{KGen}(1^n)}
\texttt{(m_0, m_1) \sample \texttt{A}(1^n, pk, c)}
\texttt{c \sample \texttt{Enc}(pk, m_b)}
\texttt{b' \sample \texttt{A}(1^n, pk, c)}
\texttt{\pcreturn b = b'}
\end{algorithm}
\end{pchstack}

which is generated by

First note that \texttt{\textbackslash pseudocode} on its own does not space itself. For laying out one (or multiple) code blocks cryptocode defines stacking environments such as \texttt{pchstack} and \texttt{pcvstack} that we discuss in Section 3.7. Wrapping a single pseudocode in a \texttt{pchstack} as in the above example generates a nicely offset code block.

As code blocks are most often not used in flow text, cryptocode offers the shorthand \texttt{\textbackslash pseudocodeblock} which centers and offsets a pseudocode block as above. We thus get the very same by writing

\vspace{1em}
\begin{pchstack} \center
\texttt{\textbackslash pseudocodeblock}
\begin{algorithm}
\texttt{b \sample \{0, 1\}}
\texttt{(pk, sk) \sample \texttt{KGen}(1^n)}
\texttt{(m_0, m_1) \sample \texttt{A}(1^n, pk, c)}
\texttt{c \sample \texttt{Enc}(pk, m_b)}
\texttt{b' \sample \texttt{A}(1^n, pk, c)}
\texttt{\pcreturn b = b'}
\end{algorithm}
\end{pchstack}

We can also define custom block commands, for example, the following defines a command \texttt{\textbackslash pcb} that offsets and centers code and draws a tight fitting box around the code block:

\vspace{1em}
\begin{pchstack} \center
\texttt{\createpseudocodeblock{pcb}{center,boxed}}
\begin{algorithm}
\texttt{b \sample \{0, 1\}}
\texttt{(pk, sk) \sample \texttt{KGen}(1^n)}
\texttt{(m_0, m_1) \sample \texttt{A}(1^n, pk, c)}
\texttt{c \sample \texttt{Enc}(pk, m_b)}
\texttt{b' \sample \texttt{A}(1^n, pk, c)}
\texttt{\pcreturn b = b'}
\end{algorithm}
\end{pchstack}

(We discuss creating custom pseudocode commands in detail in Section 3.1.2). If we now use \texttt{\textbackslash pcb} as just defined in the above example, we obtain the following nicely spaced and boxed result.
\begin{verbatim}
\pcb{
    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) \\
    return b = b'
}
\end{verbatim}

which is generated as

\begin{verbatim}
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) \\
return b = b'
\end{verbatim}

\textbf{Remark.} In the following we will use this boxed representation for the examples, but use \texttt{pseudocodeblock} in the corresponding code listings. As you can see, the pseudocode command provides a math based environment where you can simply start typing your pseudocode separating lines by  \\

3.1.1 Customizing Pseudocode

Besides the mandatory argument the \texttt{pseudocode} command can take an optional argument which consists of a list of key=value pairs separated by commas.

\begin{verbatim}
\pseudocode[options]{body}
\end{verbatim}

The following parameters 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.
- **skipfirstln** Starts line numbering on the second line.
- **minlineheight** Specify a minimum height for each line. Can be globally set by redefining \texttt{\pcminlineheight}.
- **syntaxhighlight** When set to “auto” cryptocode will attempt to automatically highlight keywords such as “for”, “foreach” and “return”. Note that this feature should be regarded as experimental. In particular, it is rather slow.
- **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 \texttt{\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 \texttt{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. By default set to 0pt which can be globally overwritten by setting length \texttt{\pcheadlinesep}.

**bodylinesep** Specifies the distance between body and the line. By default set to 0.3\emph{\baselineskip} which can be globally overwritten by setting length \texttt{\pcbodylinesep}.

**colsep** Defines the space between columns.

**headheight** Specifies the height of the header. By default set to 3.25ex which can be globally overwritten by setting length \texttt{\pcheadheight}.

**headlinecmd** Allows to overwrite which command is used to draw the bar below the headline. Defaults to \texttt{\hrule}.

**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”.

**nodraft** Forces syntax highlighting also in draft mode.

The following code

\begin{pseudocodeblock}[linenumbering,syntaxhighlight=auto,head=Header]{ return null }
3.1.2 Customized Pseudocode Commands

Besides the \pseudocode and \pseudocodeblock command the command \procedure (and its block variant \procedureblock) provides easy access to generate code with a header. They take the following form

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

Examples

\begin{verbatim}
IND-CPA\text{\textsuperscript{\textscript{A}}}(n) 
\begin{align*}
b &\leftarrow \{0,1\} \\
(pk, sk) &\leftarrow KGen(1^n) \\
(m_0, m_1) &\leftarrow A(1^n, pk, c) \\
c &\leftarrow \text{Enc}(pk, m_b) \\
b' &\leftarrow A(1^n, pk, c) \\
\text{return } b = b'
\end{align*}
\end{verbatim}

which is generated as

\begin{verbatim}
\procedureblock {$\text{\textsuperscript{\textscript{A}}}(\text{\textsuperscript{\textscript{n}}})$} 
\begin{align*}
b \&\left\leftarrow \{0,1\} \\
(pk, sk) \&\left\leftarrow KGen(1^n) \\
(m_0, m_1) \&\left\leftarrow A(1^n, pk, c) \\
c \&\left\leftarrow \text{Enc}(pk, m_b) \\
b' \&\left\leftarrow A(1^n, pk, c) \\
\text{return } b = b'
\end{align*}
\end{verbatim}

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}
\createpseudocodecommand {mypseudocode} {} {} {linenumbering} \\
\createprocedurecommand {myprocedure} {} {} {linenumbering} \\
\createpseudocodeblock {pcb} {center, boxed} {} {} {linenumbering} \\
\createprocedureblock {procb} {center, boxed} {} {} {linenumbering}
\end{verbatim}

creates the commands \mypseudocode and \myprocedure with line numbering always enabled as well as the block commands \pseudocodeblock and \procedureblock also with line numbering enabled. The created commands have an identical interface as the \pseudocode (resp. \procedure) command. The two arguments 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 (first empty argument) and a prefix for the header. For example, the command created as

\begin{verbatim}
\procedureblock {$\text{\textsuperscript{\textscript{A}}}(\text{\textsuperscript{\textscript{n}}})$} 
\begin{align*}
b \&\left\leftarrow \{0,1\} \\
(pk, sk) \&\left\leftarrow KGen(1^n) \\
(m_0, m_1) \&\left\leftarrow A(1^n, pk, c) \\
c \&\left\leftarrow \text{Enc}(pk, m_b) \\
b' \&\left\leftarrow A(1^n, pk, c) \\
\text{return } b = b'
\end{align*}
\end{verbatim}
could be used as

\begin{verbatim}
\expproc{$\indcpa_{\enc}^{\adv}(\secpar)$}{
    (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) \\
    \pcreturn b = b'
}
\end{verbatim}

This results in

\begin{verbatim}
Experiment IND-CPA_{\Enc}^A(n)
1: \quad b \leftarrow \{0, 1\}
2: \quad (pk, sk) \leftarrow \kGen(1^n)
3: \quad (m_0, m_1) \leftarrow \Adv(1^n, pk, c)
4: \quad c \leftarrow \Enc(pk, m_b)
5: \quad b' \leftarrow \Adv(1^n, pk, c)
6: \quad \text{return } b = b'
\end{verbatim}

3.2 Indentation

In order to indent code use \pcind or short \t. You can also use customized spacing such as \quad or \hspace when using the pseudocode command in math mode.

\begin{verbatim}
for i = 1..10 do
    T[i] \leftarrow \{0, 1\}^n
end
\end{verbatim}

which is generated as

\begin{verbatim}
\pseudocodeblock{for i = 1..10 }\pcode do
    \pind T[i] \sample \bin^n
end
\end{verbatim}

You can specify multiple levels via the optional first argument

\begin{verbatim}
\t[level] % \pcind[level]
\end{verbatim}

\begin{verbatim}
for i = 1..10 do
    T[i] \leftarrow \{0, 1\}^n
    T[i] \leftarrow \{0, 1\}^n
    T[i] \leftarrow \{0, 1\}^n
    T[i] \leftarrow \{0, 1\}^n
    T[i] \leftarrow \{0, 1\}^n
\end{verbatim}

21
\begin{pseudocodeblock}{space=auto}\
  \pcfor i = 1..10 \pcdo \\
  \{ T[i] \sample \bin^n \}
  \pcelse
  \{ T[i] \sample \bin^n \}
  \pcendif \\
\end{pseudocodeblock}

You can customize the indentation shortcut by redefining
\renewcommand{\pcindentname}{t}

\section*{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
\begin{enumerate}
  \item \pcif, \pcfor, \pcwhile, \pcrepeat, \pcforeach
\end{enumerate}
it will increase the indentation starting from the next line. It will again remove the indentation on seeing
\begin{enumerate}
  \item \pcfi, \pcendif, \pcendfor, \pcendwhile, \pcuntil, \pcendforeach
\end{enumerate}

Additionally, on seeing
\begin{enumerate}
  \item \pcelse, \pcelseif
\end{enumerate}
it will remove the indentation for that particular line. Thus the following
\begin{verbatim}
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
\end{verbatim}
can be obtained by:
\begin{pseudocodeblock}{space=auto}\
  \pcfor a \in [10] \pcdo \\
  \pcfor a \in [10] \pcdo \\
  \pcfor a \in [10] \pcdo \\
  \pcif a = b \pcthen \\
  \text{some operation} \\
\end{pseudocodeblock}
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 (experimental)

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

This yields the following result

\begin{center}
\begin{pseudocode}[space=keep]\
\for i = 1..10 \do \% \\
T[i] \sample \bin^n \\
T[i] \sample \bin^n \\
T[i] \sample \bin^n \\
T[i] \sample \bin^n \\
\end{pseudocode}\end{center}

Note that automatic spacing only works when the \texttt{pseudocode} command is not wrapped within another command. Thus in order to get a frame box \texttt{\fbox{pseudocode[space=keep]{code}}} will not work but you would need to use an environment such as one offered by the \texttt{mdframed} package (\texttt{https://www.ctan.org/pkg/mdframed}). Also see Section 8.1.

3.3 Textmode

By default pseudocode enables \LaTeX’ 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”.

This is simply text

\begin{pseudocodeblock}[mode=text]\
This is \% \\
\text{simply text}\
\end{pseudocodeblock}


3.4 Syntax Highlighting

In the above examples we have used commands `\pcreturn` and `\pcfor` to highlight certain keywords. Besides the `pcreturn`, `pcfor` and `pedo` (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>command</th>
<th>outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>\pcabort</td>
<td>abort</td>
</tr>
<tr>
<td>\pccontinue</td>
<td>continue</td>
</tr>
<tr>
<td>\pccomment{comment}</td>
<td>// comment</td>
</tr>
<tr>
<td>\pccomment[2em]{comment}</td>
<td>// comment</td>
</tr>
<tr>
<td>\pclinecomment{comment}</td>
<td>// comment</td>
</tr>
<tr>
<td>\pcdo</td>
<td>do</td>
</tr>
<tr>
<td>\pcdone</td>
<td>done</td>
</tr>
<tr>
<td>\pcfail</td>
<td>fail</td>
</tr>
<tr>
<td>\pcfalse</td>
<td>false</td>
</tr>
<tr>
<td>\pcif</td>
<td>if</td>
</tr>
<tr>
<td>\pcfi</td>
<td>fi</td>
</tr>
<tr>
<td>\pcendif</td>
<td>endif</td>
</tr>
<tr>
<td>\pcelse</td>
<td>else</td>
</tr>
<tr>
<td>\pcelsif</td>
<td>elseif</td>
</tr>
<tr>
<td>\pcfor</td>
<td>for</td>
</tr>
<tr>
<td>\pcendfor</td>
<td>endfor</td>
</tr>
<tr>
<td>\pcforeach</td>
<td>foreach</td>
</tr>
<tr>
<td>\pcendforeach</td>
<td>endforeach</td>
</tr>
<tr>
<td>\pcglobvar</td>
<td>gbl</td>
</tr>
<tr>
<td>\pcin</td>
<td>in</td>
</tr>
<tr>
<td>\pcnew</td>
<td>new</td>
</tr>
<tr>
<td>\pcnull</td>
<td>null</td>
</tr>
<tr>
<td>\pcparse</td>
<td>parse</td>
</tr>
<tr>
<td>\pcrepeat{10}</td>
<td>repeat 10 times</td>
</tr>
<tr>
<td>\pcreturn</td>
<td>return</td>
</tr>
<tr>
<td>\pcuntil</td>
<td>until</td>
</tr>
<tr>
<td>\pcthen</td>
<td>then</td>
</tr>
<tr>
<td>\pctrue</td>
<td>true</td>
</tr>
<tr>
<td>\pcwhile</td>
<td>while</td>
</tr>
<tr>
<td>\pcendwhile</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

```
\texttt{for } i \texttt{ in}\{1,\ldots,10\}
```

Furthermore all constants have a trailing space. This can be removed by adding the optional parameter `[]` such as

```
\texttt{for } i \texttt{ in}\{1,\ldots,10\}
```

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

```
1 \pseudocodeblock{\pcfor i \pcin\{}\{1,\ldots,10\}\}
```
3.4.1 Automatic Syntax Highlighting (Experimental)

The pseudocode command comes with an experimental (and rather slow) 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}, return, {do}, {in}, new, if, null, true, {until}, {to},
false, {then}, repeat, else if, elseif, while, else, done
```

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)

```
Bubblesort(A : list of items)
\n\ns ← length(A)
\nrepeat
\n s ← false
\n for i = 1 to n − 1 do
\n  if this pair is out of order
\n   if A[i − 1] > A[i] then
\n    swap(A[i − 1], A[i])
\n    s ← true
\nuntil ¬s
```

can be typeset as

```
\procedureblock[syntaxhighlight=auto]{Bubblesort(A : list of items)}{
 n \gets length(A)
 repeat
   s \gets false
   for i = 1 to n − 1 do
     \begin{comment} if this pair is out of order \end{comment}
     if A[i − 1] > A[i] then
       \begin{comment} swap them and remember something changed \end{comment}
       swap(A[i − 1], A[i])
       s \gets true
 until ¬s
}
```

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

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

\begin{procedureblock}[space=auto,syntaxhighlight=auto,addkeywords={swap, length}]
  Bubblesort(A : list of items)
  \begin{itemize}
    \item \textcolor{blue}{n} \leftarrow \text{length}(A) \\
    \item \textcolor{blue}{repeat} \\
    \item \textcolor{blue}{s} \leftarrow \text{false} \\
    \item \textcolor{blue}{for i = 1 to n - 1 do} \\
    \item \textcolor{blue}{\texttt{// assuming this pair is out of order}} \\
    \item \textcolor{blue}{if A[i - 1] > A[i] then} \\
    \item \textcolor{blue}{\texttt{// swap them and remember something changed}} \\
    \item \textcolor{blue}{\texttt{swap}(A[i - 1], A[i])} \\
    \item \textcolor{blue}{s} \leftarrow \text{true} \\
    \item \textcolor{blue}{endif} \\
    \item \textcolor{blue}{endfor} \\
    \item \textcolor{blue}{until \neg s}
  \end{itemize}
\end{procedureblock}
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`.

```
\procedureblock{linenumbering}{\indcpa_{\enc^\adv}(\secparam)}{%
1 \pseudocodeblock{linenumbering, skipfirstln, mode=text}{
\pclinecomment{Some comment on first line}
1: \sample b \{0, 1\}
2: (pk, sk) \sample KGen(\secparam)
3: (m_0, m_1) \sample A(1^n, pk, c)
4: c \sample Enc(pk, m_b)
5: b' \sample A(1^n, pk, c)
6: return b = b'
\}
\}
```

is generated by

```
\procedureblock{linenumbering}{\indcpa_{\enc^\adv}(\secparam)}{%
1: b \sample \{0, 1\}
2: (pk, sk) \sample KGen(\secparam)
3: (m_0, m_1) \sample A(1^n, pk, c)
4: c \sample Enc(pk, m_b)
5: b' \sample A(1^n, pk, c)
6: return b = b'
}
```

Note that you can use labels. In the above example `\label{my:line:label}` points to 3.

3.5.1 Skipping Line Numbers

When using automatic line numbering, you can skip line numbers by inserting a `\pcskipln` command. This causes the line number on the next line to be suppressed. In order to suppress the first line number use the option `skipfirstln`. Thus the following

```
// Some comment on first line
1: Some code
// Some other comment
2: Some code
```

is generated by

```
\pseudocodeblock{linenumbering, skipfirstln, mode=text}{
\pclinecomment{Some comment on first line}
1: Some code
\pclinecomment{Some other comment}
2: Some code
}
```

3.5.2 Manually Inserting Line Numbers

In order to manually insert line numbers use the command `\pcln`.
IND-CPA$^A_{Enc}(1^n)$

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$b \leftarrow {0, 1}$</td>
</tr>
<tr>
<td>2</td>
<td>$(pk, sk) \leftarrow KGen(1^n)$</td>
</tr>
<tr>
<td>3</td>
<td>$(m_0, m_1) \leftarrow A(1^n, pk, c)$</td>
</tr>
<tr>
<td>4</td>
<td>$c \leftarrow Enc(pk, m_b)$</td>
</tr>
<tr>
<td>5</td>
<td>$b' \leftarrow A(1^n, pk, c)$</td>
</tr>
<tr>
<td>6</td>
<td>return $b = b'$</td>
</tr>
</tbody>
</table>

is generated by

```latex
\procedure{$\indcpa_{\text{enc}}^A(\secparam)$}
\pcln b \sample \bin
\pcln (pk, sk) \sample \kgen(\secparam)
\pcln \label{my:line:label2} \label{my:line:label2} (m_0, m_1) \sample \adv(\secparam, \pk, c)
\pcln c \sample \enc(\pk, m_b)
\pcln b' \sample \adv(\secparam, \pk, c)
\pcln \text{return } b = b' 
```

Note that labels also work when manually placing line numbers. In the above example label `my:line:label2` points to line number 3.

### 3.5.3 Start Values

You can specify the start value (minus one) of the counter by setting the option `\lnstart`.

```latex
\procedure[\lnstart=10,linenumbering]{Header}{Body}
```

### 3.5.4 Separators

The command `\pclnseparator` defines the separator between code and line number. By default the left separator is set to `:` colon. Also see Section 5.3.1.

### 3.5.5 Style

The style in which line numbers are set can be controlled by redefining `\pclnstyle`.

