Cryptocode

Typesetting Cryptography

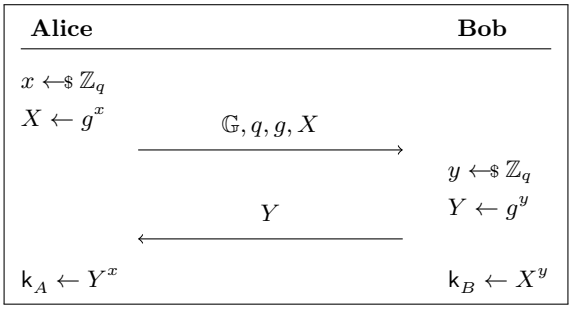
Version v0.43

Arno Mittelbach
mail@arno-mittelbach.de
https://github.com/arnomi/cryptocode*

January 17, 2021

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 $\mathsf{Adv}^{\mathrm{prf}}_{\mathcal{A},\mathsf{PRF}}(n) = \mathsf{negl}(n)$. Furthermore, it includes environments to layout game-based proofs or black-box reductions.



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

Contents

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

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

Note that all the options refer to a set of commands. That is, without any options cryptocode will provide the mechanisms for writing pseudocode, protocols, game-based proofs and black-box reductions but not define additional commands, such as `\pk` or `\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 `\pseudocode` command takes a single parameter where you can start writing code in mathmode using `\\` as line breaks. Following is an IND-CPA game definition using various commands from cryptocode to ease writing keys (`\pk`,`\sk`), sampling (`\sample`), and more:

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

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

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 `\pchstack` and `\pcvstack`. For typesetting a code block nicely centered and boxed

$b \leftarrow\$ \{0,1\}$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow\$ \mathsf{KGen}(1^n)$
$(\mathsf{state}, m_0, m_1) \leftarrow\$ \mathcal{A}(1^n, \mathsf{pk}, c)$
$c \leftarrow\$ \mathsf{Enc}(\mathsf{pk}, m_b)$
$b' \leftarrow\$ \mathcal{A}(1^n, \mathsf{pk}, c, \mathsf{state})$
return $b = b'$

you could thus use:

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

As this is a common task, cryptocode offers the `\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 `\pcb`) you can easily define the command via

```
\createpseudocodeblock{pcb}{center,boxed}{}{}{}
```

The above could now be written, more succinctly as

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

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 `\procedure` command (or its block variant `\procedureblock`) which takes an additional argument to specify the headline.

$\text{IND-CPA}^{\mathcal{A}}_{\mathsf{Enc}}(n)$

$b \leftarrow\$ \{0,1\}$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow\$ \mathsf{KGen}(1^n)$
$(\mathsf{state}, m_0, m_1) \leftarrow\$ \mathcal{A}(1^n, \mathsf{pk}, c)$
$c \leftarrow\$ \mathsf{Enc}(\mathsf{pk}, m_b)$
$b' \leftarrow\$ \mathcal{A}(1^n, \mathsf{pk}, c, \mathsf{state})$
return $b = b'$

```
\procedureblock[linenumbering]{$\indcpa_\enc^\adv(\secpar)$}{
  b \sample \bin   \\
  (\pk,\sk) \sample \kgen (\secparam)   \\
  (\state ,m_0,m_1) \sample \adv(\secparam , \pk, c)    \\
  c \sample \enc(\pk,m_b)   \\
  b' \sample \adv(\secparam , \pk, c, \state) \\
  \pcreturn b = b' }
```

Similarly to before, we can define a shorthand and boxed variant as

```
\createprocedureblock{procb}{center,boxed}{}{}{}
```

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.

$\text{IND-CPA}^{\mathcal{A}}_{\mathsf{Enc}}(n)$

1 : $b \leftarrow\$ \{0,1\}$
2 : $(\mathsf{pk},\mathsf{sk}) \leftarrow\$ \mathsf{KGen}(1^n)$

. . . . Setup Completed . . . .

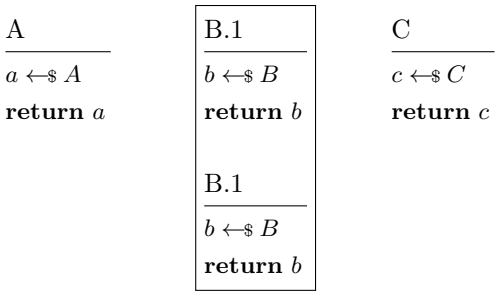
3 : $(m_0,m_1) \leftarrow\$ \mathcal{A}(1^n,\mathsf{pk},c)$
4 : $c \leftarrow\$ \mathsf{Enc}(\mathsf{pk},m_b)$
5 : $b' \leftarrow\$ \mathcal{A}(1^n,\mathsf{pk},c,\mathsf{state})$
6 : **return** $b = b'$

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

```
\begin{pcimage}
\procedureblock[linenumbering]{$\indcpa_\enc^\adv(\secpar)$}{%
  b \sample \bin   \\
  (\pk,\sk) \sample \kgen(\secparam)\pcnode{kgen}   \pclb
  \pcintertext[dotted]{Setup Completed}
  (m_0,m_1) \sample \adv(\secparam , \pk, c)   \\
   c \sample \enc(\pk,m_b)   \\
   b' \sample \adv(\secparam , \pk, c, \state) \\
   \pcreturn b = b' }

\pcdraw{
  \node[rectangle callout,callout absolute pointer=(kgen),fill=orange]
      at ([shift={(+3,-1)}]kgen) {
   \begin{varwidth}{3cm}
     $\kgen(\secparam)$ samples a public key $\pk$ and a private key $\sk$.
   \end{varwidth}
  };
}
\end{pcimage}
```

1.2 Stacking

To arrange multiple procedures, cryptocode offers horizontal and vertical stacking environments `\pchstack` 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.




```
\begin{pchstack}[center,space=1cm]
  \procedure{A}{
     a \sample A \\
    \pcreturn a
  }

  \begin{pcvstack}[boxed,space=0.5cm]
    \procedure{B.1}{
    b \sample B \\
    \pcreturn b
    }
    \procedure{B.1}{
    b \sample B \\
    \pcreturn b
    }
  \end{pcvstack}

  \procedure{C}{
     c \sample C \\
    \pcreturn c
  }
\end{pchstack}
```

1.3 Columns

The `\pseudocode` and `\procedure` commands allow the usage of multiple columns. You switch to a new column by inserting a `\>`. This is similar to using an `align` environment and placing a tabbing & character.[1]

First	**Second**	**Third**	**Fourth**
$b \leftarrow\$ \{0,1\}$	$b \leftarrow\$ \{0,1\}$	$b \leftarrow\$ \{0,1\}$	$b \leftarrow\$ \{0,1\}$

```
 \pseudocodeblock{%
  \textbf{First} \> \textbf{Second} \> \textbf{Third} \> \textbf{Fourth} \\
   b \sample \bin \> b \sample \bin \> b \sample \bin \> b \sample \bin}
```

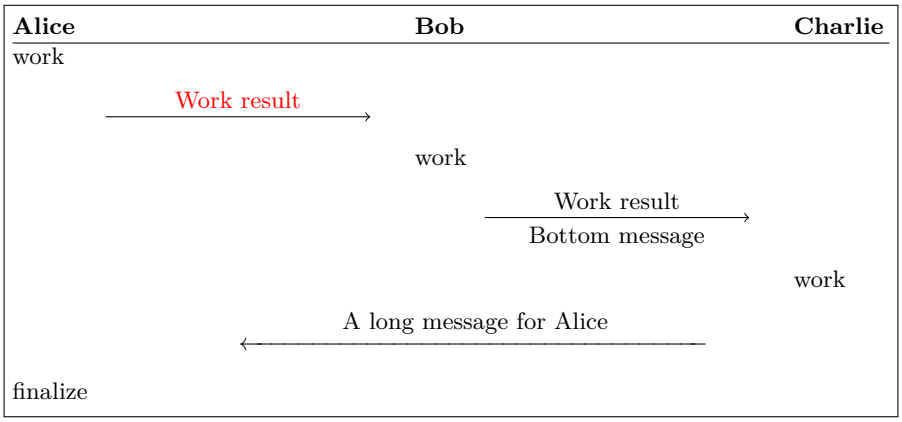
As you can see the first column is left aligned the second right, the third left and so forth. In order to get only left aligned columns you could thus always skip a column by using `\>\>` or you can alternatively use `\<` as a shorthand for `\>\>`.

First	**Second**	**Third**	**Fourth**
$b \leftarrow\$ \{0,1\}$	$b \leftarrow\$ \{0,1\}$	$b \leftarrow\$ \{0,1\}$	$b \leftarrow\$ \{0,1\}$

[1] In fact, the *pseudocode* command is based on amsmath's flalign environment.

1.4 Protocols

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

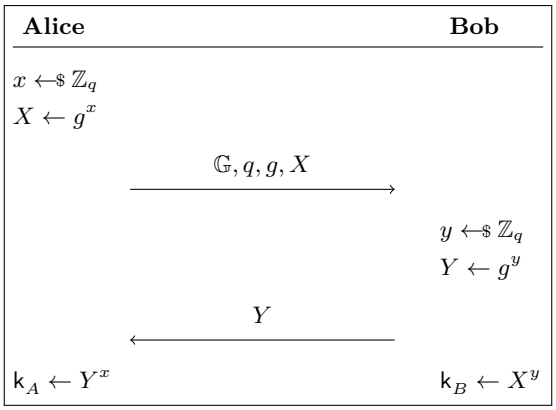



```
1  \pseudocodeblock{
2   \textbf{Alice} \< \< \textbf{Bob} \< \< \textbf{Charlie} \\[][\hline]
3   \text{work} \< \< \< \< \\
4   \< \sendmessageright{top=Work result,topstyle=red} \< \< \< \\
5   \< \< \text{work} \< \< \\
6   \< \< \<  \sendmessageright{top=Work result,bottom=Bottom message} \< \\
7   \< \<  \< \< \text{work} \\
8   \< \sendmessageleftx{8}{\text{A long message for Alice}} \< \\
9   \text{finalize} \< \< \< \< }
```

The commands `\sendmessageright` and `\sendmessageleft` are very flexible and allow to style the sending of messages in various ways. Also note the `\\[][\hline]` at the end of the first line. Here the first optional argument allows us to specify the lineheight (similarly to the behavior in an align environment) 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 `\sendmessagerightx` and `\sendmessageleftx` (note the x at the end) allow to send messages over multiple columns. In the example, as we were using `\<` the final message thus spans 8 columns.

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



```
\pseudocodeblock{
  \textbf{ Alice} \< \< \textbf{ Bob}  \\[0.1\baselineskip][\hline]
  \<\< \\[-0.5\baselineskip]
  x \sample \ZZ_q \< \< \\
  X \gets g^x \<\< \\
  \< \sendmessageright*{\GG,q,g,X} \< \\
  \<\< y \sample \ZZ_q \\
  \<\< Y \gets g^y \\
  \< \sendmessageleft*{Y} \< \\
  \key_A \gets Y^x \<\< \key_B \gets X^y }
```

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.

$\mathsf{Game}_1(n)$	$\mathsf{Game}_2(n)$ $\boxed{\mathsf{Game}_3(n)}$	$\mathsf{Game}_4(n)$
1 : Step 1	Step 1	Step 1
2 :	From game 3 on	From game 3 on
3 : Step 2	Step 3 is different	Step 3 adapted again

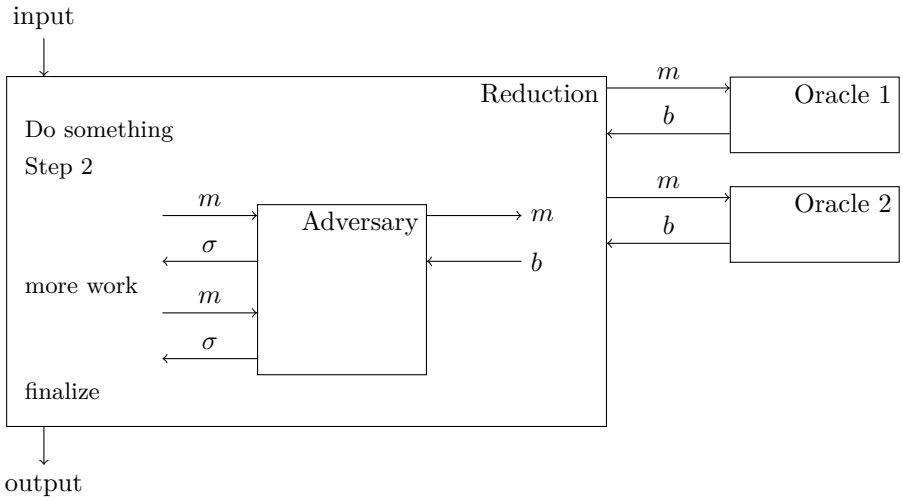
```
\begin{gameproof}
  \begin{pchstack}[center,space=1em]
    \gameprocedure[linenumbering,minlineheight=1.5em]{%
      \text{Step 1}  \\
      \\
      \text{Step 3}
     }

    \tbxgameprocedure[minlineheight=1.5em]{%
       \text{Step 1} \\
       \pcbox{\text{From game 3 on}}  \\
       \gamechange{\text{Step 3 is different}}
     }

    \gameprocedure[minlineheight=1.5em]{%
      \text{Step 1}  \\
      \text{From game 3 on}\\
      \text{\gamechange{Step 3 adapted again}}
     }

  \end{pchstack}
\end{gameproof}
```

Note that we made use of the option “mode=text” in the above example which tells the underlying pseudocode command to not work in math mode but in plain text mode. We will discuss how to visualize game-based proofs in Section 6.

1.6 Black-Box Reductions

Cryptocode provides a strucured 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

plain TIKZ commands. Following is an example reduction. We discuss the details in Section 7.




```
\begin{bbrenv}{A}
  \begin{bbrbox}[name=Reduction]
  \pseudocode{
    \text{Do something} \\
    \text{Step 2}
  }

  \begin{bbrenv}{B}
    \begin{bbrbox}[name=Adversary,minheight=2.25cm]
    \end{bbrbox}

    \bbrmsgto{top=$m$}
    \bbrmsgfrom{top=$\sigma$}
    \bbrmsgtxt{\pseudocode{%
      \text{more work}
    }}
    \bbrmsgto{top=$m$}
    \bbrmsgfrom{top=$\sigma$}

    \bbrqryto{side=$m$}
    \bbrqryfrom{side=$b$}
  \end{bbrenv}

  \pseudocode{
    \text{finalize}
  }

  \end{bbrbox}
  \bbrinput{input}
  \bbroutput{output}

  \begin{bbroracle}{OraA}
    \begin{bbrbox}[name=Oracle 1,minheight=1cm]
    \end{bbrbox}
  \end{bbroracle}
  \bbroracleqryto{top=$m$}
  \bbroracleqryfrom{top=$b$}

  \begin{bbroracle}{OraB}
    \begin{bbrbox}[name=Oracle 2,minheight=1cm]
    \end{bbrbox}
  \end{bbroracle}
  \bbroracleqryto{top=$m$}
  \bbroracleqryfrom{top=$b$}
\end{bbrenv}
```

2 Notation Macros

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

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

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^λ. The cryptocode package, when loading either option "n" or option "lambda" will define the commands

```
  \secpar
  \secparam
  \SECPAR
```

The first command provides the "letter", i.e., either n or λ, whereas `\secparam` prints `\1^\secpar` (i.e., 1^n for option "n"). Finally, `\SECPAR` yields N_0 (resp. Λ) 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:

$$\mathsf{Adv}^{\mathrm{prf}}_{\mathcal{A},\mathsf{PRF}}(n)$$

```
  \advantage{prf}{\adv,\prf}
```

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

```
  \advantage{prf}{\adv,\prf}[(arg)]
```

In order to redefine the styles in which superscript and subscript are set, or in case you want to replace the term Adv, redefine:

```
\renewcommand{\pcadvantagename}{\mathsf{Adv}}
\renewcommand{\pcadvantagesuperstyle}[1]{\mathrm{\MakeLowercase{#1}}}
\renewcommand{\pcadvantagesubstyle}[1]{#1}
```

2.3 Math Operators

The "operators" option provides the following list of commands:

Command	Description	Result	Example
`\sample`	Sampling from a distribution, or running a randomized procedure	$\leftarrow\!\!\$$	$b \leftarrow\!\!\$ \{0,1\}$
`\floor{42.5}`	Rounding down	$\lfloor 42.5 \rfloor$	
`\ceil{41.5}`	Rounding up	$\lceil 41.5 \rceil$	
`\Angle{x,y}`	Vector product	$\langle x, y \rangle$	
`\abs{\frac{a}{b}}`	Absolute number	$\left\lvert \frac{a}{b} \right\rvert$	
`\norm{x}`	Norm	$\lVert x \rVert$	
`\concat`	Verbose concatenation (I usually prefer simply `\\|`)	$\Vert$	$x \leftarrow a \Vert b$
`\emptystring`	The empty string	ε	$x \leftarrow \varepsilon$
`\argmax`	arg max	$\arg\max$	$\arg\max_{x \in S} f(x)$
`\argmin`	arg min	$\arg\min$	$\arg\min_{x \in S} f(x)$
`\pindist`	Perfect indistinguishability	$\overset{\text{p}}{=}$	$X \overset{\text{p}}{=} Y$
`\sindist`	Statistical indistinguishability	$\overset{\text{s}}{\approx}$	$X \overset{\text{s}}{\approx} Y$
`\cindist`	Computational indistinguishability	$\overset{\text{c}}{\approx}$	$X \overset{\text{c}}{\approx} Y$

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

Note that $\arg\max$ and $\arg\min$ 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:

Command	Description	Result
`\adv`	Adversary	$\mathcal{A}$
`\bdv`	Adversary	$\mathcal{B}$
`\cdv`	Adversary	$\mathcal{C}$
`\ddv`	Adversary	$\mathcal{D}$
`\edv`	Adversary	$\mathcal{E}$
`\mdv`	Adversary	$\mathcal{M}$
`\pdv`	Adversary	$\mathcal{P}$
`\rdv`	Adversary	$\mathcal{R}$
`\sdv`	Adversary	$\mathcal{S}$

The style in which an adversary is rendered is controlled via

```
\renewcommand{\pcadvstyle}[1]{\ensuremath{\mathcal{#1}}}
```

2.5 Landau

The "landau" option provides the following list of commands:

Command	Description	Result
`\bigO{n^2}`	Big O(micron) notation	$\mathcal{O}(n^2)$
`\smallO{n^2}`	small o(micron) notation	$\mathrm{o}(n^2)$
`\bigOmega{n^2}`	Big Omega notation	$\Omega(n^2)$
`\bigTheta{n^2}`	Big Theta	$\Theta(n^2)$
`\orderOf`	On the order of	$f(n) \sim g(n)$

2.6 Probabilities

The "probability" option provides commands for writing probabilities. Use

```
\prob{X=x}
\probsub{x\sample{\bin^n}}{x=5}
\condprob{X=x}{A=b}
\condprobsub{x\sample{\bin^n}}{x=5}{A=b}
```

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

$$\Pr[X = x]$$
$$\Pr_{x \leftarrow\$ \{0,1\}^n}[X = x]$$
$$\Pr[X = x \mid A = b]$$
$$\Pr_{x \leftarrow\$ \{0,1\}^n}[x = 5 \mid A = b]$$

You can control the probability symbol (Pr) by redefining

```
\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

```
\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

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

yielding

$$\mathbb{E}[X]$$
$$\mathbb{E}_{x,y \leftarrow\$ \{1,\ldots,6\}}[x + y]$$
$$\mathbb{E}[X + Y \mid Y > 3]$$
$$\mathbb{E}_{x,y \leftarrow\$ \{1,\ldots,6\}}[x + y \mid y > 3]$$

Again flowtext versions such as `\texpect{X}` are available. To control the expactation symbol ($\mathbb{E}$), redefine

```
\renewcommand{\expectationname}{\ensuremath{\mathbb{E}}}
```

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

```
\supp{X}
```

where again the name can be controlled via

```
\renewcommand{\supportname}{Supp}
```

For denoting entropy and min-entropy use

```
\entropy{X}
\minentropy{X}
\condentropy{X}{Y=5}
\condminentropy{X}{Y=5}
\condavgminentropy(X}{Y=5}
```

This yields

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

2.7 Sets

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

```
\set{1, \ldots, 10}
\sequence{1, \ldots, 10}
```

which are typeset as

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

In addition, the following commands are provided

Command	Description	Result
`\bin`	The set containing 0 and 1	$\{0, 1\}$
`\NN`	Natural numbers	$\mathbb{N}$
`\ZZ`	Integers	$\mathbb{Z}$
`\QQ`	Rational numbers	$\mathbb{Q}$
`\CC`	Complex numbers	$\mathbb{C}$
`\RR`	Reals	$\mathbb{R}$
`\PP`		$\mathbb{P}$
`\FF`		$\mathbb{F}$
`\GG`		$\mathbb{G}$

The style in which sets are being set can be adapted by redefining

```
\renewcommand{\pcsetstyle}[1]{\ensuremath{\mathbb{#1}}}
```

2.8 Cryptographic Notions

The "notions" option defines the following list of commands:

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

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

```
\renewcommand{\pcnotionstyle}[1]{\ensuremath{\mathrm{#1}}}
```

2.9 Logic

The "logic" option provides the following list of commands:

Command	Description	Result
`\AND`	Logical AND	AND
`\NAND`	Logical NAND	NAND
`\OR`	Logical OR	OR
`\NOR`	Logical NOR	NOR
`\XOR`	Logical XOR	XOR
`\XNOR`	Logical XNOR	XNOR
`\notimplies`	Negated implication	$\not\Rightarrow$
`\NOT`	not	NOT
`\xor`	exclusive or	$\oplus$
`\false`	false	false
`\true`	true	true

2.10 Function Families

The "ff" option provides the following list of commands:

Command	Description	Result
\kgen	Key generation	KGen
\pgen	Parameter generation	Pgen
\eval	Evaluation	Eval
\invert	Inversion	Inv
\il	Input length	il
\ol	Output length	ol
\kl	Key length	kl
\nl	Nonce length	nl
\rl	Randomness length	rl

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

```
\renewcommand{\pcalgostyle}[1]{\ensuremath{\mathsf{#1}}}
```

2.11 Machine Model

The "mm" option provides the following list of commands:

Command	Description	Result
\CRKT	A circuit	C
\TM	A Turing machine	M
\PROG	A program	P
\uTM	A universal Turing machine	UM
\uC	A universal Circuit	UC
\uP	A universal Program	UEval
\tmtime	Time (of a TM)	time
\ppt	Probabilistic polynomial time	PPT

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

```
\renewcommand{\pcmachinemodelstyle}[1]{\ensuremath{\mathsf{#1}}}
```

2.12 Crypto Primitives

The "primitives" option provides the following list of commands:

Command	Description	Result
`\prover`	Proover	P
`\verifier`	Verifier	V
`\nizk`	Non interactive zero knowledge	NIZK
`\hash`	A hash function	H
`\gash`	A hash function	G
`\fash`	A hash function	F
`\pad`	A padding function	pad
`\enc`	Encryption	Enc
`\dec`	Decryption	Dec
`\sig`	Signing	Sig
`\sign`	Signing	Sign
`\verify`	Verifying	Vf
`\owf`	One-way function	OWF
`\prf`	Pseudorandom function	PRF
`\prp`	Pseudorandom permutation	PRP
`\prg`	Pseudorandom generator	PRG
`\obf`	Obfuscation	O
`\iO`	Indistinguishability obfuscation	iO
`\diO`	Differing inputs obfuscation	diO
`\mac`	Message authentication	MAC
`\puncture`	Puncturing	Puncture
`\source`	A source	S
`\predictor`	A predictor	P
`\sam`	A sampler	Sam
`\distinguisher`	A distinguisher	Dist
`\dist`	A distinguisher	D
`\simulator`	A simulator	Sim
`\extractor`	An extractor	Ext
`\ext`	Shorthand for `\extractor`	Ext

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

```
\renewcommand{\pcalgostyle}[1]{\ensuremath{\mathsf{#1}}}
```

2.13 Oracles

The "oracles" option provides the following list of commands:

Command	Description	Result
`\oracle`	Generic oracle	O
`\oracle[LoR]`	Custom oracle	LoR
`\ro`	Random oracle	RO
`\Oracle{\sign}`	Oracle version of procedure	OSign

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

```
\renewcommand{\pcoraclestyle}[1]{\ensuremath{\mathsf{#1}}}
```

2.14 Events

The "events" option provides the following list of commands.