```latex
\renewcommand\pclnstyle[1]{\text{\scriptsize#1}}
```

For example, to set line numbers in normal font and dot separated use

```latex
\renewcommand{\pclnstyle}[1]{\text{#1}}
\renewcommand{\pclnseparator}{.}
```
3.6 Subprocedures

The pseudocode package allows the typesetting of subprocedures such as

\begin{procedureblock}[linenumbering]{$\indcpa^d_{\text{Enc}}(1^n)$}{%}
\begin{dbox}{\procedure{$\adv(1^n, \pk, c)$}{%}
\begin{itemize}
  \item \text{Step 1}
  \item \text{Step 2}
  \item \text{return $m_0, m_1$}
\end{itemize}
\end{dbox}
\item $b \leftarrow \{0, 1\}$
\item $(\pk, \sk) \leftarrow \text{KGen}(1^n)$
\item $(m_0, m_1) \leftarrow \mathcal{A}(1^n, \pk, c)$
\item $c \leftarrow \text{Enc}(\pk, m_b)$
\item $b' \leftarrow \mathcal{A}(1^n, \pk, c)$
\item \text{return $b = b'$}
\end{procedureblock}

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

\begin{verbatim}
\procedureblock[linenumbering]{$\indcpa^d_{\text{Enc}}(1^n)$}{%}
\begin{dbox}{\procedure{$\adv(1^n, \pk, c)$}{%}
\begin{itemize}
  \item \text{Step 1}
  \item \text{Step 2}
  \item \text{return $m_0, m_1$}
\end{itemize}
\end{dbox}
\item $b \leftarrow \{0, 1\}$
\item $(\pk, \sk) \leftarrow \text{KGen}(1^n)$
\item $(m_0, m_1) \leftarrow \mathcal{A}(1^n, \pk, c)$
\item $c \leftarrow \text{Enc}(\pk, m_b)$
\item $b' \leftarrow \mathcal{A}(1^n, \pk, c)$
\item \text{return $b = b'$}
\end{verbatim}

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

3.6.1 Numbering in Subprocedures

As subprocedures are simply normal pseudocode blocks, you can use easily add 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 \pseudocode command within the subprocedure.
3.7 Stacking Procedures

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

\begin{pchstack} [options] \begin{p}{
\procedure{$\indcpa^A_{\text{Enc}}(1^n)$} {\begin{align*}
1: & \quad b \leftarrow \{0, 1\} \\
2: & \quad (pk, sk) \leftarrow \text{kgen}(1^n) \\
3: & \quad (m_0, m_1) \leftarrow \text{Adv}^O(1^n, pk) \\
4: & \quad c \leftarrow \text{Enc}(pk, m_b) \\
5: & \quad b' \leftarrow \text{Adv}(1^n, pk, c) \\
6: & \quad \text{return } b = b' \\
\end{align*}} \end{p} \end{pchstack}

The following example displays two procedures next to one another. To space two horizontally outlined procedures use the \texttt{space} option or manually insert spaces via \texttt{pchspace} which takes an optional length as a parameter.

Similarly you can stack two procedures vertically using the “pcvstack” environment. As a spacing between two vertically stacked procedures again use either the \texttt{space} option or insert space manually via \texttt{pcvspace} which takes an optional length as a parameter.
Horizontal and vertical stacking can be combined

<table>
<thead>
<tr>
<th>IND-CPA$_{\text{Enc}}^A(1^n)$</th>
<th>IND-CPA$_{\text{Enc}}^A(1^n)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: $b \leftarrow {0,1}$</td>
<td>1: $b \leftarrow {0,1}$</td>
</tr>
<tr>
<td>2: $(\pk,\sk) \leftarrow \text{KGen}(1^n)$</td>
<td>2: $(\pk,\sk) \leftarrow \text{KGen}(1^n)$</td>
</tr>
<tr>
<td>3: $(m_0,m_1) \leftarrow \text{A}_O(1^n,\pk)$</td>
<td>3: $(m_0,m_1) \leftarrow \text{A}_O(1^n,\pk)$</td>
</tr>
<tr>
<td>4: $c \leftarrow \text{Enc}(\pk,m_b)$</td>
<td>4: $c \leftarrow \text{Enc}(\pk,m_b)$</td>
</tr>
<tr>
<td>5: $b' \leftarrow \text{A}(1^n,\pk,c)$</td>
<td>5: $b' \leftarrow \text{A}(1^n,\pk,c)$</td>
</tr>
<tr>
<td>6: return $b = b'$</td>
<td>6: return $b = b'$</td>
</tr>
</tbody>
</table>

**Oracle $O$**

1: Some code
2: Some more code

**Oracle $H_1$**

1: Some code
2: Some more code

**Oracle $H_2$**

1: Some code
2: Some more code
3.7.1 Stacking Options

The following keys are available on both \texttt{pchstack} and \texttt{pcvstack} environments.

\texttt{center} Centers the stack.

\texttt{boxed} Draws a box around the stack.

\texttt{space} Controls the space between two pseudocode blocks within a stack. The default is 0pt which can be adapted globally by redefining \texttt{pchstackspace} or \texttt{pcvstackspace}.

\texttt{noindent} Does not indent the stack. Only applies if option \texttt{center} is not used.

\texttt{inline} Ensures that no paragraph is added by \texttt{pchstack}. This cannot be used together with either \texttt{center} or \texttt{noindent}.

\texttt{aboveskip} By default the outer most stack adds vertical space above. The default space added is \texttt{abovedisplayskip} and can be adapted by redefining \texttt{pcaboveskip}.

\texttt{belowskip} By default the outer most stack adds vertical space below. The default space added is \texttt{belowdisplayskip} and can be adapted by redefining \texttt{pcbelowskip}. Note that the default space below will not be added when used in a floating environment such as a figure. However, when manually setting belowskip it will always be added.

3.8 Default Arguments

You can set the default arguments to be used with pseudocode blocks via \texttt{pcsetargs}. This is especially handy in stacking environments to add arguments to all enclosed code blocks.

\begin{center}
\begin{tabular}{ccc}
Some Procedure A & Some Procedure B & Some Procedure C \\
1 : Step 1 & 1 : Step 1 & 1 : Step 1 \\
2 : Step 2 & 2 : \( \frac{A}{B+C} \) & 2 : Step 2 \\
3 : Step 3 & & \\
\end{tabular}
\end{center}

\begin{Verbatim}
\begin{pchstack}[space=1em, center, boxed]
  \% Do not change size to scriptsize for line numbers
  \renewcommand\pclnstyle[1]{#1}
  \% set default arguments for all pseudocode blocks in this hstack
\end{pchstack}
\end{Verbatim}
### Default Arguments for Stacking

Similarly to \texttt{pcsetargs} you can define default arguments for \texttt{hstack} and \texttt{vstack} environments via \texttt{pcsethstackargs} and \texttt{pcsetvstackargs}.

### 3.9 Divisions and Linebreaks

Within the pseudocode command you generate linebreaks as \\.

In order to specify the linewdith you can add an optional argument

```
\\[ \text{[height]} \]
```

Furthermore, you can add horizontal lines by using the second optional argument and write

```
\\[][\text{hline}]
```

### IND-CPA$^A_{\text{Enc}}(1^n)$

1: \( b \leftarrow \{0, 1\} \)

2: \((pk, sk) \leftarrow KGen(1^n)\)

3: \((m_0, m_1) \leftarrow A^O(1^n, pk)\)

4: \(c \leftarrow \text{Enc}(pk, m_b)\)

5: \(b' \leftarrow A(1^n, pk, c)\)

6: \(\text{return} b = b'\)
3.9.1 Optimizing Layout

In case you are laying out multiple procedures horizontally, procedures may be slightly misaligned if the procedure headings are not of the same height. As an example, Consider the following setup

<table>
<thead>
<tr>
<th>Procedure A</th>
<th>Procedure $B_{G_1}^{F^{h^*}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: do</td>
<td>1: do</td>
</tr>
<tr>
<td>2: some</td>
<td>2: some</td>
</tr>
<tr>
<td>3: work</td>
<td>3: work</td>
</tr>
</tbody>
</table>

Here the sub and double superscripts in Procedure $B$ make the header slightly larger than the maximum allotted space provided for headers which causes procedure $B$ to be slightly shifted to the bottom. The best way to remedy such a situation is to use a combination of the headheight and headlinesep properties to increase the header space in both procedures and shift back the headline for a more compact visualization. As we here want to set some arguments for all procedure blocks within the stacking environment we can use \texttt{\textbackslash pcsetargs}.

\begin{pchstack}
\texttt{\textbackslash pcsetargs\{headheight=5ex, headlinesep=\textbackslash -1ex\}}
\procedure[linenumbering]{Procedure $A$}{
\text{do}\\
\text{some}\\
\text{work}
}
\procedure[linenumbering]{Procedure $B_{G_1}^{F^{h^*}}$}{
\text{do}\\
\text{some}\\
\text{work}
}
\end{pchstack}

3.10 Matrices and Math Environments within Pseudocode

In order to work its magic, cryptocode (in particular within the \texttt{\textbackslash pseudocode} command) mingles with a few low level commands such as \texttt{\textbackslash \textbackslash} or \texttt{\textbackslash 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{pseudocodeblock}
\texttt{\textbackslash pseudocodeblock}\{
\text{compute } P = \begin{pmatrix} A \& B + C \\ \end{pmatrix}
\}
\end{pseudocodeblock}

which, somewhat unexpectedly, yields

\[
\text{compute } P = \begin{pmatrix} A \\ B + C \end{pmatrix}
\]
Here, the alignment is somewhat off. In order, to allow for the `pmatrix` environment to properly work without interference from `\pseudocode` you can wrap it into a `pcmbox` environment (where pcmbox is short for pseudocode math box). This ensures that the low-level changes introduced by `\pseudocode` are not active.

```
\pseudocodeblock{
\begin{pmatrix}
A \\
B + C
\end{pmatrix}
}
```

### 3.11 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 and which is then given the ciphertext of one of the messages in order 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 idea in a slightly more fancy way.

```
\procedureblock[linenumbering]{$\text{indcpa}^A_{\text{Enc}}(1^n)$}{%
1: \text{\texttt{b}} \leftarrow \{0,1\} \\
2: (\text{\texttt{pk}}, \text{\texttt{sk}}) \leftarrow \text{\texttt{KGen}}(1^n) \\
3: (\text{\texttt{state}}, \text{\texttt{m}}_0, \text{\texttt{m}}_1) \leftarrow \text{\texttt{A}}(1^n, \text{\texttt{pk}}, \text{\texttt{c}}) \\
4: \text{\texttt{c}} \leftarrow \text{\texttt{Enc}}(\text{\texttt{pk}}, \text{\texttt{m}}_b) \\
5: \text{\texttt{b}}' \leftarrow \text{\texttt{A}}(1^n, \text{\texttt{pk}}, \text{\texttt{c}}, \text{\texttt{state}}) \\
6: \text{\texttt{return}} \text{\texttt{b}} = \text{\texttt{b}}' 
}%
```

The image on the right is generated by:

```
\begin{pcimage}
\procedureblock[linenumbering]{$\text{indcpa}^A_{\text{Enc}}(1^n)$}{%
1: \text{\texttt{b}} \leftarrow \{0,1\} \\
2: (\text{\texttt{pk}}, \text{\texttt{sk}}) \leftarrow \text{\texttt{KGen}}(1^n) \\
3: (\text{\texttt{state}}, \text{\texttt{m}}_0, \text{\texttt{m}}_1) \leftarrow \text{\texttt{A}}(1^n, \text{\texttt{pk}}, \text{\texttt{c}}) \\
4: \text{\texttt{c}} \leftarrow \text{\texttt{Enc}}(\text{\texttt{pk}}, \text{\texttt{m}}_b) \\
5: \text{\texttt{b}}' \leftarrow \text{\texttt{A}}(1^n, \text{\texttt{pk}}, \text{\texttt{c}}, \text{\texttt{state}}) \\
6: \text{\texttt{return}} \text{\texttt{b}} = \text{\texttt{b}}' 
}%
\end{pcimage}
```

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 by using the optional argument of `\pcnode` for the TIKZ code.
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.

\begin{pcimage}
\procedureblock{\linenumbering}{\textsc{ind-cpa}^{A}_{\text{Enc}(1^n)}}{%
\begin{align*}
1: & \quad b \leftarrow \{0,1\} \\
2: & \quad (pk, sk) \leftarrow \text{KGen}(1^n) \\
3: & \quad (m_0, m_1) \leftarrow \text{A}(1^n, pk) \\
4: & \quad c \leftarrow \text{Enc}(pk, m_b) \\
5: & \quad b' \leftarrow \text{A}(1^n, pk, c, \text{state}) \\
6: & \quad \text{return} \quad b = b' 
\end{align*} \}
\end{pcimage}
4 Tabbing Mode

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

<table>
<thead>
<tr>
<th>First</th>
<th>Second</th>
<th>Third</th>
<th>Fourth</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b \leftarrow {0,1}$</td>
<td>$b \leftarrow {0,1}$</td>
<td>$b \leftarrow {0,1}$</td>
<td>$b \leftarrow {0,1}$</td>
</tr>
</tbody>
</table>

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{verbatim}
\textbf{First} \> \textbf{Second} \> \textbf{Third} \> \textbf{Fourth} \\
b \sample \bin \> b \sample \bin \> b \sample \bin \> b \sample \bin
\end{verbatim}

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.

<table>
<thead>
<tr>
<th>First</th>
<th>Second</th>
<th>Third</th>
<th>Fourth</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b \leftarrow {0,1}$</td>
<td>$b \leftarrow {0,1}$</td>
<td>$b \leftarrow {0,1}$</td>
<td>$b \leftarrow {0,1}$</td>
</tr>
</tbody>
</table>

\begin{verbatim}
\textbf{First} \< \textbf{Second} \< \textbf{Third} \< \textbf{Fourth} \\
b \sample \bin \< b \sample \bin \< b \sample \bin \< b \sample \bin
\end{verbatim}

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 Section 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 notation. 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 \texttt{\>}. This mode is called \textit{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 $\textgreater$ 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></td>
</tr>
</tbody>
</table>

\begin{align*}
\text{send over } b
\end{align*}

\text{do something}

is generated by

\begin{verbatim}
\begin{pseudocodeblock}
 Alice & Bob \\
 b \leftarrow \{0, 1\} \\
 \xrightarrow{\text{send over } b} \\
 \text{do something}
\end{pseudocodeblock}
\end{verbatim}

### 4.1.1 Overriding The Tabbing Character

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

\begin{verbatim}
\renewcommand{\pctabname}{\myTab}
\begin{pseudocode}
 Alice \myTab \textbf{Bob} \\
 b \sample \bin \myTab \myTab \\
 \myTab \xrightarrow{\text{send over } b} \\
 \myTab \myTab \text{do something}
\end{pseudocode}
\end{verbatim}

### 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 $\backslash[\text{spacing}]$ to specify the spacing between two lines or of $\backslash[\text{\hline}]$ to insert a horizontal rule. In order to achieve the same effect within the pseudocode command you can use $\backslash[\text{[spacing]}[\text{\hline}]$. You can also use \pclb to get a line break which does not insert the additional alignment characters.
5 Protocols

Using tabbing, we can use \texttt{pseudocode} to also layout protocols such as

\begin{procedureblock}{My Protocol}{
\textbf{Alice} \> \textbf{Bob} \\
\textit{b} \sample \bin \> \> \\
\textit{send over } \textit{b} \> \> \\
\texti{do something} \\
\textit{send over sth. else} \\
\textit{finalize}
}\end{procedureblock}

which is generated as

\begin{verbatim}
\procedureblock{My Protocol}{
  \textbf{Alice} \> \textbf{Bob} \\
  b \sample \bin \> \> \\
  \sendmessageright*{\text{send over } b} \> \> \\
  \> \text{do something} \\
  \> \sendmessageleft*{\text{send over sth. else}} \\
  \text{finalize} 
}
\end{verbatim}

In order to get nicer message arrows use the commands \texttt{sendmessengeright*{message}}, \texttt{sendmessageleft*{message}}, and \texttt{sendmessengerightleft*{message}}. All three take an additional optional argument specifying the length of the arrow and all wrap their mandatory argument in an \texttt{aligned} environment.

To obtain granular control over how messages are set use the \texttt{sendmessage} and \texttt{sendmessage*} commands. These take two parameters, the first being the message style for the underlying TIKZ path (e.g., -> for messages to the right) and the second a key
value list of arguments. The difference between the starred version and the unstarred version is that the starred version wraps its labels in an \texttt{aligned} environment. Following is an example, that showcases various message options.

My Protocol

\begin{procedureblock}{My Protocol}{\texttt{My Protocol}}
\begin{itemize}
  \item \texttt{Alice} \> \> \texttt{Bob} \\
  \item $b \leftarrow \{0, 1\}$ \\
  \item \texttt{send over $b$} \\
  \item \texttt{do something} \\
  \item \texttt{send over sth. else} \\
  \item $a, b, c$ \\
  \item $c, d, e$ \\
  \item \texttt{foo} \\
  \item 1: you can also \\
  \item 2: use pseudocode \\
  \item \texttt{finalize}
\end{itemize}
\end{procedureblock}

\texttt{sendmessage} and \texttt{sendmessage*} support the following options:

\begin{itemize}
  \item \texttt{top} The content to display on top of the arrow.
  \item \texttt{bottom} The content to display below the arrow.
  \item \texttt{left} The content to display on the left of the arrow.
  \item \texttt{right} The content to display on the right of the arrow.
  \item \texttt{topstyle} The TIKZ style to be used for the top node.
  \item \texttt{bottomstyle} The TIKZ style to be used for the bottom node.
  \item \texttt{rightstyle} The TIKZ style to be used for the right node.
  \item \texttt{leftstyle} The TIKZ style to be used for the left node.
  \item \texttt{length} The length of the arrow.
  \item \texttt{style} The style of the arrow.
  \item \texttt{width} The width of the column
\textbf{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 Section 4).

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

\begin{procedureblock}{My Protocol}{%}
\begin{tabbing}
\textbf{Alice} \quad \textbf{Bob} \\
b \leftarrow \{0, 1\} \\
\> send over b \\
\> do something \\
\> send over sth. else \\
\> do something \\
\> left aligned \\
\> multiline message \\
\textbf{finalize} \quad \textbf{Alice} \quad \textbf{Bob}
\end{tabbing}
\end{procedureblock}

\textbf{Remark}. When using \texttt{\textbf{sendmessage}}* the tabbing character \& cannot be used. Instead use the \> command as defined within \texttt{\textbf{pseudocode}}.
5.2.1 Multiplayer Protocols

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

\begin{procedureblock}{Multiparty Protocol}{%
   \textbf{Alice} \hspace{0.5cm} \textbf{Bob} \hspace{0.5cm} \textbf{Charlie} \\
   work \\
   \hspace{0.5cm} \textbf{work} \\
   \hspace{0.5cm} \textbf{Work result} \hspace{0.5cm} \textbf{work} \\
   \hspace{0.5cm} \textbf{Work result} \hspace{0.5cm} \textbf{Bottom message} \hspace{0.5cm} \textbf{work} \\
   \hspace{0.5cm} \text{work} \\
   \hspace{0.5cm} \text{finalize} \\
%}
\end{procedureblock}

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

5.2.2 Divisions

You can use \pcintertext in order to divide protocols (or other pseudocode for that matter).

\begin{pcintertext}{dotted\hspace{0.5cm}center}{Division\hspace{0.5cm}Text}
\end{pcintertext}

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

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

My Protocol
\begin{procedureblock}{My Protocol}{
\textbf{Alice} \rightarrow \textbf{Bob} \\
b \leftarrow \{0, 1\} \\
\begin{center}
\begin{tabular}{c}
\text{send over } b \\
\hline
do something \\
\hline
\text{send over sth. else} \\
\hline
\text{finalize}
\end{tabular}
\end{center}
\end{procedureblock}

Which is generated as

\begin{verbatim}
\procedureblock{My Protocol}{% \\
  \textbf{Alice} \rightarrow \textbf{Bob} \\
b \leftarrow \{0, 1\} \\
  \begin{center}
  \begin{tabular}{c}
    \text{send over } b \\
    \hline
do something \\
    \hline
    \text{send over sth. else} \\
    \hline
  \end{tabular}
  \end{center}
\end{procedureblock}
\end{verbatim}
And using \texttt{pcrln} we obtain:

<table>
<thead>
<tr>
<th>My Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: Alice</td>
</tr>
<tr>
<td>2: $b \leftarrow {0,1}$</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>3: finalize</td>
</tr>
</tbody>
</table>

This is generated as