Command	Description	Result
\event{E}	Event E	E
\nevent{E}	Negated event E	$\overline{\mathsf{E}}$
\bad	Bad event	bad
\nbad	Bad event	$\overline{\mathsf{bad}}$

2.15 Complexity

The "complexity" option provides the following list of commands:

Command	Result
\complclass{myClass}	myClass
\cocomplclass{myClass}	co-myClass
\npol	NP
\conpol	co-NP
\pol	P
\bpp	BPP
\ppoly	P/poly
\NC{1}	NC^1
\AC{1}	AC^1
\TC{1}	TC^1
\AM	AM
\coAM	co-AM
\PH	PH
\csigma{1}	Σ_1^p
\cpi{1}	Π_1^p
\cosigma{1}	co-Σ_1^p
\copi{1}	co-Π_1^p

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

```
\renewcommand{\pccomplexitystyle}[1]{\ensuremath{\mathsf{#1}}}
```

2.16 Asymptotics

The "asymptotics" option provides the following list of commands:

Command	Description	Result
`\negl`	A negligible function	$\mathsf{negl}(n)$ (n is `\secpar`)
`\negl[x]`	A negligible function	$\mathsf{negl}(x)$
`\negl[]`	A negligible function	negl
`\poly`	A polynomial	$\mathsf{poly}(n)$ (n is `\secpar`)
`\poly[x]`	A polynomial	$\mathsf{poly}(x)$
`\poly[]`	A polynomial	poly
`\pp`	some polynomial `p`	p
`\pp[t]`	some custom polynomial `t`	t
`\cc`	some polynomial `c`	c
`\ee`	some polynomial `e`	e
`\kk`	some polynomial `k`	k
`\mm`	some polynomial `m`	m
`\nn`	some polynomial `n`	n
`\qq`	some polynomial `q`	q
`\rr`	some polynomial `r`	r

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

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

2.17 Keys

The "keys" option provides the following list of commands:

Command	Description	Result
`pk`	public key	pk
`vk`	verification key	vk
`sk`	secret key	sk
`key`	a plain key	k
`key[xk]`	custom key	xk
`hk`	hash key	hk
`gk`	gash key	gk
`fk`	function key	fk
`st`	state	st
`state`	state	state
`state{myState}`	custom state	$\mathsf{state}_{myState}$

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

```
\renewcommand{\pckeystyle}[1]{\ensuremath{\mathsf{#1}}}
```

3 Pseudocode

3.1 Basics

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

$$
\begin{aligned}
&b \leftarrow\$ \{0,1\} \\
&(\mathsf{pk}, \mathsf{sk}) \leftarrow\$ \mathsf{KGen}(1^n) \\
&(m_0, m_1) \leftarrow\$ \mathcal{A}(1^n, \mathsf{pk}, c) \\
&c \leftarrow\$ \mathsf{Enc}(\mathsf{pk}, m_b) \\
&b' \leftarrow\$ \mathcal{A}(1^n, \mathsf{pk}, c) \\
&\textbf{return } b = b'
\end{aligned}
$$

which is generated by

```
\begin{pchstack}[center]
\pseudocode{
    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)  \\
  \pcreturn b = b' }
\end{pchstack}
```

First note that `\pseudocode` on its own does not space itself. For laying out one (or multiple) code blocks cryptocode defines stacking environemnts such as `\pchstack` and `\pcvstack` that we discuss in Section 3.7. Wrapping a single pseudocode in a `\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 `\pseudocodeblock` which centers and offsets a pseudocode block as above. We thus get the very same by writing

```
\pseudocodeblock{
    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)  \\
  \pcreturn b = b' }
```

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

```
\createpseudocodeblock{pcb}{center,boxed}{}{}{}
```

(We discuss creating custom pseudocode commands in detail in Section 3.1.2). If we now use `\pcb` as just defined in the above example, we obtain the following nicely spaced and boxed result.

$b \leftarrow\$ \{0,1\}$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow\$ \mathsf{KGen}(1^n)$
$(m_0, m_1) \leftarrow\$ \mathcal{A}(1^n, \mathsf{pk}, c)$
$c \leftarrow\$ \mathsf{Enc}(\mathsf{pk}, m_b)$
$b' \leftarrow\$ \mathcal{A}(1^n, \mathsf{pk}, c)$
return $b = b'$

which is generated as

```
\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)  \\
   \pcreturn b = b' }
```

Remark. In the following we will use this boxed representation for the examples, but use `\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 `\pseudocode` command can take an optional argument which consists of a list of key=value pairs separated by commas.

```
\pseudocode[options]{body}
```

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 numberswhen 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 `\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 `\scriptsize` for a smaller size.

colspace Allows to insert spacing between columns. In particular this allows to also overlap columns by inserting negative space.

jot Allows to specify extra space between each line. Use `jot=1mm`.

beginline Allows to specify a macro that is placed at the beginning of each line.

endline Allows to specify a macro that is placed at the end of each line.

xshift Allows horizontal shifting

yshift Allows horizontal shifting

headlinesep Specifies the distance between header and the line. By default set to 0pt which can be globally overwritten by setting length `\pcheadlinesep`.

bodylinesep Specifies the distance between body and the line. By default set to `0.3\baselineskip` which can be globally overwritten by setting length `\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 `\pcheadheight`.

headlinecmd Allows to overwrite which command is used to draw the bar below the headline. Defaults to `\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

```
1 \pseudocodeblock[linenumbering,syntaxhighlight=auto,head=Header]{ return null }
```

creates

Header
1 : **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

```
\procedure[options]{Header}{Body}
\procedureblock[options]{Header}{Body}
```

Examples

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

which is generated as

```
\procedureblock{$\indcpa_\enc^\adv(\secpar)$}{
   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)  \\
   \pcreturn b = b' }
```

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

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

creates the commands `\mypseudocode` and `\myprocedure` with line numbering always enabled as well as the block commands `\pcb` and `\procb` 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

```
\createprocedureblock{expproc}{center,boxed}{}{$\mathrm{Experiment}$\xspace}{
    linenumbering}
```

could be used as

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

This results in

$\underline{\text{Experiment IND-CPA}^{\mathcal{A}}_{\mathsf{Enc}}(n)}$

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

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.

for $i = 1..10$ **do**
  $T[i] \leftarrow\$ \{0,1\}^n$
for $i = 1..10$ **do**
  $T[i] \leftarrow\$ \{0,1\}^n$

which is generated as

```
\pseudocodeblock{
    \pcfor i = 1..10 \pcdo  \\
    \pcind T[i] \sample \bin^n  \\
    \pcfor i = 1..10 \pcdo  \\
    \t T[i] \sample \bin^n }
```

You can specify multiple levels via the optional first argument

```
\t[level] % \pcind[level]
```

for $i = 1..10$ **do**
  $T[i] \leftarrow\$ \{0,1\}^n$
    $T[i] \leftarrow\$ \{0,1\}^n$
      $T[i] \leftarrow\$ \{0,1\}^n$
        $T[i] \leftarrow\$ \{0,1\}^n$
          $T[i] \leftarrow\$ \{0,1\}^n$

```
\pseudocodeblock{
    \pcfor i = 1..10 \pcdo  \\
    \t T[i] \sample \bin^n  \\
    \t\t T[i] \sample \bin^n \\
    \t[3] T[i] \sample \bin^n \\
    \t[4] T[i] \sample \bin^n \\
    \t[5] T[i] \sample \bin^n }
```

You can customize the indentation shortcut by redefining

```
\renewcommand{\pcindentname}{t}
```

Automatic Indentation

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

```
\pcif, \pcfor, \pcwhile, \pcrepeat, \pcforeach
```

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

```
\pcfi, \pcendif, \pcendfor, \pcendwhile, \pcuntil, \pcendforeach
```

Additionally, on seeing

```
\pcelse, \pcelseif
```

it will remove the indentation for that particular line. Thus the following

for $a \in [10]$ **do**
  for $a \in [10]$ **do**
    for $a \in [10]$ **do**
      if $a = b$ **then**
        some operation
      elseif $a = c$ **then**
        some operation
      else
        some default operation
      fi
    endfor
  endfor
endfor
return a

can be obtained by:

```
\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} \\
```

```
      \pcelseif a = c \pcthen \\
        \text{some operation} \\
      \pcelse \\
        \text{some default operation} \\
      \pcfi \\
    \pcendfor \\
  \pcendfor \\
\pcendfor \\
\pcreturn a}
```

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

Keep Input Indentation (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".

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

This yields the following result

$$\begin{array}{l}\textbf{for } i = 1..10 \textbf{ do}\\ \quad T[i] \leftarrow\$ \{0,1\}^n\\ \quad\quad T[i] \leftarrow\$ \{0,1\}^n\\ \quad\quad\quad T[i] \leftarrow\$ \{0,1\}^n\\ \quad\quad\quad\quad T[i] \leftarrow\$ \{0,1\}^n\\ \quad\quad\quad\quad\quad T[i] \leftarrow\$ \{0,1\}^n\end{array}$$

Note that automatic spacing only works when the `\pseudocode` command is not wrapped within another command. Thus in order to get a frame box `\fbox{\pseudocode[space=keep]{code}}` will not work but you would need to use an environment such as one offered by the *mdframed* package ((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

```
\pseudocodeblock[mode=text]{%
This is \\
\t simply text}
```

3.4 Syntax Highlighting

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

command	outcome
`\pcabort`	**abort**
`\pcassert`	**assert**
`\pccontinue`	**continue**
`\pccomment{comment}`	// comment
`\pccomment[2em]{comment}`	// comment
`\pclinecomment{comment}`	// comment
`\pcdo`	**do**
`\pcdone`	**done**
`\pcfail`	**fail**
`\pcfalse`	**false**
`\pcif`	**if**
`\pcfi`	**fi**
`\pcendif`	**endif**
`\pcelse`	**else**
`\pcelseif`	**elseif**
`\pcfor`	**for**
`\pcendfor`	**endfor**
`\pcforeach`	**foreach**
`\pcendforeach`	**endforeach**
`\pcglobvar`	**gbl**
`\pcin`	**in**
`\pcnew`	**new**
`\pcnull`	**null**
`\pcparse`	**parse**
`\pcrepeat{10}`	**repeat** 10 **times**
`\pcreturn`	**return**
`\pcuntil`	**until**
`\pcthen`	**then**
`\pctrue`	**true**
`\pcwhile`	**while**
`\pcendwhile`	**endwhile**

Note that `\pcdo`, `\pcin` and `\pcthen` have a leading space. This is due to their usual usage scenarios such as

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

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

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

```
1 \pseudocodeblock{\pcfor i \pcin[] \{1,\ldots,10\}}
```

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

```
\newcommand{\highlightkeyword}[2][\ ]{\ensuremath{\mathbf{#2}}#1}
```

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)

$n \leftarrow \mathsf{length}(A)$
repeat
  $s \leftarrow$ **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
      $\mathsf{swap}(A[i-1], A[i])$
      $s \leftarrow$ **true**
until $\neg s$

can be typeset as

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

can be typeset as

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

We can also combine automatic syntax highlighting with automatic spacing in which case we need to insert "group end" keywords:

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

```
\procedureblock[space=auto,syntaxhighlight=auto,addkeywords={swap,length}]{
    Bubblesort(A : list of items)}{
n \gets length(A) \\
repeat \\
    s \gets false \\
    for i=1 to n-1 do \\
        \pclinecomment{assuming this pair is out of order} \\
        if A[i-1]>A[i] then \\
            \pclinecomment{swap them and remember something changed} \\
```

```
9               swap(A[i-1], A[i]) \\
10              s \gets true \\
11          endif \\
12      endfor \\
13  until \neg s }
```

Alternative Keywords

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

```
\newcommand{\highlightkeyword}[2][\ ]{\ensuremath{\mathbf{#2}}#1}
\newcommand{\highlightaltkeyword}[1]{\ensuremath{\mathsf{#1}}}
```

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

Bubblesort(A : list of items)

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

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

Draft Mode

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

3.5 Line Numbering

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

$\underline{\text{IND-CPA}^{\mathcal{A}}_{\mathsf{Enc}}(1^n)}$

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

is generated by

```
\procedureblock[linenumbering]{$\indcpa_\enc^\adv(\secparam)$}{%
  b \sample \bin \\
 (\pk,\sk) \sample \kgen(\secparam)  \\
  \label{my:line:label}   (m_0,m_1) \sample \adv(\secparam, \pk, c)  \\
  c \sample \enc(\pk,m_b)  \\
  b' \sample \adv(\secparam, \pk, c)  \\
 \pcreturn 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 supressed. 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} \\
 Some code \pcskipln\\
 \pclinecomment{Some other comment } \\
 Some code }
```

3.5.2 Manually Inserting Line Numbers

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

$\text{IND-CPA}^{\mathcal{A}}_{\mathsf{Enc}}(1^n)$

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

is generated by

```
\procedure{$\indcpa_\enc^\adv(\secparam)$}{
 \pcln  b \sample \bin \\
 \pcln   (\pk,\sk) \sample \kgen(\secparam)  \\
 \pcln\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   \pcreturn 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`.

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

$\text{IND-CPA}^{\mathcal{A}}_{\mathsf{Enc}}(1^n)$

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

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`.

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

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

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

$\text{IND-CPA}^{\mathcal{A}}_{\mathsf{Enc}}(1^n)$

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

3.6 Subprocedures

The pseudocode package allows the typesetting of subprocedures such as

$\text{IND-CPA}^{\mathcal{A}}_{\mathsf{Enc}}(1^n)$

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

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

```
\procedureblock[linenumbering]{$\indcpa_\enc^\adv(\secparam)$}{%
   b \sample \bin  \\
   (\pk,\sk) \sample \kgen(\secparam)  \\
   (m_0,m_1) \sample \begin{subprocedure}%
 \dbox{\procedure{$\adv(\secparam, \pk, c)$}{%
    \text{Step 1}  \\
    \text{Step 2}  \\
  \pcreturn m_0, m_1 }}
  \end{subprocedure}  \\
   c \sample \enc(\pk,m_b)  \\
   b' \sample \adv(\secparam, \pk, c)  \\
   \pcreturn b = b' }
```

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] body \end{pchstack}
\begin{pcvstack}[options] body \end{pcvstack}
```

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

$\text{IND-CPA}^{\mathcal{A}}_{\mathsf{Enc}}(1^n)$

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

Oracle O

1 : Some code
2 : Some more code

```
\begin{pchstack}[boxed,center,space=1em]
  \procedure[linenumbering]{$\indcpa_\enc^\adv(\secparam)$}{%
    b \sample \bin  \\
    (\pk,\sk) \sample \kgen(\secparam)  \\
    (m_0,m_1) \sample \adv^O(\secparam, \pk)  \\
    c \sample \enc(\pk,m_b)  \\
    b' \sample \adv(\secparam, \pk, c)  \\
    \pcreturn b = b' }

  % alternatively use \pchspace for spacing

  \procedure[linenumbering,mode=text]{Oracle $O$}{%
      Some code  \\
      Some more code
  }
\end{pchstack}
```

Similarly you can stack two procedures vertically using the "pcvstack" environment. As a spacing between two vertically stacked procedures again use either the `space` option or insert space manually via `\pcvspace` which takes an optional length as a parameter.

$\text{IND-CPA}^{\mathcal{A}}_{\mathsf{Enc}}(1^n)$

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

Oracle O

1 : Some code
2 : Some more code

```
\begin{pcvstack}[boxed,center,space=0.5em]]
 \procedure[linenumbering]{$\indcpa_\enc^\adv(\secparam)$}{%
   b \sample \bin   \\
   (\pk,\sk) \sample \kgen(\secparam)   \\
   (m_0,m_1) \sample \adv^O(\secparam, \pk)   \\
   c \sample \enc(\pk,m_b)   \\
   b' \sample \adv(\secparam, \pk, c)   \\
   \pcreturn b = b' }

 % alternatively use \pcvspace for spacing

 \procedure[linenumbering,mode=text]{Oracle $O$}{%
     Some code   \\
     Some more code
 }
\end{pcvstack}
```

Horizontal and vertical stacking can be combined

$\underline{\text{IND-CPA}^{\mathcal{A}}_{\mathsf{Enc}}(1^n)}$

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

$\underline{\text{IND-CPA}^{\mathcal{A}}_{\mathsf{Enc}}(1^n)}$

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

Oracle O	Oracle H_1	Oracle H_2
1 : Some code	1 : Some code	1 : Some code
2 : Some more code	2 : Some more code	2 : Some more code

```
\begin{pcvstack}[boxed,center,space=1em]
  \begin{pchstack}[center,space=2em]

    \procedure[linenumbering]{$\indcpa_\enc^\adv(\secparam)$}{
      b \sample \bin   \\
      (\pk,\sk) \sample \kgen(\secparam)   \\
      (m_0,m_1) \sample \adv^O(\secparam, \pk)   \\
      c \sample \enc(\pk,m_b)   \\
      b' \sample \adv(\secparam, \pk, c)   \\
      \pcreturn b = b' }

    % alternatively use \pchspace for spacing

    \procedure[linenumbering]{$\indcpa_\enc^\adv(\secparam)$}{%
      b \sample \bin   \\
      (\pk,\sk) \sample \kgen(\secparam)   \\
      (m_0,m_1) \sample \adv^O(\secparam, \pk)   \\
      c \sample \enc(\pk,m_b)   \\
      b' \sample \adv(\secparam, \pk, c)   \\
      \pcreturn b = b' }

  \end{pchstack}

  % alternatively use \pcvspace for spacing

  \begin{pchstack}[space=0.25em]
    \procedure[linenumbering,mode=text]{Oracle $O$}{
      Some code   \\
      Some more code
    }

    \procedure[linenumbering,mode=text]{Oracle $H_1$}{
      Some code   \\
```

```
      Some more code
    }

    \procedure[linenumbering,mode=text]{Oracle $H_2$}{
      Some code  \\
      Some more code
    }
  \end{pchstack}
\end{pcvstack}
```

3.7.1 Stacking Options

The following keys are available on both `pchstack` and `pcvstack` environments

center Centers the stack.

boxed Draws a box around the stack.

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

noindent Does not indent the stack. Only applies if option `center` is not used.

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

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

belowskip By default the outer most stack adds vertical space below. The default space added is `\belowdisplayskip` and can be adapted by redefining `\pcbelowskip`. Note that the defualt 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 `\pcsetargs`. This is especially handy in stacking environments to add arguments to all enclosed code blocks.

Some Procedue A	Some Procedue B	Some Procedue C
1 : Step 1	1 : Step 1	1 : Step 1
2 : Step 2	2 : $\binom{A}{B+C}$	2 : Step 2
	3 : Step 3	

```
\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
```

```
  \pcsetargs{linenumbering,mode=text,minlineheight=1cm,codesize=\Large{}}

  \procedure{Some Procedue A}{
    Step 1\\
    Step 2 }

  \procedure{Some Procedue B}{
    \text{Step 1}\\
    \scriptsize$\begin{pcmbox}\begin{pmatrix}A \\ B + C \end{pmatrix}\end{pcmbox
    }$\\
    \text{Step 3}}

  \procedure{Some Procedue C}{
    Step 1\\
     Step 2 }
\end{pchstack}
```

Default Arguments for Stacking

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

3.9 Divisions and Linebreaks

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

```
\\[height]
```

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

```
\\[][\hline]
```

$\text{IND-CPA}^{\mathcal{A}}_{\mathsf{Enc}}(1^n)$

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

```
\procedureblock[linenumbering]{$\indcpa_\enc^\adv(\secparam)$}{%
    b \sample \bin   \\[2\baselineskip][\hline\hline]
    (\pk,\sk) \sample \kgen(\secparam)   \\
    (m_0,m_1) \sample \adv^O(\secparam, \pk)   \\
    c \sample \enc(\pk,m_b)   \\
    b' \sample \adv(\secparam, \pk, c)   \\
    \pcreturn 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

Procedure A	Procedure $B_{G_1}^{F^{h^*}}$
1 : do	1 : do
2 : some	2 : some
3 : work	3 : work

Here the sub and double superscripts in Procedure B make the header slightly larger than the maximum alotted 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 `\pcsetargs`.

Procedure A	Procedure $B_{G_1}^{F^{h^*}}$
1 : do	1 : do
2 : some	2 : some
3 : work	3 : work

```
\begin{pchstack}[center,space=1ex]
  \pcsetargs{headheight=5ex,headlinesep=-1ex}

  \procedure[linenumbering]{Procedure $A$}{
    \text{do}\\
    \text{some} \\
    \text{work}
  }

  \procedure[linenumbering]{Procedure $B^{F^{h^*}}_{G_1}$}{
    \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 `\pseudocode` command) mingles with a few low level commands such as `\\` or `\halign`. The effect of this is, that when you use certain math environments, for example, to create matrices, within pseudocode the result may be unexpected. Consider the following example

```
\pseudocodeblock{
\text{compute } P = \begin{pmatrix}
    A \\ B + C
\end{pmatrix}
}
```

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{
\text{compute } P = \begin{pcmbox}\begin{pmatrix}
    A \\ B + C
\end{pmatrix}\end{pcmbox}
}
```

$$\boxed{\text{compute } P = \begin{pmatrix} A \\ B + C \end{pmatrix}}$$

3.11 Fancy Code with Overlays

Consider the IND-CPA game. Here we have a single adversary $\mathcal{A}$ 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 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.

$\text{IND-CPA}^{\mathcal{A}}_{\mathsf{Enc}}(1^n)$

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

$\text{IND-CPA}^{\mathcal{A}}_{\mathsf{Enc}}(1^n)$

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

state

The image on the right is generated by:

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

 \pcdraw{
   \path[->] (start) edge[bend left=50] node[right]{$\state$} (start|-end);
 }
\end{pcimage}
```

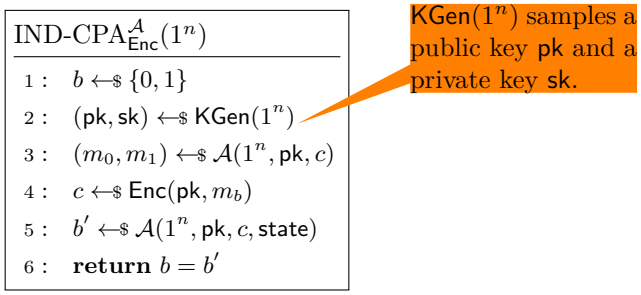
In order to achieve the above effect cryptocode utilizes TIKZ underneath. The `\pcnode` command generates TIKZ nodes and additionally we wrapped the pseudocode (or procedure) command in an `\begin{pcimage}\end{pcimage}` environment which allows us to utilize these nodes later, for example using the `\pcdraw` command. We can achieve a similar effect without an additional pcimage environment by using the optional argument of `\pcnode` for the TIKZ code.

```
\procedureblock[linenumbering]{$\indcpa_\enc^\adv(\secparam)$}{%
 b \sample \bin   \\
 (\pk,\sk) \sample \kgen (\secparam)   \\
 (m_0,m_1) \sample \adv(\secparam, \pk, c) \pcnode{start}   \\
```

```
5    c \sample \enc(\pk,m_b)  \\
6    b' \sample \adv(\secparam, \pk, c, \state) \pcnode{end}[draw={
7      \path[->] (start) edge[bend left=50] node[right]{$\state$} (start|-end);
8    }] \\
9    \pcreturn b = b' }
```

Example: Explain your Code

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




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

\pcdraw{
  \node[rectangle callout,callout absolute pointer=(kgen),fill=orange]
    at ([shift={(+3,+1)}]kgen) {
    \begin{varwidth}{3cm}
      $\kgen(\secparam)$ samples a public key $\pk$ and a private key $\sk$.
    \end{varwidth}
  };
}
\end{pcimage}
```

4 Tabbing Mode

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

First	**SecondThird**	**Fourth**
$b \leftarrow\$ \{0,1\}$	$b \leftarrow\$ \{0,1\}$	$b \leftarrow\$ \{0,1\}$ $b \leftarrow\$ \{0,1\}$

```
\pseudocodeblock{
  \textbf{First} \> \textbf{Second} \> \textbf{Third} \> \textbf{Fourth} \\
   b \sample \bin \> b \sample \bin \> b \sample \bin \> b \sample \bin}
```

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

First	**Second**	**Third**	**Fourth**
$b \leftarrow\$ \{0,1\}$	$b \leftarrow\$ \{0,1\}$	$b \leftarrow\$ \{0,1\}$	$b \leftarrow\$ \{0,1\}$

```
\pseudocodeblock{
  \textbf{First} \< \textbf{Second} \< \textbf{Third} \< \textbf{Fourth} \\
   b \sample \bin \< b \sample \bin \< b \sample \bin \< b \sample \bin}
```

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.

First	**Second**	**Third**	**Fourth**
$b \leftarrow\$ \{0,1\}$	$b \leftarrow\$ \{0,1\}$	$b \leftarrow\$ \{0,1\}$	$b \leftarrow\$ \{0,1\}$

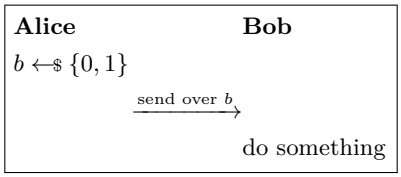
```
 \pseudocodeblock[colsep=1em,addtolength=10em]{%
  \textbf{First} \< \textbf{Second} \< \textbf{Third} \< \textbf{Fourth} \\
   b \sample \bin \< b \sample \bin \< b \sample \bin \< b \sample \bin}
```

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 `\>`. This mode is called *mintabmode* and is active by default.

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



is generated by

```
\pseudocodeblock{
 \textbf{Alice} \> \> \textbf{Bob}  \\
 b \sample \bin \> \> \\
 \> \xrightarrow{\text{send over } b} \>  \\
 \> \> \text{do something}}
```

4.1.1 Overriding The Tabbing Character

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

```
\renewcommand{\pctabname}{\myTab}

\pseudocode{
  \textbf{Alice} \myTab \myTab \textbf{Bob}  \\
  b \sample \bin \myTab \myTab \\
  \myTab \xrightarrow{\text{send over } b} \myTab  \\
  \myTab \myTab \text{do something}}
```

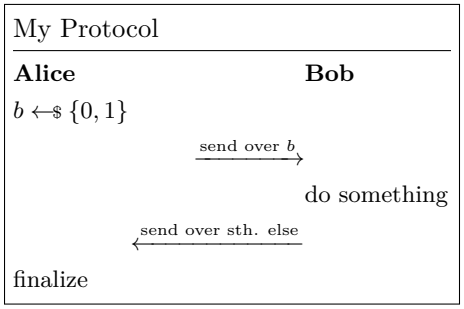
Similarly you can redefine the double tabbing character `\<` by overwriting `\pcdbltabname` (also see Section 5).

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 `\\[spacing]` to specifiy the spacing between two lines or of `[\\\hline]` to insert a horizontal rule. In order to achieve the same effect within the pseudocode command you can use `\\[spacing][\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 `\pseudocode` to also layout protocols such as

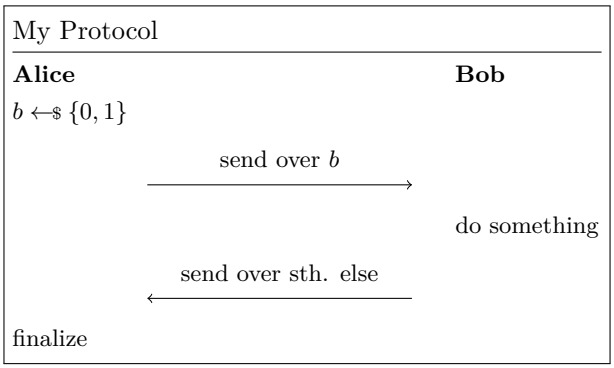


which is generated as

```
\procedureblock{My Protocol}{
  \textbf{Alice} \> \> \textbf{Bob}  \\
  b \sample \bin \> \> \\
  \> \xrightarrow{\text{send over } b} \>  \\
  \> \> \text{do something}    \\
  \> \xleftarrow{\text{send over sth. else}} \>   \\
  \text{finalize} \> \>}
```

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

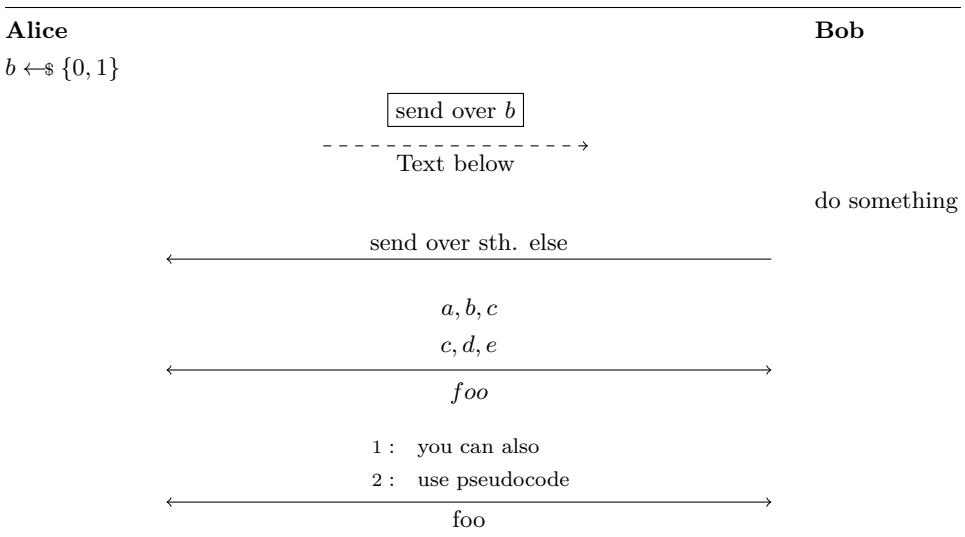
```
\sendmessageright*[3.5cm]{message}
\sendmessageleft*[3.5cm]{message}
```




```
\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} \> \> }
```

To obtain granular control over how messages are set use the `\sendmessage` and `\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 `aligned` environment. Following is an example, that showcases various message options.