```
\begin{procedureblock}{My Protocol}{%}
  \textbf{Alice} \, \textbf{Bob} \\
  b \sample \bin \sendmessageright*{send over \, b} \pcrln\sendmessageleft*{send over sth. else} \pcrln
  \text{finalize} \, \text{do something} \ \text{send over sth. else} \cup \pcrln\cup
\end{procedureblock}
```

### 5.3.1 Separators

The commands \texttt{pclnseparator} and \texttt{pcrlnseparator} define the separators between code and line number. By default the left separator is set to (:) colon and the right separator is set to an empty string.

### 5.3.2 Spacing

Spacings after the left separator and in front of the right separator can be controlled by \texttt{pclnspace} and \texttt{pclnrspace} which are set to 1em and 0.5em, respectively.

### 5.4 Sub Protocols

Use the \texttt{subprocedure} environment to also create sub protocols.
My Protocol

Alice  Bob

\[
b \leftarrow \{0, 1\}
\]

\[
\begin{array}{c}
\text{send over } b \\
\end{array}
\]

do something

\[
\begin{array}{c}
\text{Subprotocol} \\
\end{array}
\]

\[
\begin{array}{c}
\text{Charlie} \\
\end{array}
\]

\[
\begin{array}{c}
\text{something more} \\
\end{array}
\]

\[
\begin{array}{c}
\text{message} \\
\end{array}
\]

\[
\begin{array}{c}
\text{some processing} \\
\end{array}
\]

\[
\begin{array}{c}
\text{more processing} \\
\end{array}
\]

\[
\begin{array}{c}
\text{message} \\
\end{array}
\]

\[
\begin{array}{c}
\text{message} \\
\end{array}
\]

\[
\begin{array}{c}
\text{message} \\
\end{array}
\]

\[
\begin{array}{c}
\text{message} \\
\end{array}
\]

\[
\begin{array}{c}
\text{message} \\
\end{array}
\]

\[
\begin{array}{c}
\text{send over sth. else} \\
\end{array}
\]

\[
\begin{array}{c}
\text{finalize} \\
\end{array}
\]
6 Game-Based Proofs

6.1 Basics

Besides displaying pseudocode the package also comes with commands to help present game-based proofs. The \texttt{gameproof} environment wraps the pseudocode block of a game-based proof.

```latex
\begin{gameproof}
\begin{pchstack}[space=1em, center, boxed]
\gameprocedure[linenumbering, mode=text]{% 
  Step 1  \\
  Step 2 }
\gameprocedure[mode=text]{% 
  Step 1  \\
  Step 2 }
\end{pchstack}
\end{gameproof}
```

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

<table>
<thead>
<tr>
<th>Game\textsubscript{1}(\textit{n})</th>
<th>Game\textsubscript{2}(\textit{n})</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: Step 1</td>
<td>Step 1</td>
</tr>
<tr>
<td>2: Step 2</td>
<td>Step 2</td>
</tr>
</tbody>
</table>

by using

```latex
\begin{gameproof}
\begin{pchstack}[space=1em, center, boxed]
\gameprocedure[linenumbering, mode=text]{% 
  Step 1  \\
  Step 2 }
\gameprocedure[mode=text]{% 
  Step 1  \\
  Step 2 }
\end{pchstack}
\end{gameproof}
```

For discussing individual games, cryptocode provides the \texttt{\textbackslash pgame} command which without argument prints \texttt{Game} and with (optional) argument \texttt{\textbackslash pgame[n]} prints \texttt{Game}\textsubscript{n}.

6.1.1 Highlight Changes

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

```latex
\begin{gameproof}
\begin{pchstack}[space=1em, center, boxed]
\gameprocedure[linenumbering, mode=text]{% 
  Step 1  \\
  Step 2 }
\gameprocedure[mode=text]{% 
  Step 1  \\
  Step 2 }
\gamechange{Step 2}
\end{pchstack}
\end{gameproof}
```

```latex
\begin{gameproof}
\begin{pchstack}[space=1em, center, boxed]
\gameprocedure[linenumbering, mode=text]{% 
  Step 1  \\
  Step 2 }
\gameprocedure[mode=text]{% 
  Step 1  \\
  Step 2 }
\gamechange{Step 2}
\end{pchstack}
\end{gameproof}
```
The background color can be controlled by redefining \texttt{\gamechangecolor} which by default is defined as

\begin{verbatim}
\definecolor{gamechangecolor}{gray}{0.90}
\end{verbatim}

**Remark.** Note that \texttt{\gamechange} is always in text mode.

### 6.1.2 Boxed Games

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

```
\begin{gameproof}
\begin{pchstack}[ space=1em, boxed , center ]
\gameprocedure[ linenumbering ]{
\text{Step 1} \\
\text{Step 2}
}
\tbxgameprocedure{
\text{Step 1}; \pcbox{\text{Alternative step 1}} \\
\gamechange{\text{Step 2 is different}}
}
\gameprocedure{
\text{Step 1} \\
\text{\gamechange{Step 2}}
}
\end{pchstack}
\end{gameproof}
```

### 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 \texttt{PRG}. To give a hint within the pseudocode that the difference between two games is down to “something” you can use the \texttt{\addgamehop} command.

```
\addgamehop{startgame}{endgame}{options}
```

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

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

48
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

\begin{gameproof}
\begin{pchstack}[center, space=2em]
\text{Step 1} \\
\text{Step 2}
\end{pchstack}
\addstartgamehop{hint=\footnotesize some hint, edgestyle=}
\addgamehop{1}{2}{hint=\footnotesize some hint}
\addendgamehop{hint=\footnotesize some outgoing hint, edgestyle=}
\end{gameproof}

6.1.4 Numbering and Names

By default the \texttt{gameproof} environment starts to count from 1 onwards. Its optional parameters allow you to specify a custom name for the game as well as defining the starting number.
The following parameters are available which, as usual, are provided in a key=value based form.

\textbf{nr} The starting number minus 1. Thus, when setting nr=5, the first game will be Game_6.

\textbf{name} The name for the game

\textbf{arg} The argument to be used for the game.

\begin{gameproof}[nr=5, name=\mathsf{MyGame}, arg=(1^n)]
\begin{pchstack}[center, space=2em]
\gameprocedure[linenumbering]{
\text{Step 1} \\
\text{Step 2} \\
\text{Step 1} \\
\gamechange{Step 2 is different}
}
\end{pchstack}
\addstartgamehop{hint=\footnotesize some ingoing hint, edgestyle=}
\addgamehop{6}{7}{hint=\footnotesize some hint}
\addendgamehop{hint=\footnotesize some outgoing hint, edgestyle=}
\end{gameproof}

6.1.5 Default Name and Argument

The default name and argument are controlled via the commands $\texttt{pcgamename}$ and $\texttt{gameprocedurearg}$.

\begin{center}
\begin{tabular}{|l|l|}
\hline
\textbf{Command} & \textbf{Default} \\
\hline
\texttt{pcgamename} & $\mathsf{Game}$ \\
\texttt{gameprocedurearg} & $\texttt{secpar}$ \\
\hline
\end{tabular}
\end{center}

6.1.6 Bi-Directional Games

You can use the $\texttt{bxgameprocedure}$ to generate games for going in two directions. Use the $\texttt{addloopgamehop}$ to add the gamehop in the middle.

\begin{gameproof}
\begin{pchstack}[center, space=2em]
\text{Game}_1(n) & \text{Game}_2(n) \\
\text{Game}_3(n) & \text{Game}_4(n) \\
\text{Step 1; Alternative} & \text{Step 1; Alternative} \\
\text{Step 2; Alternative} & \text{Step 2 is different}
\end{pchstack}
\addstartgamehop{hint=\footnotesize some ingoing hint, edgestyle=}
\addgamehop{6}{7}{hint=\footnotesize some hint}
\addendgamehop{hint=\footnotesize some outgoing hint, edgestyle=}
\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 \texttt{\gameprocedure}. For this you can define the default arguments by calling \texttt{\setgameproceduredefaultstyle} to for example:

\begin{verbatim}
\setgameproceduredefaultstyle{beginline=&vphantom{\bin^\mathrm{prg}\_prg}}
\end{verbatim}

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

\begin{itemize}
\item \texttt{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.
\item \texttt{MyGame}_4(n): This is the fourth game. The arrow at the side indicates the reduction target.
\end{itemize}

The above example is generated as

\texttt{\begin{gamedescription}[name=MyGame, nr=2]}\texttt{\describegame}
\begin{itemize}
\item 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.
\end{itemize}
\texttt{\describegame[inhint=reduction target]}
\begin{itemize}
\item This is the second game. The arrow at the side indicates the reduction target.
\end{itemize}\texttt{\end{gamedescription}}

The \texttt{gamedescription} environment takes an optional argument to specify name and counter (defaults to 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}.

\texttt{inhint} Displays an ingoing arrow to denote the reduction target for this game hop.
**length**  Allows to control the length of the arrow.

**nodestyle**  Allows to control the style of the node.

**hint**  Instead of having an ingoing arrow, this adds an outgoing arrow.
7 Black-Box Reductions

The cryptocode package comes with support for drawing basic black box reductions. A reduction always takes 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 \texttt{bbrenv} environment (where bbr is short for black-box reduction) which takes a single \texttt{bbrbox} environment plus some additional commands.

Following is a simple example with a single (black)box and some code plus inputs/outputs:

\begin{verbatim}
step 1
step 2
for some condition do
  step 3
\end{verbatim}

This box is generated as

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

The commands \texttt{bbrinput} and \texttt{bbroutput} allow to specify input and output for the latest \texttt{bbrenv} environment. The optional parameter for the \texttt{bbrenv} environment allows to specify leading and trailing space (this may become necessary when using inputs and outputs). For this provide \texttt{aboveskip} and \texttt{belowskip} keys. (Note that in an earlier version of cryptocode you could write \texttt{\begin{bbrenv}[]{A}}. While this format is still supported it should be regarded deprecated.) The single mandatory argument to the bbrenv environment needs to specify a unique identifier (unique for the current reduction). This id is used as an internal TIKZ node name (\url{https://www.ctan.org/pkg/pgf}).

\begin{verbatim}
\begin{bbrenv}[options]{UNIQUE IDENTIFIER}
% deprecated version
\begin{bbrenv}{UNIQUE IDENTIFIER}[vspace before][vspace after]
\end{bbrenv}
\end{verbatim}
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.

```latex
\begin{tikzpicture}
  \node (A) at (0,0) {Box Name};
  \node (input) [above=of A] {input};
  \node (output) [below=of A] {output};
  \draw [fill=blue] (A.north) circle (4pt);
  \draw [fill=blue] (A.west) circle (4pt);
  \draw [fill=blue] (A.east) circle (4pt);
  \draw [fill=blue] (A.south) circle (4pt);
\end{tikzpicture}
```

The `bbrbox` takes as single argument a comma separated list of key value pairs. In the example we used

```
\begin{bbrbox}[name=Box Name]
\pseudocode{
  \text{step 1} \\
  \text{step 2} \\
  \pcfor \text{some condition} \pcdo \\
  \pcind\text{step 3}
}
\end{bbrbox}
```

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`. 
### 7.2 Messages and Queries

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

```latex
\begin{brello}
\begin{bbrbox}[name=Box Name]
\begin{pseudocode}
\text{step 1} \\
\text{step 2} \\
\textbf{for some condition do} \\
\text{step 3} \\
\end{pseudocode}
\end{bbrbox}
\begin{bbrbox}[name=Inner Box]
\begin{pseudocode}
\text{inner step 1} \\
\text{inner step 2} \\
\end{pseudocode}
\end{bbrbox}
\begin{pseudocode}
\text{step 4} \\
\text{step 5} \\
\end{pseudocode}
\end{brello}
```

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 list. The `tofrom` and `fromto` variants draw two messages (back and forth) that are
more compactly set together. Here usually, the fist message should be drawn on top (\texttt{top=Label}) while the second message should be drawn on the bottom (\texttt{bottom=Label}).

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

\begin{verbatim}
\bbmsgto{top=Top Label, side=Side Label}
\end{verbatim}

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

\begin{verbatim}
\bbmsgto{top=Top Label, side={Side, Label}}
\end{verbatim}

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{
  step 1 \\
  step 2 \\
  \textbf{for some condition do} \\
  step 3 \\
  $m_0, m_1$ \\
  \textbf{inner step 1} \\
  \textbf{inner step 2} \\
  $q$
}
\end{bbrbox}
\bbmsgto{top={$m_0, m_1$}}
\bbmsgfrom{top=$q$}
\bbqrytofrom{top={$m_0, m_1$}}{bottom=$q$}
\end{bbrenv}

\begin{bbrenv}{B}
\begin{bbrbox}[name=Inner Box]
pseudocode{
  \textbf{inner step 1} \\
  \textbf{inner step 2}
}
\end{bbrbox}
\end{bbrenv}
\end{verbatim}
7.2.1 Options

Following is a list of all available options. Remember that underneath the reduction commands is a TIKZ image (https://www.ctan.org/pkg/pgf/) and for each label position (top, side, bottom) a node is generated which can be further customized via low-level TIKZ.

- **top** Label on top
- **bottom** Label on the bottom
- **side** Label on the far side of the box. For challengers and oracles, on the side of the box.
- **oside** Label on the “other” side.
- **topstyle** Style for label on top
- **bottomstyle** Style for label on bottom
- **sidestyle** Style for label on side
- **osidestyle** Style for label on other side
- **edgestyle** Style for edge
- **length** Length of arrow
- **topname** Name for node on top
- **bottomname** Name for node on bottom
- **sidename** Name for node on side
- **osidename** Name for node on other side
- **aboveskip** Space before message
- **belowskip** Space after message
- **fixedoffset** Ignores automatic spacing and sets the message at the provided offset from the top.
- **fixedboffset** Ignores automatic spacing and sets the message at the provided offset from the bottom.
- **islast** Places the message at the bottom.
\begin{bbrenv}{A}
\begin{bbrbox}[name=Box Name]
\pseudocode{
\text{step 1} \text{ for some condition do} \\
\text{step 2} \\
\text{step 3} \\
\text{inner step 1} \\
\text{inner step 2} \\
}\pcfor \text{some condition} \pcdo \\
\pcind \text{step 3}
}
\end{bbrbox}
\bbrmsgto{top={$m_0,m_1$},side=Side Label , bottom=$b$, length=2cm, topstyle={draw , solid }, sidestyle={red}, bottomstyle={draw, dashed}}
\end{bbrenv}

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

7.2.2 First Message

The first message is offset by $\bbrfirstmessageoffset$ which defaults to $1\text{ex}$.

7.2.3 Add Space

If the spacing between messages is not sufficient you can use the $\bbrmsgspace$ and $\bbrqryspace$ commands to add additional space.
Note that for placing a message at the bottom, `islast` or fixed offsets often allow obtain more accurate results.
7.2.4 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}
\begin{pseudocode}{}
\text{Do something}
\end{pseudocode}
\end{bbrbox}
\end{bbrenv}
```

```
\begin{bbrenv}{B}
\begin{bbrbox}{name=Adversary, minheight=15ex, xshift=4cm}
\begin{bbrinput}{input}
\begin{bbroutput}{output}
\end{bbrenv}
```

\begin{center}
\begin{tikzpicture}
\node (input) at (0,0) {input};
\node (reduction) at (2,0) {Reduction};
\node (adversary) at (2,-1) {Adversary};
\node (output) at (0,-1) {output};
\draw[->] (input) -- (reduction);
\draw[->] (reduction) -- (adversary) node[midway, above] {$m$};
\draw[->] (adversary) -- (output) node[midway, below] {$\sigma$};
\end{tikzpicture}
\end{center}
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

```latex
\bbrloop{BeginLoop}{EndLoop}{center=$q$}
```

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.

```
\begin{bbrenv}{A}
\begin{bbrbox}[name=Reduction]
\pseudocode{
\text{Do something}
}
\end{bbrbox}
\end{bbrenv}
```

The \texttt{bbrloop} command supports the following parameters:

- **center**: Label displayed within the loop
- **left**: Label displayed left of the loop
- **right**: Label displayed right of the loop
- **centerstyle**: Style for center label
- **leftstyle**: Style for left label
- **rightstyle**: Style for right label
- **clockwise**: Loop going in clockwise direction
7.2.5 Intertext

If your reduction needs to do some extra work between queries use the \texttt{\textbackslash bbrmsgtxt} and \texttt{\textbackslash bbrqrytxt} commands.

```
1 \texttt{\textbackslash bbrmsgtxt \{options\} \{Text\}}
2 \texttt{\textbackslash bbrqrytxt \{options\} \{Text\}}
```

```
\begin{bbrenv}{A}
\begin{bbrbox}[name=Reduction]
\texttt{\textbackslash pseudocode{Do something}}
\texttt{\textbackslash text{Do something}}
\end{bbrbox}
\begin{bbrenv}{B}
\begin{bbrbox}[name=Adversary, minheight=12ex, xshift=4cm]
\texttt{\textbackslash bbrmsgto{top=$m$}}
\texttt{\textbackslash bbrmsgtxt{\texttt{\textbackslash pseudocode{do \textbackslash \ \ \ \ \ \ some \ \ \ \ \ \ work}}}}
\texttt{\textbackslash bbrmsgfrom{top=$\sigma$}}
\texttt{\textbackslash bbrqryto{top=$m$}}
\texttt{\textbackslash bbrqrytxt{\texttt{\textbackslash pseudocode{do \textbackslash \ \ \ \ \ \ some \ \ \ \ \ \ work}}}}
\texttt{\textbackslash bbrqryfrom{top=$\sigma$}}
\end{bbrbox}
\texttt{\textbackslash bbrinput{input}}
\texttt{\textbackslash bbroutput{output}}
\end{bbrenv}
```
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{bbrbox} environment within an \texttt{bbrenv} environment.