```
\procedureblock{My Protocol}{
  \textbf{Alice} \> \> \textbf{Bob}  \\
  b \sample \bin \> \> \\
  \> \sendmessage{->}{centercol=3,top=send over $b$,bottom=Text below,topstyle={
    draw,solid,yshift=0.25cm},style={dashed}} \>  \\
  \> \> \text{do something}    \\
  \> \sendmessage{<-}{length=8cm,top=send over sth. else} \>   \\
  \> \sendmessage*{<->}{length=8cm,top={{a,b, c\\c,d, e}},bottom={foo}} \>   \\
  \> \sendmessage{<->}{length=8cm,top=\pseudocode[linenumbering]{\text{you can
    also}\\\text{use pseudocode}},bottom={foo}} \>   \\
  \text{finalize} \> \> }
```

`sendmessage` and `sendmessage*` support the following options:

top The content to display on top of the arrow.

bottom The content to display below the arrow.

left The content to display on the left of the arrow.

right The content to display on the right of the arrow.

topstyle The TIKZ style to be used for the top node.

bottomstyle The TIKZ style to be used for the bottom node.

rightstyle The TIKZ style to be used for the right node.

leftstyle The TIKZ style to be used for the left node.

length The length of the arrow.

style The style of the arrow.

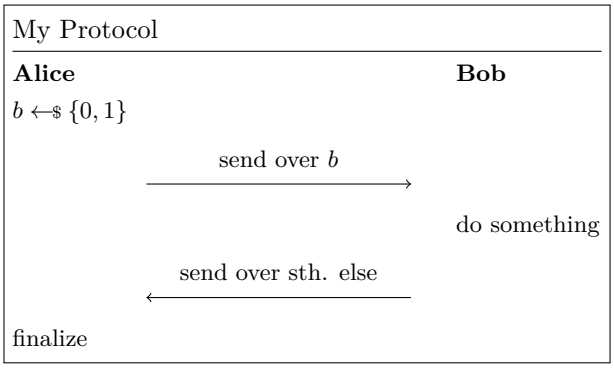
width The width of the column

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

5.1 Tabbing

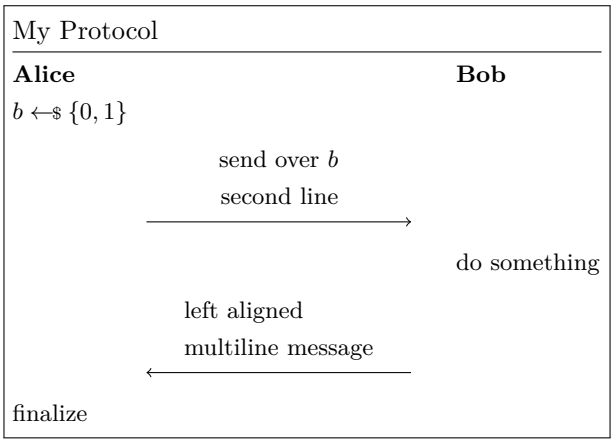
When typesetting protocols you might find that using two tabs instead of a single tab usually provides a better result as this ensures that all columns are left aligned. For this you can use `\<` instead of `\>` (see Section 4).

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



5.2 Multiline Messages

Using the starred send message commands you can easily generate multiline messages as the command wraps an *aligned* environment around the message.



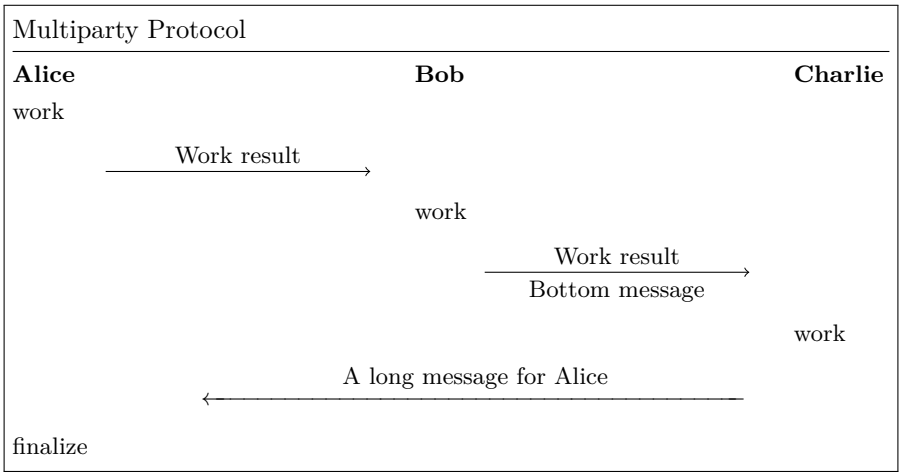

```
\procedureblock{My Protocol}{%
 \textbf{Alice} \< \< \textbf{Bob}  \\
 b \sample \bin \< \< \\
 \< \sendmessageright*{\text{send over } b\\ \text{second line}} \<  \\
 \< \< \text{do something}    \\
 \< \sendmessage*{<-}{top={\>\text{left aligned}\\ \> \text{multiline message
   }}} \<   \\
 \text{finalize} \< \<}
```

Remark. When using `\sendmessage*` the tabbing character `&` cannot be used. Instead use the `\>` command as defined within `\pseudocode`.

5.2.1 Multiplayer Protocols

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

```
\sendmessagerightx[width]{columnspan}{Text}
\sendmessageleftx[width]{columnspan}{Text}
```




```
\procedureblock{Multiparty Protocol}{%
 \textbf{Alice} \< \< \textbf{Bob} \< \< \textbf{Charlie} \\
 \text{work} \< \< \< \< \\
 \< \sendmessageright{top=Work result} \< \< \< \\
 \< \< \text{work} \< \< \\
 \< \< \<  \sendmessageright{top=Work result,bottom=Bottom message} \< \\
 \< \<  \< \< \text{work} \\
 \< \sendmessageleftx[7cm]{8}{\text{A long message for Alice}} \< \\
 \text{finalize} \< \< \< \< }
```

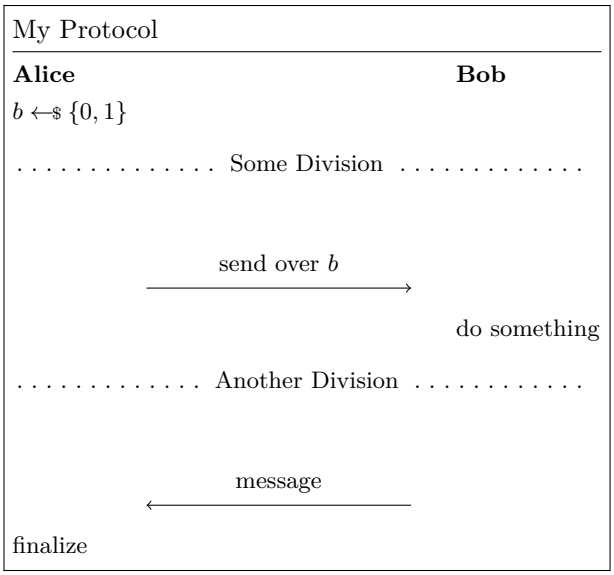
Note that for the last message from Charlie to Alice we needed to specify the number of passed over colums (`\sendmessageleftx[7cm]{8}{message}`). As we were passing 4 `\<` where each creates 2 columns, the total was 8 columns.

5.2.2 Divisions

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

```
\pcintertext[dotted|center]{Division Text}
```

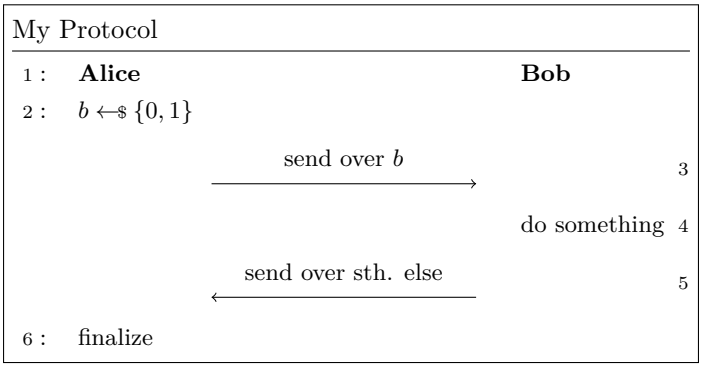
Note that in order to use the `\pcintertext` you need to use `\pclb` as the line break for the line before. Also see Section 4.




```
\procedureblock{My Protocol}{%
  \textbf{Alice} \< \< \textbf{Bob}  \\
  b \sample \bin \< \<  \pclb
  \pcintertext[dotted]{Some Division} \\
  \< \sendmessageright*{\text{send over } b} \<  \\
  \< \< \text{do something} \pclb
  \pcintertext[dotted]{Another Division} \\
  \< \sendmessageleft*{\text{message}} \<   \\
  \text{finalize} \< \< }
```

5.3 Line Numbering in Protocols

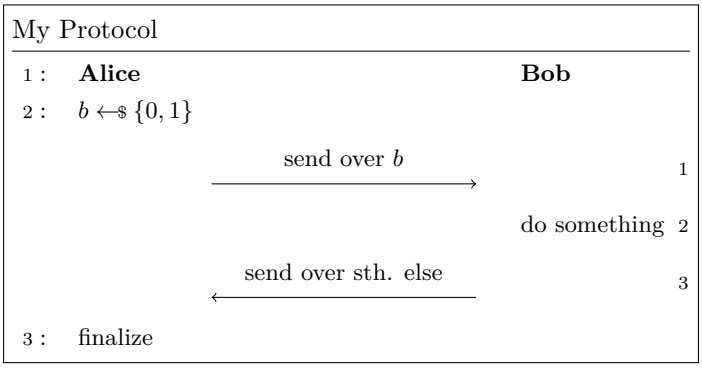
Protocols can be numbered similarly to plain pseudocode. Additionally to the `\pcln` there are the commands `\pclnr` and `\pcrln`. The first allows you to right align line numbers but uses the same counter as `\pcln`. The second uses a different counter.



Which is generated as

```
\procedureblock{My Protocol}{
  \pcln  \textbf{Alice} \< \< \textbf{Bob} \< \\
  \pcln b \sample \bin \< \< \< \\
  \< \sendmessageright*{\text{send over } b} \< \< \pclnr\\
  \< \< \text{do something} \<  \pclnr \\
  \< \sendmessageleft*{\text{send over sth. else}} \<\<  \pclnr \\
  \pcln \text{finalize} \< \< \<}
```

And using `\pcrln` we obtain:



This is generated as

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

5.3.1 Separators

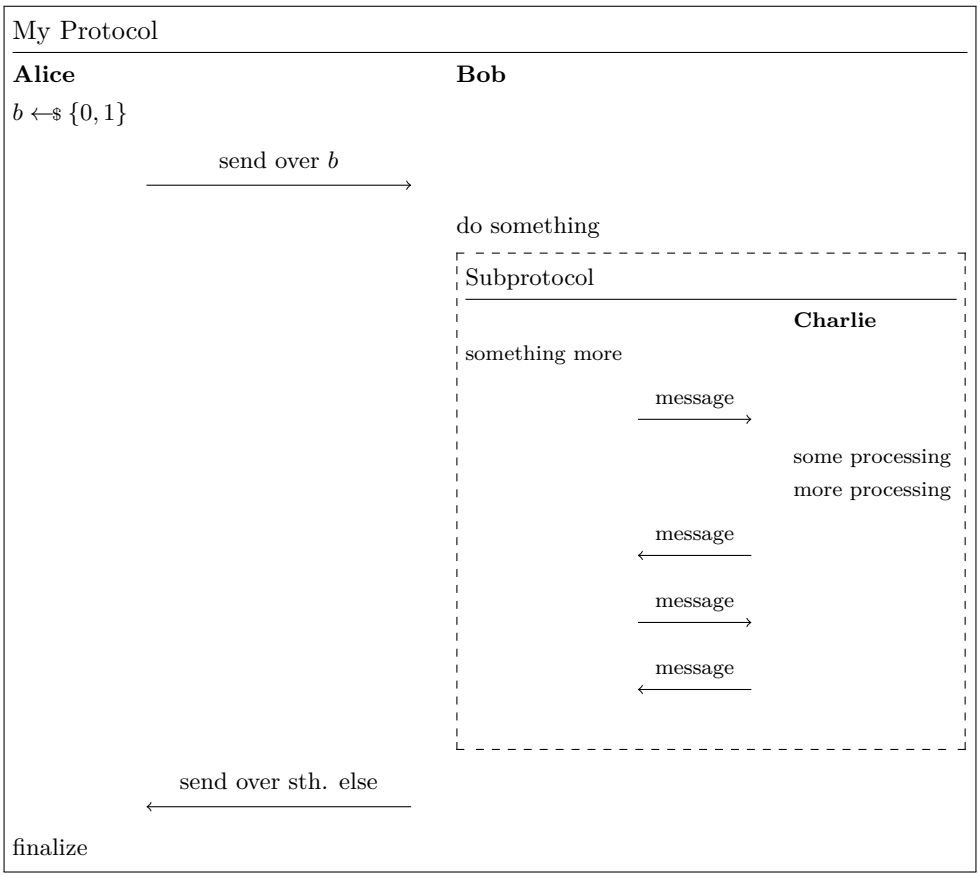
The commands `\pclnseparator` and `\pcrlnseparator` define the separators between code and line number. By default the left separator is set to (:) colon and the right separator is set to an empty string.

5.3.2 Spacing

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

5.4 Sub Protocols

Use the `subprocedure` environemnt to also create sub protocols.

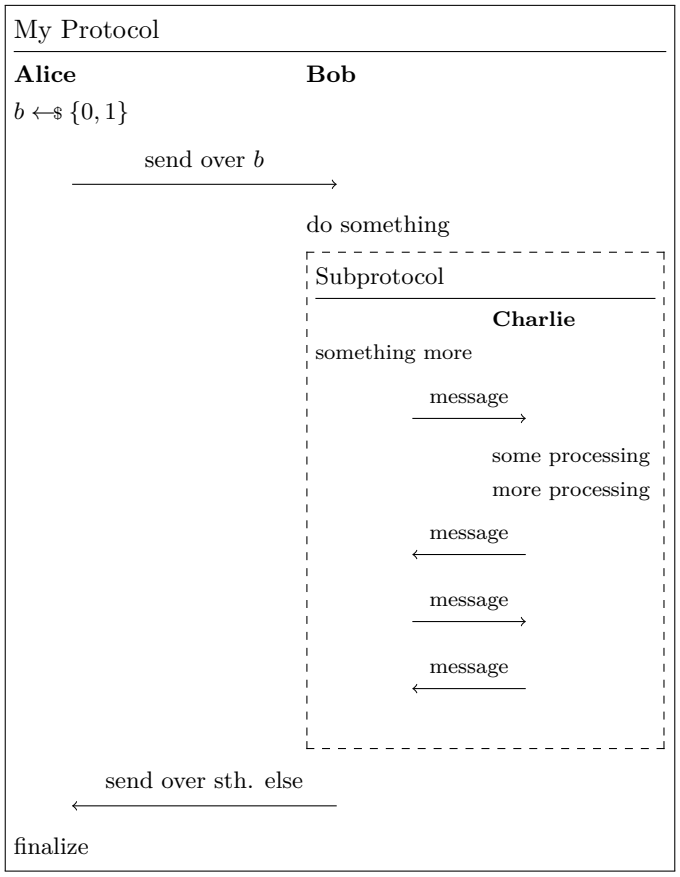



```
\procedureblock{My Protocol}{
  \textbf{Alice} \< \< \textbf{Bob}  \\
  b \sample \bin \< \< \\
  \< \sendmessageright*{\text{send over } b} \<  \\
  \< \< \text{do something}    \\
  \<\< \dbox{\begin{subprocedure}\procedure{Subprotocol}{
  \< \< \textbf{Charlie}  \\
  \text{something more} \< \<  \\
  \< \sendmessageright*[1.5cm]{\text{message}} \<  \\
  \< \< \text{some processing}  \\
  \< \< \text{more processing}  \\
  \< \sendmessageleft*[1.5cm]{\text{message}} \<  \\
  \< \sendmessageright*[1.5cm]{\text{message}} \<  \\
  \< \sendmessageleft*[1.5cm]{\text{message}} \<  \\
  }\end{subprocedure}}  \\
  \< \sendmessageleft*{\text{send over sth. else}} \<   \\
  \text{finalize} \< \< }
```

5.5 Compact Presentation of Protocols

In order to present protocols more compactly you can use the `colspace` option which adds space inbetween two columns. When set to a negative space, this has the effect that columns overlap. The following example is once more our above example using a

sub protocol but this time with `colspace=-1cm`. Note that the sub protocol inherits the option which is why both the outer and the inner protocol now have overlapping columns.




```
\procedureblock[colspace=-1cm]{My Protocol}{
  \textbf{Alice} \< \< \textbf{Bob}  \\
  b \sample \bin \< \< \\
  \< \sendmessageright*{\text{send over } b} \<  \\
  \< \< \text{do something}    \\
  \<\< \dbox{\begin{subprocedure}\procedure{Subprotocol}{
  \< \< \textbf{Charlie}  \\
  \text{something more} \< \<  \\
  \< \sendmessageright*[1.5cm]{\text{message}} \<  \\
  \< \< \text{some processing}  \\
  \< \< \text{more processing}  \\
  \< \sendmessageleft*[1.5cm]{\text{message}} \<  \\
  \< \sendmessageright*[1.5cm]{\text{message}} \<  \\
  \< \sendmessageleft*[1.5cm]{\text{message}} \<  \\
  }\end{subprocedure}}  \\
  \< \sendmessageleft*{\text{send over sth. else}} \<  \\
  \text{finalize} \< \< }
```

6 Game-Based Proofs

6.1 Basics

Besides displaying pseudocode the package also comes with commands to help presetn game-based proofs. The `gameproof` environment wraps the pseudocode block of a game-based proof.

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

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

$\mathsf{Game}_1(n)$	$\mathsf{Game}_2(n)$
1 : Step 1	Step 1
2 : Step 2	Step 2

by using

```
\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 `\pcgame` command which without argument prints Game and with (optional) argument `\pcgame[n]` prints Game_n.

6.1.1 Highlight Changes

In order to highlight changes from one game to the next use `\gamechange`.

$\mathsf{Game}_1(n)$	$\mathsf{Game}_2(n)$
1 : Step 1	Step 1
2 : Step 2	Step 2

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

The background color can be controlled by redefining `\gamechangecolor` which by default is defined as

```
\definecolor{gamechangecolor}{gray}{0.90}
```

Remark. Note that `\gamechange` is always in text mode.

6.1.2 Boxed Games

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

$\mathsf{Game}_1(n)$	$\mathsf{Game}_2(n)$ $\boxed{\mathsf{Game}_3(n)}$	$\mathsf{Game}_4(n)$
1 : Step 1	Step 1; $\boxed{\text{Alternative step 1}}$	Step 1
2 : Step 2	Step 2 is different	Step 2

```
\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 PRG. To give a hint within the pseudocode that the difference between two games is down to "something" you can use the `\addgamehop` command.

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

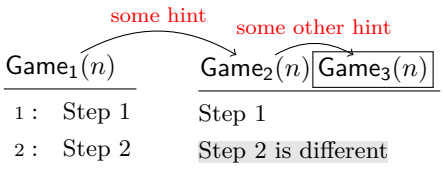
Here options allows you to specify the hint as well as the style. The following options are available

hint The hint text

nodestyle A TIKZ style to be used for the node.

pathstyle A TIKZ style to be used for the path.

edgestyle A TIKZ style to be used for the edge. This defaults to "bend left".



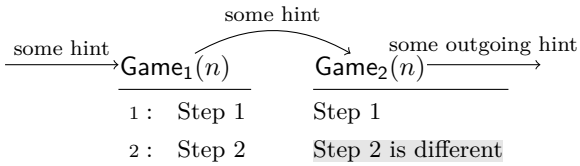

```
\begin{gameproof}
\begin{pchstack}[center,space=2em]
  \gameprocedure[linenumbering]{
    \text{Step 1}  \\
    \text{Step 2}
  }
  \tbxgameprocedure{
    \text{Step 1}  \\
    \gamechange{\text{Step 2 is different}}
  }
\end{pchstack}

\addgamehop{1}{2}{hint=\footnotesize some hint,nodestyle=red}
\addgamehop{2}{3}{hint=\footnotesize some other hint}
\end{gameproof}
```

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

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

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

```
\addstartgamehop[first game]{options}
\addendgamehop[last game]{options}
```




```
\begin{gameproof}
\begin{pchstack}[center,space=2em]
  \gameprocedure[linenumbering]{
    \text{Step 1}  \\
    \text{Step 2}
    }
  \gameprocedure{
    \text{Step 1}  \\
    \gamechange{\text{Step 2 is different}}
  }
\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 `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.

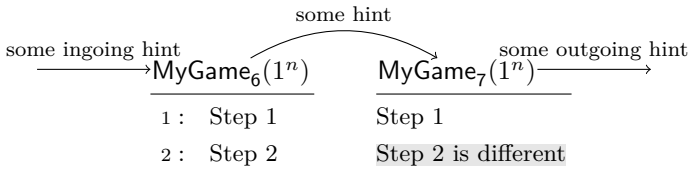
```
\begin{gameproof}[options]
```

The following parameters are available which, as usual, are provided in a key=value based form.

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

name The name for the game

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} }
  \gameprocedure{
    \text{Step 1}  \\
    \gamechange{\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}
```

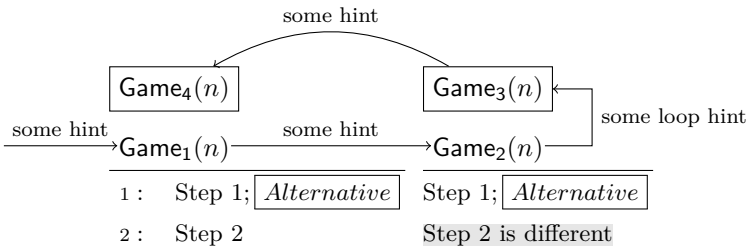
6.1.5 Default Name and Argument

The default name and argument are controlled via the commands `\pcgamename` and `\gameprocedurearg`.

Command	Default
`\pcgamename`	`\mathsf{Game}`
`\gameprocedurearg`	`\secpar`

6.1.6 Bi-Directional Games

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



```
\begin{gameproof}
\bxgameprocedure{4}{%
 \pcln  \text{Step 1}; \pcbox{Alternative} \\
 \pcln  \text{Step 2}
 }
\bxgameprocedure{3}{%
   \text{Step 1}; \pcbox{Alternative}  \\
   \gamechange{\text{Step 2 is different}}
 }
 \addstartgamehop{hint=\footnotesize some hint,edgestyle=}
\addgamehop{1}{2}{hint=\footnotesize some hint,edgestyle=}
 \addloopgamehop{hint=\footnotesize some loop hint}
 \addgamehop{2}{1}{hint=\footnotesize some hint}
\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 `\gameprocedure`. For this you can define the default arguments by calling \setgameproceduredefaultstyle to for example:

```
\setgameproceduredefaultstyle{beginline=\vphantom{\bin^\prg_\prg}
```

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

reduction target

$\mathsf{MyGame}_3(n)$**:** 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.

$\mathsf{MyGame}_4(n)$**:** This is the fourt game. The arrow at the side indicates the reduction target.