\begin{verbatim}
\begin{bbrenv}{A}
\begin{bbrbox}[name=Reduction]
\texttt{pseudocode}\{ \\
\texttt{Do something} \\
\}
\end{bbrbox}
\end{bbrenv}
\begin{bbrbox}[name=Adversary , minheight=3cm, xshift=4cm]
\end{bbrbox}
\bbrinput{input}
\bbroutput{output}
\begin{bbroracle}{OraA}
\begin{bbrbox}[name=Oracle 1]
\end{bbrbox}
\end{bbroracle}
\begin{bbroracle}{OraB}[ vdistance=2cm, hdistance=3cm]
\begin{bbrbox}[name=Oracle 2]
\end{bbrbox}
\end{bbroracle}
\end{bbrenv}
\end{verbatim}

Via the option “hdistance=length” and “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{bbroracleqryto}.

\begin{verbatim}
\bbroracleqryfrom{options}
\bbroracleqryto{options}
\end{verbatim}
Here options allow you to specify where the label goes (top, bottom). In addition you can use \texttt{oraclequeryspace} to generate extra space between oracle messages. Note that oracle messages need to be added after the closing \texttt{endoracle} 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 \texttt{brenv} environment using the \texttt{brcaller} environment. Challengers go behind the single \texttt{bbrbox} environment within an \texttt{brenv} 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 sketch for full domain hash.

\begin{bbrenv}{Red}
\end{bbrenv}
\begin{bbrbox}[name=\textsc{Reduction }$\bdv$]
\pseudocode{
  j \sample \{q\}
}
\vspace{2ex}
\emph{/∗ begin simulation */}
\begin{bbrenv}[aboveskip=2em]
\begin{bbrbox}[name=$\adv$,minheight=8.5cm, style={fill=black},namestyle={color=white}, xshift=3cm]
  \bbrinput{$\fk$}
  \bbroutput{$\sigma$}
  \bbrmsgfrom{top=$m_1$,afterskip=−0.5\baselineskip}
  \bbrmsgto{bottom=$\$$,afterskip=0.5\baselineskip}
  \bbrmsgvdots
  \bbrmsgfrom{top=$m_{j-1}$,beforeskip=0.5\baselineskip , afterskip=−0.5\baselineskip}
  \bbrmsgto{bottom=$\$$,afterskip=1.5\baselineskip}
  \bbrmsgfrom{top=$m_j $, afterskip=−0.5\baselineskip}
  \bbrmsgto{bottom=$y$, afterskip=1.5\baselineskip}
  \bbrmsgfrom{top=$m_{j+1}$,afterskip=−0.5\baselineskip}
  \bbrmsgto{bottom=$\$$,afterskip=0.5\baselineskip}
  \bbrmsgvdots
  \bbrmsgfrom{top=$m_q$, beforeskip=0.5\baselineskip , afterskip=−0.5\baselineskip}
  \bbrmsgto{bottom=$\$$}
  \begin{bbroracle}{Sign}
  \begin{bbrbox}[name=Sign ,namepos=center , style={draw},minheight=1cm]
  \end{bbrbox}
  \end{bbroracle}
  \begin{bbroracleqryto}{top=$m$}
  \begin{bbroracleqryfrom}{top=$\sigma$}
  \end{bbrenv}
\end{bbrbox}
\pcdraw{
  \node[ left=2cm of Adv. north west ] ( startsim ) {};
  \node[ left=2cm of Adv. south west ] (endsim) {};
  \draw[−>,thick] ( startsim ) −− (endsim);
  \node[ rotate=90, left =2.75cm of Adv.west , anchor=center ] () \textsc{Simulation of Random Oracle};
}
\emph{/∗ end simulation */}
\end{bbrbox}
\pseudocode{
y \gets \sigma
}
\bbrquerytobeforeskip=0.25cm,top={$($\fk , \ y$)$},side={\dbox{\pseudocode{\fk \ sample \fash.\kgen(\secparam) \ \ \ x \ \sample \bin^{\fash.\il(\secpar)} \ \ \ y \ \gets \fash.\eval(\fk , x)}\}}
\bbrquerytobeforeskip=11.75cm,side=\pseudocode{y \in \fash^{−1}(\fk , x)}}
\end{bbrenv}
8 Known Issues

8.1 Pseudocode KeepSpacing within Commands

The (experimental) “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 \texttt{pseudocode} command in an \texttt{fbox} or in a stacking environment such as \texttt{pchstack}. As a workaround for generating frame boxes you should hence use a package such as \texttt{mdframed} (https://www.ctan.org/pkg/mdframed) which provides a frame environment.

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

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

\begin{verbatim}
\newsavebox{\mypcbox}
\begin{lrbox}{\mypcbox}
\pseudocode [ space=keep ,mode=text ]{ Pseudocode with - spaces - }
\end{lrbox}
\fbox{\usebox{\mypcbox}}
\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 \texttt{\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 \texttt{pcfixhyperref} after \texttt{\begin{document}}.
9 Implementation

Following is the implementation of cryptocode. The source code documentation is a work in progress.

Note that most macros are prefixed with pc short for pseudocode. This is a general design choice to not conflict with macros defined by other packages. One exception are the macros defined via the various package options.

Load amsmath and mathtools early on, before defining various macros.

\RequirePackage{amsmath}
\RequirePackage{mathtools}

9.1 Package Options

Definitions of boolean flags used to determine whether or not to load amsfonts.

\newif\ifpc@opt@amsfonts
\newif\ifpc@opt@advantage
\newif\ifpc@opt@centernot

9.1.1 operators

Definitions of macros for the operators package option.

\DeclareOption{operators}{
7 \providecommand{\sample}{\hskip2.3pt{\gets\!\!\!\mbox{scriptsize\$\$}\normalsize}}\,}
8 \providecommand{\floor}{\pc@floor\{#1\}}
9 \providecommand{\tfloor}{\pc@floor{#1}}
10 \providecommand{\pc@floor}{\lfloor}{\rfloor}\{\lfloor\}\{\rfloor\}
11 \providecommand{\pc@ceil}{\lceil}{\rceil}\{\lceil\}\{\rceil\}
12 \providecommand{\Angle}{\pc@Angle\{#1\}}
13 \providecommand{\tAngle}{\pc@Angle{#1}}
14 \providecommand{\abs}{\pc@abs\{#1\}}
15 \providecommand{\tabs}{\pc@abs{#1}}
16 \providecommand{\norm}{\pc@norm\{#1\}}
17 \providecommand{\tnorm}{\pc@norm{#1}}
18 \providecommand{\concat}{\ensuremath{\|}}
19 \providecommand{\emptystring}{\ensuremath{\varepsilon}}
20 \DeclareMathOperator*{\argmax}{\text{arg\,max}}
21 \DeclareMathOperator*{\argmin}{\text{arg\,min}}
22 \DeclareMathOperator*{\pindist}{\text{pindist}}
23 \DeclareMathOperator*{\cindist}{\text{cindist}}
24 \DeclareMathOperator*{\sindist}{\text{sindist}}
25 \DeclareMathOperator*{\argmax}{\text{arg\,max}}
26 \DeclareMathOperator*{\argmin}{\text{arg\,min}}
27 \DeclareMathOperator*{\pindist}{\text{pindist}}
28 \DeclareMathOperator*{\cindist}{\text{cindist}}
29 \DeclareMathOperator*{\sindist}{\text{sindist}}
30 \DeclareMathOperator*{\concat}{\text{concat}}
31 \providecommand{\emptystring}{\ensuremath{\varepsilon}}
32 \providecommand{\emptystring}{\ensuremath{\varepsilon}}
33 \DeclareMathOperator*{\argmax}{\text{arg\,max}}
\DeclareMathOperator*{\argmin}{\text{arg}, \text{min}}

\%indistinguishability
\newcommand{\pindist}{\@pc@oset{\text{p}}{\lower.2ex\hbox{$\approx$}}}
\newcommand{\sindist}{\@pc@oset{\text{s}}{\lower.1ex\hbox{$\approx$}}}
\newcommand{\cindist}{\@pc@oset{\text{c}}{\lower.1ex\hbox{$\approx$}}}

9.1.2 adversary
\adversary
\begin{verbatim}
\DeclareOption{adversary}{
  \providecommand{\adversary}{\pcadvstyle{A}}
  \providecommand{\bdv}{\pcadvstyle{B}}
  \providecommand{\cdv}{\pcadvstyle{C}}
  \providecommand{\ddv}{\pcadvstyle{D}}
  \providecommand{\edv}{\pcadvstyle{E}}
  \providecommand{\mdv}{\pcadvstyle{M}}
  \providecommand{\pdv}{\pcadvstyle{P}}
  \providecommand{\rdv}{\pcadvstyle{R}}
  \providecommand{\sdv}{\pcadvstyle{S}}
}
\end{verbatim}

9.1.3 landau
\begin{verbatim}
\bigO
\smallO
\bigOmega
\smallOmega
\bigsmallO
\bigTheta
\orderOf
\end{verbatim}

9.1.4 probability
\begin{verbatim}
\probnname
\expectationname
\supportname
\tprob
\prob
\probsub
\probsublong
\tcondprob
\condprob
\tcondprobsub
\condprobsub
\texpect
\expect
\texpsub
\expsub
\end{verbatim}

The \textit{probability} package option defines various macros for typesetting probabilities. Sets flags \@pc@opt@amsfontstrue.
The \texttt{sets} option defines various macros for standard sets such as natural numbers \texttt{NN}, integers \texttt{ZZ}, complex numbers \texttt{CC}, rational numbers \texttt{QQ}, real numbers \texttt{RR}, and integers \texttt{PP}.

As we usually work with bit strings, the macro \texttt{bin} defines the set \{0,1\}. Sets the \texttt{flags} \texttt{\@pc@opt@amsfontstrue}.

The style can be configured via \texttt{\pcsetstyle}.

\set\sequence\bin\NN\ZZ\CC\QQ\RR\PP\FF\GG\choice
\begin{verbatim}
\providecommand\NN{\pcsetstyle{N}}
\providecommand\ZZ{\pcsetstyle{Z}}
\providecommand\CC{\pcsetstyle{C}}
\providecommand\QQ{\pcsetstyle{Q}}
\providecommand\RR{\pcsetstyle{R}}
\providecommand\PP{\pcsetstyle{P}}
\providecommand\FF{\pcsetstyle{F}}
\providecommand\GG{\pcsetstyle{G}}
\end{verbatim}

\begin{verbatim}
\set\sequence\bin\NN\ZZ\CC\QQ\RR\PP\FF\GG\choice
\end{verbatim}

9.1.5 \texttt{sets}
9.1.6 noamsfonts

Package option noamsfonts ensures that ams fonts are not loaded. For this flag \@pc@opt@amsfontsfalse is set to false.
\DeclareOption{noamsfonts}{
\@pc@opt@amsfontsfalse
}

9.1.7 notions

The notion package option defines various cryptographic security notions. The style to be can be defined via \pcnotionstyle.
\DeclareOption{notions}{
\providecommand{\indcpa}{\pcnotionstyle{IND-CPA}}
\providecommand{\indcca}{\pcnotionstyle{IND-CCA}}
\providecommand{\indccai}{\pcnotionstyle{IND-CCA1}}
\providecommand{\indccaii}{\pcnotionstyle{IND-CCA2}}
\providecommand{\priv}{\pcnotionstyle{PRIV}}
\providecommand{\ind}{\pcnotionstyle{IND}}
\providecommand{\indcda}{\pcnotionstyle{IND-CDA}}
\providecommand{\prvcda}{\pcnotionstyle{PRV-CDA}}
\providecommand{\prvrcda}{\pcnotionstyle{PRV\$-CDA}}
\providecommand{\kiae}{\pcnotionstyle{KIAE}}
\providecommand{\kdae}{\pcnotionstyle{KDAE}}
\providecommand{\mle}{\pcnotionstyle{MLE}}
\providecommand{\uce}{\pcnotionstyle{UCE}}
\providecommand{\eufcma}{\pcnotionstyle{EUF-CMA}}
\providecommand{\eufnacma}{\pcnotionstyle{EUF-naCMA}}
\providecommand{\seufcma}{\pcnotionstyle{SUF-CMA}}
\providecommand{\eufko}{\pcnotionstyle{EUF-KO}}
}

9.1.8 logic

\AND
\OR
\NOR
\NOT
\AND
\XOR
\XNOR
\xor
\false
\true

\notimplies
The \texttt{ff} option defines macros for function families. Algorithms are typeset via \texttt{pcalgsstyle}.

\begin{verbatim}
\providecommand{\kgen}{\pcalgsstyle{KGen}}
\providecommand{\pgen}{\pcalgsstyle{Pgen}}
\providecommand{\eval}{\pcalgsstyle{Eval}}
\providecommand{\invert}{\pcalgsstyle{Inv}}
\providecommand{\il}{\pcalgsstyle{il}}
\providecommand{\ol}{\pcalgsstyle{ol}}
\providecommand{\kl}{\pcalgsstyle{kl}}
\providecommand{\nl}{\pcalgsstyle{nl}}
\providecommand{\rl}{\pcalgsstyle{rl}}
\end{verbatim}

\subsection*{9.1.10 \texttt{mm} (machine models)}

\begin{verbatim}
\providecommand{\CRKT}{\pcmachinemodelstyle{C}}
\providecommand{\TM}{\pcmachinemodelstyle{M}}
\providecommand{\PROG}{\pcmachinemodelstyle{P}}
\providecommand{\uTM}{\pcmachinemodelstyle{UM}}
\providecommand{\uC}{\pcmachinemodelstyle{UC}}
\providecommand{\uP}{\pcmachinemodelstyle{UEval}}
\providecommand{\csize}{\pcmachinemodelstyle{size}}
\providecommand{\tmtime}{\pcmachinemodelstyle{time}}
\providecommand{\ppt}{\pcalgsstyle{PPT}}
\end{verbatim}

\subsection*{9.1.11 advantage}

The \texttt{advantage} option defines an \texttt{advantage} command for typesetting advantage declarations of adversaries.

\begin{verbatim}
\providecommand{\prover}{\pcalgsstyle{prover}}
\providecommand{\verifier}{\pcalgsstyle{verifier}}
\providecommand{\nizk}{\pcalgsstyle{nizk}}
\providecommand{\hash}{\pcalgsstyle{hash}}
\providecommand{\gash}{\pcalgsstyle{gash}}
\providecommand{\fash}{\pcalgsstyle{fash}}
\providecommand{\enc}{\pcalgsstyle{enc}}
\providecommand{\dec}{\pcalgsstyle{dec}}
\providecommand{\sig}{\pcalgsstyle{sig}}
\providecommand{\verify}{\pcalgsstyle{verify}}
\providecommand{\obf}{\pcalgsstyle{obf}}
\providecommand{\iO}{\pcalgsstyle{iO}}
\providecommand{\diO}{\pcalgsstyle{diO}}
\providecommand{\owf}{\pcalgsstyle{owf}}
\providecommand{\prf}{\pcalgsstyle{prf}}
\providecommand{\prp}{\pcalgsstyle{prp}}
\providecommand{\prg}{\pcalgsstyle{prg}}
\providecommand{\mac}{\pcalgsstyle{mac}}
\providecommand{\puncture}{\pcalgsstyle{puncture}}
\providecommand{\source}{\pcalgsstyle{source}}
\providecommand{\predictor}{\pcalgsstyle{predictor}}
\providecommand{\dist}{\pcalgsstyle{dist}}
\providecommand{\distinguisher}{\pcalgsstyle{distinguisher}}
\providecommand{\simulator}{\pcalgsstyle{simulator}}
\providecommand{\ext}{\pcalgsstyle{ext}}
\providecommand{\extractor}{\pcalgsstyle{extractor}}
\end{verbatim}

\subsection*{9.1.12 primitives}

The \texttt{primitives} package option defines various cryptographic primitives.
\DeclareOption{primitives}{

Zero knowledge
\providecommand{\prover}{\pcalgostyle{P}}
\providecommand{\verifier}{\pcalgostyle{V}}
\providecommand{\nizk}{\pcalgostyle{NIZK}}

Hash
\providecommand{\hash}{\pcalgostyle{H}}
\providecommand{\gash}{\pcalgostyle{G}}
\providecommand{\fash}{\pcalgostyle{F}}
\providecommand{\pad}{\pcalgostyle{pad}}

Encryption
\providecommand{\enc}{\pcalgostyle{Enc}}
\providecommand{\dec}{\pcalgostyle{Dec}}

Signatures
\providecommand{\sig}{\pcalgostyle{Sig}}
\providecommand{\sign}{\pcalgostyle{Sign}}
\providecommand{\verify}{\pcalgostyle{Vf}}

Obfuscation
\providecommand{\obf}{\pcalgostyle{O}}
\providecommand{\iO}{\pcalgostyle{iO}}
\providecommand{\diO}{\pcalgostyle{diO}}

One-wayness
\providecommand{\owf}{\pcalgostyle{OWF}}
\providecommand{\owp}{\pcalgostyle{OWP}}
\providecommand{\tdf}{\pcalgostyle{TF}}
\providecommand{\inv}{\pcalgostyle{Inv}}
\providecommand{\hcf}{\pcalgostyle{HC}}

Pseudorandomness
\providecommand{\prf}{\pcalgostyle{PRF}}
\providecommand{\prp}{\pcalgostyle{PRP}}
\providecommand{\prg}{\pcalgostyle{PRG}}

Message authentication code
\providecommand{\mac}{\pcalgostyle{MAC}}

Puncture
\providecommand{\puncture}{\pcalgostyle{Puncture}}

Misc
\providecommand{\source}{\pcalgostyle{S}}
\providecommand{\predictor}{\pcalgostyle{P}}
\providecommand{\sam}{\pcalgostyle{Sam}}
\providecommand{\dist}{\pcalgostyle{D}}
\providecommand{\distinguisher}{\pcalgostyle{Distinguish}}
\providecommand{\simulator}{\pcalgostyle{Sim}}
\providecommand{\ext}{\pcalgostyle{Ext}}
}\endinput
9.1.13 oracles

The \textit{oracles} package option defines macros for typesetting oracles.