The above example is generated as

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

\describegame[inhint=reduction target]
This is the second game. The arrow at the side indicates the reduction target.
\end{gamedescription}
```

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

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.

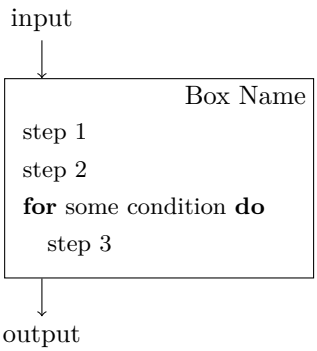
7 Black-Box Reductions

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 `bbrenv` environment (where bbr is short for black-box reduction) which takes a single `bbrbox` environment plus some additional commands.

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



This box is generated as

```
\begin{bbrenv}[aboveskip=1cm,belowskip=1cm]{A}
 \begin{bbrbox}[name=Box Name]
 \pseudocode{
   \text{step 1} \\
   \text{step 2} \\
   \pcfor \text{some condition} \pcdo \\
   \t\text{step 3}
 }
 \end{bbrbox}
 \bbrinput{input}
 \bbroutput{output}
\end{bbrenv}
```

The commands `bbrinput` and `bbroutput` allow to specify input and output for the latest `bbrenv` environment. The optional parameter for the `bbrenv` environment allows to specify leading and trailing space (this may become necessary when using inputs and outputs). For this provide `aboveskip` and `belowskip` keys. (Note that in an earlier version of cryptocode you could write `\begin{bbrenv}[1cm]{A}[1cm]`. While this format is still supported it should be regarded deprecated.)

The `bbrenv` environment takes the following options as optional first parameter:

aboveskip Space above.

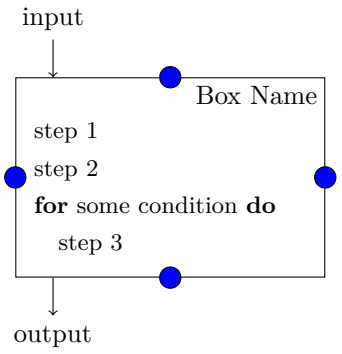
belowskip Space below.

tikzargs Underneath `bbrenv` is a `tikzpicture` and via `tikzargs` you can pass in arguments.

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 (https://www.ctan.org/pkg/pgf).

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

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.




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

 \filldraw[fill=blue] (A.north) circle (4pt);
 \filldraw[fill=blue] (A.west) circle (4pt);
 \filldraw[fill=blue] (A.east) circle (4pt);
 \filldraw[fill=blue] (A.south) circle (4pt);
\end{bbrenv}
```

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

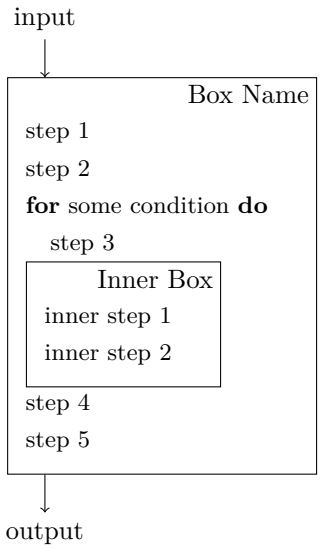
```
name=Box Name
```

to specify the label. The following options are available

Option	Description
name	Specifies the box's label
namepos	Specifies the position (left, center, right, top left, top center, top right, middle)
namestyle	Specifies the style of the name
abovesep	Space above box (defaults to `\baselineskip`)
minheight	The minimal height
addheight	Additional height at the end of the box
xshift	Allows horizontal positioning
yshift	Allows horizontal positioning
style	allows to customize the node

7.1 Nesting of Boxes

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




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

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

 \pseudocode{
   \text{step 4} \\
   \text{step 5}
 }
 \end{bbrbox}
 \bbrinput{input}
 \bbroutput{output}
\end{bbrenv}
```

7.2 Messages and Queries

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

```
\bbrmsgto{options}
\bbrmsgfrom{options}
\bbrmsgtofrom{options}{options}
\bbrmsgfromto{options}{options}
\bbrqryto{options}
\bbrqryfrom{options}
\bbrqrytofrom{options}{options}
\bbrqryfromto{options}{options}
```

By convention messages are on the left of boxes and queries on the right. Commands ending on `to` make an arrow to the right while commands ending on `from` make an arrow to the left. The `options` define how the message is drawn and consists of a key-value 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 (`top=Label`) while the second message should be drawn on the bottom (`bottom=Label`).

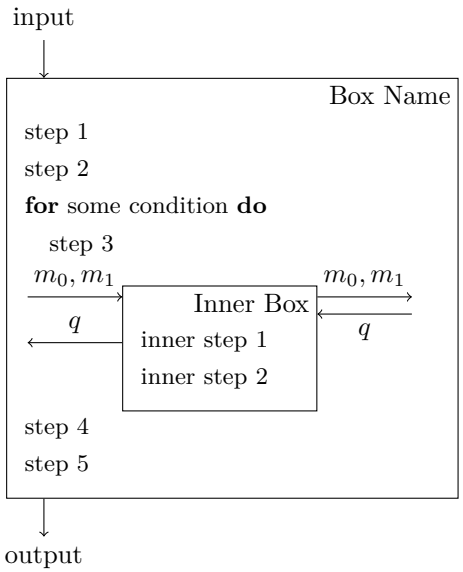
For example, to draw a message with a label on top and on the side use

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

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

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

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




```
\begin{bbrenv}{A}
 \begin{bbrbox}[name=Box Name]
 \pseudocode{
```

```
    \text{step 1} \\
    \text{step 2} \\
    \pcfor \text{some condition} \pcdo \\
    \pcind\text{step 3}
  }

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

    \bbrmsgto{top={$m_0,m_1$}}
    \bbrmsgfrom{top=$q$}

    \bbrqrytofrom{top={$m_0,m_1$}}{bottom=$q$}

  \end{bbrenv}

  \pseudocode{
    \text{step 4} \\
    \text{step 5}
  }
  \end{bbrbox}
  \bbrinput{input}
  \bbroutput{output}
\end{bbrenv}
```

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

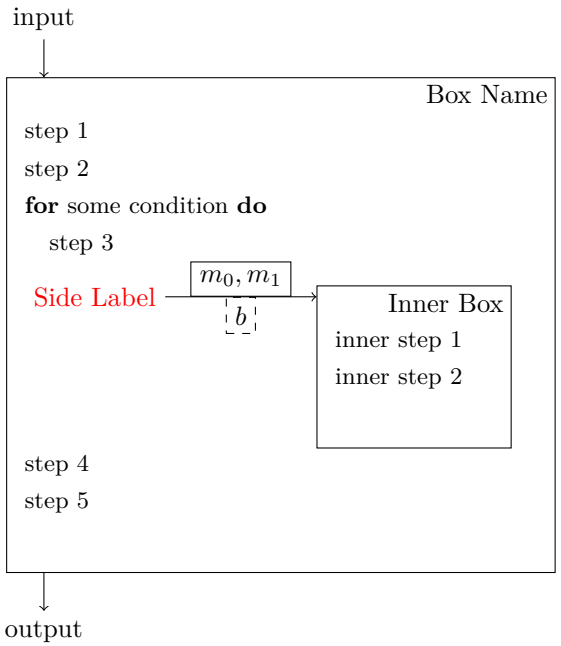
aboveskip Space before message

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{step 2} \\
   \pcfor \text{some condition} \pcdo \\
   \pcind\text{step 3}
 }

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

   \bbrmsgto{top={$m_0,m_1$},side=Side Label, bottom=$b$, length=2cm,
         topstyle={draw, solid}, sidestyle={red}, bottomstyle={draw, dashed}}

 \end{bbrenv}

 \pseudocode{
   \text{step 4} \\
   \text{step 5} \\
 }
 \end{bbrbox}
 \bbrinput{input}
```

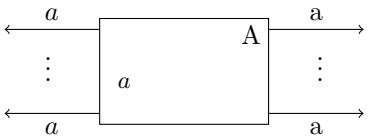
```
  \bbroutput{output}
\end{bbrenv}
```

First Message

The first message is offset by `\bbrfirstmessageoffset` which defaults to 1ex.

7.2.2 Vdots

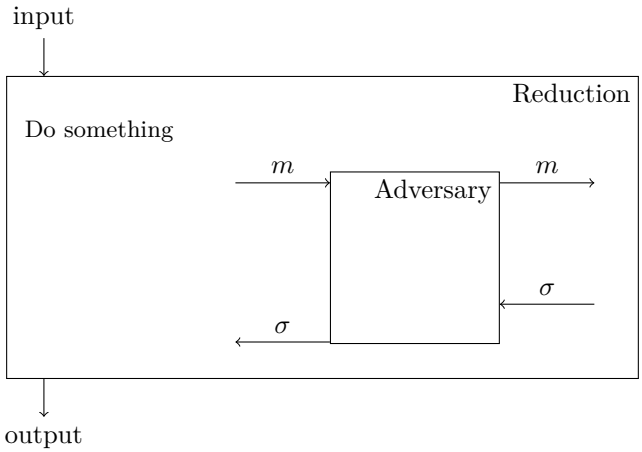
You can use `\bbrmsgvdots` and `\bbrqryvdots` to add `\vdots` in between messages or queries.




```
\begin{bbrenv}{A}
  \begin{bbrbox}[name=A, minheight=14mm]
    \pseudocode{a}
  \end{bbrbox}
  \bbrmsgfrom{top={$a$}}
  \bbrmsgvdots
  \bbrmsgfrom{islast=true, bottom={$a$}}
  \bbrqryto{top=a}
  \bbrqryvdots%
  \bbrqryto{bottom=a, islast=true}
\end{bbrenv}
```

7.2.3 Add Space

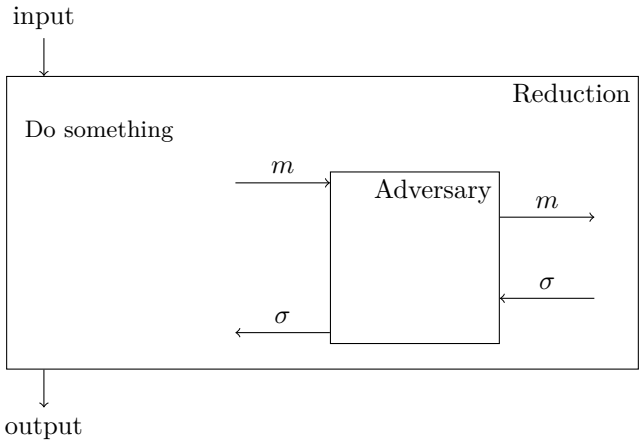
If the spacing between messages is not sufficient you can use the `bbrmsgspace` and `bbrqryspace` commands to add additional space. Alternatively, you can use the options `aboveskip` and `belowskip` on the individual message or query commands.




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

```

```
  \begin{bbrenv}{B}

    \begin{bbrbox}[name=Adversary, minheight=15ex, xshift=4cm]

    \end{bbrbox}

    \bbrmsgto{top=$m$}
    \bbrmsgspace{1.5cm}
    \bbrmsgfrom{top=$\sigma$}

    \bbrqryto{top=$m$}
    \bbrqryspace{1cm}
    \bbrqryfrom{top=$\sigma$}

  \end{bbrenv}

  \end{bbrbox}
  \bbrinput{input}
  \bbroutput{output}
\end{bbrenv}
```

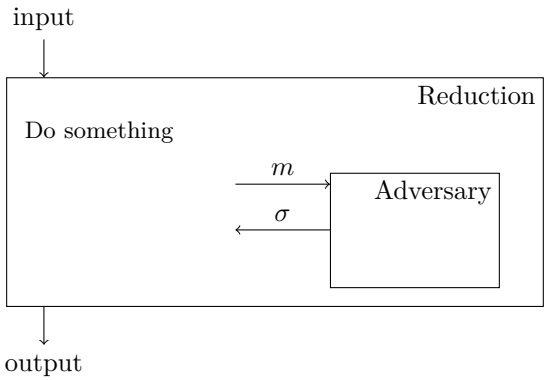
Note that for placing a message at the bottom, `islast` or fixed offsets often allow obtain more accurate results.




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

    \begin{bbrenv}{B}

      \begin{bbrbox}[name=Adversary, minheight=15ex, xshift=4cm]

      \end{bbrbox}

      \bbrmsgto{top=$m$}
      \bbrmsgfrom{top=$\sigma$, islast}

      \bbrqryto{top=$m$, fixedoffset=4ex}
      \bbrqryfrom{top=$\sigma$, fixedboffset=4ex}

    \end{bbrenv}

    \end{bbrbox}
    \bbrinput{input}
    \bbroutput{output}
  \end{bbrenv}
```

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

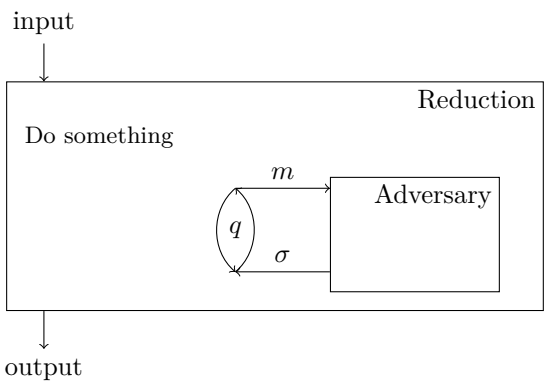



```
\begin{bbrenv}{A}
 \begin{bbrbox}[name=Reduction]
 \pseudocode{
   \text{Do something}
 }

 \begin{bbrenv}{B}

   \begin{bbrbox}[name=Adversary, minheight=10ex, xshift=4cm]

   \end{bbrbox}

   \bbrmsgto{top=$m$}
   \bbrmsgfrom{top=$\sigma$}

 \end{bbrenv}

 \end{bbrbox}
 \bbrinput{input}
 \bbroutput{output}
\end{bbrenv}
```

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

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

The `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}
 }

 \begin{bbrenv}{B}

   \begin{bbrbox}[name=Adversary,minheight=10ex,xshift=4cm]

   \end{bbrbox}

   \bbrmsgto{top=$m$,sidename=BeginLoop}
   \bbrmsgspace{0.5cm}
   \bbrmsgfrom{top=$\sigma$,sidename=EndLoop}
   \bbrloop{BeginLoop}{EndLoop}{center=$q$}

 \end{bbrenv}

 \end{bbrbox}
 \bbrinput{input}
 \bbroutput{output}
\end{bbrenv}
```

The `\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

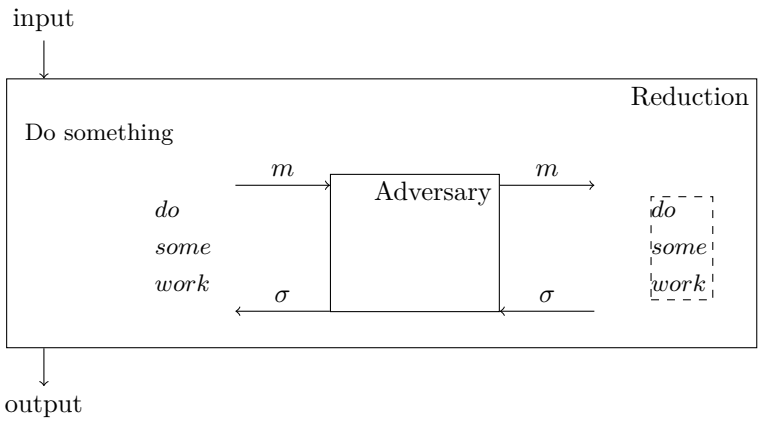
clockwise Loop going in clockwise direction

angle Angle of the arrows

7.2.5 Intertext

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

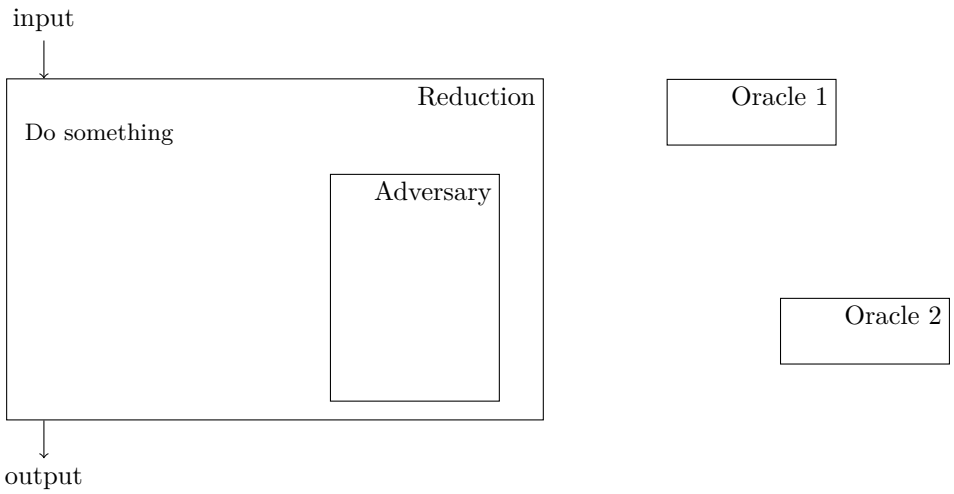
```
\bbrmsgtxt[options]{Text}
\bbrqrytxt[options]{Text}
```




```
1  \begin{bbrenv}{A}
2   \begin{bbrbox}[name=Reduction]
3   \pseudocode{
4     \text{Do something}
5   }
6
7   \begin{bbrenv}{B}
8
9     \begin{bbrbox}[name=Adversary, minheight=12ex, xshift=4cm]
10
11    \end{bbrbox}
12
13    \bbrmsgto{top=$m$}
14    \bbrmsgtxt{\pseudocode{
15      do \\
16      some \\
17      work
18    }}
19    \bbrmsgfrom{top=$\sigma$}
20
21    \bbrqryto{top=$m$}
22    \bbrqrytxt[nodestyle={draw,dashed},xshift=2cm]{\pseudocode{
23      do \\
24      some \\
25      work
26    }}
27    \bbrqryfrom{top=$\sigma$}
28
29  \end{bbrenv}
30
31  \end{bbrbox}
32  \bbrinput{input}
33  \bbroutput{output}
34 \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 `bbrenv` environment using the `bbroracle` environment. Oracles go behind the single `bbrbox` environment within an `bbrenv` enviornment.




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

 \begin{bbrenv}{B}
   \begin{bbrbox}[name=Adversary, minheight=3cm, xshift=4cm]
   \end{bbrbox}

 \end{bbrenv}

 \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}
```

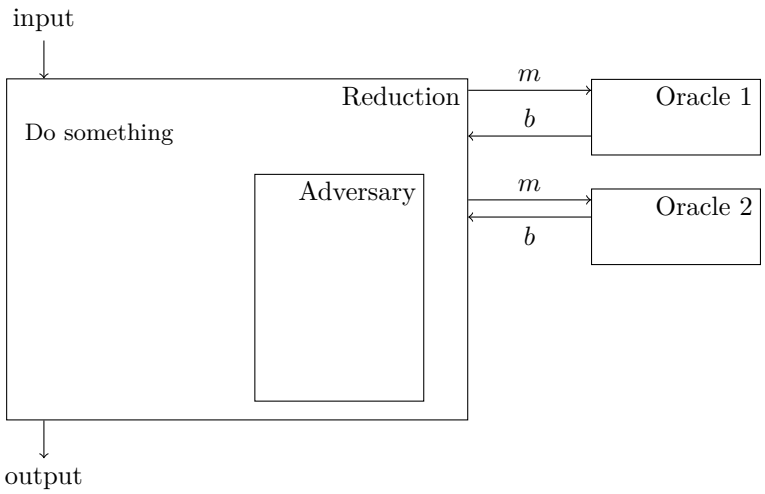
Via the option “hdistance=length” and “vdistance=length” you can control the horizontal and vertical position of the oracle. By default this value is set to 1.5cm and `\baselineskip`.

7.3.1 Communicating with Oracles

As oracles use the *bbrbox* environment we can directly use the established ways to send messages and queries to oracles. In addition you can use the `\bbroracleqryfrom` and `\bbroracleqryto`.

```
\bbroracleqryfrom{options}
\bbroracleqryto{options}
\bbroracleqrytofrom{options}{options}
\bbroracleqryfromto{options}{options}
```

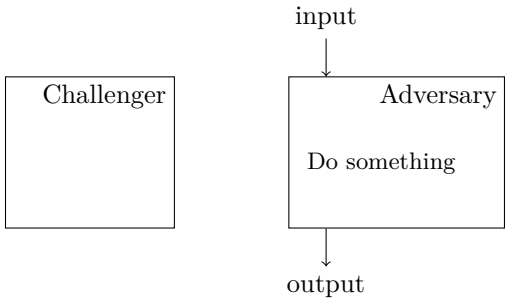
Here options allow you to specify where the label goes (top, bottom). In addition you can use `\bbroracleqryspace` to generate extra space between oracle messages. Note that oracle messages need to be added after the closing `\end{bbroracle}` command.




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

 \begin{bbrenv}{B}
   \begin{bbrbox}[name=Adversary,minheight=3cm,xshift=3cm]
   \end{bbrbox}

 \end{bbrenv}

 \end{bbrbox}
 \bbrinput{input}
 \bbroutput{output}

 \begin{bbroracle}{OraA}
   \begin{bbrbox}[name=Oracle 1,minheight=1cm]
   \end{bbrbox}
 \end{bbroracle}
 \bbroracleqrytotop=$m$}
 \bbroracleqryfrom{top=$b$}

 \begin{bbroracle}{OraB}
   \begin{bbrbox}[name=Oracle 2,minheight=1cm]
   \end{bbrbox}
 \end{bbroracle}
 \bbroracleqrytofrom{top=$m$}{bottom=$b$}
\end{bbrenv}
```

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 *bbrenv* environment using the *bbrchallenger* environment. Challengers go behind the single *bbrbox* environment within an *bbrenv* enviornment.




```
\begin{bbrenv}{A}
 \begin{bbrbox}[name=Adversary, minheight=2cm]
 \pseudocode{
   \text{Do something}
 }

 \end{bbrbox}
 \bbrinput{input}
 \bbroutput{output}

 \begin{bbrchallenger}{ChaA}
   \begin{bbrbox}[name=Challenger, minheight=2cm]

   \end{bbrbox}
 \end{bbrchallenger}
\end{bbrenv}
```

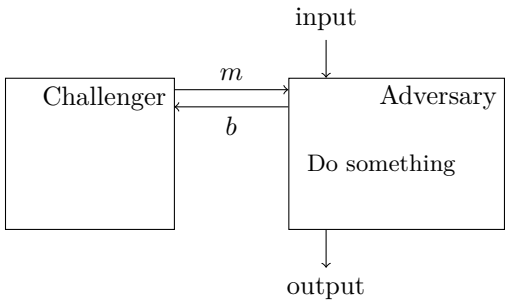
Via the option "hdistance=length" and "vdistance=length" you can control the horizontal and vertical position of the challenger. By default this value is set to 1.5cm and `\baselineskip`.

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`.

```
\bbrchallengerqryfrom{options}
\bbrchallengerqryto{options}
\bbrchallengerqrytofrom{options}{options}
\bbrchallengerqryfromto{options}{options}
```

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.

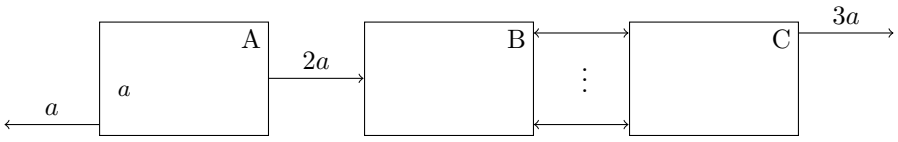



```
\begin{bbrenv}{A}
 \begin{bbrbox}[name=Adversary, minheight=2cm]
```

```
  \pseudocode{
    \text{Do something}
  }

  \end{bbrbox}
  \bbrinput{input}
  \bbroutput{output}

  \begin{bbrchallenger}{ChaA}
    \begin{bbrbox}[name=Challenger, minheight=2cm]

    \end{bbrbox}
  \end{bbrchallenger}

  \bbrchallengerqryfromto{top=$m$}{bottom=$b$}
\end{bbrenv}
```

7.5 Horizontal Stacking

`bbrenv` environments can be stacked horizontally.

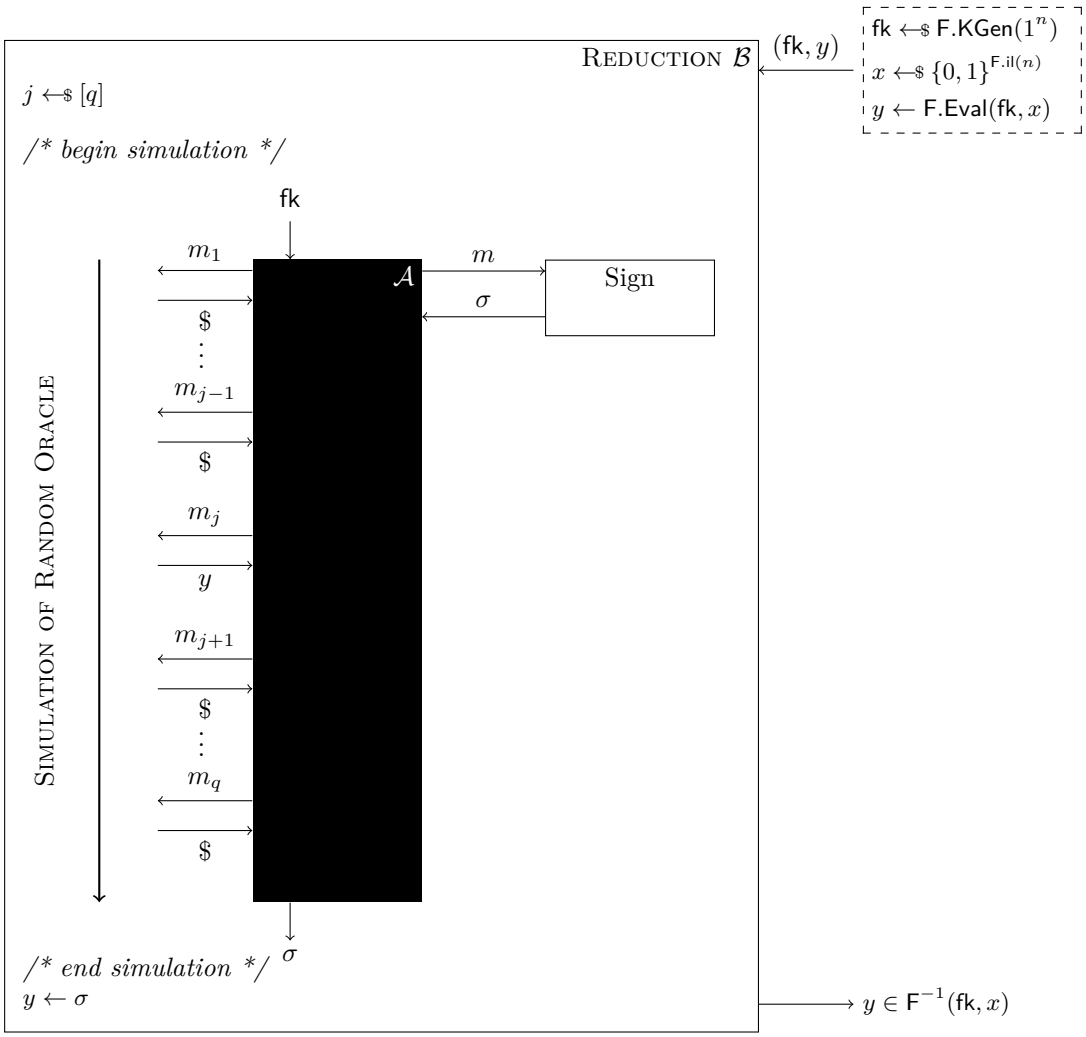


Note that in order to not have horizontal space inbetween the boxes, that you need to not leave any space. In the following code this is handled via comments.

```
\begin{figure}[h]
  \centering
  \begin{bbrenv}{A}
    \begin{bbrbox}[name=A, minheight=15mm]
    \pseudocode{a}
    \end{bbrbox}
    \bbrmsgfrom{islast=true, top={$a$}}
    \bbrqryto{aboveskip=6mm, top={$2a$}}
  \end{bbrenv}%
\begin{bbrenv}{B}
    \begin{bbrbox}[name=B, minheight=15mm]
    \end{bbrbox}
    \bbrqryto{edgestyle={<->}}
    \bbrqryvdots[aboveskip=1mm]
    \bbrqryto{islast=true, edgestyle={<->}}
  \end{bbrenv}%
\begin{bbrenv}{C}
    \begin{bbrbox}[name=C, minheight=15mm]
    \end{bbrbox}
    \bbrqryto{top={$3a$}}
  \end{bbrenv}%
\end{figure}
```

7.6 Examples

A reduction sketch for full domain hash.