\begin{verbatim}
\DeclareOption{oracles}{
  \providecommand{\Oracle}[1]{\pcalgostyle{O{#1}}}
  \def\oracle#1{\bgroup\oracle@}
  \newcommand{\oracle@}[1]{\ifthenelse{\equal{#1}{}}{\oracle@@{O}}{\oracle@@{#1}}}
  \def\oracle@@#1{\pcoraclestyle{#1}\egroup}
  \providecommand{\ro}{\pcoraclestyle{RO}}}
\end{verbatim}

9.1.14 events

The \textit{events} package option defines macros for typesetting events (probabilistic). Also defines \texttt{\textbackslash bad} as a \textit{bad event} often used in game based proofs.

\begin{verbatim}
\DeclareOption{events}{
  \providecommand{\event}[1]{\ensuremath{\mathsf{#1}}}
  \providecommand{\nevent}[1]{\ensuremath{\overline{\event{#1}}}}
  \providecommand{\bad}{\ensuremath{\event{bad}}}
  \providecommand{\nbad}{\ensuremath{\nevent{bad}}}
}\end{verbatim}

9.1.15 complexity

The \textit{complexity} package option defines various complexity classes. The style can be adjusted via \texttt{\pccomplexitystyle}.

\begin{verbatim}
\DeclareOption{complexity}{
  \providecommand{\complclass}[1]{\pccomplexitystyle{#1}}
  \providecommand{\cocomplclass}[1]{\pccomplexitystyle{co}\pmathhyphen{}\pccomplexitystyle{#1}}
  \providecommand{\npol}{\pccomplexitystyle{NP}}
  \providecommand{\conpol}{\cocomplclass{NP}}
  \providecommand{\pol}{\pccomplexitystyle{P}}
  \providecommand{\bpp}{\pccomplexitystyle{BPP}}
  \providecommand{\ppoly}{\ensuremath{\pol/\mathrm{poly}}}
  \providecommand{\AM}{\pccomplexitystyle{AM}}
  \providecommand{\coAM}{\cocomplclass{AM}}
  \providecommand{\AC}[1]{\ensuremath{\ifthenelse{\equal{#1}{}}{\pccomplexitystyle{AC}}{\pccomplexitystyle{AC}^{#1}}}}
  \providecommand{\NC}[1]{\ensuremath{\ifthenelse{\equal{#1}{}}{\pccomplexitystyle{NC}}{\pccomplexitystyle{NC}^{#1}}}}
  \providecommand{\TC}[1]{\ensuremath{\ifthenelse{\equal{#1}{}}{\pccomplexitystyle{TC}}{\pccomplexitystyle{TC}^{#1}}}}
  \providecommand{\PH}{\pccomplexitystyle{PH}}
  \providecommand{\csigma}[1]{\pccomplexitystyle{\Sigma}^p_{#1}}
  \providecommand{\cpi}[1]{\pccomplexitystyle{\Pi}^p_{#1}}
  \providecommand{\cosigma}[1]{\cocomplclass{\Sigma}^p_{#1}}
  \providecommand{\copi}[1]{\cocomplclass{\Pi}^p_{#1}}}
\end{verbatim}
The \textit{asymptotics} package option defines "polynomials" \(c\) (\texttt{cc}), \(e\) (\texttt{ee}), \(k\) (\texttt{kk}), \(m\) (\texttt{mm}), \(n\) (\texttt{nn}), \(p\) (\texttt{pp}), and \(q\) (\texttt{qq}) as well as macros \texttt{negl} and \texttt{poly}.

\begin{verbatim}
\providecommand{\negl}{\pcpolynomialstyle{negl}\ifthenelse{\equal{#1}{}}{}{\pc@olrk*{#1}}}
\providecommand{\poly}{\pcpolynomialstyle{poly}\ifthenelse{\equal{#1}{}}{}{\pc@olrk*{#1}}}
\def\pp{\bgroup\pp@}
\newcommand{\pp@}[1]{\ifthenelse{\equal{#1}{}}{\pp@@{p}}{\pp@@{#1}}}
\def\pp@@#1{\pcpolynomialstyle{#1}\egroup}
\providecommand{\cc}{\pcpolynomialstyle{c}}
\providecommand{\ee}{\pcpolynomialstyle{e}}
\providecommand{\kk}{\pcpolynomialstyle{k}}
\providecommand{\mm}{\pcpolynomialstyle{m}}
\providecommand{\nn}{\pcpolynomialstyle{n}}
\providecommand{\qq}{\pcpolynomialstyle{q}}
\providecommand{\rr}{\pcpolynomialstyle{r}}
\end{verbatim}

\textbf{keys}

\(\texttt{pk}\) The \textit{keys} package option defines various "keys" such as a symmetric and general purpose \(k\) (\texttt{key}) or an asymmetric key pair \(pk\), \(sk\) (\texttt{pk} and \texttt{sk})

\begin{verbatim}
\providecommand{\pk}{\pckeystyle{pk}}
\providecommand{\vk}{\pckeystyle{vk}}
\providecommand{\sk}{\pckeystyle{sk}}
\providecommand{\hk}{\pckeystyle{hk}}
\providecommand{\gk}{\pckeystyle{gk}}
\providecommand{\fk}{\pckeystyle{fk}}
\providecommand{\st}{\pckeystyle{st}}
\end{verbatim}

\(\texttt{state}\)

\begin{verbatim}
\def\key{\bgroup\key@}
\newcommand{\key@}[1]{\ifthenelse{\equal{#1}{}}{\key@@{k}}{\key@@{#1}}}
\def\key@@#1{\pckeystyle{#1}\egroup}
\providecommand{\hk}{\pckeystyle{hk}}
\providecommand{\gk}{\pckeystyle{gk}}
\providecommand{\fk}{\pckeystyle{fk}}
\providecommand{\st}{\pckeystyle{st}}
\end{verbatim}

\(\texttt{state}\)

\textbf{Security parameter}

\(\texttt{SECPAR}\) The \(n\) option defines security parameter macros \texttt{secpar} and \texttt{secparam} using \(n\). See also "\textit{lambda}" package option.

\begin{verbatim}
\providecommand{\SECPAR}{\ensuremath{{N_0}}}
\providecommand{\secpar}{\ensuremath{\text{Secpar}}}\providecommand{\secparam}{\ensuremath{\text{Secparam}}}
\end{verbatim}
The \n option defines security parameter macros \secpar and \secparam using \lambda. See also “n” package option.

\textbf{9.2 Preamble and Option Parsing}

Print a warning in case an undefined package option is provided.

By default, only the \n option (security parameter as \(n\) and \(1^n\)) is loaded

We are now ready to process all package options

The cryptocode package depends on various external packages which are loaded next.

Note that the amsfonts package is optional and can be disabled via the noamsfonts package option.

Note that amsmath and mathtools have been loaded already earlier.

The \advantage option defines an \advantage command for typesetting advantage declarations of adversaries.


9.3 Global Macros

9.3.1 Styles

Definition of styles for algorithms, sets, complexity classes, polynomials, adversaries, notions, keys, and machine models.

9.3.2 Order of Growth

Define order of growth helper macros. These are optionally defined depending on the loaded package options.

9.3.3 Spacing

Control the spacing before (resp. after) pseudocode and stacking blocks both vertically and horizontally.

9.3.4 Keywords and Highlighting

Commands for highlighting primary and secondary keywords. Both commands take an optional first parameter to control spacing.

All predefined (highlightable) keywords.
9.3.5 Misc
\texttt{\textbackslash pcmathhyphen} Definition of a hyphen to be used within math formulas.
\texttt{\textbackslash mathchardef\pcmathhyphen = "2D}

\texttt{\textbackslash pccomment} Programming style line comment prefixing the comment with a double slash. An optional first parameter allows to control the spacing before the comment (defaults to 1em).
\texttt{\textbackslash pclinecomment} An optional first parameter allows to control the spacing before the comment (defaults to 1em).

9.4 Internal Helper Functions
\texttt{\@expandedsetkeys} Calls \texttt{\textbackslash setkeys} from the xkeyval package but before expands argument number 4. Arguments \{\texttt{\textbackslash (families)}\} \{\texttt{\textbackslash (na)}\} \{\texttt{\textbackslash (first set of keys)}\} \{\texttt{\textbackslash (keys to be expanded)}\} \{\texttt{\textbackslash (final set of keys)}\}
\texttt{\@expandedsetkeys[5]} \texttt{\textbackslash expandafter\expandedsetkeys[0]@\expandafter[\#4][\#1][\#2][\#3][\#5]}
\newcommand{\@pc@settowidthofalign}[2]{\setbox\z@ = \vbox{\@pseudocodecodesize\begin{flalign*}#2\end{flalign*}\ifmeasuring@\else\global\let\got@maxcolwd\maxcolumn@widths\fi}\end{flalign*}}

\newcommand{\@pc@settowidthofaligned}[2]{\settowidth{#1}{\@pseudocodesubcodesize$\begin{aligned}#2\end{aligned}$}}

\def\@pc@ifdraft{\ifdim\overfullrule>\z@\expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}

\newcommand{\@pc@executeblindly}[1]{\setbox\z@ = \vbox{#1}}

\AtBeginDocument{\let\@pc@original@label\ltx@label}

\providecommand{\pcfixhyperref}{\global\let\textlabel\label\global\let\@pc@original@label\textlabel}

9.5 Stacking

In the following we define two stacking environments \texttt{pchstack} and \texttt{pcvstack} to layout multiple pseudocode blocks.
9.5.1 Manual Spacing

\newcommand{\pchspace}[1][1em]{\hspace{#1}}
\newcommand{\pcvspace}[1][\baselineskip]{\par\vspace{#1}}

9.5.2 Misc

\@pc@stackdepth
Counter to keep track of nesting level of stacks.
\@pc@incstackdepth
\@pc@decstackdepth

9.5.3 Stacking Options

\@pc@centerstack
Allows to center the stack.
\@pc@centerstack
\define@key{pcstack}{center}[true]{\ifthenelse{\equal{#1}{true}}{\renewcommand{\@pc@centerstack}{true}}{\renewcommand{\@pc@centerstack}{false}}}

\@pc@boxedstack
Allows to draw a box around the stack.
\@pc@boxedstack
\define@key{pcstack}{boxed}[true]{\ifthenelse{\equal{#1}{true}}{\renewcommand{\@pc@boxedstack}{true}}{\renewcommand{\@pc@boxedstack}{false}}}

\@pc@noindentstack
Allows to draw a box around the stack.
\@pc@noindentstack
\define@key{pcstack}{noindent}[true]{\ifthenelse{\equal{#1}{true}}{\renewcommand{\@pc@noindentstack}{true}}{\renewcommand{\@pc@noindentstack}{false}}}

\@pc@inlinestack
Allows to keep the pchstack inline and not creating a paragraph.
\@pc@inlinestack
\define@key{pcstack}{inline}[true]{\ifthenelse{\equal{#1}{true}}{\renewcommand{\@pc@inlinestack}{true}}{\renewcommand{\@pc@inlinestack}{false}}}

\@pc@stackspace
Introduces horizontal (resp. vertical) space in-between pseudocode blocks in stacking environments.
\@pc@stackspace
\@pc@centerstack
\@pc@boxedstack
\@pc@noindentstack
\@pc@inlinestack

\@pc@stackspace\@pc@centerstack\@pc@boxedstack
\@pc@noindentstack\@pc@inlinestack

aboveskip
By default \pcaboveskip is applied on the outer most stacking environment. Can be overridden using aboveskip.
\@pc@applyaboveskip
\@pc@spaceunlessstacking
\@pc@applyaboveskip
By default \pcbelowskip is applied on the outer most stacking environment. Can be overridden using belowskip.

\pcbeforehstackskip \pcafterhstackskip Allows adding global skips before and after \pchstack blocks.

\pcboxedstack For \pchstack and \pcvstack we use a box to store temporary results.

9.5.4 The Stacking Environments

pccenter

A stacking environment for horizontally stacked pseudocode blocks.

pchstack
pchstack  A stacking environment for vertically stacked pseudocode blocks.

\begin{pcvstack}[]
\end{pcvstack}
9.6 The pseudocode command

Define internal lengths used for measurements within pseudocode.
\newlength{\@pc@minipage@length}
\newlength{\@pc@alt@minipage@length}
\newlength{\@pc@length@tmp@width@vstack}

Define flags used in game based proofs.
\newcommand{\@withingame}{false}
\newcommand{\@withinbxgame}{false}
\newcommand{\@withingamedescription}{false}
\@bxgameheader

Define a placeholder command which will take the current game header.
\newcommand{\@bxgameheader}{}
\@pc@beginnewline
An internal helper that is called at the beginning of each new line.
\newlength{\@pseudocodecodeminlineheight@len}
\newcommand{\@pc@beginnewline}{\@pseudocodecodeatbeginline\@pseudocodelinenumber\@pc@and\@pcln@stephiddenlncnt%}
\setlength{\@pseudocodecodeminlineheight@len}{\@pseudocodecodeminlineheight}%
\vphantom{\rule{0.5ex-0.5\@pseudocodecodeminlineheight@len}{\@pseudocodecodeminlineheight@len}}%
%checkspace
\ifthenelse{\equal{\@pseudocodespace}{auto}}{\expandafter\pcind\expandafter{\value{@pc@indentationlevel}}%}{%reset column counter
\setcounter{pccolumncounter}{2}%beginmode
\@pc@modebegin}
\@pc@and@wrap@end
\@pc@and@wrap@start
Every pseudocode line is wrapped in between \@pc@and@wrap@start and \@pc@and@wrap@end.
\newcommand{\@pc@and@wrap@start}{\@pc@beginnewline}
\newcommand{\@pc@and@wrap@end}{\@pc@modeend&\@pseudocodecodeatendline}
\@pc@and
An internal helper to store the ampersand. As this is a special character this is the easiest
in order to place custom alignment tags.
\newcommand{\@pc@and}{&}
\pctabname
\pcdbltabname
\pcindentname
Shorthands for alignment tabs and indentation. These are defined only within the pseu-
docode scope.
\newcommand{\pctabname}{>}
\newcommand{\pcdbltabname}{<}
\newcommand{\pcindentname}{t}

The following commands handle line numbering within the pseudocode command.
The pseudocode command itself does need to do some counter magic. We start with
a definition of various helper counters. The H version of counters is needed to make hyperref happy

\newcounter{pclinenumber}
\newcounter{Hpclinenumber}
\newcounter{@pclinenumber}
\newcounter{H@pclinenumber}
\newcounter{@pclinenumbertmp}
\newcounter{pcgamecounter}
\newcounter{Hpcgamecounter}
\newcounter{pcrlinenumber}
\newcounter{Hpcrlinenumber}
\newcounter{pcrlinenumber} TMP

The following implements some counter magic. When using automatic linenumbering line numbers are nicely aligned before the first alignment tag. This, however confuses hyperref and we thus have a second counter that is updated after the first tag. This is done with the \@pcln@stephiddenlncnt

\renewcommand{\the@pclinenumber}{\thepclinenumber}
\providecommand{\@pcln@stephiddenlncnt}{\refstepcounter{@pclinenumber}\stepcounter{H@pclinenumber}}

\pclnseparator
\pcrlnseparator
Define separators between line numbers and code (left and right). Note that line numbers can be displayed either to the left or to the right of code.

\providecommand{\pclnseparator}{:}
\providecommand{\pcrlnseparator}{}

\pclnspace
\pclnrspace
Define spacing between line numbers and code (left and right).

\providecommand{\pclnspace}{1em}
\providecommand{\pclnrspace}{0.5em}

\pclnstyle
\providecommand{\pclnstyle}[1]{\text{\scriptsize#1}}

\pcln
Manually place (left aligned) line numbers. This command is also used by the automatic placement of line numbers.

\providecommand{\pcln}{}% \ifthenelse{\equal{\@pc@skiplnmarker}{1}}{\ifmeasuring@\else\@pc@resetskipln{}\fi}{% \refstepcounter{pclinenumber}\stepcounter{Hpclinenumber}\ifthenelse{\value{pclinenumber}<10}{\hspace{1ex}}{}\pclnstyle{\arabic{pclinenumber}}\pclnseparator\hspace{\pclnspace}}% \pcskipln
\@pc@skiplnmarker
skipfirstln
allow to skip numbering single lines if linenumbering=on

\def{\@pc@skiplnmarker}{}
\providecommand{\pcskipln}{{\ifmeasuring@{\@global{\def{\@pc@skiplnmarker}{1}}}}% \newcommand{\@pc@resetskipln}{\@global{\def{\@pc@skiplnmarker}{}}}% \define@key{pseudocode}{skipfirstln}[1]{\@global{\def{\@pc@skiplnmarker}{1}}}% }

\pclnr
\pcrln
Manual placement of right aligned line numbers using the same counter (\pclnr) or a separate counter (\pcrln).

\providecommand{\pclnr}{}\refstepcounter{pclinenumber}
9.6.1 Options

The following commands define a bunch of placeholders (plus their default values) that are defined via the various options of the pseudocode command.

Distance between header and line.

Distance between header and line.
Specification of the various options of the \pseudocode command.

The \pseudocode{mode} key (with values \texttt{text} or \texttt{math} (default)) specifies whether within a pseudocode block input is by default typeset in text mode or in math mode. The \texttt{\pc...} variables are variables that help typesetting each line in a pseudocode block.

\newcommand*{\pseudocodemodebegin}{}
\newcommand*{\pseudocodemodeend}{}
\define@key{pseudocode}{mode}[math]{%
Control the minimal line height of pseudocode blocks.

This is the core of the (experimental) automatic syntax highlighting and automatic spacing. The code is ugly, and very slow. It is not really recommended to be used in larger projects.