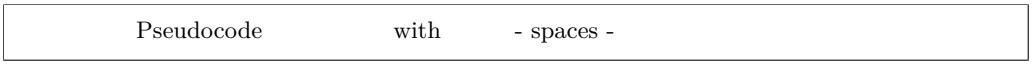

```
\begin{bbrenv}{Red}

  \begin{bbrbox}[name=\textsc{Reduction }$\bdv$]

    \pseudocode{
      j \sample [q]
    }

    \vspace{2ex}
    \emph{/* begin simulation */}

    \begin{bbrenv}[aboveskip=2em]{Adv}
      \begin{bbrbox}[name=$\adv$,minheight=8.5cm,style={fill=black},namestyle={color=white},xshift=3cm]
      \end{bbrbox}

      \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}
```

```
26
27        \bbrmsgfrom{top=$m_j$,afterskip=-0.5\baselineskip}
28        \bbrmsgto{bottom=$y$,afterskip=1.5\baselineskip}
29
30        \bbrmsgfrom{top=$m_{j+1}$,afterskip=-0.5\baselineskip}
31        \bbrmsgto{bottom=$\$$,afterskip=0.5\baselineskip}
32
33        \bbrmsgvdots
34
35        \bbrmsgfrom{top=$m_q$,beforeskip=0.5\baselineskip,afterskip=-0.5\
      baselineskip}
36        \bbrmsgto{bottom=$\$$}
37
38        \begin{bbroracle}{Sign}
39          \begin{bbrbox}[name=Sign,namepos=center,style={draw},minheight=1cm]
40          \end{bbrbox}
41        \end{bbroracle}
42
43        \bbroracleqryto{top=$m$}
44        \bbroracleqryfrom{top=$\sigma$}
45
46      \end{bbrenv}
47
48      \pcdraw{
49        \node[left=2cm of Adv.north west] (startsim) {};
50        \node[left=2cm of Adv.south west] (endsim) {};
51        \draw[->,thick] (startsim) -- (endsim);
52        \node[rotate=90, left=2.75cm of Adv.west,anchor=center] () {\textsc{
      Simulation of Random Oracle}};
53      }
54
55      \emph{/* end simulation */}
56
57      \pseudocode{
58        y \gets \sigma
59      }
60
61    \end{bbrbox}
62    \bbrqryfrom{beforeskip=0.25cm,top={$(\fk, y)$},side={\dbox{\pseudocode{
63      \fk \sample \fash.\kgen(\secparam) \\ x \sample \bin^{\fash.\il(\secpar)} \\
        y \gets \fash.\eval(\fk, x)}
64    }}}
65    \bbrqryto{beforeskip=11.75cm,side=\pseudocode{y \in \fash^{-1}(\fk, x)}}
66  \end{bbrenv}
```

8 Known Issues

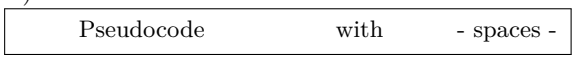
8.1 Pseudocode KeepSpacing within Commands

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




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

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




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

8.2 AMSFonts

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

8.3 Hyperref

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

8.4 Cleveref

In order to support the cleveref package (https://ctan.org/pkg/cleveref) load cleveref after cryptocode and subsequently (but still in the preamble) call `\pcfixcleveref`.

8.5 Babel - Spanish

The spanish version of the babel package uses < and > as shorthands which are used by cryptocode as tabbing characters. The easisest workaround is to tell cryptocode to use different tabbing characters, for example:

```
\renewcommand{\pctabname}{ctab}
\renewcommand{\pcdbltabname}{cdtab}
```

9 Implementation

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

```
1 ⟨*cryptocode.sty⟩
```

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.

```
2 \RequirePackage{amsmath}
3 \RequirePackage{mathtools}
```

9.1 Package Options

`\@pc@opt@amsfonts` Definitions of boolean flags used to determin whether or not to load amsfonts.

```
4 \newif\if@pc@opt@amsfonts
```

`\@pc@opt@advantage` Whether or not to define commands for the given option.

```
5 \newif\if@pc@opt@advantage
```

`\@pc@opt@centernot` Whether or not to load centernot

```
6 \newif\if@pc@opt@centernot
```

9.1.1 operators

`\sample` `\floor` `\tfloor` `\ceil` `\tceil` `\Angle` `\tAngle` `\abs` `\tabs` `\norm` `\tnorm` `\concat` `\emptystring` `\argmax` `\argmin` `\pindist` `\cindist` `\sindist`

Definitions of macros for the *operators* pacakge option.

```
7 \DeclareOption{operators}{
```

Robust sample operator that also works in subscripts. Is based on egreg's solution given in https://tex.stackexchange.com/questions/418740/how-to-write-left-arrow-with-a-dollar-sig

```
8 \providecommand\sample{\leftarrow\mathrel{\mkern-2.0mu}\pc@smalldollar}
9 \newcommand{\pc@smalldollar}{\mathrel{\mathpalette\pc@small@dollar\relax}}
10 \newcommand{\pc@small@dollar}[2]{%
11   \vcenter{\hbox{%
12     $#1\textnormal{\fontsize{0.7\dimexpr\f@size pt}{0}\selectfont\$\hskip-0.05em plus 0.5em}$%
13   }}%
14 }
15
16 \DeclarePairedDelimiter\pc@floor{\lfloor}{\rfloor}
17 \providecommand{\floor}[1]{\pc@floor*{#1}}
18 \providecommand{\tfloor}[1]{\pc@floor{#1}}
19
20 \DeclarePairedDelimiter\pc@ceil{\lceil}{\rceil}
21 \providecommand{\ceil}[1]{\pc@ceil*{#1}}
22 \providecommand{\tceil}[1]{\pc@ceil{#1}}
23
24 \DeclarePairedDelimiter\pc@Angle{\langle}{\rangle}
25 \providecommand{\Angle}[1]{\pc@Angle*{#1}}
26 \providecommand{\tAngle}[1]{\pc@Angle{#1}}
27
28 \DeclarePairedDelimiter\pc@abs{\lvert}{\rvert}
29 \providecommand{\abs}[1]{\pc@abs*{#1}}
30 \providecommand{\tabs}[1]{\pc@abs{#1}}
```

```
31
32 \DeclarePairedDelimiter\pc@norm{\lVert}{\rVert}
33 \providecommand{\norm}[1]{\pc@norm*{#1}}
34 \providecommand{\tnorm}[1]{\pc@tnorm{#1}}
35
36 \providecommand{\concat}{\ensuremath{\|}}
37 \providecommand{\emptystring}{\ensuremath{\varepsilon}}
38
39 \DeclareMathOperator*{\argmax}{arg\,max}
40 \DeclareMathOperator*{\argmin}{arg\,min}
41
42 %indistinguishability
43 \newcommand{\@pc@oset}[3][0ex]{%
44   \mathrel{\mathop{#3}\limits^{
45     \vbox to#1{\kern-2\ex@
46     \hbox{$\scriptstyle#2$}\vss}}}}
47
48 \newcommand{\pindist}{\@pc@oset{\text{p}}{\lower.2ex\hbox{$=$}}}
49 \newcommand{\sindist}{\@pc@oset{\text{s}}{\lower.1ex\hbox{$\approx$}}}
50 \newcommand{\cindist}{\@pc@oset{\text{c}}{\lower.1ex\hbox{$\approx$}}}
51 }
```

9.1.2 adversary

`\adversary` `\adv` `\bdv` `\cdv` `\ddv` `\edv` `\mdv` `\pdv` `\rdv` `\sdv`

Definitions of adversaries $\mathcal{A}$ (`\adv`), $\mathcal{B}$ (`\bdv`), etc. together with a style `\pcadvstyle`.

```
52 \DeclareOption{adversary}{
53 \providecommand{\adversary}[1]{\pcadvstyle{#1}}
54
55 \providecommand{\adv}{\pcadvstyle{A}}
56 \providecommand{\bdv}{\pcadvstyle{B}}
57 \providecommand{\cdv}{\pcadvstyle{C}}
58 \providecommand{\ddv}{\pcadvstyle{D}}
59 \providecommand{\edv}{\pcadvstyle{E}}
60 \providecommand{\mdv}{\pcadvstyle{M}}
61 \providecommand{\pdv}{\pcadvstyle{P}}
62 \providecommand{\rdv}{\pcadvstyle{R}}
63 \providecommand{\sdv}{\pcadvstyle{S}}
64 }
```

9.1.3 landau

`\bigO` `\smallO` `\bigOmega` `\smallOmega` `\bigsmallO` `\bigTheta` `\orderOf`

Defines several *Landau symbols*.

```
65 \DeclareOption{landau}{
66 \providecommand{\bigO}[1]{\ensuremath{\mathcal{O}\pc@olrk*{#1}}}
67 \providecommand{\smallO}[1]{\ensuremath{\text{o}\pc@olrk*{#1}}}
68 \providecommand{\bigOmega}[1]{\ensuremath{\Omega\pc@olrk*{#1}}}
69 \providecommand{\smallOmega}[1]{\ensuremath{\omega\pc@olrk*{#1}}}
70 \providecommand{\bigsmallO}[1]{%
71 \PackageWarning{cryptocode}{bigsmallO is deprecated. Use bigTheta instead.}%
72 \ensuremath{\Theta\pc@olrk*{#1}}}
73 \providecommand{\bigTheta}[1]{\ensuremath{\Theta\pc@olrk*{#1}}}
74 \providecommand{\orderOf}{\ensuremath{\sim}}
75 }
```

9.1.4 probability

\probname
\expectationname
\supportname
\tprob
\prob
\tprobsub
\probsub
\probsublong
\tcondprob
\condprob
\tcondprobsub
\condprobsub
\texpect
\expect
\texpsub
\expsub
\tcondexp
\condexp
\tcondexpsub
\condexpsub
\supp
\entropy
\condentropy
\minentropy
\condminentropy
\condavgminentropy

The *probability* package option defines various macros for typesetting probabilities.

Sets flags `\@pc@opt@amsfontstrue`.

```
\DeclareOption{probability}{
\@pc@opt@amsfontstrue

\providecommand{\probname}{Pr}
\providecommand{\expectationname}{\ensuremath{\mathbb{E}}}
\providecommand{\supportname}{Supp}

\providecommand{\tprob}[1]{\ensuremath{\operatorname{\probname}\pc@elrk{#1}}}
\providecommand{\prob}[1]{\ensuremath{\operatorname{\probname}\pc@elrk*{#1}}}

\providecommand{\tprobsub}[2]{\ensuremath{\operatorname{\probname}_{#1}\pc@elrk{#2}}}
\providecommand{\probsub}[2]{\ensuremath{\operatorname{\probname}_{#1}\pc@elrk*{#2}}}
\providecommand{\probsublong}[2]{\ensuremath{\prob{#2\,:\,#1}}}

\providecommand{\tcondprob}[2]{\ensuremath{\tprob{#1\,\left|\,#2\vphantom{#1}\right.}}}
\providecommand{\condprob}[2]{\ensuremath{\prob{#1\,\left|\,#2\vphantom{#1}\right.}}}

\providecommand{\tcondprobsub}[3]{\ensuremath{\tprobsub{#1}{#2\,\left|\,#3\vphantom{#1}\right.}}}
\providecommand{\condprobsub}[3]{\ensuremath{\probsub{#1}{#2\,\left|\,#3\vphantom{#1}\right.}}}

\providecommand{\texpect}[1]{\ensuremath{\operatorname{\expectationname}\pc@elrk{#1}}}
\providecommand{\expect}[1]{\ensuremath{\operatorname{\expectationname}\pc@elrk*{#1}}}

\providecommand{\texpsub}[2]{\ensuremath{\operatorname{\expectationname}_{#1}\pc@elrk{#2}}}
\providecommand{\expsub}[2]{\ensuremath{\operatorname{\expectationname}_{#1}\pc@elrk*{#2}}}

\providecommand{\tcondexp}[2]{\ensuremath{\texpect{#1\,\left|\,#2\vphantom{#1}\right.}}}
\providecommand{\condexp}[2]{\ensuremath{\expect{#1\,\left|\,#2\vphantom{#1}\right.}}}

\providecommand{\tcondexpsub}[3]{\ensuremath{\texpsub{#1}{#2\,\left|\,#3\vphantom{#1}\right.}}}
\providecommand{\condexpsub}[3]{\ensuremath{\expsub{#1}{#2\,\left|\,#3\vphantom{#1}\right.}}}

\providecommand{\supp}[1]{\ensuremath{\operatorname{Supp}\pc@olrk*{#1}}}

\providecommand{\entropy}[1]{\ensuremath{\operatorname{H}\pc@olrk*{#1}}}
\providecommand{\condentropy}[2]{%
\ensuremath{\operatorname{H}\pc@olrk*{#1\,\left|\,#2\vphantom{#1}\right.}}}

\providecommand{\minentropy}[1]{\ensuremath{\operatorname{H_\infty}\pc@olrk*{#1}}}
\providecommand{\tminentropy}[1]{\ensuremath{\operatorname{H_\infty}\pc@olrk{#1}}}
\providecommand{\condminentropy}[2]{%
\ensuremath{\operatorname{H_\infty}\pc@olrk*{#1\,\left|\,#2\vphantom{#1}\right.}}}
\providecommand{\tcondminentropy}[2]{%
\ensuremath{\operatorname{H_\infty}\pc@olrk{#1\,\left|\,#2\vphantom{#1}\right.}}}
\providecommand{\condavgminentropy}[2]{%
\ensuremath{\operatorname{\tilde{H}_\infty}\pc@olrk*{#1\,\left|\,#2\vphantom{#1}\right.}}}
\providecommand{\tcondavgminentropy}[2]{%
\ensuremath{\operatorname{\tilde{H}_\infty}\pc@olrk{#1\,\left|\,#2\vphantom{#1}\right.}}}
}
```

9.1.5 sets

\NN
\ZZ
\CC
\QQ
\RR
\PP
\FF
\GG
\set
\sequence
\bin

The *sets* option defines various macros for standard sets such as natural numbers \NN ($\mathbb{N}$). The style can be configured via \pcsetstyle.

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

```
\DeclareOption{sets}{
\@pc@opt@amsfontstrue

\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}}

\providecommand{\set}[1]{\ensuremath{\pc@clrk*{#1}}}
\providecommand{\sequence}[1]{\ensuremath{\pc@olrk*{#1}}}
\providecommand{\bin}{\ensuremath{\{0,1\}}}
}
```

9.1.6 noamsfonts

\@pc@opt@amsfontsfalse

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

\indcpa
\indcca
\indccai
\indccaii
\priv
\ind
\indcda
\prvcda
\prvrcda
\kiae
\kdae
\mle
\uce
\eufcma
\eufko
\eufnacma
\seufcma

The *notion* package option defines various cryptographic security notions. The style to be can be defined via \pcnotionstyle.

```
\DeclareOption{notions}{
\providecommand{\indcpa}{\pcnotionstyle{IND\pcmathhyphen{}CPA}}
\providecommand{\indcca}{\pcnotionstyle{IND\pcmathhyphen{}CCA}}
\providecommand{\indccai}{\pcnotionstyle{IND\pcmathhyphen{}CCA1}}
\providecommand{\indccaii}{\pcnotionstyle{IND\pcmathhyphen{}CCA2}}
\providecommand{\priv}{\pcnotionstyle{PRIV}}
\providecommand{\ind}{\pcnotionstyle{IND}}
\providecommand{\indcda}{\pcnotionstyle{IND\pcmathhyphen{}CDA}}
\providecommand{\prvcda}{\pcnotionstyle{PRV\pcmathhyphen{}CDA}}
\providecommand{\prvrcda}{\pcnotionstyle{PRV\$\pcmathhyphen{}CDA}}
\providecommand{\kiae}{\pcnotionstyle{KIAE}}
\providecommand{\kdae}{\pcnotionstyle{KDAE}}
\providecommand{\mle}{\pcnotionstyle{MLE}}
\providecommand{\uce}{\pcnotionstyle{UCE}}

\providecommand{\eufcma}{\pcnotionstyle{EUF\pcmathhyphen{}CMA}}
\providecommand{\eufnacma}{\pcnotionstyle{EUF\pcmathhyphen{}naCMA}}
\providecommand{\seufcma}{\pcnotionstyle{SUF\pcmathhyphen{}CMA}}

\providecommand{\eufko}{\pcnotionstyle{EUF\pcmathhyphen{}KO}}
```

```
}
```

9.1.8 logic

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

```
\DeclareOption{logic}{
% load centernot needed for notimplies
\@pc@opt@centernottrue

\providecommand{\AND}{\ensuremath{\mathrm{AND}}}
\providecommand{\OR}{\ensuremath{\mathrm{OR}}}
\providecommand{\NOR}{\ensuremath{\mathrm{NOR}}}
\providecommand{\NOT}{\ensuremath{\mathrm{NOT}}}
\providecommand{\NAND}{\ensuremath{\mathrm{NAND}}}
\providecommand{\XOR}{\ensuremath{\mathrm{XOR}}}
\providecommand{\XNOR}{\ensuremath{\mathrm{XNOR}}}
\providecommand{\xor}{\ensuremath{\oplus}}
\providecommand{\false}{\mathsf{false}}
\providecommand{\true}{\mathsf{true}}
\providecommand{\notimplies}{\centernot\implies}
}
```

9.1.9 ff (function families)

\kgen
\pgen
\eval
\invert
\il
\ol
\kl
\nl
\rl

The *ff* option defines macros for function families. Algorithms are typeset via `\pcalgostyle`.

```
\DeclareOption{ff}{
\providecommand{\kgen}{\pcalgostyle{KGen}}
\providecommand{\pgen}{\pcalgostyle{Pgen}}
\providecommand{\eval}{\pcalgostyle{Eval}}
\providecommand{\invert}{\pcalgostyle{Inv}}

\providecommand{\il}{\pcalgostyle{il}}
\providecommand{\ol}{\pcalgostyle{ol}}
\providecommand{\kl}{\pcalgostyle{kl}}
\providecommand{\nl}{\pcalgostyle{nl}}
\providecommand{\rl}{\pcalgostyle{rl}}
}
```

9.1.10 mm (machine models)

\pcmachinemodelstyle
\CRKT
\TM
\PROG
\uTM
\uC
\uP
\csize
\tmtime
\ppt

The *mm* option defines macros for machine models.

```
\DeclareOption{mm}{
\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}{\pcalgostyle{PPT}}
}
```

9.1.11 advantage

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

```
\DeclareOption{advantage}{
\@pc@opt@advantagetrue
}
```

9.1.12 primitives

The *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}}
```

`\prover` `\verifier` `\nizk` `\hash` `\gash` `\fash` `\enc` `\dec` `\sig` `\sign` `\verify` `\obf` `\iO` `\diO` `\owf` `\prf` `\prp` `\prg` `\mac` `\puncture` `\source` `\predictor` `\sam` `\dist` `\distinguisher` `\simulator` `\ext` `\extractor`

```
\providecommand{\dist}{\pcalgostyle{D}}
\providecommand{\distinguisher}{\pcalgostyle{Dist}}
\providecommand{\simulator}{\pcalgostyle{Sim}}
\providecommand{\ext}{\pcalgostyle{Ext}}
\providecommand{\extractor}{\ext}
}
```

9.1.13 oracles

\Oracle \oracle \ro

The *oracles* package option defines macros for typesetting oracles.

```
\DeclareOption{oracles}{
\providecommand{\Oracle}[1]{\pcalgostyle{O{#1}}}

\def\oracle{\bgroup\oracle@}
\newcommand{\oracle@}[1][]{\ifthenelse{\equal{#1}{}}{\oracle@@{O}}{\oracle@@{#1}}}
\def\oracle@@#1{\pcoraclestyle{#1}\egroup}

\providecommand{\ro}{\pcoraclestyle{RO}}
}
```

9.1.14 events

\event \nevent \bad \nbad

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

```
\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}}}
}
```

9.1.15 complexity

\complclass \cocomplclass \npol \conpol \pol \bpp \ppoly \AM \coAM \AC \NC \TC \PH \csigma \cpi \cosigma \copi

The *complexity* package option defines various complexity classes. The style can be adjusted via \pccomplexitystyle

```
\DeclareOption{complexity}{
\providecommand{\complclass}[1]{\pccomplexitystyle{#1}}
\providecommand{\cocomplclass}[1]{\pccomplexitystyle{co}\pcmathhyphen{}\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}}{\pccomplexitysty
\providecommand{\NC}[1]{\ensuremath{\ifthenelse{\equal{#1}{}}{\pccomplexitystyle{NC}}{\pccomplexitysty
\providecommand{\TC}[1]{\ensuremath{\ifthenelse{\equal{#1}{}}{\pccomplexitystyle{TC}}{\pccomplexitysty

\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}}
}
```

9.1.16 asymptotics

\negl
\poly
\cc
\ee
\kk
\mm
\nn
\pp
\qq
\rr

The *asymptotics* package option defines "polynomials" c (\cc), e (\ee), k (\kk), m (\mm), n (\nn), p (\pp), and q (\qq) as well as macros \negl and \poly.

```
\DeclareOption{asymptotics}{
\providecommand{\negl}[1][\secpar]{%
\pcpolynomialstyle{negl}\ifthenelse{\equal{#1}{}}{}{\pc@olrk*{#1}}}

\providecommand{\poly}[1][\secpar]{%
\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}}
}
```

9.1.17 keys

\pk
\vk
\sk
\key
\hk
\gk
\fk
\st
\state

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

```
\DeclareOption{keys}{
\providecommand{\pk}{\pckeystyle{pk}}
\providecommand{\vk}{\pckeystyle{vk}}
\providecommand{\sk}{\pckeystyle{sk}}

\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}}

\def\state{\bgroup\state@}
\newcommand{\state@}[1][]{\ifthenelse{\equal{#1}{}}{\state@@{state}}{\state@@{#1}}}
\def\state@@#1{\pckeystyle{#1}\egroup}
}
```

9.1.18 Security parameter

\SECPAR
\secpar
\secparam

The n option defines security parameter macros \secpar and \secparam using n. See also "lambda" package option.

```
\DeclareOption{n}{
\providecommand{\SECPAR}{\ensuremath{{N_0}}}
\providecommand{\secpar}{\ensuremath{n}}
\providecommand{\secparam}{\ensuremath{1^\secpar}}
}
```

\SECPAR
\secpar
\secparam

The n option defines security parameter macros \secpar and \secparam using λ. See also "n" package option.

```
\DeclareOption{lambda}{
\renewcommand{\SECPAR}{\ensuremath{\Lambda}}
\renewcommand{\secpar}{\ensuremath{\lambda}}
\renewcommand{\secparam}{\ensuremath{1^\secpar}}
}
```

9.2 Preamble and Option Parsing

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

```
\DeclareOption*{%
\PackageError{cryptocode}{Unknown option `\CurrentOption'}%
}
```

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

```
\ExecuteOptions{n}
```

We are now ready to process all package options

```
\ProcessOptions\relax
```

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.