9.6.2 Automatic Syntax Highlighting and Spacing (Experimental)

Experimental \LaTeX3 string substitution helpers for automatic keyword highlighting. The regex parsing is (regrettably) super slow.
9.6.3 Helper Variables

\@pc@thecontent
\@pc@colspace

Helper variables used within pseudocode

\newcommand{\@pc@thecontent}{}
\newcommand{\@pc@colspace}{}

\@withinspaces
\@keepspaces

Helper variables for controlling automatic spacing

\newcommand{\@withinspaces}{false}%
\newcommand{\@keepspaces}{%}
\renewcommand{\@withinspaces}{true}\@pc@withspaces%

\newcommand*{\pseudocodespace}{}
\define@key{pcspace}{space}{\ifthenelse{\equal{#1}{keep}}{\@keepspaces}{}\renewcommand*{\pseudocodespace}{#1}}

\newcommand*{\@pc@defaultargs}{}
\newcommand*{\pcsetargs}[1]{\renewcommand*{\@pc@defaultargs}{#1}}

\newcounter{@pc@indentationlevel}
\newcommand{\@pc@increaseindent}{\addtocounter{@pc@indentationlevel}{1}}
\newcommand{\@pc@decreaseindent}{\setcounter{@pc@indentationlevel}{\value{@pc@indentationlevel}-1}}
\newcommand{\@pc@tmpdecreaseindent}{\setcounter{@pc@indentationlevel}{\value{@pc@indentationlevel}-1}}
\newcommand{\@pc@althighlight}{%}
\ifthenelse{\equal{\@pseudocodespace}{keep}}{\highlightaltkeyword{#1}}{}\renewcommand*{\@pseudocodespace}{#1}\@pc@stringsubstitution{#1}{ }{~}
9.6.4 The Actual Pseudocode Command

% Check if the pseudocode command is called with an optional argument
\providecommand{\pseudocode}{%}
\begingroup%
\renewcommand{\@withinspaces}{false}%
@ifnextchar[{}{\@pseudocodeA}]{\@pseudocode{}%}
\endgroup%
\def{\@pseudocodeA}[#1]{%
\setkeys*{pcspace}{#1}%test if there is a space assignment within the keys .. make the necessary arrangements
\@pseudocode{}%}
\def{\@pseudocode}[#1]#2{%
\begingroup%
% reset skip marker before parsing options, as this might set it
\@pc@resetskipln%
% parse options
% this is the same as \setkeys{pseudocode}{space}{\@pc@defaultargs,#1}%ignore the space key.
% expect that we expand the default args
@expandedsetkeys{pseudocode}{space}{head=}\@pc@defaultargs{#1}%
% check draft mode and disable syntax highlighting
@pc@ifdraft{\ifthenelse{\equal{\@pseudocodenodraft}{true}}{}{\renewcommand{\@pseudocodesyntaxhighlighting{}}}{}%}
% create tabbing command
\ifcsname \pctabname\endcsname%
\edef\pctabnameend{\@pc@colspace}\edef\pctabnamebegin{\@pc@colspace}
\expandafter\if\csname \pctabname\endcsname%
\edef\pctabnameend{\@pc@colspace}\edef\pctabnamebegin{\@pc@colspace}
\else%
\% adjust row width
\addtolength{\jot}{\@pseudocodecodejot}\%
\% create indent command
\expandafter\let\csname \pcindentname \endcsname\pcind\%
\% store and wrap (do syntax highlighting) argument
\renewcommand{\@pc@thecontent}{\@pc@and@wrap@start\@pc@syntaxhighlight{#2}\@pc@and@wrap@end}%
\% take care of counters
\stepcounter{\pcglobal@pc@cnt}%
\setcounter{\pclinenumber}{\@pseudocodelnstart}%
\setcounter{\pcrlinenumber}{\@pseudocodelnstartright}%
\setlength{\@pc@minipage@length}{\@pseudocodelnstart}%
\setlength{\@pc@alt@minipage@length}{\@pseudocodelnstartright}%
\setcounter{\@pclinenumbertmp}{\value{pclinenumber}}%
\setcounter{\@pcrlinenumbertmp}{\value{pcrlinenumber}}%
\% reset column counter
\setcounter{\pccolumncounter}{2}%
\% vertical space
\vspace{\@pseudocodeyshift}%
\% line magic
\ifthenelse{\value{\pcglobal@pc@nestcnt}=1}{%\%
\let\@pc@halign\halign%\renewenvironment{pcmbox}{\let\halign\@pc@halign}{}%\let\halign{\renewcommand{\label}{\ifmeasuring@\else\@pc@original@label{####1}\fi}}%\let\@pc@lb\%\renewcommandx*{\\}[2][1=,2=]{\@pc@modeend\@pc@and\@pseudocodecodeatendline\ifthenelse{\equal{####1}}{\@pc@lb}{\@pc@lb[####1]}####2\@pc@beginnewline}%\def\pclb{\let\\@pc@lb\relax\@pc@modeend\\}%\@pc@halign}%\}%\% align column separation
\renewcommand*{\minalignsep}{\@pseudocodecolsep}%\%
\% as the following block will execute the pseudocode we need to store the skip command
\edef\@pc@org@skiplnmarker{\@pc@skiplnmarker}%
\% if no width is set compute width and store in circuitlength
\ifthenelse{\equal{\@pseudocodewidth}}{%\%
\ifthenelse{\value{\pcsubprogstep}=0}{%\%
\@pc@settowidthofalign{\@pc@minipage@length}{\@pc@thecontent}%%
\@pc@settowidthofaligned{\@pc@minipage@length}{\@pc@thecontent}%%
\@pc@settowidthofaligned{\@pc@minipage@length}{\@pc@thecontent}%%
\% compute length of pseudocode
\ifthenelse{\value{\pcsubprogstep}=0}{%\%
\@pc@settowidthofalign{\@pc@minipage@length}{\@pc@thecontent}%%
\% compute length of header
\ifthenelse{\equal{\@withingame}}{true}{%\%
\ifthenelse{\equal{\@pc@secondheader}}{true}{%\%
}{\addtolength{\@pc@alt@minipage@length}{\widthof{x\ensuremath{\@pc@gametitle[1]}\@pc@gametitle[1]}}}%\%
}{\addtolength{\@pc@alt@minipage@length}{\widthof{\ensuremath{\@pc@gametitle[1]}}}}%\%
}{\addtolength{\@pc@alt@minipage@length}{\widthof{\@pseudocodehead}}}%
92
% use header length if longer and add some points for good measure
\ifdim\@pc@alt@minipage@length>\@pc@minipage@length\%
\setlength{\@pc@minipage@length}{\@pc@alt@minipage@length}\%
\fi%
\addtolength{\@pc@minipage@length}{\@pseudocodeaddtolength}\%
{\addtolength{\@pc@minipage@length}{\@pseudocodewidth}\%}
\reset counter and skip command
\setcounter{pclinenumber}{\value{@pclinenumbertmp}}\%
\setcounter{pcrlinenumber}{\value{@pcrlinenumbertmp}}\%
\setcounter{@pc@indentationlevel}{0}\%
\edef\@pc@skiplnmarker{\@pc@org@skiplnmarker}\%
% do the actual mini page
\hspace{\pcbeforeskip}\hspace{\@pseudocodexshift}\%
\ifthenelse{\equal{\@pseudocodeminipagealign}{t}}{% raisebox{\dimexpr\ht\strutbox-\height}{\@pc@pseudocodeminipage{t}}%
\@pc@pseudocodeminipage{\@pseudocodeminipagealign}%
\hspace{\pcafterskip}%
% tikz usage
\@pc@releaseremember\%
\addtocounter{@pcsubprogstep}{-1}\%
\endgroup\%
\ifthenelse{\equal{\@withinspaces}{true}}{\end@pc@withspaces}{}\%
\endgroup\%
\hspace{\pcafter skip}%
% begin actual output
\% insert space from stacking
\@pc@stackspace@forpseudocode\%
\newcommand{\@pc@pseudocodeminipage}[1]{% 
\begin{minipage}{\@pc@minipage@length}\%
\ifthenelse{\value{@pcsubprogstep}=0}{% pc@display@pseudocode{\@pseudocodehead}{\@pc@thecontent}%
\pc@display@subcode{\@pseudocodehead}{\@pc@thecontent}%
\end{minipage}%
\newcommand{\@pc@display@gameheader}[1]{% 
\begin{minipage}{\@pc@minipage@length}\%
\ifthenelse{\value{@pcsubprogstep}=0}{% pc@display@pseudocode{\@pseudocodehead}{\@pc@thecontent}%
\tikz\{\gdef\i{\thepcgamecounter}\% node[anchor=base, text depth=0pt, inner sep=0.05em, outer sep=0pt] (gamenode\i) \{#1\};
\ifthenelse{\equal{\@withinbxgame}{true}}{% \node[draw, anchor=base, above=2ex of gamenode\i] (bgamenode\i) \{\@bxgameheader\};
\end{tikzpicture} \}
\end{minipage}%;
\let\pclb\relax
\newcommand{\@pc@display@pseudocode}[2]{%
Create Pseudocode/Procedure Commands

Define pseudocode command with parameters:

1. name
2. code to execute after begin group
3. head prefix
4. other config

\newcommand*{\@pc@createproc@headmode}{text}
\newcommand{\createprocedurecommand}[4]{
\setkeys*{pcspace}{#2,#3}\@pseudocode[head={#1},#2,#3]
\begin{aligned}#4\end{aligned}
\endgroup
\renewcommand{\@withinspaces}{false}\
#2\
\@ifnextchar\[%
{\@pseudocodeB{#3}{#4}}
{\@pseudocodeC{#3}{#4}}\%
% 
\}

\createpseudocodecommand

\newcommand{\createpseudocodecommand}[4]{{
\expandafter\gdef\csname #1\endcsname{{
\begingroup
\renewcommand{\@withinspaces}{false}
#2\
\@ifnextchar\[%
{\@pseudocodeE{#3}{#4}}
{\@pseudocodeF{#3}{#4}}\%
}\}}
\}}

\createpseudocodeblock

\newcommand{\createpseudocodeblock}[5]{{
\createpseudocodecommand{#1@pc}{#3}{#4}{#5}
\expandafter\gdef\csname #1\endcsname{{
\@ifnextchar\[%
{\csname #1@@\endcsname}
{\csname #1@\endcsname}
}\}}
\expandafter\gdef\csname #1@@\endcsname[##1]{{
\begin{pchstack}\[#2\]
\csname #1@pc\endcsname[##1]{##2}
\end{pchstack}
}}
\expandafter\gdef\csname #1@@\endcsname[##1][##2]{{
\begin{pchstack}\[#2\]
\csname #1@pc\endcsname[##1][##2]
\end{pchstack}
}}
\}}

\createprocedureblock

\newcommand{\createprocedureblock}[5]{{
\createpseudocodeblock{#1@pc}{#3}{#4}{#5}
\expandafter\gdef\csname #1\endcsname{{
\@ifnextchar\[%
{\csname #1@@\endcsname}
{\csname #1@\endcsname}
}\}}
\expandafter\gdef\csname #1@@\endcsname[##1]{{
\begin{pchstack}\[#2\]
\csname #1@pc\endcsname[##1]{##2}
\end{pchstack}
}}
\expandafter\gdef\csname #1@@\endcsname[##1][##2]{{
\begin{pchstack}\[#2\]
\csname #1@pc\endcsname[##1][##2]
\end{pchstack}
}}
\}}

\createprocedureblock

\newcommand{\createprocedureblock}[5]{{
\createpseudocodeblock{#1@pc}{#3}{#4}{#5}
\expandafter\gdef\csname #1\endcsname{{
\@ifnextchar\[%
{\csname #1@@\endcsname}
{\csname #1@\endcsname}
}\}}
\expandafter\gdef\csname #1@@\endcsname[##1]{{
\begin{pchstack}\[#2\]
\csname #1@pc\endcsname[##1]{##2}
\end{pchstack}
}}
\expandafter\gdef\csname #1@@\endcsname[##1][##2]{{
\begin{pchstack}\[#2\]
\csname #1@pc\endcsname[##1][##2]
\end{pchstack}
}}
\}}

\createprocedureblock

\newcommand{\createprocedureblock}[5]{{
\createpseudocodeblock{#1@pc}{#3}{#4}{#5}
\expandafter\gdef\csname #1\endcsname{{
\@ifnextchar\[%
{\csname #1@@\endcsname}
{\csname #1@\endcsname}
}\}}
\expandafter\gdef\csname #1@@\endcsname[##1]{{
\begin{pchstack}\[#2\]
\csname #1@pc\endcsname[##1]{##2}
\end{pchstack}
}}
\expandafter\gdef\csname #1@@\endcsname[##1][##2]{{
\begin{pchstack}\[#2\]
\csname #1@pc\endcsname[##1][##2]
\end{pchstack}
}}
\}}
other config

\newcommand{\createprocedureblock}[5]{
  \createprocedurecommand{#1@pc}{#3}{#4}{#5}
}\@ifnextchar[%
  \csname #1\endcsname\%
}{\csname #1@\endcsname}

\expandafter\gdef\csname #1@\endcsname##1##2{\begin{pchstack}[#2]
  \csname #1@pc\endcsname{##1}{##2}
  \end{pchstack}}
\expandafter\gdef\csname #1@@\endcsname[##1]##2##3{\begin{pchstack}[#2]
  \csname #1@pc\endcsname[##1]{##2}{##3}
  \end{pchstack}}

Create \procedure command.

\createprocedurecommand{procedure}{\{}{\}{}\}
\createpseudocodeblock{pseudocodeblock}{center}{\}{}\}
\createprocedureblock{procedureblock}{center}{\}{}\}

9.8 Subprocedures

\newcounter{pcs\subprog\cnt\un{1}}
\newcounter{pc\subprog\cnt\un{1}}
\newcounter{pcs\subprog\cnt\un{2}}
\newcounter{pc\subprog\cnt\un{2}}
\newcounter{pcs\subprog\cnt\un{3}}
\newcounter{pc\subprog\cnt\un{3}}
\newcounter{pcs\subprog\cnt\un{4}}
\newcounter{pc\subprog\cnt\un{4}}
\newcounter{pcs\subprog\cnt\un{5}}
\newcounter{pc\subprog\cnt\un{5}}
\newcounter{pcs\subprog\cnt\un{6}}
\newcounter{pc\subprog\cnt\un{6}}
\newcounter{pcs\subprog\cnt\un{7}}
\newcounter{pc\subprog\cnt\un{7}}
\newcounter{pcs\subprog\cnt\un{8}}
\newcounter{pc\subprog\cnt\un{8}}
\newcounter{pcs\subprog\cnt\un{9}}
\newcounter{pc\subprog\cnt\un{9}}
\newcounter{pc\subprog\step}
\newenvironment{subprocedure}{\addtocounter{pc\subprog\step}{1}}{\setcounter{pc\subprog\cnt\un{\thepc\subprog\step}}{\value{pclinenumber}}\setcounter{pc\subprog\cnt\un{\thepc\subprog\step}}{\value{pcrlinenumber}}}

96
9.9 Protocols

% send message
\newcommand{\pcshortmessageoffset}{0.5cm}
\newcommand{\pcdefaultmessagelength}{3.5cm}
\newcommand{\pcdefaultlongmessagelength}{6cm}
\newcommand{\pcbeforemessageskip}{opt}
\newcommand{\pcaftermessageskip}{opt}
\newlength{\pcmessagearrow}
\newcommand{\@pcsendmessagelength}{\pcdefaultmessagelength}
\newcommand{\@pcsendmessagecol}{}
\newcommand{\@pcsendmessagewidth}{}
\newcommand{\@pcsendmessagestyle}{}
\newcommand{\@pcsendmessagetop}{}
\newcommand{\@pcsendmessagebottom}{}
\newcommand{\@pcsendmessageright}{}
\newcommand{\@pcsendmessageleft}{}
\newcommand{\@pcsendmessagetopname}{t}
\newcommand{\@pcsendmessagebottomname}{b}
\newcommand{\@pcsendmessagerightname}{r}
\newcommand{\@pcsendmessageleftname}{l}
\newcommand{\@pcsendmessagetopstyle}{}
\newcommand{\@pcsendmessagebottomstyle}{}
\newcommand{\@pcsendmessagerightstyle}{}
\newcommand{\@pcsendmessageleftstyle}{}
\newcommand{\@pcsendmessagebeforeskip}{\pcbeforemessageskip}
\newcommand{\@pcsendmessageafterskip}{\pcaftermessageskip}
\newcommand{\@pcsendmessagecenteredtop}{false}
\define@key{pcsendmessage}{centercol}[]{\renewcommand{\@pcsendmessagecol}{#1}}
\define@key{pcsendmessage}{width}[]{\renewcommand{\@pcsendmessagewidth}{#1}}
\define@key{pcsendmessage}{style}[]{\renewcommand{\@pcsendmessagestyle}{#1}}
\define@key{pcsendmessage}{length}[]{\renewcommand{\@pcsendmessagelength}{#1}}
\define@key{pcsendmessage}{top}[]{\renewcommand{\@pcsendmessagetop}{#1}}
\define@key{pcsendmessage}{bottom}[]{\renewcommand{\@pcsendmessagebottom}{#1}}
\define@key{pcsendmessage}{right}[]{\renewcommand{\@pcsendmessageright}{#1}}
\define@key{pcsendmessage}{left}[]{\renewcommand{\@pcsendmessageleft}{#1}}
\define@key{pcsendmessage}{topname}[]{\renewcommand{\@pcsendmessagetopname}{#1}}
\define@key{pcsendmessage}{bottomname}[]{\renewcommand{\@pcsendmessagebottomname}{#1}}
\define@key{pcsendmessage}{rightname}[]{\renewcommand{\@pcsendmessagerightname}{#1}}
\define@key{pcsendmessage}{leftname}[]{\renewcommand{\@pcsendmessageleftname}{#1}}
\define@key{pcsendmessage}{topstyle}[]{\renewcommand{\@pcsendmessagetopstyle}{#1}}
\define@key{pcsendmessage}{bottomstyle}[]{\renewcommand{\@pcsendmessagebottomstyle}{#1}}
\define@key{pcsendmessage}{rightstyle}[]{\renewcommand{\@pcsendmessagerightstyle}{#1}}
\define@key{pcsendmessage}{leftstyle}[]{\renewcommand{\@pcsendmessageleftstyle}{#1}}
\define@key{pcsendmessage}{beforeskip}[]{\renewcommand{\@pcsendmessagebeforeskip}{#1}}
\define@key{pcsendmessage}{afterskip}[]{\renewcommand{\@pcsendmessageafterskip}{#1}}
\newcommand{\@pcsendmessagealignedtop}{false}
\define@key{pcsendmessage}{topaligned}{true}{\renewcommand*{\@pcsendmessagealignedtop}{#1}}
\newcommand*{\@pcsendmessagealignedbottom}{false}
\define@key{pcsendmessage}{bottomaligned}{true}{\renewcommand*{\@pcsendmessagealignedbottom}{#1}}
\newcommand*{\@pcsendmessagealignedleft}{false}
\define@key{pcsendmessage}{leftaligned}{true}{\renewcommand*{\@pcsendmessagealignedleft}{#1}}
\newcommand*{\@pcsendmessagealignedright}{false}
\define@key{pcsendmessage}{rightaligned}{true}{\renewcommand*{\@pcsendmessagealignedright}{#1}}

\newcommand{\@pc@centerincol}[2]{%  
  \ifmeasuring@#2%  
  \else%  
    \makebox[\ifcase\expandafter #1\maxcolumn@widths\fi]{\$\displaystyle#2\$}%  
  \fi%}
\newcommand{\centerincol}[1]{\@pc@centerincol{\thepccolumncounter}{#1}}

\newcommand{\@do@sendmessage}[1]{%  
  \ifthenelse{\equal{\@pcsendmessagecol}{}}%  
  {\ifthenelse{\equal{\@pcsendmessagewidth}{}}%  
   {\makebox[\@pcsendmessagewidth]{\$\displaystyle#1\$}}%  
  {\let\@pc@fin@sendmessagetop\@pcsendmessagetop}%  
  {\newcommand{\@pc@fin@sendmessagetop}{\let\halign\@pc@halign$\begin{aligned}\@pcsendmessagetop\end{aligned}$}}}%  
  {\let\@pc@fin@sendmessagetop\@pcsendmessagetop}%  
%  \ifthenelse{\equal{\@pcsendmessagealignedtop}{true}}%  
%    {\ifthenelse{\equal{\@pcsendmessagetop}{}}%  
%      {\let\@pc@fin@sendmessagetop\@pcsendmessagetop}%  
%      {\newcommand{\@pc@fin@sendmessagetop}{\let\halign\@pc@halign$\begin{aligned}\@pcsendmessagetop\end{aligned}$}}}%  
%  {\let\@pc@fin@sendmessagetop\@pcsendmessagetop}%  
%  \ifthenelse{\equal{\@pcsendmessagealignedbottom}{true}}%  
%    {\ifthenelse{\equal{\@pcsendmessagemessagebottom}{}}%  
%      {\let\@pc@fin@sendmessagemessagebottom\@pcsendmessagemessagebottom}%  
%      {\newcommand{\@pc@fin@sendmessagemessagebottom}{\let\halign\@pc@halign$\begin{aligned}\@pcsendmessagemessagebottom\end{aligned}$}}}%  
%  {\let\@pc@fin@sendmessagemessagebottom\@pcsendmessagemessagebottom}%  
%  \ifthenelse{\equal{\@pcsendmessagealignedleft}{true}}%  
%    {\ifthenelse{\equal{\@pcsendmessagemessageleft}{}}%  
%      {\let\@pc@fin@sendmessagemessageleft\@pcsendmessagemessageleft}%  
%      {\newcommand{\@pc@fin@sendmessagemessageleft}{\let\halign\@pc@halign$\begin{aligned}\@pcsendmessagemessageleft\end{aligned}$}}}%  
%  {\let\@pc@fin@sendmessagemessageleft\@pcsendmessagemessageleft}%  
%  \ifthenelse{\equal{\@pcsendmessagealignedright}{true}}%  
%    {\ifthenelse{\equal{\@pcsendmessagemessageright}{}}%  
%      {\let\@pc@fin@sendmessagemessageright\@pcsendmessagemessageright}%  
%      {\newcommand{\@pc@fin@sendmessagemessageright}{\let\halign\@pc@halign$\begin{aligned}\@pcsendmessagemessageright\end{aligned}$}}}%  
%  {\let\@pc@fin@sendmessagemessageright\@pcsendmessagemessageright}%  
\begin{group}
\setkeys{pcsendmessage}{#2}%  
\tikzset{PCSENDMSG-PATH-STYLE/.style/.expand once=\@pcsendmessagestyle}%  
\tikzset{PCSENDMSG-TOP-STYLE/.style/.expand once=\@pcsendmessagetopstyle}%  
\tikzset{PCSENDMSG-BOTTOM-STYLE/.style/.expand once=\@pcsendmessagebottomstyle}%  
\tikzset{PCSENDMSG-LEFT-STYLE/.style/.expand once=\@pcsendmessageleftstyle}%  
\tikzset{PCSENDMSG-RIGHT-STYLE/.style/.expand once=\@pcsendmessagerightstyle}%  
\%  
\%  \ifthenelse{\equal{\@pcsendmessagealignedtop}{true}}%  
%    {\ifthenelse{\equal{\@pcsendmessagetop}{}}%  
%      {\let\@pc@fin@sendmessagetop\@pcsendmessagetop}%  
%      {\newcommand{\@pc@fin@sendmessagetop}{\let\halign\@pc@halign$\begin{aligned}\@pcsendmessagetop\end{aligned}$}}}%  
%  {\let\@pc@fin@sendmessagetop\@pcsendmessagetop}%  
%  \ifthenelse{\equal{\@pcsendmessagealignedbottom}{true}}%  
%    {\ifthenelse{\equal{\@pcsendmessagemessagebottom}{}}%  
%      {\let\@pc@fin@sendmessagemessagebottom\@pcsendmessagemessagebottom}%  
%      {\newcommand{\@pc@fin@sendmessagemessagebottom}{\let\halign\@pc@halign$\begin{aligned}\@pcsendmessagemessagebottom\end{aligned}$}}}%  
%  {\let\@pc@fin@sendmessagemessagebottom\@pcsendmessagemessagebottom}%  
%  \ifthenelse{\equal{\@pcsendmessagealignedleft}{true}}%  
%    {\ifthenelse{\equal{\@pcsendmessagemessageleft}{}}%  
%      {\let\@pc@fin@sendmessagemessageleft\@pcsendmessagemessageleft}%  
%      {\newcommand{\@pc@fin@sendmessagemessageleft}{\let\halign\@pc@halign$\begin{aligned}\@pcsendmessagemessageleft\end{aligned}$}}}%  
%  {\let\@pc@fin@sendmessagemessageleft\@pcsendmessagemessageleft}%  
%  \ifthenelse{\equal{\@pcsendmessagealignedright}{true}}%  
%    {\ifthenelse{\equal{\@pcsendmessagemessageright}{}}%  
%      {\let\@pc@fin@sendmessagemessageright\@pcsendmessagemessageright}%  
%      {\newcommand{\@pc@fin@sendmessagemessageright}{\let\halign\@pc@halign$\begin{aligned}\@pcsendmessagemessageright\end{aligned}$}}}%  
%  {\let\@pc@fin@sendmessagemessageright\@pcsendmessagemessageright}%  
\end{group}
9.10 Tikz within Pseudocode
9.11 Black Box Reductions

\ifdefempty{\@pcnodedraw}{}{%
\begin{tikzpicture}[overlay,inner sep=0ex,baseline=0pt]\@pcnodedraw\end{tikzpicture}
%
\endgroup}

% Reductions
\newcommandx*{\pcdraw}[2]{%
\begin{tikzpicture}[overlay,inner sep=0ex,baseline=0pt,#2]
#1
\end{tikzpicture}}

9.11 Black Box Reductions

\ifdefempty{\@pcnodedraw}{}{%
\begin{tikzpicture}[overlay,inner sep=0ex,baseline=0pt]\@pcnodedraw\end{tikzpicture}
%
\endgroup}

% Reductions
\newcommand{\@bb@lastbox}{}
\newcommand{\@bb@lastoracle}{}
\newcommand{\@bb@lastchallenger}{}
\newlength{\@bb@message@voffset}
\newlength{\@bb@query@voffset}
\newlength{\@bb@oraclequery@voffset}
\newlength{\@bb@challengerquery@voffset}
\newcounter{\@bb@oracle@cnt}
\newcounter{\@bb@oracle@nestcnt}
\newcounter{\@bb@challenger@cnt}
\newcounter{\@bb@challenger@nestcnt}
\newcounter{\@bb@env@nestcnt}
\newcommand{\bbroraclenodenameprefix}{ora-}
\newcommand{\bbrchallengernodenameprefix}{challenger-}
\newcommand{\bbrenvnodenameprefix}{env-}
\newcommand*{\@pc@bbrenvaiveskip}{0pt}
\define@key{pcbbrenv}{aboveskip}{\renewcommand*{\@pc@bbrenvaiveskip}{#1}}
\newcommand*{\@pc@bbrenvbelowskip}{0pt}
\define@key{pcbbrenv}{belowskip}{\renewcommand*{\@pc@bbrenvbelowskip}{#1}}
\newcommand*{\@pc@bbrenv@argstring}{}
\def\@pc@bbrenv@remfinalequals#1=#2=\relax{\renewcommand*{\@pc@bbrenv@argstring}{#1}}
\def\@pc@bbrenv@legacyargcheck#1=#2\relax{\ifthenelse{\equal{#2}{}}{\PackageWarning{cryptocode}{Deprecated option for bbrenv. Please use key value list as first parameter with keys aboveskip and/or belowskip.}\renewcommand*{\@pc@bbrenv@argstring}{aboveskip=#1}}{\@pc@bbrenv@remfinalequals#1=#2}}
\providecommand{\bbrfirstmessageoffset}{1ex}
\providecommand{\bbrfirstmessageoffset}{1ex}
\newenvironment{bbrenv}{\begin{tikzpicture}}{\end{tikzpicture}}

% parse args and allow old style #1=opt
\@pc@bbrenv@legacyargcheck#1=\relax
\@expandedsetkeys{pcbbrenv}{}{belowskip=#3}{\@pc@bbrenv@argstring}{}

% reset lengths
\@pc@globalsetlength{\@bb@message@voffset}{\bbrfirstmessageoffset}
\@pc@globalsetlength{\@bb@query@voffset}{\bbrfirstmessageoffset}
\@pc@globalsetlength{\@bb@oraclequery@voffset}{\bbrfirstmessageoffset}
\@pc@globalsetlength{\@bb@challengerquery@voffset}{\bbrfirstmessageoffset}

% reset oracle counter and oracle query offset
\ifthenelse{\value{@bb@oracle@nestcnt}=0}{\setcounter{@bb@oracle@cnt}{0}}{}
\ifthenelse{\value{@bb@challenger@nestcnt}=0}{\setcounter{@bb@challenger@cnt}{0}}{}

\vspace{\@pc@bbrenvaboveskip}
\ifthenelse{\value{@bb@env@nestcnt}=1}{\@pc@ensureremember{\begin{tikzpicture}}}{\tikz\bgroup}
\ifthenelse{\value{@bb@env@nestcnt}=1}{\end{tikzpicture}\@pc@releaseremember}{\egroup}
\vspace{\@pc@bbrenvbelowskip}
\addtocounter{@bb@env@nestcnt}{-1}

\define@key{bbrbox}{abovesep}{\renewcommand{\bbrboxabovesep}{#1}}
\define@key{bbrbox}{name}{\renewcommand{\bbrboxname}{#1}}
\define@key{bbrbox}{namestyle}{\renewcommand{\bbrboxnamestyle}{#1}}
\define@key{bbrbox}{namepos}{\renewcommand{\bbrboxnamepos}{#1}}

\newcommand*{\bbrboxname}{}
\newcommand*{\bbrboxnamepos}{right}
\newcommand*{\bbrboxnamestyle}{}
\newcommand*{\@bbrboxnameposoffset}{below left=0.5ex and -0.5ex of \@bb@lastbox.north east,anchor=north east}
\newcommand*{\bbrboxabovesep}{\baselineskip}
\newcommand*{\bbrboxnameoffset}{below left=\bbrboxabovesep of phantomname.south west}
\newcommand*{\bbrboxstyle}{draw}
\newcommand*{\bbrboxafterskip}{}
\newcommand*{\bbrboxminheight}{0cm}
\newcommand*{\bbrboxminwidth}{2cm}
\newcommand*{\bbrboxxshift}{0cm}
\newcommand*{\bbrboxyshift}{0cm}
\newcommand*{\define@key{bbrbox}[1]}{\renewcommand*{\bbrboxabovesep[#1]}{}}
\newcommand*{\define@key}[1]{\renewcommand*{\bbrboxname[#1]}{}}
\newcommand*{\define@key}[1]{\renewcommand*{\bbrboxnamepos[#1]}{}}
\newcommand*{\define@key}[1]{\renewcommand*{\bbrboxnamestyle[#1]}{}}
\newcommand*{\define@key}[1]{\renewcommand*{\bbrboxnameoffset[#1]}{}}
\newenvironment{x{bbroracle}[2][2=]}{% 
  \begingroup
  \setkeys{x{bbroracle}}{#2}
  %reset query boolean. This is a bit crude and does not allow nesting oracles
  \gdef\@bbr@first@oraclequery{true}
  %\add to nesting cout
  \addtocounter{#2}{1}
  %\if first oracle, then put it to the right, else stack them vertically
  \addtocounter{#2}{1}
  %\setlength{\@bb@tmplength@b}{\bbroraclevdistance-\baselineskip}
  \node[inner sep=0pt, below right=\@bb@tmplength@b and \bbroraclehdistance of \@bb@lastbox.north east, anchor=north west] (\bbroraclenodenameprefix#1) {\begin{bbrenv}{#1}}
  \global\def\@bb@lastoracle{#1} 
  \end{bbrenv}
}{
  \end{bbrenv};
  \addtocounter{#2}{-1}
\endgroup
}

\newcommand\bbrchallengerhdistance{1.5cm}
\newcommand\bbrchallengervdistance{\baselineskip}
\define@key{x{bbrchallenger}}{distance}[]{\renewcommand\bbrchallengerhdistance{#1}}
\define@key{x{bbrchallenger}}{hdistance}[]{\renewcommand\bbrchallengerhdistance{#1}}
\define@key{x{bbrchallenger}}{vdistance}[]{\renewcommand\bbrchallengervdistance{#1}}

% Challenger
\newenvironment{x{bbrchallenger}[2][2=]}{% 
  \begingroup
  \setkeys{x{bbrchallenger}}{#2}
  %reset query boolean. This is a bit crude and does not allow nesting oracles
  \gdef\@bbr@first@challengerquery{true}
  %\add to nesting cout
  \addtocounter{#2}{1}
  %\setlength{\@bb@tmplength@b}{\bbrchallengervdistance-\baselineskip}
  \node[inner sep=0pt, below right=\@bb@tmplength@b and \bbrchallengervdistance of \@bb@lastbox.north east, anchor=north west] (\bbroraclenodenameprefix#1) {\begin{bbrenv}{#1}}
  \global\def\@bb@lastchallenger{#1} 
  \end{bbrenv};
  \addtocounter{#2}{-1}
\endgroup
}
\node[inner sep=0pt,outer sep=0pt,below left=\@bb@tmmptlen0b and \bbrchallengerhdistance of \@bb@lastchallenger.north west]{#1};
\coordinate (@bbtmpcoord) at (\@bb@tmmptlen0b.north west);% 
\path (@bbtmpcoord);% 
\pgfgetlastxy{\XCoord}{\YCoordA}%;
\coordinate (@bbtmpcoord) at (\bbrchallengernameprefix \@bb@lastchallenger.south east);% 
\path (@bbtmpcoord);% 
\pgfgetlastxy{\XCoord}{\YCoordB}%;
\setlength{\@bb@tmplength}{\YCoordA-\YCoordB+\bbrchallengervdistance}%;
\node[inner sep=0pt,below left=\@bb@tmplength0b and \bbrchallengerhdistance of \@bb@lastchallenger.north west]{#1};
\global\def\@bb@lastchallenger{#1}
\begin{bbrenv}{#1}%;
\end{bbrenv}%;
\addtocounter{@bb@challenger@nestcnt}{-1}%;
\endgroup%;
\let\msgfrom\bbrchallengerqueryto%
\
\newcommand*{\bbrinputlength}{0.5cm}
\newcommand*{\bbrinputhoffset}{0.5cm}
\newcommand*{\bbrinputbottom}{}
\newcommand*{\bbrinputtop}{}
\newcommand*{\bbrinputedgestyle}{}
\newcommand*{\bbrinputtopstyle}{}
\newcommand*{\bbrinputbottomstyle}{}
\newcommand*{\bbrinputnodestyle}{}
\newcommand*{\bbrinputnodename}{}
\define@key{bbrinput}{length}{\renewcommand*{\bbrinputlength}{#1}}
\define@key{bbrinput}{hoffset}{\renewcommand*{\bbrinputhoffset}{#1}}
\define@key{bbrinput}{name}{\renewcommand*{\bbrinputnodename}{#1}}
\define@key{bbrinput}{top}{\renewcommand*{\bbrinputtop}{#1}}
\define@key{bbrinput}{bottom}{\renewcommand*{\bbrinputbottom}{#1}}
\newcommand*{\bbrinputsetup}[1]{% 
\setkeys{bbrinput}{#1}%;
\tikzset{BBRINPUT-NODESTYLE/.style/.expand once=\bbrinputnodestyle}%;
\tikzset{BBRINPUT-TOPSTYLE/.style/.expand once=\bbrinputtopstyle}%;
\tikzset{BBRINPUT-BOTTOMSTYLE/.style/.expand once=\bbrinputbottomstyle}%;
\tikzset{BBRINPUT-EDGESTYLE/.style/.expand once=\bbrinputedgestyle}%;
\newcommand\bbrinput[1][2]{%
\texttt{\textbackslash define@key{bbrcom}{\textbackslash length}[]{\textbackslash renewcommand*{\textbackslash bbrcomlength}[#1]}}

\texttt{\textbackslash define@key{bbrcom}{\textbackslash topname}[]{\textbackslash renewcommand*{\textbackslash bbrcomtopname}[#1]}}

\texttt{\textbackslash define@key{bbrcom}{\textbackslash bottomname}[]{\textbackslash renewcommand*{\textbackslash bbrcombottomname}[#1]}}

\texttt{\textbackslash define@key{bbrcom}{\textbackslash sidename}[]{\textbackslash renewcommand*{\textbackslash bbrcomsidename}[#1]}}

\texttt{\textbackslash define@key{bbrcom}{\textbackslash osidename}[]{\textbackslash renewcommand*{\textbackslash bbrcomosidename}[#1]}}

\texttt{\textbackslash define@key{bbrcom}{\textbackslash beforerskip}[]{\textbackslash renewcommand*{\textbackslash bbrcombeforerskip}[#1]}}

\texttt{\textbackslash define@key{bbrcom}{\textbackslash aboveskip}[]{\textbackslash renewcommand*{\textbackslash bbrcomaboveskip}[#1]}}

\texttt{\textbackslash define@key{bbrcom}{\textbackslash afterskip}[]{\textbackslash renewcommand*{\textbackslash bbrcomafterskip}[#1]}}

\texttt{\textbackslash define@key{bbrcom}{\textbackslash belowskip}[]{\textbackslash renewcommand*{\textbackslash bbrcombelowskip}[#1]}}

Provide means for fixed message offset from top or bottom

\texttt{\textbackslash newcommand*{\textbackslash bbrcomfixedoffset}{}}

\texttt{\textbackslash newcommand*{\textbackslash bbrcomfixedboffset}{false}}

\texttt{\textbackslash define@key{bbrcom}{\textbackslash fixedoffset}[]{\textbackslash renewcommand*{\textbackslash bbrcomfixedoffset}[#1]}}