```
\RequirePackage{etex}
\if@pc@opt@amsfonts
\RequirePackage{amsfonts}
\fi
\if@pc@opt@centernot
\RequirePackage{centernot}
\fi
\RequirePackage{xcolor}
\RequirePackage{calc}
\RequirePackage{tikz}
\usetikzlibrary{positioning,calc}
\RequirePackage{ifthen}
\RequirePackage{xargs}
\RequirePackage{pgf}
\RequirePackage{forloop}
\RequirePackage{array}
\RequirePackage{xparse}
\RequirePackage{expl3}
\RequirePackage{pbox}
\RequirePackage{varwidth}
\RequirePackage{suffix}
```

```
\RequirePackage{etoolbox}
\RequirePackage{environ}
\RequirePackage{xkeyval}
```

\pcadvantagesuperstyle
\pcadvantagename
\pcadvantagesubstyle
\advantage

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

```
\if@pc@opt@advantage
\providecommand{\pcadvantagesuperstyle}[1]{\mathrm{\MakeLowercase{#1}}}
\providecommand{\pcadvantagesubstyle}[1]{#1}
\providecommand{\pcadvantagename}{\mathsf{Adv}}

\newcommandx*{\advantage}[3][3=(\secpar)]{\ensuremath{\pcadvantagename^{\pcadvantagesuperstyle{#1}}_{\p
\fi
```

9.3 Global Macros

9.3.1 Styles

\pcalgostyle
\pcsetstyle
\pccomplexitystyle
\pcpolynomialstyle
\pcadvstyle
\pcnotionstyle
\pckeystyle
\pcmachinemodelstyle
\pcoraclestyle

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

```
\providecommand{\pcalgostyle}[1]{\ensuremath{\mathsf{#1}}}
\providecommand{\pcsetstyle}[1]{\ensuremath{\mathbb{#1}}}
\providecommand{\pccomplexitystyle}[1]{\ensuremath{\mathsf{#1}}}
\providecommand{\pcpolynomialstyle}[1]{\ensuremath{\mathsf{#1}}}
\providecommand{\pcadvstyle}[1]{\ensuremath{\mathcal{#1}}}
\providecommand{\pcnotionstyle}[1]{\ensuremath{\mathrm{#1}}}
\providecommand{\pckeystyle}[1]{\ensuremath{\mathsf{\protect\vphantom{p}#1}}}
\providecommand{\pcmachinemodelstyle}[1]{\ensuremath{\mathsf{#1}}}
\providecommand{\pcoraclestyle}[1]{\ensuremath{\mathsf{#1}}}
```

9.3.2 Order of Growth

\pc@olrk
\pc@olrk*
\pc@elrk
\pc@elrk*
\pc@clrk
\pc@clrk*

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

```
\DeclarePairedDelimiter\pc@olrk{(}{)}
\DeclarePairedDelimiter\pc@elrk{[}{]}
\DeclarePairedDelimiter\pc@clrk{\{}{\}}
```

9.3.3 Spacing

\pcaboveskip
\pcbelowskip
\pcbeforeskip
\pcafterskip

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

```
\newlength\pcaboveskip
\setlength\pcaboveskip{\abovedisplayskip}

\newlength\pcbelowskip
\setlength\pcbelowskip{\belowdisplayskip}

\newlength\pcbeforeskip
\newlength\pcafterskip
```

9.3.4 Keywords and Highlighting

\highlightkeyword
\highlightaltkeyword

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

```
\newcommand{\highlightkeyword}[2][\ ]{\ensuremath{\mathbf{#2}}#1}
\newcommand{\highlightaltkeyword}[2][\ ]{\ensuremath{\mathsf{#2}}#1}
```

\pcglobvar All predefined (highlightable) keywords.

```
\newcommand{\pcglobvar}{\highlightkeyword{gbl}}
\newcommand{\pcnew}{\highlightkeyword{new}}
\newcommand{\pcwhile}{\@pc@increaseindent\highlightkeyword{while}}
\newcommand{\pcendwhile}{\@pc@decreaseindent\highlightkeyword{endwhile}}
\newcommandx*{\pcdo}[2][1=\ ,2=]{#1\highlightkeyword[#2]{do}}
\newcommandx*{\pcif}[1][1=\ ]{\@pc@increaseindent\highlightkeyword[#1]{if}}
\newcommandx*{\pcunless}[1][1=\ ]{\@pc@increaseindent\highlightkeyword[#1]{unless}}
\newcommandx*{\pcelse}[1][1=\ ]{\@pc@tmpdecreaseindent\highlightkeyword[#1]{else}}
\newcommandx*{\pcelseif}[1][1=\ ]{\@pc@tmpdecreaseindent\highlightkeyword[#1]{else if}}
\newcommand{\pcfi}{\@pc@decreaseindent\highlightkeyword{fi}}
\newcommand{\pcendif}{\@pc@decreaseindent\highlightkeyword{endif}}
\newcommand{\pcendfor}{\@pc@decreaseindent\highlightkeyword{endfor}}
\newcommandx*{\pcthen}[2][1=\ ,2=\ ]{#1\highlightkeyword[#2]{then}}
\newcommand{\pcreturn}{\highlightkeyword{return}}
\newcommandx*{\pcin}[2][1=\ ,2=]{#1\highlightkeyword[#2]{in}}
\newcommandx*{\pcfor}[1][1=\ ]{\@pc@increaseindent\highlightkeyword[#1]{for}}
\newcommand{\pcrepeat}[1]{%
\@pc@increaseindent\ensuremath{%
\highlightkeyword{repeat} #1\ \highlightkeyword{times}%
}}
\newcommand{\pcrepeatuntil}[2]{%
\ensuremath{\highlightkeyword{repeat}\ #1\ \highlightkeyword{until}\ #2}}
\newcommand{\pcforeach}{\@pc@increaseindent\highlightkeyword{foreach}}
\newcommand{\pcendforeach}{\@pc@decreaseindent\highlightkeyword{endforeach}}
\newcommand{\pcuntil}{\@pc@decreaseindent\highlightkeyword{until}}
\newcommand{\pccontinue}{\highlightkeyword{continue}}
\newcommandx*{\pcfalse}[2][1=\ ,2=]{\highlightkeyword[#2]{false}}
\newcommandx*{\pctrue}[2][1=\ ,2=]{\highlightkeyword[#2]{true}}
\newcommandx*{\pcnull}[2][1=\ ,2=]{\highlightkeyword[#2]{null}}
\newcommand{\pcdone}{\highlightkeyword{done}}
\newcommand{\pcparse}{\highlightkeyword{parse}}
\newcommand{\pcfail}{\highlightkeyword{fail}}
\newcommand{\pcabort}{\highlightkeyword{abort}}
\newcommand{\pcassert}{\highlightkeyword{assert}}
```

9.3.5 Misc

\pcmathhyphen Definition of a hyphen to be used within math formulas.

```
\mathchardef\pcmathhyphen ="2D
```

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

```
\newcommand{\pccomment}[2][1em]{\hspace{#1}{\mbox{/\!\!/ } \text{\scriptsize#2}}}
\newcommand{\pclinecomment}[2][0em]{\hspace{#1}{\mbox{/\!\!/ } \text{\scriptsize#2}}}
```

9.4 Internal Helper Functions

\@expandedsetkeys

```
\newcommand\@pc@ifinfloat[2]{\ifnum\@floatpenalty<0\relax#1\else#2\fi}
```

`\@expandedsetkeys` Calls `\setkeys` from the xkeyval package but before exapands argument number 4. Arguments `{`⟨*families*⟩`}` `{`⟨*na*⟩`}` `{`⟨*first set of keys*⟩`}` `{`⟨*keys to be expanded*⟩`}` `{`⟨*final set of keys*⟩`}`

```
\newcommand*\@expandedsetkeys[5]{\expandafter\@expandedsetkeys@\expandafter{#4}{#1}{#2}{#3}{#5}}
\def\@expandedsetkeys@#1#2#3#4#5{\setkeys{#2}[#3]{#4,#1,#5}}

\newenvironment{@pc@withspaces}
 {\obeyspaces\begingroup\lccode`~=` \lowercase{\endgroup\let~}\ }
 {}
```

`\@pc@settowidthofalign` `\@pc@settowidthofaligned` Commands to measure width of an align (resp. aligned) environment. Takes two arguments a length in which to store the resulting width and the content.

```
\newcommand{\@pc@settowidthofalign}[2]{%
  \setbox\z@=\vbox{\@pseudocodecodesize
    \begin{flalign*}
    #2
    \ifmeasuring@\else\global\let\got@maxcolwd\maxcolumn@widths\fi
    \end{flalign*}
  }%
  \begingroup
  \def\or{+}\edef\x{\endgroup#1=\dimexpr\got@maxcolwd\relax}\x}

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

`\@pc@ifdraft` Check for draft mode.

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

`\@pc@executeblindly` Run stuff in an empty box

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

We need to fiddle with the label command to use it in `\pseudocode`. To access the original, we store it in

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

`\@pc@globaladdtolength` `\@pc@globalsetlength` A helper command to set (resp. add to) the length to a given value globally even when being within a scoped grouping.

```
\newcommand*{\@pc@globaladdtolength}[2]{%
\addtolength{#1}{#2}%
\global#1=#1\relax}

\newcommand*{\@pc@globalsetlength}[2]{%
\setlength{#1}{#2}%
\global#1=#1\relax}
```

`@pc@global@pc@cnt` `@pc@global@pc@nestcnt` A global counter storing the number of times the pseudocode command was triggered.

```
\newcounter{@pc@global@pc@cnt}
\newcounter{@pc@global@pc@nestcnt}
```

Fix hyperref package.. gnarl http://tex.stackexchange.com/questions/130319/incompatibility-between-etoolbox-and-hyperref

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

Allow to support cleveref package. It wants to be loaded after amsmath which is why it needs to be loaded after cryptocode. To do the necessary label fixes, we need to run these after its label fixes at begin document.

```
\providecommand{\pcfixcleveref}{
\AtBeginDocument{%
\pcfixhyperref%
\makeatletter%
\crefformat{@pclinenumber}{line~##2##1##3}%
\crefrangeformat{@pclinenumber}{lines~##3##1##4 to~##5##2##6}%
\makeatother%
}}
```

9.5 Stacking

In the following we define two stacking environments `pchstack` and `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`

```
\newcounter{@pc@stackdepth}
\newcommand{\@pc@incstackdepth}{\addtocounter{@pc@stackdepth}{1}}
\newcommand{\@pc@decstackdepth}{\addtocounter{@pc@stackdepth}{-1}}
```

9.5.3 Stacking Options

`center` Allows to center the stack.

`\@pc@centerstack`

```
\newcommand{\@pc@centerstack}{false}
\define@key{pcstack}{center}[true]{\ifthenelse{\equal{#1}{true}}
{\renewcommand{\@pc@centerstack}{true}}
{\renewcommand{\@pc@centerstack}{false}}}%
```

`boxed` Allows to draw a box around the stack.

`\@pc@boxedstack`

```
\newcommand{\@pc@boxedstack}{false}
\define@key{pcstack}{boxed}[true]{\ifthenelse{\equal{#1}{true}}
{\renewcommand{\@pc@boxedstack}{true}}
{\renewcommand{\@pc@boxedstack}{false}}}%
```

noindent
\@pc@noindentstack

Allows to draw a box around the stack.

```
\newcommand{\@pc@noindentstack}{false}
\define@key{pcstack}{noindent}[true]{\ifthenelse{\equal{#1}{true}}
{\renewcommand{\@pc@noindentstack}{true}}
{\renewcommand{\@pc@noindentstack}{false}}}%
```

inline
\@pc@inlinestack

Allows to keep the pchstack inline and not creating a paragraph.

```
\newcommand{\@pc@inlinestack}{false}
\define@key{pcstack}{inline}[true]{\ifthenelse{\equal{#1}{true}}
{\renewcommand{\@pc@inlinestack}{true}}
{\renewcommand{\@pc@inlinestack}{false}}}%
```

space
\pchstackspace
\pcvstackspace
\@pc@centerstack

Introduces horizontal (resp. vertical) space in-between pseudocode blocks in stacking environments .

```
\providecommand{\pchstackspace}{0pt}
\providecommand{\pcvstackspace}{0pt}
\newcommand{\@pc@stackspace@forpseudocode}{}
\newlength{\@pc@stackspace@len}
\newcommand*{\@pc@stackspace}{0pt}
\newcommand*{\@pc@reset@stackspace}{\setlength{\@pc@stackspace@len}{\@pc@stackspace}}
\define@key{pcstack}{space}[0pt]{\renewcommand*{\@pc@stackspace}{#1}}%
```

aboveskip
@applyaboveskipinstack
ovespaceunlessstacking

By default \pcaboveskip is applied on the outer most stacking environment. Can be overriden using aboveskip.

```
\newcommand{\@pc@addabovespaceunlessstacking}{%
\ifthenelse{\value{@pc@stackdepth}=0}{\par\addvspace{\pcaboveskip}}{}}

\newcommand{\@pc@applyaboveskipinstack}{\@pc@addabovespaceunlessstacking}
\let\org@pc@applyaboveskipinstack\@pc@applyaboveskipinstack

\define@key{pcstack}{aboveskip}[default]{\ifthenelse{\equal{#1}{default}}
{\renewcommand{\@pc@applyaboveskipinstack}{\org@pc@applyaboveskipinstack}}
{\renewcommand{\@pc@applyaboveskipinstack}{\vspace{#1}}}}%
```

belowskip
@applybelowskipinstack
lowspaceunlessstacking

By default \pcbelowskip is applied on the outer most stacking environment. Can be overriden using belowskip.

```
\newcommand{\@pc@addbelowspaceunlessstacking}{%
\ifthenelse{\value{@pc@stackdepth}=0}
{\@pc@ifinfloat{}{\par\addvspace{\pcbelowskip}}}
{}}

\newcommand{\@pc@applybelowskipinstack}{\@pc@addbelowspaceunlessstacking}
\let\org@pc@applybelowskipinstack\@pc@addbelowspaceunlessstacking

\define@key{pcstack}{belowskip}[default]{\ifthenelse{\equal{#1}{default}}
{\renewcommand{\@pc@applybelowskipinstack}{\org@pc@applybelowskipinstack}}
{\renewcommand{\@pc@applybelowskipinstack}{\par\addvspace{#1}}}}%
```

\pcbeforehstackskip
\pcafterhstackskip

Allows adding global skips before and after \pchstack blocks.

```
\newlength{\pcbeforehstackskip}
\newlength{\pcafterhstackskip}
```

\@pc@boxedstack

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

```
\newsavebox{\@pc@stackcontentbox}%
```

\pcsethstackargs
\pcsetvstackargs

```
\newcommand*\@pc@hstack@defaultargs{}
\newcommand*\pcsethstackargs[1]{\renewcommand*\@pc@hstack@defaultargs{#1}}
\newcommand*\@pc@vstack@defaultargs{}
\newcommand*\pcsetvstackargs[1]{\renewcommand*\@pc@vstack@defaultargs{#1}}
```

9.5.4 The Stacking Environments

pccenter

```
\newenvironment{pccenter}{%
\setlength\topsep{0pt}\setlength\parskip{0pt}%
\begin{center}}{\end{center}}
```

pchstack A stacking environment for horizontally stacked pseudocode blocks.

```
\NewEnviron{pchstack}[1][]{%
%Ensure that the parameters are defaulted
\begingroup%
% parse args this is the same as
% \setkeys{pcstack}{center=false,boxed=false,aboveskip=default,belowskip=default,space=\pchstackspace,#
% expect that we expand the default args
\@expandedsetkeys{pcstack}{}{center=false,boxed=false,aboveskip=default,belowskip=default,space=\pchsta
\@pc@reset@stackspace%
%add above skip except when in inline mode
\ifthenelse{\equal{\@pc@inlinestack}{true}}{}{\@pc@applyaboveskipinstack}%
\@pc@incstackdepth%
\renewcommand{\@pc@stackspace@forpseudocode}{\hspace{\@pc@stackspace}}%
%Store main content in a box
\ifthenelse{\equal{\@pc@boxedstack}{true}}%
{\sbox{\@pc@stackcontentbox}
   {\fbox{\mbox{\hspace{\pcbeforehstackskip}\BODY\hspace{\pcafterhstackskip}\hspace{-\@pc@stackspace}}}
{\sbox{\@pc@stackcontentbox}
   {\mbox{\hspace{\pcbeforehstackskip}\BODY\hspace{\pcafterhstackskip}\hspace{-\@pc@stackspace}}}}%
% handle noindent
\ifthenelse{\equal{\@pc@noindentstack}{true}}{\par\noindent\ignorespaces}{}%
%set content either centered or directly
\ifthenelse{\equal{\@pc@centerstack}{true}}%
{\begin{pccenter}\usebox{\@pc@stackcontentbox}\end{pccenter}}
{\usebox{\@pc@stackcontentbox}}%
% cleanup
\@pc@decstackdepth%
\ifthenelse{\equal{\@pc@inlinestack}{true}}{}{\@pc@applybelowskipinstack}%
\endgroup%reset space outside group
\@pc@reset@stackspace%
\@pc@stackspace@forpseudocode%
%ignore any spaces after, to allow staying within paragraph
\ignorespacesafterend\noindent%
}
```

pchstack A stacking environment for vertically stacked pseudocode blocks.

```
\NewEnviron{pcvstack}[1][]{%
%Ensure that the parameters are defaulted
\begingroup%
% parse args this is the same as
% \setkeys{pcstack}{center=false,boxed=false,aboveskip=default,belowskip=default,space=\pcvstackspace,#
% expect that we expand the default args
```

```
\@expandedsetkeys{pcstack}{}{center=false,boxed=false,aboveskip=default,belowskip=default,space=\pcvsta
\@pc@reset@stackspace%
\@pc@applyaboveskipinstack%
\@pc@incstackdepth%
\renewcommand{\@pc@stackspace@forpseudocode}{\par\vspace{\@pc@stackspace}}%
%Store main content in a box
\sbox{\@pc@stackcontentbox}{%
\ifthenelse{\equal{\@pc@boxedstack}{true}}%
{\fbox{\raisebox{\dimexpr\ht\strutbox-\height}{\begin{varwidth}[t]{2\linewidth}\BODY\end{varwidth}}}}%
{\raisebox{\dimexpr\ht\strutbox-\height}{\begin{varwidth}[t]{2\linewidth}\BODY\end{varwidth}}}%
\vspace{-\@pc@stackspace}}%
% handle noindent
\ifthenelse{\equal{\@pc@noindentstack}{true}}{\par\noindent\ignorespaces}{}%
% display content
\ifthenelse{\equal{\@pc@centerstack}{true}}%
{\begin{pccenter}\usebox{\@pc@stackcontentbox}\end{pccenter}}%
{\usebox{\@pc@stackcontentbox}}%
% cleanup
\@pc@decstackdepth%
\@pc@applybelowskipinstack%
\endgroup%reset space outside group
\@pc@reset@stackspace%
\@pc@stackspace@forpseudocode%
%ignore any spaces after, to allow staying within paragraph
\ignorespacesafterend\noindent%
}
```

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]{0pt}{\@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}{&}
```

\pcind

An indentation macro to be used within pseudocode. As writing \pcind is a bit cumbersome, there is a shorthand that can be defined via \pcindentname (defaults to t). See below.

```
\newlength{\@pcindentwidth}
\providecommand{\pcind}[1][1]{%
\setlength{\@pcindentwidth}{\widthof{\ensuremath{\quad}}*#1}%
\ensuremath{\mathmakebox[\@pcindentwidth]{}}}
```

\pctabname
\pcdbltabname
\pcindentname

Shorthands for alignment tabs and indentation. These are defined only within the pseudocode 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{@pcrlinenumbertmp}
```

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 allow to skip numbering single lines if linenumbering=on

\@pc@skiplnmarker

skipfirstln

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

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

\pcrln separate counter (\pcrln).

```
\providecommand{\pclnr}{%
\refstepcounter{pclinenumber}%
\stepcounter{Hpclinenumber}%
\hspace{\pclnrspace}\pcrlnseparator\pclnstyle{\arabic{pclinenumber}}}
\providecommand{\pcrln}{
\refstepcounter{pcrlinenumber}%
\stepcounter{Hpcrlinenumber}%
\hspace{\pclnrspace}\pcrlnseparator\pclnstyle{\arabic{pcrlinenumber}}}
```

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.

```
\newcommand*\@pseudocodehead{}
\newcommand*\@pseudocodewidth{}
\newcommand*\@pseudocodexshift{0pt}
\newcommand*\@pseudocodeyshift{0pt}
\newcommand*\@pseudocodelinenumber{}
\newcommand*\@pseudocodebeforeskip{0ex}
\newcommand*\@pseudocodeafterskip{0ex}
\newcommand*\@pseudocodelnstart{0}
\newcommand*\@pseudocodelnstartright{0}
\newcommand*\@pseudocodesyntaxhighlighting{}
\newcommand*\@pseudocodenodraft{false}
\newcommand*\@pseudocodecolspace{} % empty per default, use length,

\newcommand*\@pseudocodeheadlinecmd{\hrule}

```

headlinesep Distance between header and line.

\pcheadlinesep

@pseudocodeheadlinesep

```
\newlength\pcheadlinesep
\setlength\pcheadlinesep{0pt}
\newcommand*\@pseudocodeheadlinesep{0em}
\define@key{pseudocode}{headlinesep}[0em]{\renewcommand*\@pseudocodeheadlinesep{#1}}
```

bodylinesep
\pcbodylinesep
@pseudocodebodylinesep

```
\newlength\pcbodylinesep
\setlength\pcbodylinesep{0.3\baselineskip}
\newcommand*\@pseudocodebodylinesep{0em}
\define@key{pseudocode}{bodylinesep}[0em]{\renewcommand*\@pseudocodebodylinesep{#1}}
```

headheight
\pcheadheight
\@pc@headheightskip
\@pseudocodeheadheight

```
\newlength\@pseudocodeheadheight@len
\newcommand{\@pc@headheightskip}{%
\setlength{\@pseudocodeheadheight@len}{\@pseudocodeheadheight}%
\vphantom{\rule[0.5ex-0.5\@pseudocodeheadheight@len]{0pt}{\@pseudocodeheadheight@len}}%
}
\newlength\pcheadheight
\setlength{\pcheadheight}{3.25ex}
\newcommand*\@pseudocodeheadheight{\pcheadheight}
\define@key{pseudocode}{headheight}[0em]{\renewcommand*\@pseudocodeheadheight{#1}}

\newcommand*\@pseudocodecolsep{0em}
\newcommand*\@pseudocodeaddtolength{2pt}

\newcommand*\@pseudocodecodeatbeginline{}
\newcommand*\@pseudocodecodeatendline{}
\newcommand*\@pseudocodecodejot{0em}
\newcommand*\@pseudocodecodesize{\small}
\newcommand*\@pseudocodesubcodesize{\footnotesize}

\newcommand*\@pseudocodeminipagealign{t}

%
% Define keywords for the automatic syntax highlighting
% the accompanying add provides additional keywords.
% The space version for automatic spacing
\newcommand*\@pseudocodekeywordsindent{for ,foreach ,if ,repeat ,while }
\newcommand*\@pseudocodekeywordsunindent{endfor,endforeach,fi,endif,until,endwhile}
\newcommand*\@pseudocodekeywordsuninindent{else if ,elseif ,else }
\newcommand*\@pseudocodekeywords{for,foreach,{return },return,{ do },{ in },new,if, null, true,{until }
\newcommand*\@pseudocodeaddkeywords{}
\newcommand*\@pseudocodealtkeywords{}
\begin{@pc@withspaces}
\global\def\@pseudocodekeywordsspace{for,endfor,foreach,endforeach,return,do,in,new,if,null,true,until
\end{@pc@withspaces}
```

Specification of the various options of the \pseudocode command.

```
\define@key{pseudocode}{beginline}[]{\renewcommand*\@pseudocodecodeatbeginline{#1}}
\define@key{pseudocode}{endline}[]{\renewcommand*\@pseudocodecodeatendline{#1}}
\define@key{pseudocode}{jot}[0em]{\renewcommand*\@pseudocodecodejot{#1}}
\define@key{pseudocode}{codesize}[\small]{\renewcommand*\@pseudocodecodesize{#1}}
\define@key{pseudocode}{subcodesize}[\small]{\renewcommand*\@pseudocodesubcodesize{#1}}
\define@key{pseudocode}{head}[]{\renewcommand*\@pseudocodehead{#1}}
\define@key{pseudocode}{width}[]{\renewcommand*\@pseudocodewidth{#1}}
\define@key{pseudocode}{valign}[t]{\renewcommand*\@pseudocodeminipagealign{#1}}
\define@key{pseudocode}{xshift}[]{\renewcommand*\@pseudocodexshift{#1}}
\define@key{pseudocode}{yshift}[]{\renewcommand*\@pseudocodeyshift{#1}}
\define@key{pseudocode}{colspace}[]{\renewcommand*\@pseudocodecolspace{#1}}
\define@key{pseudocode}{linenumbering}[on]{\ifthenelse{\equal{#1}{on}}{\renewcommand*\@pseudocodelinenu
```

```
\define@key{pseudocode}{beforeskip}[]{\renewcommand*\@pseudocodebeforeskip{#1}}
\define@key{pseudocode}{afterskip}[]{\renewcommand*\@pseudocodeafterskip{#1}}
\define@key{pseudocode}{lnstart}[0]{\renewcommand*\@pseudocodelnstart{#1}}
\define@key{pseudocode}{lnstartright}[0]{\renewcommand*\@pseudocodelnstartright{#1}}
\define@key{pseudocode}{colsep}[0em]{\renewcommand*\@pseudocodecolsep{#1}}
\define@key{pseudocode}{headlinecmd}[\hrule]{\renewcommand*\@pseudocodeheadlinecmd{#1}}
\define@key{pseudocode}{addtolength}[2pt]{\renewcommand*\@pseudocodeaddtolength{#1}}
\define@key{pseudocode}{nodraft}[true]{\renewcommand*\@pseudocodenodraft{#1}}
\define@key{pseudocode}{keywords}[]{\renewcommand*\@pseudocodekeywords{#1}}
\define@key{pseudocode}{keywordsindent}[]{\renewcommand*\@pseudocodekeywordsindent{#1}}
\define@key{pseudocode}{keywordsunindent}[]{\renewcommand*\@pseudocodekeywordsunindent{#1}}
\define@key{pseudocode}{keywordsuninindent}[]{\renewcommand*\@pseudocodekeywordsuninindent{#1}}
\define@key{pseudocode}{addkeywords}[]{\renewcommand*\@pseudocodeaddkeywords{#1}}
\define@key{pseudocode}{altkeywords}[]{\renewcommand*\@pseudocodealtkeywords{#1}}
\define@key{pseudocode}{syntaxhighlight}[]{\renewcommand*\@pseudocodesyntaxhighlighting{#1}}
```

mode The [⟨*mode*⟩] key (with values ⟨*text*⟩ or ⟨*math*⟩ (default)) specifies whether within a pseudocode block input is by default typeset in text mode or in math mode. The \@pc... variables are variables that help typesetting each line in a pseudocode block.

```
\newcommand{\@pc@modebegin}{}
\newcommand{\@pc@modeend}{}
\define@key{pseudocode}{mode}[math]{%
\ifthenelse{\equal{#1}{text}}{%
\renewcommand*\@pc@modebegin{\begin{varwidth}{\textwidth}%
%introduce line magic for text mode
\let\@pc@lb\\%
\renewcommandx*{\\}[2][1=,2=]{\@pc@modeend\@pc@and \@pseudocodecodeatendline\ifthenelse{\equal{####1}{]
\def\pclb{\let\\\@pc@lb\relax\@pc@modeend\\}%
\def\pcolb{\let\\\@pc@lb\relax\@pc@modeend\\}%
}%
\renewcommand*\@pc@modeend{\end{varwidth}}%
}{\renewcommand{\@pc@modebegin}{}\renewcommand{\@pc@modeend}{}}}
```

minlineheight Control the minimal line height of pseudocode blocks.

\pcminlineheight

ocodecodeminlineheight

```
\providecommand{\pcminlineheight}{0pt}
\newcommand*\@pseudocodecodeminlineheight{\pcminlineheight}
\define@key{pseudocode}{minlineheight}[0pt]{\renewcommand*\@pseudocodecodeminlineheight{#1}}
```

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.

```
\ExplSyntaxOn
\tl_new:N \l_pc_strsub_input_tl
\tl_new:N \l_pc_strsub_search_tl
\tl_new:N \l_pc_strsub_replace_tl

\NewDocumentCommand{\@pc@stringsubstitution}{mmm}
 {
  \tl_set:Nn \l_pc_strsub_input_tl { #1 }
  \tl_set:Nn \l_pc_strsub_search_tl { #2 }
  \tl_set:Nn \l_pc_strsub_replace_tl { #3 }
 %  \tl_show_analysis:N  \l_pc_strsub_input_tl % uncomment for debugging
 %  \tl_show_analysis:N  \l_pc_strsub_search_tl % uncomment for debugging
 %  \tl_show_analysis:N  \l_pc_strsub_replace_tl % uncomment for debugging
```

```
   \regex_replace_all:nnN
      { \u{l_pc_strsub_search_tl} } %only match if keyword does not have a word character preceding
      { \u{l_pc_strsub_replace_tl} }
      \l_pc_strsub_input_tl
       %  \tl_show_analysis:N \l_tmpa_tl % uncomment for debugging
      \tl_use:N \l_pc_strsub_input_tl
 }
\ExplSyntaxOff
```

\@pc@syntaxhighlight
\@pc@highlight
\@pc@highlightindent
\@pc@highlightunindent
\@pc@highlightuninindent
\@pc@althighlight

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.

```
\newcommand{\@pc@syntaxhighlight}[1]{%
%don't highlight during measuring runs for performance improvements.
\ifmeasuring@#1\else%
\ifthenelse{\equal{\@pseudocodesyntaxhighlighting}{auto}}{%
\def\@shtmp{#1}% first step
% Depending on space mode, we might later run the indent/unindent/... lists
% if not, we add them now to tmp lists in order to have a complete list.
\ifthenelse{\equal{\@pseudocodespace}{keep}}
   {\edef\@tmpkeywords{\@pseudocodekeywordsspace,\@pseudocodeaddkeywords}}
   {\ifthenelse{\equal{\@pseudocodespace}{auto}}
      {\edef\@tmpkeywords{\@pseudocodekeywords,\@pseudocodeaddkeywords}}
      {\edef\@tmpkeywords{\@pseudocodekeywords,\@pseudocodekeywordsindent,\@pseudocodekeywordsunindent
\foreach \@pckw in \@tmpkeywords{%
\ifthenelse{\equal{\@pckw}{}}{}{%
% we are doing a simple strsub and storing the result (globally) in @shtmp
\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter
                  \gdef\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter
                  \@shtmp\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter
                 {\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter
                 \@pc@stringsubstitution\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter
                 {\expandafter\expandafter\expandafter\@shtmp\expandafter\expandafter\expandafter
                  }\expandafter\expandafter\expandafter{\expandafter\@pckw\expandafter}\expandafter{\expandafter
}}% alt keywords
\foreach \@pckw in \@pseudocodealtkeywords{%
\ifthenelse{\equal{\@pckw}{}}{}{%
% we are doing a simple strsub and storing the result (globally) in @shtmp
\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter
                  \gdef\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter
                  \@shtmp\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter
                 {\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter
                 \@pc@stringsubstitution\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter
                 {\expandafter\expandafter\expandafter\@shtmp\expandafter\expandafter\expandafter
                  }\expandafter\expandafter\expandafter{\expandafter\@pckw\expandafter}\expandafter{\expandafter
}}%
% if automatic spacing
\ifthenelse{\equal{\@pseudocodespace}{auto}}
{%
\foreach \@pckw in \@pseudocodekeywordsindent{% indentation keywords
\ifthenelse{\equal{\@pckw}{}}{}{%
% we are doing a simple strsub and storing the result (globally) in @shtmp
\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter
                  \gdef\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter
                  \@shtmp\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter
                 {\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter
```

```
                \@pc@stringsubstitution\expandafter\expandafter\expandafter\expandafter\expandafter\exp
                {\expandafter\expandafter\expandafter\@shtmp\expandafter\expandafter\expandafter
                 }\expandafter\expandafter\expandafter{\expandafter\@pckw\expandafter}\expandafter{\exp
}}%
\foreach \@pckw in \@pseudocodekeywordsunindent{% unindentation keywords
\ifthenelse{\equal{\@pckw}{}}{}{%
% we are doing a simple strsub and storing the result (globally) in @shtmp
\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter
                 \gdef\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expanda
                 \@shtmp\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expan
                {\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter
                \@pc@stringsubstitution\expandafter\expandafter\expandafter\expandafter\expandafter\exp
                {\expandafter\expandafter\expandafter\@shtmp\expandafter\expandafter\expandafter
                 }\expandafter\expandafter\expandafter{\expandafter\@pckw\expandafter}\expandafter{\exp
}}%
\foreach \@pckw in \@pseudocodekeywordsuninindent{% uninindentation keywords
\ifthenelse{\equal{\@pckw}{}}{}{%
% we are doing a simple strsub and storing the result (globally) in @shtmp
\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter
                 \gdef\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expanda
                 \@shtmp\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expan
                {\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter
                \@pc@stringsubstitution\expandafter\expandafter\expandafter\expandafter\expandafter\exp
                {\expandafter\expandafter\expandafter\@shtmp\expandafter\expandafter\expandafter
                 }\expandafter\expandafter\expandafter{\expandafter\@pckw\expandafter}\expandafter{\exp
}}%
}{}%
% return result
\@shtmp%
}{#1}% nothing to highlight
\fi}

\newcommand{\@pc@highlight}[1]{%
\ifthenelse{\equal{\@pseudocodespace}{keep}}
    {\highlightkeyword[]{#1}}%
    {\highlightkeyword[]{\@pc@stringsubstitution{#1}{ }{~}}}%
}

\newcommand{\@pc@highlightindent}[1]{%
\@pc@increaseindent\@pc@highlight{#1}%
}

\newcommand{\@pc@highlightunindent}[1]{%
\@pc@decreaseindent\@pc@highlight{#1}%
}

\newcommand{\@pc@highlightuninindent}[1]{%
\@pc@tmpdecreaseindent\@pc@highlight{#1}%
}

\newcommand{\@pc@althighlight}[1]{%
\ifthenelse{\equal{\@pseudocodespace}{keep}}
    {\highlightaltkeyword{#1}}%
    {\highlightaltkeyword{\@pc@stringsubstitution{#1}{ }{~}}}%
}
```

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*\@pseudocodesp


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

% automatic indentation
\newcounter{@pc@indentationlevel}
\newcommand{\@pc@increaseindent}{\addtocounter{@pc@indentationlevel}{1}}
\newcommand{\@pc@decreaseindent}{\ifthenelse{\equal{\@pseudocodespace}{auto}}{\pcind[-1]}{}\addtocounte
\newcommand{\@pc@tmpdecreaseindent}{\ifthenelse{\equal{\@pseudocodespace}{auto}}{\pcind[-1]}{}}

\newcounter{pccolumncounter}
\setcounter{pccolumncounter}{2}

% store original halign
\let\@pc@halign\halign%
```

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[]}%
}

\def\@pseudocodeA[#1]{%
\setkeys*{pcspace}{#1}%test if there is a space assignment within the keys .. make the necessary arrang
\@pseudocode[#1]%
}

\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\@pseudocodesyntaxhighlight
%
%
\addtocounter{@pc@global@pc@nestcnt}{1}%
% allow for tikz usage
\@pc@ensureremember%
%
% create tabbing command
\ifcsname \pctabname\endcsname%
\expandafter\renewcommand\csname \pctabname\endcsname{\@pc@modeend&\@pc@colspace\@pc@modebegin}%
\else%
\expandafter\newcommand\csname \pctabname\endcsname{\@pc@modeend&\@pc@colspace\@pc@modebegin}%
\fi%
\ifcsname \pcdbltabname\endcsname%
\expandafter\renewcommand\csname \pcdbltabname\endcsname{\@pc@modeend&&\@pc@colspace\@pc@modebegin}%
\else%
\expandafter\newcommand\csname \pcdbltabname\endcsname{\@pc@modeend&&\@pc@colspace\@pc@modebegin}%
\fi%
% create colspace command if necessary (must be empty for multicolumns
\ifthenelse{\equal{\@pseudocodecolspace}{}}
 {}
 {\renewcommand{\@pc@colspace}{\hspace{\@pseudocodecolspace}}}%
%
%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{@pc@global@pc@cnt}%
\setcounter{pclinenumber}{\@pseudocodelnstart}%
\setcounter{pcrlinenumber}{\@pseudocodelnstartright}%
\setlength{\@pc@minipage@length}{0pt}%
\setlength{\@pc@alt@minipage@length}{0pt}%
\setcounter{@pclinenumbertmp}{\value{pclinenumber}}%
\setcounter{@pcrlinenumbertmp}{\value{pcrlinenumber}}%
%reset column counter
\setcounter{pccolumncounter}{2}%
%
% vertical space
\vspace{\@pseudocodeyshift}%
%
%
%
% line magic
\ifthenelse{\value{@pc@global@pc@nestcnt}=1}{%
\let\@pc@halign\halign%
\newenvironment{pcmbox}{\let\halign\@pc@halign}{}%
\def\halign{%
\renewcommand{\label}[1]{\ifmeasuring@\else\@pc@original@label{####1}\fi}%
\let\@pc@lb\\%
\renewcommandx*{\\}[2][1=,2=]{\@pc@modeend\@pc@and\@pseudocodecodeatendline \ifthenelse{\equal{####1}{
\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}{}}{%
% compute length of pseudocode
\ifthenelse{\value{@pcsubprogstep}=0}{%
\@pc@settowidthofalign{\@pc@minipage@length}{\@pc@thecontent}%
}{%
\@pc@settowidthofaligned{\@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}}}%
% 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}%
% begin actual output
%
%
%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{@pc@global@pc@nestcnt}{-1}%
\endgroup%
% close spacing and potentially a single group generated by the space tester
\ifthenelse{\equal{\@withinspaces}{true}}{\end@pc@withspaces}{}%
\endgroup%
%insert space from stacking
\@pc@stackspace@forpseudocode%
}

\newcommand{\@pc@pseudocodeminipage}[1]{%
```

```
\begin{minipage}[#1]{\@pc@minipage@length}%
\ifthenelse{\value{@pcsubprogstep}=0}{%
\pc@display@pseudocode{\@pseudocodehead}{\@pc@thecontent}%
}{% if sub procedure
\pc@display@subcode{\@pseudocodehead}{\@pc@thecontent}%
}%
\end{minipage}%
}


\newcommand{\@pc@display@gameheader}[1]{%
\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};}
  {}%
}%
}

\let\pclb\relax
%
\newcommand{\pc@display@pseudocode}[2]{%
\ifthenelse{\equal{#1}{}}{\vspace{-\baselineskip}\@pseudocodecodesize{}}{%
\ifthenelse{\equal{\@withingame}{true}}
{\ifthenelse{\equal{\@pc@secondheader}{true}}
 {\@pc@display@gameheader{#1}\addtocounter{pcgamecounter}{1}\fboxsep=1pt\fbox{\vphantom{#1}\@pc@displa
 {\@pc@display@gameheader{#1}}}
{#1}%
\@pc@headheightskip\vspace{\pcheadlinesep}\vspace{\@pseudocodeheadlinesep}\@pseudocodeheadlinecmd%
\vspace{-\baselineskip}\vspace{\pcbodylinesep}\vspace{\@pseudocodebodylinesep}\@pseudocodecodesize}%
\begin{flalign*}#2\end{flalign*}%
}


\newcommand{\pc@display@subcode}[2]{%
\begingroup%
\ifthenelse{\equal{#1}{}}{}{#1\@pc@headheightskip%
\vspace{\pcheadlinesep}\vspace{\@pseudocodeheadlinesep}\@pseudocodeheadlinecmd{}%
\vspace{\pcbodylinesep}\vspace{\@pseudocodebodylinesep}}%
\@pseudocodesubcodesize%
$\begin{aligned}#2\end{aligned}$%
\endgroup%
}


\newcommand{\@pc@gettikzwidth}[2]{ % #1 = width, #2 = height
 \pgfextractx{\@tempdima}{\pgfpointdiff{\pgfpointanchor{current bounding box}{south west}}
 {\pgfpointanchor{current bounding box}{north east}}}
 \global#1=\@tempdima
 \pgfextracty{\@tempdima}{\pgfpointdiff{\pgfpointanchor{current bounding box}{south west}}
 {\pgfpointanchor{current bounding box}{north east}}}
 \global#2=\@tempdima
}


```

9.7 Create Pseudocode/Procedure Commands

```
%
% parameter reordering
\def\@pseudocodeB#1#2[#3]#4{\setkeys*{pcspace}{#2,#3}\@pseudocode[head={#1#4},#2,#3]}
\def\@pseudocodeC#1#2#3{\setkeys*{pcspace}{#2}\@pseudocode[head={#1#3},#2]}
%for no headers
\def\@pseudocodeE#1#2[#3]{\setkeys*{pcspace}{#2,#3}\@pseudocode[head={#1},#2,#3]}
\def\@pseudocodeF#1#2{\setkeys*{pcspace}{#2}\@pseudocode[head={#1},#2]}
%
```

createprocedurecommand Define pseudocode command with parameters:

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

```
\newcommand*{\@pc@createproc@headmode}{text}
\newcommand{\createprocedurecommand}[4]{
\expandafter\gdef\csname #1\endcsname{%
\begingroup%
\renewcommand{\@withinspaces}{false}%
#2%
\@ifnextchar[%]
  {\@pseudocodeB{#3}{#4}}
  {\@pseudocodeC{#3}{#4}}%
}%
}
```

reatepseudocodecommand

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

\createpseudocodeblock Creates a command that has pseudocode wrapped in an \pchstack.

name

options for \pchstack

code to execute after begingroup

head prefix

other config

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

\createprocedureblock Creates a command that has procedure wrapped in an \pchstack.

name

options for \pchstack

code to execute after begingroup

head prefix

other config

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

\procedure Create \procedure command.

\pseudocodeblock

\procedureblock

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

9.8 Subprocedures

```

%
% subprocedures
\newcounter{@pcsubprogcnt1}
\newcounter{@pcrsubprogcnt1}
\newcounter{@pcsubprogcnt2}
\newcounter{@pcrsubprogcnt2}
\newcounter{@pcsubprogcnt3}
```

```
\newcounter{@pcrsubprogcnt3}
\newcounter{@pcsubprogcnt4}
\newcounter{@pcrsubprogcnt4}
\newcounter{@pcsubprogcnt5}
\newcounter{@pcrsubprogcnt5}
\newcounter{@pcsubprogcnt6}
\newcounter{@pcrsubprogcnt6}
\newcounter{@pcsubprogcnt7}
\newcounter{@pcrsubprogcnt7}
\newcounter{@pcsubprogcnt8}
\newcounter{@pcrsubprogcnt8}
\newcounter{@pcsubprogcnt9}
\newcounter{@pcrsubprogcnt9}
\newcounter{@pcsubprogstep}

\newenvironment{subprocedure}{%
\addtocounter{@pcsubprogstep}{1}%
% store old counter values
\setcounter{@pcsubprogcnt\the@pcsubprogstep}{\value{pclinenumber}}%
\setcounter{@pcrsubprogcnt\the@pcsubprogstep}{\value{pcrlinenumber}}%
}{%
\setcounter{pclinenumber}{\value{@pcsubprogcnt\the@pcsubprogstep}}%
\setcounter{pcrlinenumber}{\value{@pcrsubprogcnt\the@pcsubprogstep}}%
\addtocounter{@pcsubprogstep}{-1}}


```

9.9 Protocols

```

%
% send message
\newcommand{\pcshortmessageoffset}{0.5cm}
\newcommand{\pcdefaultmessagelength}{3.5cm}
\newcommand{\pcdefaultlongmessagelength}{6cm}
\newcommand{\pcbeforemessageskip}{0pt}
\newcommand{\pcaftermessageskip}{10pt}
\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}
```

```
1234
1235 \define@key{pcsendmessage}{centercol}[]{\renewcommand*\@pcsendmessagecol{#1}}
1236 \define@key{pcsendmessage}{width}[]{\renewcommand*\@pcsendmessagewidth{#1}}
1237 \define@key{pcsendmessage}{style}[]{\renewcommand*\@pcsendmessagestyle{#1}}
1238 \define@key{pcsendmessage}{length}[]{\renewcommand*\@pcsendmessagelength{#1}}
1239 \define@key{pcsendmessage}{top}[]{\renewcommand*\@pcsendmessagetop{#1}}
1240 \define@key{pcsendmessage}{bottom}[]{\renewcommand*\@pcsendmessagebottom{#1}}
1241 \define@key{pcsendmessage}{right}[]{\renewcommand*\@pcsendmessageright{#1}}
1242 \define@key{pcsendmessage}{left}[]{\renewcommand*\@pcsendmessageleft{#1}}
1243 \define@key{pcsendmessage}{topname}[]{\renewcommand*\@pcsendmessagetopname{#1}}
1244 \define@key{pcsendmessage}{bottomname}[]{\renewcommand*\@pcsendmessagebottomname{#1}}
1245 \define@key{pcsendmessage}{rightname}[]{\renewcommand*\@pcsendmessagerightname{#1}}
1246 \define@key{pcsendmessage}{leftname}[]{\renewcommand*\@pcsendmessageleftname{#1}}
1247 \define@key{pcsendmessage}{topstyle}[]{\renewcommand*\@pcsendmessagetopstyle{#1}}
1248 \define@key{pcsendmessage}{bottomstyle}[]{\renewcommand*\@pcsendmessagebottomstyle{#1}}
1249 \define@key{pcsendmessage}{rightstyle}[]{\renewcommand*\@pcsendmessagerightstyle{#1}}
1250 \define@key{pcsendmessage}{leftstyle}[]{\renewcommand*\@pcsendmessageleftstyle{#1}}
1251 \define@key{pcsendmessage}{beforeskip}[]{\renewcommand*\@pcsendmessagebeforeskip{#1}}
1252 \define@key{pcsendmessage}{afterskip}[]{\renewcommand*\@pcsendmessageafterskip{#1}}
1253
1254 \newcommand*\@pcsendmessagealignedtop{false}
1255 \define@key{pcsendmessage}{topaligned}[true]{\renewcommand*\@pcsendmessagealignedtop{#1}}
1256 \newcommand*\@pcsendmessagealignedbottom{false}
1257 \define@key{pcsendmessage}{bottomaligned}[true]{\renewcommand*\@pcsendmessagealignedbottom{#1}}
1258 \newcommand*\@pcsendmessagealignedleft{false}
1259 \define@key{pcsendmessage}{leftaligned}[true]{\renewcommand*\@pcsendmessagealignedleft{#1}}
1260 \newcommand*\@pcsendmessagealignedright{false}
1261 \define@key{pcsendmessage}{rightaligned}[true]{\renewcommand*\@pcsendmessagealignedright{#1}}
1262
1263
1264 \newcommand{\@pc@centerincol}[2]{%
1265 \ifmeasuring@%
1266 #2%
1267 \else%
1268 \makebox[\ifcase\expandafter #1\maxcolumn@widths\fi]{$\displaystyle#2$}%
1269 \fi%
1270 }
1271
1272 \newcommand{\centerincol}[1]{\@pc@centerincol{\thepccolumncounter}{#1}}
1273
1274 \newcommand{\@do@sendmessage}[1]{%
1275 \ifthenelse{\equal{\@pcsendmessagecol}{}}{%
1276 \ifthenelse{\equal{\@pcsendmessagewidth}{}}{#1}{% we have some width
1277 \makebox[\@pcsendmessagewidth]{$\displaystyle#1$}%
1278 }}{%we know the column to center on
1279 \@pc@centerincol{\@pcsendmessagecol}{#1}%
1280 }%
1281 }
1282
1283 \newcommand*{\sendmessage}[2]{%
1284 \begingroup\setkeys{pcsendmessage}{#2}%
1285 \tikzset{PCSENDMSG-PATH-STYLE/.style/.expand once=\@pcsendmessagestyle}%
1286 \tikzset{PCSENDMSG-TOP-STYLE/.style/.expand once=\@pcsendmessagetopstyle}%
1287 \tikzset{PCSENDMSG-BOTTOM-STYLE/.style/.expand once=\@pcsendmessagebottomstyle}%
1288 \tikzset{PCSENDMSG-LEFT-STYLE/.style/.expand once=\@pcsendmessageleftstyle}%
1289 \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$\begin{aligned}\@pcsendmessagetop\end{alig
{\let\@pc@fin@sendmessagetop\@pcsendmessagetop}%
%
\ifthenelse{\equal{\@pcsendmessagealignedbottom}{true}}
{\ifthenelse{\equal{\@pcsendmessagebottom}{}}
{\let\@pc@fin@sendmessagebottom\@pcsendmessagebottom}%
{\newcommand{\@pc@fin@sendmessagebottom}{\let\halign\@pc@halign$\begin{aligned}\@pcsendmessagebottom\en
{\let\@pc@fin@sendmessagebottom\@pcsendmessagebottom}%
%
\ifthenelse{\equal{\@pcsendmessagealignedright}{true}}
{\ifthenelse{\equal{\@pcsendmessageright}{}}
{\let\@pc@fin@sendmessageright\@pcsendmessageright}
{\newcommand{\@pc@finsendmessageright}{\let\halign\@pc@halign$\begin{aligned}\@pcsendmessageright\end{a
{\let\@pc@fin@sendmessageright\@pcsendmessageright}%
%
\ifthenelse{\equal{\@pcsendmessagealignedleft}{true}}
{\ifthenelse{\equal{\@pcsendmessageleft}{}}
{\let\@pc@fin@sendmessageleft\@pcsendmessageleft}
{\newcommand{\@pc@fin@sendmessageleft}{\let\halign\@pc@halign$\begin{aligned}\@pcsendmessageleft\end{al
{\let\@pc@fin@sendmessageleft\@pcsendmessageleft}%
%restore halign
%
\addtocounter{@pcsubprogstep}{1}%
\hspace{\@pcsendmessagebeforeskip}%
\begin{varwidth}{\linewidth}
\@do@sendmessage{
\begin{tikzpicture}%
 \node[PCSENDMSG-LEFT-STYLE] (\@pcsendmessageleftname) {\@pc@fin@sendmessageleft};
 \node[right=\@pcsendmessagelength of \@pcsendmessageleftname,PCSENDMSG-RIGHT-STYLE] (\@pcsendmessageri
 \path[#1,PCSENDMSG-PATH-STYLE] (\@pcsendmessageleftname) edge[] node[above,PCSENDMSG-TOP-STYLE] (\@pcs
\end{tikzpicture}%
}%
\end{varwidth}
\addtocounter{@pcsubprogstep}{-1}%
\hspace{\@pcsendmessageafterskip}%
\endgroup%
}

\WithSuffix\newcommand\sendmessage*[2]{%
\sendmessage{#1}{topaligned,leftaligned,bottomaligned,rightaligned,#2}%
}

\newcommandx*{\sendmessageright}[2][1=->]{%
\sendmessage{#1}{#2}%
}

\newcommandx*{\sendmessageleft}[2][1=<-]{%
\sendmessage{#1}{#2}%
}

\WithSuffix\newcommand\sendmessageleft*[2][\pcdefaultmessagelength]{%
```

```
\begingroup%
\renewcommand{\@pcsendmessagetop}{\let\halign\@pc@halign$\begin{aligned}#2\end{aligned}$}%
\sendmessage{<-}{length=#1}%
\endgroup%
}


\WithSuffix\newcommand\sendmessageright*[2][\pcdefaultmessagelength]{%
\begingroup%
\renewcommand{\@pcsendmessagetop}{\let\halign\@pc@halign$\begin{aligned}#2\end{aligned}$}%
\sendmessage{->}{length=#1}%
\endgroup%
}

\WithSuffix\newcommand\sendmessagerightleft*[2][\pcdefaultmessagelength]{%
\begingroup%
\renewcommand{\@pcsendmessagetop}{\let\halign\@pc@halign$\begin{aligned}#2\end{aligned}$}%
\sendmessage{<->}{length=#1}%
\endgroup%
}

\DeclareExpandableDocumentCommand{\sendmessagerightx}{O{\pcdefaultlongmessagelength}mO{}m}{%
\multicolumn{#2}{c}{\ensuremath{\hspace{\pcbeforemessageskip}\xrightarrow[\begin{aligned}#3\end{aligne
}

\DeclareExpandableDocumentCommand{\sendmessageleftx}{O{\pcdefaultlongmessagelength}mO{}m}{%
\multicolumn{#2}{c}{\ensuremath{\hspace{\pcbeforemessageskip}\xleftarrow[\begin{aligned}#3\end{aligned
}

%
% Division
\DeclareExpandableDocumentCommand{\pcintertext}{O{}m}{\intertext{%
\ifthenelse{\equal{#1}{center}}{\makebox[\linewidth][c]{#2}}{}%
\ifthenelse{\equal{#1}{dotted}}{\dotfill#2\dotfill}{}%
\ifthenelse{\equal{#1}{}}{#2}{}%
}\@pc@beginnewline}
```

9.10 Tikz within Pseudocode

```
%
% remember pictues
\newcounter{@pc@remember}

\newcommand{\@pc@ensureremember}{%
\ifthenelse{\value{@pc@remember}=0}{\tikzstyle{every picture}+=[remember picture]}{}%
\addtocounter{@pc@remember}{1}}

\newcommand{\@pc@releaseremember}{%
\addtocounter{@pc@remember}{-1}%
\ifthenelse{\value{@pc@remember}=0}{\tikzstyle{every picture}-=[remember picture]}{}%
}
```

```
%
% pcimage
\newenvironment{pcimage}{%
\begingroup\@pc@ensureremember%
}{%
\@pc@releaseremember\endgroup%
}

\newcommand*\@pcnodecontent{}
\newcommand*\@pcnodestyle{}
\newcommand*\@pcnodedraw{}
\define@key{pcnode}{content}[]{\renewcommand*\@pcnodecontent{#1}}
\define@key{pcnode}{style}[]{\renewcommand*\@pcnodestyle{#1}}
\define@key{pcnode}{draw}[]{\renewcommand*\@pcnodedraw{#1}}

\newcommandx*{\pcnode}[2][2=]{%
\begingroup\setkeys{pcnode}{#2}%
\tikzset{PCNODE-STYLE/.style/.expand once=\@pcnodestyle}%
\begin{tikzpicture}[inner sep=0ex,baseline=0pt]%
\node[PCNODE-STYLE] (#1) {\@pcnodecontent}; %
\end{tikzpicture}%
\ifdefempty{\@pcnodedraw}{}{%
\begin{tikzpicture}[overlay,inner sep=0ex,baseline=0pt]\@pcnodedraw\end{tikzpicture}
}%
\endgroup}

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

```

9.11 Black Box Reductions

```

%
% 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-}
```

aboveskip
\@pc@bbrenvaboveskip
```
\newcommand*\@pc@bbrenvaboveskip{0pt}
\define@key{pcbbrenv}{aboveskip}[0pt]{\renewcommand*\@pc@bbrenvaboveskip{#1}}
```

belowskip
\@pc@bbrenvbelowskip
```
\newcommand*\@pc@bbrenvbelowskip{0pt}
\define@key{pcbbrenv}{belowskip}[0pt]{\renewcommand*\@pc@bbrenvbelowskip{#1}}
```

@bbrenv@legacyargcheck ensures that first command can still be 5cm which is rewritten as aboveskip=5cm

\@pc@bbrenv@argstring
```
\newcommand*\@pc@bbrenv@argstring{}
\def\@pc@bbrenv@remfinalequals#1=#2=\relax{\renewcommand*\@pc@bbrenv@argstring{#1=#2}}
\def\@pc@bbrenv@legacyargcheck#1=#2\relax{%
\ifthenelse{\equal{#2}{}}
{\PackageWarning{cryptocode}{Deprecated option for bbrenv. Please use key value list as first parameter
\renewcommand*\@pc@bbrenv@argstring{aboveskip=#1}}
{\@pc@bbrenv@remfinalequals#1=#2\relax}%
}
```

\bbrfirstmessageoffset offset of the first message from top
```
\providecommand{\bbrfirstmessageoffset}{1ex}
```

tikzargs Allow passing in arguments to `tikzpicture`.