\texttt{\textbackslash define@key{bbrcom}{\textbackslash fixedboffset}[]{\textbackslash renewcommand*{\textbackslash bbrcomfixedoffset}[#1]\textbackslash renewcommand*{\textbackslash bbrcomfixedboffset}{true}}}
\begin{document}
\begin{tikzpicture}[BBRCOM-SIDESTYLE/.style/.expand once=\bbrcomsidestyle]
\setkeys{bbrcom}{#1}
\tikzset{BBRCOM-OSIDESTYLE/.style/.expand once=\bbrcomosidestyle}
\tikzset{BBRCOM-TOPSTYLE/.style/.expand once=\bbrcomtopstyle}
\tikzset{BBRCOM-BOTTOMSTYLE/.style/.expand once=\bbrcombottomstyle}
\tikzset{BBRCOM-EDGESTYLE/.style/.expand once=\bbrcomedgestyle}
\@bbrcom@check@islast{}
\@bbrcombeforeskip
\ifthenelse{\equal{\@bbrcomfixedoffset}{}}{\ifthenelse{\equal{\@tmp@bbr@isfirst}{true}}{\@bbr@lastskip}{\@bbr@lastskip}}{\setlength{\@bb@com@tmpoffset}{#2}}
\setlength{\@bb@com@tmpoffset}{\@bbrcomfixedoffset}
\@bb@comfinalize
\newcommand{\@bb@comfinalize}[1]{#1{\@bbrcomafterskip}
\endgroup
\def\@bbr@lastskip{\@bbr@intermessage@skip}
\}
\@bbrmsg
\newcommand{\@bbrmsg}[9]{\@bb@comsetup{#1}{#7}{#8}{#9}
\ifthenelse{\equal{\@bbrcomfixedboffset}{true}}{\% from bottom
\ifthenelse{\equal{#4}{north east}}{\def\@bbr@tmp@bottomanchor{south east}}{}
\ifthenelse{\equal{#4}{north west}}{\def\@bbr@tmp@bottomanchor{south west}}{}
\node[#3=-\@bb@com@tmpoffset and \@bbrcomlength of \@bb@lastbox.#4,anchor=#6,BBRCOM-SIDESTYLE] (#@bbrcomside) {\@bbrcomside};
\node[#3=\@bb@com@tmpoffset and \@bbrcomlength of \@bb@lastbox.#4,anchor=#6,BBRCOM-SIDESTYLE] (#@bbrcomside) {\@bbrcomside};
\node[#3=\@bb@com@tmpoffset and \@bbrcomlength of \@bb@lastbox.#4,anchor=#6,BBRCOM-SIDESTYLE] (#@bbrcomside) {\@bbrcomside};
\node[#3=\@bb@com@tmpoffset and \@bbrcomlength of \@bb@lastbox.#4,anchor=#6,BBRCOM-SIDESTYLE] (#@bbrcomside) {\@bbrcomside};
\node[#3=\@bb@com@tmpoffset and \@bbrcomlength of \@bb@lastbox.#4,anchor=#6,BBRCOM-SIDESTYLE] (#@bbrcomside) {\@bbrcomside};
\node[#3=\@bb@com@tmpoffset and \@bbrcomlength of \@bb@lastbox.#4,anchor=#6,BBRCOM-SIDESTYLE] (#@bbrcomside) {\@bbrcomside};
\node[#3=\@bb@com@tmpoffset and \@bbrcomlength of \@bb@lastbox.#4,anchor=#6,BBRCOM-SIDESTYLE] (#@bbrcomside) {\@bbrcomside};
\node[#3=\@bb@com@tmpoffset and \@bbrcomlength of \@bb@lastbox.#4,anchor=#6,BBRCOM-SIDESTYLE] (#@bbrcomside) {\@bbrcomside};
\node[#3=\@bb@com@tmpoffset and \@bbrcomlength of \@bb@lastbox.#4,anchor=#6,BBRCOM-SIDESTYLE] (#@bbrcomside) {\@bbrcomside};
\node[#3=\@bb@com@tmpoffset and \@bbrcomlength of \@bb@lastbox.#4,anchor=#6,BBRCOM-SIDESTYLE] (#@bbrcomside) {\@bbrcomside};
\node[#3=\@bb@com@tmpoffset and \@bbrcomlength of \@bb@lastbox.#4,anchor=#6,BBBRCOM-SIDESTYLE] (#@bbrcomside) {\@bbrcomside};
\node[#3=\@bb@com@tmpoffset and \@bbrcomlength of \@bb@lastbox.#4,anchor=#6,BBBRCOM-SIDESTYLE] (#@bbrcomside) {\@bbrcomside};
\node[#3=\@bb@com@tmpoffset and \@bbrcomlength of \@bb@lastbox.#4,anchor=#6,BBBRCOM-SIDESTYLE] (#@bbrcomside) {\@bbrcomside};
\node[#3=\@bb@com@tmpoffset and \@bbrcomlength of \@bb@lastbox.#4,anchor=#6,BBBRCOM-SIDESTYLE] (#@bbrcomside) {\@bbrcomside};
\node[#3=\@bb@com@tmpoffset and \@bbrcomlength of \@bb@lastbox.#4,anchor=#6,BBBRCOM-SIDESTYLE] (#@bbrcomside) {\@bbrcomside};
\node[#3=\@bb@com@tmpoffset and \@bbrcomlength of \@bb@lastbox.#4,anchor=#6,BBBRCOM-SIDESTYLE] (#@bbrcomside) {\@bbrcomside};
\node[#3=\@bb@com@tmpoffset and \@bbrcomlength of \@bb@lastbox.#4,anchor=#6,BBBRCOM-SIDESTYLE] (#@bbrcomside) {\@bbrcomside};
\node[#3=\@bb@com@tmpoffset and \@bbrcomlength of \@bb@lastbox.#4,anchor=#6,BBBRCOM-SIDESTYLE] (#@bbrcomside) {\@bbrcomside};
\node[#3=\@bb@com@tmpoffset and \@bbrcomlength of \@bb@lastbox.#4,anchor=#6,BBBRCOM-SIDESTYLE] (#@bbrcomside) {\@bbrcomside};
\node[#3=\@bb@com@tmpoffset and \@bbrcomlength of \@bb@lastbox.#4,anchor=#6,BBBRCOM-SIDESTYLE] (#@bbrcomside) {\@bbrcomside};
\node[#3=\@bb@com@tmpoffset and \@bbrcomlength of \@bb@lastbox.#4,anchor=#6,BBBRCOM-SIDESTYLE] (#@bbrcomside) {\@bbrcomside};
\node[#3=\@bb@com@tmpoffset and \@bbrcomlength of \@bb@lastbox.#4,anchor=#6,BBBRCOM-SIDESTYLE] (#@bbrcomside) {\@bbrcomside};
\path[#2](\@bbrcomsidename.#6) edge[BBRCOM-EDGESTYLE] node[above,BBRCOM-TOPSTYLE] (#@bbrcomtopname) {\@bbrcomtop} ...
\@bb@comfinalize{#8}
\}
\@bbrmsgto
\@bbrmsgfrom
\@bbrmsgtofrom
\@bbrmsgfromto
\end{document}
\node[at=($(bbrleft.west)!0.5!(bbright.east)$),anchor=center,BBRLOOP-CENTERSTYLE]() {brcomloopcenter};
\endgroup

\newcommand*\bbrintertexthoffset{1.5cm}
\define@key{bbrintertext}{xshift}[]{\renewcommand*\bbrintertexthoffset{#1}}
\newcommand{\@bb@intertextsetup}[1]{
\load keys
\begingroup % for local keys
% fix align environment (e.g. for use of pseudocode)
% \pretocmd\start@align{\if@minipage\kern-0.5\abovedisplayskip\fi}{\if\iffalse}{\fi}
\setkeys{bbrcom,bbrabase,bbrintertext}{#1}\
\@bbrcom@check@islast{}
\tikzset{BBRBASE-NODESTYLE/.style/.expand once=\@bbrbasenodestyle}\
\@bbrintertext 7 -> whether or not this is the first msg/query
\newcommand{\@bb@intertext}[7]{
\edef\@tmp@bbr@isfirst{#7}
\renewcommand#7{false}
\newcommand{\@bb@intertextsetup}[#1]
\newcommand{\@bb@intertextfinalize}[1]{\setlength\@bb@com@tmpoffset{#4}\
\ifthenelse{\equal{\@bbrcomfixedoffset}{true}}{
\setlength\@bb@com@tmpoffset{\@bbr@intermessage@veryshortskip}}{\setlength\@bb@com@tmpoffset{\@bbr@com@bottomanchor{south east}}}}
}
\ifthenelse{\equal{#3}{\north west}}{\def\bbrtmpbottomanchor{\south west}}{}

\node[#2=-\@bb@com@tmpoffset and \bbrintertexthoffset of \@bb@lastbox.\bbrtmpbottomanchor, inner sep=0, outer sep=0, BBRBASE-NODESTYLE] {#6};

\node[#2=\@bb@com@tmpoffset and \bbrintertexthoffset of \@bb@lastbox.\north west, inner sep=0, outer sep=0, BBRBASE-NODESTYLE] {#6};

% compute height of node
\coordinate (@bbtmpcoord) at (@bbasenodename.north);
\path (@bbtmpcoord);
\pgfgetlastxy{\XCoord}{\YCoordA}
\coordinate (@bbtmpcoord) at (@bbasenodename.south);
\path (@bbtmpcoord);
\pgfgetlastxy{\XCoord}{\YCoordB}
\setlength{\@bb@tmplength@b}{\YCoordA-\YCoordB}
#5{\the\@bb@tmplength@b}
\@bb@intertextfinalize{#5}

\newcommand{\bbrmsgtxt}[2][]{
\bbrintertext[#1]{\below left}{\north west}{\bbmessage@voffset}{\bbrmsgspace}{#2}{\bbr@first@msg}
}

\newcommand{\bbrquerytxt}[2][]{
\bbrintertext[#1]{\below right}{\north east}{\bbquery@voffset}{\bbrqueryspace}{#2}{\bbr@first@query}
}

\newcommand{\bbrchallengertxt}[2][]{
\begingroup
\setlength{\@bb@tmplength@b}{\bbrchallengerhdistance/2}%
\renewcommand{\bbrintertexthoffset}{\the\@bb@tmplength@b}%
\bbrintertext[#1]{\below left}{\north west}{\bbchallengerquery@voffset}{\bbrchallengerqryspace}{#2}{\bbr@first@challengerquery}
\endgroup
}

\newcommand{\bbroracletxt}[2][]{
\begingroup
\setlength{\@bb@tmplength@b}{\bbroracledistance/2}%
\renewcommand{\bbrintertexthoffset}{\the\@bb@tmplength@b}%
\bbrintertext[#1]{\below left}{\north west}{\bboraclequery@voffset}{\bbroracleqryspace}{#2}{\bbr@first@oraclequery}
\endgroup
}

\newcommand{\bbrmsgspace}[1]{\@pc@globaladdtolength{\bbmessage@voffset}{#1}}

\newcommand{\bbrqueryspace}[1]{\@pc@globaladdtolength{\bbquery@voffset}{#1}}

\newcommand{\bbroracleqryspace}[1]{\@pc@globaladdtolength{\bboraclequery@voffset}{#1}}
9.12 Game-Based Proofs

```
\newcounter{pcstartgamecounter}
\% \gamechange
Highlighting of changes between games. Highlight color can be set via \gamechangecolor
\definecolor{gamechangecolor}{gray}{0.90}
\newcommand{\gamechange}[2]{gamechangecolor}{0.90}
\setlength{\fboxsep}{0pt}\colorbox{#1}{\ifmmode\displaystyle#2\else#2\fi}}%
\pcbox
A simple box for conditional (ie., boxed) lines.
\newcommand{\pcbox}[1]{\setlength{\fboxsep}{3pt}\fbox{$\displaystyle#1$}}%
\pcgame
\pcgamename
\pcgameprocedurestyle
\newcommand*{\pcgamename}{Game}
\newcommand*{\pcgameprocedurestyle}[1]{\ensuremath{\mathsf{#1}}}
\def{\pcgame}{\bgroup\pcgame@}
\newcommand{\pcgame@}[1]\{\ifthenelse{\equal{#1}{}}{\pcgame@@}{\pcgame@@@{#1}}
\def{\pcgame@@}{\pcgameprocedurestyle{\pcgamename}\egroup}
\def{\pcgame@@@#1}{\ensuremath{\pcgameprocedurestyle{\pcgamename_{\normalfont{#1}}}\egroup}
\@pc@gametitle
Creates the header/title of a game
\newcommand{\@pc@gametitle}[1]\{\ifthenelse{\equal{#1}{}}{\pcgame\\gameprocedurearg}{\pcgame\[#1\]\gameprocedurearg}
\\ensuremath{\pcgame\\gameprocedurearg}{}
\gameproof
\newcommand*{\@pcgameproofgamenr}{0}
\define@key{pcgameproof}{nr}\{/\renewcommand{\@pcgameproofgamenr}{#1}}
\define@key{pcgameproof}{name}\{/\renewcommand{\pcgamename}{\ensuremath{#1}}
\define@key{pcgameproof}{arg}\{/\renewcommand{\gameprocedurearg}{\ensuremath{#1}}
\newenvironment{gameproof}[1]\{\renewcommand*{\@pcgameproofgamenr}{0}
\renewcommand{\@pcgameproofgamenr}{#1}\}
\begin{gameproof}[1]\%
\setkeys{pcgameproof}{#1}\%
```

9.12.1 Game Descriptions

\begin{macrocode}
\newenvironment{gamedescription}[1][]{% 
  \begingroup 
  \setkeys{pcgameproof}{#1} 
  \renewcommand{\withingamedescription}{true} 
  \@pc@ensureremember 
  \setcounter{pcgamecounter}{\@pcgameproofgamenr} \stepcounter{pcstartgamecounter} 
  \begin{description} 
} {\end{description} \@pc@releaseremember \endgroup}
\newcommandx*{\describegame}[1][1]{% 
  \addtocounter{pcgamecounter}{1} 
  \item[\pcdraw{ 
    \gdef\i{\thepcgamecounter} 
    \node[inner sep=0.0em,outer sep=0, xshift=-1ex, yshift=0.5ex] (gamenode\i) {}; 
  }]{\withingamedescription:} 
  \ifthenelse{\equal{}{\@pcgamehophint}}{}{ 
    \hspace{-0.7ex}\pcdraw{ 
      \tikzset{GAMEHOP-PATH-STYLE/.style/.expand once=\@pcgamehoppathestyle} 
      \tikzset{GAMEHOP-NODE-STYLE/.style/.expand once=\@pcgamehopnodestyle} 
      \draw[->,GAMEHOP-PATH-STYLE] (gamenode\thepcgamecounter) --++ (0,-\@pcgamehoplength) node[midway,above,xshift=-1mm,rotate=90,GAMEHOP-NODE-STYLE] {\@pcgamehophint}; 
  }} 
  \ifthenelse{\equal{}{\@pcgamehopinhint}}{}{ 
    \hspace{-0.7ex}\pcdraw{ 
      \tikzset{GAMEHOP-PATH-STYLE/.style/.expand once=\@pcgamehoppathestyle} 
      \tikzset{GAMEHOP-NODE-STYLE/.style/.expand once=\@pcgamehopnodestyle} 
      \draw[<-,GAMEHOP-PATH-STYLE] (gamenode\thepcgamecounter) --++ (0,\@pcgamehoplength) node[midway,above,xshift=-1mm,rotate=90,GAMEHOP-NODE-STYLE] {\@pcgamehopinhint}; 
  }} 
  \end{macrocode}

\fi
Change History

v0.04  
General: added \pcabort. ............... 1  
        better control whitespace for \pcif,  
        \pcelse, \pcelseif. ............... 1

v0.05  
General: add bottom to namepos in  
        bbrbox ....................... 1  
        angle for bbrloop ........... 1  
        fix length for bbrinput .... 1  
        introduce hoffset for bbrinput . 1  
        names for bbrinput and bbroutput . 1  
        side and oside support to  
        \bbroraclecryto and  
        \bbroraclequeryfrom ............ 1

v0.06  
General: added \pcunless .......... 1

v0.10  
General: Initial version ........... 1

v0.11  
General: Added \pcmbox environment  
        for matrices in pseudocode. .... 1  
        Added \NAND command. ......... 1  
        changed command pckeystyle to  
        ensure that subscripts on sk and pk  
        are aligned the same before,  
        (sk_R, pk_R) had slightly misaligned  
        subscripts due to Tex treating  
        subscripts on composite objects  
        with descenders differently than  
        without. ...................... 1

v0.20  
General: Added \pccommand. ......... 1  
        Added namepos middle for bbrbox. . 1  
        Added valign to pseudocode to  
        allow minipage vertical alignment. . 1  
        Changed minheight for bbrbox  
        environment to actually reflect a  
        minimum height in tikz. The old  
        minheight which added space at  
        the bottom was preserved as  
        addheight. ..................... 1  
        Ensure line numbers are right  
        aligned to allow for two digit  
        line numbers having the same  
        width. ......................... 1

v0.30  
General: replace obsolete 3regex .... 1

v0.31  
General: added \prp ................. 1  
        added \tprob (variants for prob and  
        co for in-text) .................. 1

v0.32  
General: allow overwriting rule  
        command in pseudocode via  
        headlinecmd (defaults to \hrule) .. 1  
        allow to control spacing with \pcfor 1

v0.40  
General: Adapted bbenv environment  
        to take key value option list. Old  
        format is still supported but  
        deprecated. .................... 1  
        Added \argmax and \argmin to  
        operators. ..................... 1  
        Added \pindist, \sindist, and  
        \cindist to operators. .......... 1  
        Added aboveskip and belowskip  
        option to \pchstack and \pcvstack. 1  
        Added additional adversaries. .... 1  
        Added additional complexity classes. 1  
        Added additional polynomials. .... 1  
        Added block forms for pseudocode  
        and procedure commands  
        (\pseudocodeblock and  
        \procedureblock). ................ 1  
        Added boxed, inline, noindent  
        options to \pchstack and  
        \pcvstack. ...................... 1  
        Added clockwise, leftstyle,  
        centerstyle, rightstyle for bbrloop.  
        Adjusted placing of center. ....... 1  
        Added command \pcsetargs to  
        define default arguments for  
        pseudocode blocks. ............ 1  
        Added command \pcsethstackargs  
        and \pcsetvstackargs to define  
        default arguments for hstack and  
        vstack environments. ............ 1  
        Added fixedoffset, fixedboffset, islast  
        for reduction messages. .......... 1  
        Added headheight option to  
        \pseudocode. ................... 1  
        Added minlineheight option to  
        \pseudocode. ................... 1  
        Added oracles package option. .... 1  
        Added space option to \pchstack  
        and \pcvstack. ................... 1  
        Adjusted spacing via \pcaboveskip  
        and \pcbelowskip which are added
to \pseudocode blocks and \pchstack environments ............ 1

Bigger refactoring. Not completely backwards compatible. In particular, optimized spacing of pseudocode blocks and black box reductions. ............... 1

Fixed spacing issues with black box reduction messages. ............ 1

Renamed horizontal spacing commands \beforepcskip and \afterpcskip to \pcbeforeskip and \pcafterskip. ....................... 1

Switched to mathtools \DeclarePairedDelimiter for paired operators. Each paired operator comes in two forms, e.g, \texttt{abs} and \texttt{tabs} the latter to be used in flowtext which does not scale the outer delimiters. ....................... 1