\bbrtikzargs
```
\newcommand*\bbrtikzargs{}
\define@key{pcbbrenv}{tikzargs}[]{\renewcommand*\bbrtikzargs{#1}}
```

bbrenv Black Box Reduction Environment
```
\newenvironmentx{bbrenv}[3][1={aboveskip=0pt,belowskip=0pt},3=0pt]{%
\addtocounter{@bb@env@nestcnt}{1}%
\renewcommand{\@bb@lastbox}{#2}%
% parse args and allow old style #1=0pt
\@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}[baseline=0pt,\bbrtikzargs]
}{\tikz\bgroup}
}{%
\ifthenelse{\value{@bb@env@nestcnt}=1}
{\end{tikzpicture}%
\@pc@releaseremember%
}{\egroup}%
```

```
\vspace{\@pc@bbrenvbelowskip}%
\addtocounter{@bb@env@nestcnt}{-1}%
% 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}%
}
```

black box reduction box option keys

```
\newcommand*\bbrboxname{}
\newcommand*\bbrboxnamepos{right}
\newcommand*\bbrboxnamestyle{}
\newcommand*\@bbrboxnamepos{below right=0.5ex and -0.5ex of \@bb@lastbox.north east,anchor=north east}
\newcommand*\bbrboxabovesep{\baselineskip}
\newcommand*\@bbrboxnameposoffset{below left=\bbrboxabovesep of phantomname.south west}
\newcommand*\bbrboxstyle{draw}
\newcommand*\bbrboxafterskip{}
\newcommand*\bbrboxminheight{0pt}
\newcommand*\bbrboxminwidth{2cm}
\newcommand*\bbrboxxshift{0pt}
\newcommand*\bbrboxyshift{0pt}
\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}}
\define@key{bbrbox}{style}[draw]{\renewcommand*\bbrboxstyle{#1}}
\define@key{bbrbox}{minwidth}[]{\renewcommand*\bbrboxminwidth{#1}}
\define@key{bbrbox}{addheight}[]{\renewcommand*\bbrboxafterskip{#1}}
\define@key{bbrbox}{minheight}[]{\renewcommand*\bbrboxminheight{#1}}
\define@key{bbrbox}{xshift}[]{\renewcommand*\bbrboxxshift{#1}}
\define@key{bbrbox}{yshift}[]{\renewcommand*\bbrboxyshift{#1}}


\NewEnviron{bbrbox}[1][]{%
\setkeys{bbrbox}{#1}%

\ifthenelse{\equal{\bbrboxnamepos}{center}}
 {\renewcommand{\@bbrboxnamepos}{below=0.5ex of \@bb@lastbox.north,anchor=north}}{}
\ifthenelse{\equal{\bbrboxnamepos}{left}}
 {\renewcommand{\@bbrboxnamepos}{below=0.5ex of \@bb@lastbox.north west,anchor=north west}}{}
\ifthenelse{\equal{\bbrboxnamepos}{top right}}
 {\renewcommand{\@bbrboxnamepos}{above=0cm of \@bb@lastbox.north east,anchor=south east}\renewcommand{\
\ifthenelse{\equal{\bbrboxnamepos}{top center}}
 {\renewcommand{\@bbrboxnamepos}{above=0cm of \@bb@lastbox.north,anchor=south}\renewcommand{\@bbrboxnar
\ifthenelse{\equal{\bbrboxnamepos}{top left}}
 {\renewcommand{\@bbrboxnamepos}{above=0cm of \@bb@lastbox.north west,anchor=south west}\renewcommand{\
\ifthenelse{\equal{\bbrboxnamepos}{middle}}
 {\renewcommand{\@bbrboxnamepos}{above=0.5ex of \@bb@lastbox.base,anchor=south}}{}
\ifthenelse{\equal{\bbrboxnamepos}{bottom}}
 {\renewcommand{\@bbrboxnamepos}{above=0.5ex of \@bb@lastbox.base,anchor=north}}{}


\tikzset{BBRBOXSTYLE/.style/.expand once=\bbrboxstyle}%
\tikzset{BBRBOXNAMEPOS/.style/.expand once=\@bbrboxnamepos}%
\tikzset{BBRBOXNAMESTYLE/.style/.expand once=\bbrboxnamestyle}%
```

```
\tikzset{BBRBOXNAMEPOSOFFSET/.style/.expand once=\@bbrboxnameposoffset}%

\ifthenelse{\equal{\bbrboxxshift}{} \OR \equal{\bbrboxxshift}{0pt}}{
\coordinate[inner sep=0pt,outer sep=0pt] (\@bb@lastbox-tmpouter) {};
}{
\node[inner sep=0pt, outer sep=0pt] (\@bb@lastbox-tmpouter) {}; %this empty node seems needed to get th
}

\node[inner sep=.3333em,anchor=north,BBRBOXSTYLE,minimum height=\bbrboxminheight,below right=\bbrboxysh
\tikz{
\node[inner sep=0pt,outer sep=0pt,minimum height=0cm] (phantomname) {}; %minimum width
\node[BBRBOXNAMEPOSOFFSET,minimum height=0cm] (\@bb@lastbox-inner) {\begin{varwidth}{2\linewidth}\BODY\
\ifthenelse{\equal{\bbrboxafterskip}{}}{}{
\node[below=0cm of \@bb@lastbox-inner,minimum height=\bbrboxafterskip] {};
}
\node[inner sep=0pt,outer sep=0pt,at=(\@bb@lastbox-inner.south west),minimum height=0cm] () {\phantom{\
}
\egroup;
\ifthenelse{\equal{\bbrboxnamepos}{none}}
{}{\node[BBRBOXNAMEPOS,BBRBOXNAMESTYLE, inner sep=0.2ex, outer sep=0pt, overlay] () {\bbrboxname};}
}


\newcommand*\bbroraclevdistance{\baselineskip}
\newcommand*\bbroraclehdistance{1.5cm}
\define@key{bbroracle}{distance}[]{\renewcommand*\bbroraclehdistance{#1}}
\define@key{bbroracle}{hdistance}[]{\renewcommand*\bbroraclehdistance{#1}}
\define@key{bbroracle}{vdistance}[]{\renewcommand*\bbroraclevdistance{#1}}


% ORACLES
\newenvironmentx{bbroracle}[2][2=]{%
\begingroup
\setkeys{bbroracle}{#2}
%reset query boolean. This is a bit crude and does not allow nesting oracles
%in oracles but should be good enough
\gdef\@bbr@first@oraclequery{true}
%add to nesting cout
\addtocounter{@bb@oracle@nestcnt}{1}
%if first oracle, then put it to the right, else stack them vertically
\addtocounter{@bb@oracle@cnt}{1}
\ifthenelse{\value{@bb@oracle@cnt}=1}{
\setlength{\@bb@tmplength@b}{\bbroraclevdistance-\baselineskip}
\node[inner sep=0pt,below right=\@bb@tmplength@b and \bbroraclehdistance of \@bb@lastbox.north east,anc
}{
% compute distance of top of last box to bottom of last oracle
\coordinate (@bbtmpcoord) at (\@bb@lastbox.north east);
\path (@bbtmpcoord);
\pgfgetlastxy{\XCoord}{\YCoordA}
\coordinate (@bbtmpcoord) at (\bbroraclenodenameprefix \@bb@lastoracle.south west);
\path (@bbtmpcoord);
\pgfgetlastxy{\XCoord}{\YCoordB}
\setlength{\@bb@tmplength@b}{\YCoordA-\YCoordB+\bbroraclevdistance}
\node[inner sep=0pt,below right=\@bb@tmplength@b and \bbroraclehdistance of \@bb@lastbox.north east,anc
}
\global\def\@bb@lastoracle{#1}
```

```
\begin{bbrenv}{#1}
}{
\end{bbrenv}
\egroup;

\addtocounter{@bb@oracle@nestcnt}{-1}
\endgroup
}


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


% Challenger
\newenvironmentx{bbrchallenger}[2][2=]{%
\begingroup%
\setkeys{bbrchallenger}{#2}%
%reset query boolean. This is a bit crude and does not allow nesting oracles
%in oracles but should be good enough
\gdef\@bbr@first@challengerquery{true}%
%add to nesting cout
\addtocounter{@bb@challenger@nestcnt}{1}%
%if first oracle, then put it to the right, else stack them vertically
\addtocounter{@bb@challenger@cnt}{1}%
\ifthenelse{\value{@bb@challenger@cnt}=1}{%
\setlength{\@bb@tmplength@b}{\bbrchallengervdistance-\baselineskip}%
\node[inner sep=0pt,outer sep=0pt,below left=\@bb@tmplength@b and \bbrchallengerhdistance of \@bb@lastb
}{%
\coordinate (@bbtmpcoord) at (\@bb@lastbox.north west);%
\path (@bbtmpcoord);%
\pgfgetlastxy{\XCoord}{\YCoordA}%
\coordinate (@bbtmpcoord) at (\bbrchallengernodenameprefix \@bb@lastchallenger.south east);%
\path (@bbtmpcoord);%
\pgfgetlastxy{\XCoord}{\YCoordB}%
\setlength{\@bb@tmplength@b}{\YCoordA-\YCoordB+\bbrchallengervdistance}%
\node[inner sep=0pt,below left=\@bb@tmplength@b and \bbrchallengerhdistance of \@bb@lastbox.north west
}%
\global\def\@bb@lastchallenger{#1}
\begin{bbrenv}{#1}%
}{
\end{bbrenv}%
\egroup;%
\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{\@bb@inputsetup}[1]{
%load keys
\begingroup % for local keys

\setkeys{bbrinput}{#1}%

\tikzset{BBRINPUT-NODESTYLE/.style/.expand once=\bbrinputedgestyle}%
\tikzset{BBRINPUT-TOPSTYLE/.style/.expand once=\bbrinputtopstyle}%
\tikzset{BBRINPUT-BOTTOMSTYLE/.style/.expand once=\bbrinputbottomstyle}%
\tikzset{BBRINPUT-EDGESTYLE/.style/.expand once=\bbrinputedgestyle}%

}

\newcommand{\@bb@inputfinalize}{
\endgroup
}

\newcommandx*{\bbrinput}[2][2=]{%
\@bb@inputsetup{#2}
\ifthenelse{\equal{\bbrinputnodename}{}}
  {\renewcommand{\bbrinputnodename}{\@bb@lastbox-input}}{}

\node[overlay,above right={\bbrinputlength} and {\bbrinputhoffset} of \@bb@lastbox.north west, anchor=s
\path[->] (\bbrinputnodename.south)  edge[BBRINPUT-EDGESTYLE] node[above,anchor=east,BBRINPUT-TOPSTYLE]
\@bb@inputfinalize
}

\newcommandx*{\bbroutput}[2][2=]{%
\@bb@inputsetup{#2}
\ifthenelse{\equal{\bbrinputnodename}{}}
  {\renewcommand{\bbrinputnodename}{\@bb@lastbox-output}}{}

\node[overlay,below right={\bbrinputlength} and {\bbrinputhoffset} of \@bb@lastbox.south west, anchor=n
\draw[->] (\bbrinputnodename.north|-\@bb@lastbox.south)  -- (\bbrinputnodename.north|-\bbrinputnodename
\@bb@inputfinalize
}

\newenvironment{bbrpic}[1][]{%
\begin{tikzpicture}[overlay,inner sep=0ex,baseline=0pt,#1]%
}{%
\end{tikzpicture}}

%
% communication
```

```
%temporary lengths
\newlength{\@bb@com@tmpoffset}
\newlength{\@bb@tmplength@b}

%keys
\newcommand*\@bbrcomsidestyle{}
\newcommand*\@bbrcomosidestyle{}
\newcommand*\@bbrcomtopstyle{}
\newcommand*\@bbrcombottomstyle{}
\newcommand*\@bbrcomside{}
\newcommand*\@bbrcomoside{}
\newcommand*\@bbrcomtop{}
\newcommand*\@bbrcombottom{}
\newcommand*\@bbrcomedgestyle{}
\newcommand*\@bbrcomlength{1.25cm}
\newcommand*\@bbrcomtopname{bbrcomtop}
\newcommand*\@bbrcombottomname{bbrcombottom}
\newcommand*\@bbrcomsidename{bbrcomside}
\newcommand*\@bbrcomosidename{bbrcomoside}
\newcommand*\@bbrcombeforeskip{0pt}
\newcommand*\@bbrcomafterskip{0ex}
\define@key{bbrcom}{sidestyle}[]{\renewcommand*\@bbrcomsidestyle{#1}}
\define@key{bbrcom}{osidestyle}[]{\renewcommand*\@bbrcomosidestyle{#1}}
\define@key{bbrcom}{topstyle}[]{\renewcommand*\@bbrcomtopstyle{#1}}
\define@key{bbrcom}{bottomstyle}[]{\renewcommand*\@bbrcombottomstyle{#1}}
\define@key{bbrcom}{side}[]{\renewcommand*\@bbrcomside{#1}}
\define@key{bbrcom}{oside}[]{\renewcommand*\@bbrcomoside{#1}}
\define@key{bbrcom}{top}[]{\renewcommand*\@bbrcomtop{#1}}
\define@key{bbrcom}{bottom}[]{\renewcommand*\@bbrcombottom{#1}}
\define@key{bbrcom}{edgestyle}[]{\renewcommand*\@bbrcomedgestyle{#1}}
\define@key{bbrcom}{length}[]{\renewcommand*\@bbrcomlength{#1}}
\define@key{bbrcom}{topname}[]{\renewcommand*\@bbrcomtopname{#1}}
\define@key{bbrcom}{bottomname}[]{\renewcommand*\@bbrcombottomname{#1}}
\define@key{bbrcom}{sidename}[]{\renewcommand*\@bbrcomsidename{#1}}
\define@key{bbrcom}{osidename}[]{\renewcommand*\@bbrcomosidename{#1}}
\define@key{bbrcom}{beforeskip}[]{\renewcommand*\@bbrcombeforeskip{#1}}
\define@key{bbrcom}{aboveskip}[]{\renewcommand*\@bbrcombeforeskip{#1}}
\define@key{bbrcom}{afterskip}[]{\renewcommand*\@bbrcomafterskip{#1}}
\define@key{bbrcom}{belowskip}[]{\renewcommand*\@bbrcomafterskip{#1}}
```

\@bbrcomfixedoffset
\@bbrcomfixedboffset
fixedoffset
fixedboffset

Provide means for fixed message offset from top or bottom

```
\newcommand*\@bbrcomfixedoffset{}
\newcommand*\@bbrcomfixedboffset{false}
\define@key{bbrcom}{fixedoffset}[]{\renewcommand*\@bbrcomfixedoffset{#1}}
\define@key{bbrcom}{fixedboffset}[]{\renewcommand*\@bbrcomfixedoffset{#1}\renewcommand*\@bbrcomfixedbo

%
%
\newcommand*\@bbrbasenodestyle{}
\newcommand*\@bbrbasenodename{bbrtmpname}
\define@key{bbrabase}{nodestyle}[]{\renewcommand*\@bbrbasenodestyle{#1}}
\define@key{bbrabase}{nodename}[]{\renewcommand*\@bbrbasenodename{#1}}

\newcommand*\@bbr@first@msg{true}
\newcommand*\@bbr@first@query{true}
\newcommand*\@bbr@first@oraclequery{true}
```

```
\newcommand*\@bbr@first@challengerquery{true}

```

@bbr@intermessage@skip
r@intermessage@medskip
intermessage@shortskip
rmessage@veryshortskip

Skip between two messages.

```
\newcommand*\@bbr@intermessage@skip{4ex}
\newcommand*\@bbr@intermessage@veryshortskip{1ex}
\newcommand*\@bbr@intermessage@shortskip{1.5ex}
\newcommand*\@bbr@intermessage@medskip{2.5ex}
```

islast
\@bbrcomislast

Sets the message from the bottom of the box with the same distance as the first message.

```
\newcommand*\@bbrcomislast{false}
\define@key{bbrcom}{islast}[true]{\renewcommand*\@bbrcomislast{#1}}

\newcommand*\@bbrcom@check@islast{%
\ifthenelse{\equal{\@bbrcomislast}{true}}
{\renewcommand*\@bbrcomfixedoffset{\bbrfirstmessageoffset}\renewcommand*\@bbrcomfixedboffset{true}}
{}
}
```

\@bbr@lastskip

marker to set whether next skip is a short or a long one

```
\def\@bbr@lastskip{0pt}
```

\@bb@comsetup

Sets up communication parameters for message/query commands. Parameters are {⟨*key value list*⟩}, {⟨*length*⟩}, {⟨*command for adding space*⟩} {⟨*true if first message*⟩}

```
\newcommand{\@bb@comsetup}[4]{
% check if is first message and mark as false
\edef\@tmp@bbr@isfirst{#4}
\renewcommand#4{false}

%load keys
\begingroup % for local keys

\setkeys{bbrcom}{#1}%

%set styles
\tikzset{BBRCOM-SIDESTYLE/.style/.expand once=\@bbrcomsidestyle}%
\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{}

% increase space
#3{\@bbrcombeforeskip}
\ifthenelse{\equal{\@bbrcomfixedoffset}{}}
{
\ifthenelse{\equal{\@tmp@bbr@isfirst}{true}}
{}{#3{\@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 9 -> true if first message 10 -> anchor from bottom

```
\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}}{}

\ifdefempty{\@bbrcomside}{
\coordinate[#3=-\@bb@com@tmpoffset and \@bbrcomlength of \@bb@lastbox.\@bbr@tmp@bottomanchor] (\@bbrcor
}{
\node[#3=-\@bb@com@tmpoffset and \@bbrcomlength of \@bb@lastbox.\@bbr@tmp@bottomanchor,anchor=#6,BBRCO
}
}
{
% from top
\ifdefempty{\@bbrcomside}{
\coordinate[#3=\@bb@com@tmpoffset and \@bbrcomlength of \@bb@lastbox.#4] (\@bbrcomsidename);
}{
\node[#3=\@bb@com@tmpoffset and \@bbrcomlength of \@bb@lastbox.#4,anchor=#6,BBRCOM-SIDESTYLE] (\@bbrcor
}
}
\path[#2] (\@bbrcomsidename.#6) edge[BBRCOM-EDGESTYLE] node[above,BBRCOM-TOPSTYLE] (\@bbrcomtopname) {
%
\@bb@comfinalize{#8}
}
```

\bbrmsgto
\bbrmsgfrom
\bbrmsgtofrom
\bbrmsgfromto

```
\newcommandx{\bbrmsgto}[1]{%
\@bbrmsg{#1}{->}{below left}{north west}{west}{east}{\@bb@message@voffset}{\bbrmsgspace}{\@bbr@first@m
}
\newcommandx{\bbrmsgfrom}[1]{%
\@bbrmsg{#1}{<-}{below left}{north west}{west}{east}{\@bb@message@voffset}{\bbrmsgspace}{\@bbr@first@m
}

\newcommandx{\bbrmsgtofrom}[2]{%
\bbrmsgto{#1}
\bbrmsgspace{-\@bbr@intermessage@skip}
\bbrmsgspace{\@bbr@intermessage@shortskip}
\bbrmsgfrom{#2}
\bbrmsgspace{\@bbr@intermessage@medskip}
}

\newcommandx{\bbrmsgfromto}[2]{%
\bbrmsgfrom{#1}
\bbrmsgspace{-\@bbr@intermessage@skip}
\bbrmsgspace{\@bbr@intermessage@shortskip}
```

```
\bbrmsgto{#2}
\bbrmsgspace{\@bbr@intermessage@medskip}
}

\bbrmsgvdots
\newcommand{\bbrmsgvdots}[1][]{%
\bbrmsgtxt[xshift=\@bbrcomlength/2,afterskip=\@bbr@intermessage@shortskip,#1]{$\vdots$}
}

\bbrqryto
\bbrqryfrom
\bbrqrytofrom
\bbrqryfromto
\newcommandx{\bbrqryto}[1]{%
\@bbrmsg{#1}{<-}{below right}{north east}{east}{west}{\@bb@query@voffset}{\bbrqryspace}{\@bbr@first@que
}
\newcommandx{\bbrqryfrom}[1]{%
\@bbrmsg{#1}{->}{below right}{north east}{east}{west}{\@bb@query@voffset}{\bbrqryspace}{\@bbr@first@que
}

\newcommand*{\bbrqrytofrom}[2]{%
\bbrqryto{#1}
\bbrqryspace{-\@bbr@intermessage@skip}
\bbrqryspace{\@bbr@intermessage@shortskip}
\bbrqryfrom{#2}
\bbrqryspace{\@bbr@intermessage@medskip}
}

\newcommand*{\bbrqryfromto}[2]{%
\bbrqryfrom{#1}
\bbrqryspace{-\@bbr@intermessage@skip}
\bbrqryspace{\@bbr@intermessage@shortskip}
\bbrqryto{#2}
\bbrqryspace{\@bbr@intermessage@medskip}
}

\bbrqryvdots
\newcommand{\bbrqryvdots}[1][]{%
\bbrqrytxt[xshift=\@bbrcomlength/2,afterskip=\@bbr@intermessage@skip,#1]{$\vdots$}
}

\@bbroracleqry
\newcommand{\@bbroracleqry}[4]{
\@bb@comsetup{#1}{#3}{#4}{\@bbr@first@oraclequery}
%
\ifthenelse{\equal{\@bbrcomfixedboffset}{true}}
{
% from bottom
\path[#2] (\@bb@lastoracle.south west) -- ++ (0,\@bb@com@tmpoffset) node[inner sep=0pt,outer sep=0pt,an
}
{
\path[#2] (\@bb@lastoracle.north west) -- ++ (0,-\@bb@com@tmpoffset) node[inner sep=0pt,outer sep=0pt,a
}
%
\@bb@comfinalize{#4}
}

\bbroracleqryto
\bbroracleqryfrom
\bbroracleqrytofrom
\bbroracleqryfromto
```

```
\newcommand{\bbroracleqryfrom}[1]{
\@bbroracleqry{#1}{->}{\@bb@oraclequery@voffset}{\bbroracleqryspace}
}

\newcommand{\bbroracleqryto}[1]{
\@bbroracleqry{#1}{<-}{\@bb@oraclequery@voffset}{\bbroracleqryspace}
}

\newcommand*{\bbroracleqrytofrom}[2]{%
\bbroracleqryto{#1}
\bbroracleqryspace{-\@bbr@intermessage@skip}
\bbroracleqryspace{\@bbr@intermessage@shortskip}
\bbroracleqryfrom{#2}
\bbroracleqryspace{\@bbr@intermessage@medskip}
}

\newcommand*{\bbroracleqryfromto}[2]{%
\bbroracleqryfrom{#1}
\bbroracleqryspace{-\@bbr@intermessage@skip}
\bbroracleqryspace{\@bbr@intermessage@shortskip}
\bbroracleqryto{#2}
\bbroracleqryspace{\@bbr@intermessage@medskip}
}
```

\@bbrchallengerqry

```
\newcommand{\@bbrchallengerqry}[4]{
\@bb@comsetup{#1}{#3}{#4}{\@bbr@first@challengerquery}
%
\ifthenelse{\equal{\@bbrcomfixedboffset}{true}}
{
\path[#2] (\@bb@lastchallenger.south east) -- ++ (0,\@bb@com@tmpoffset) node[inner sep=0pt,outer sep=0p
}
{
\path[#2] (\@bb@lastchallenger.north east) -- ++ (0,-\@bb@com@tmpoffset) node[inner sep=0pt,outer sep=0
}
%
\@bb@comfinalize{#4}
}
```

\bbroracleqryto
\bbroracleqryfrom
\bbroracleqrytofrom
\bbroracleqryfromto

```
\newcommand{\bbrchallengerqryfrom}[1]{
\@bbrchallengerqry{#1}{<-}{\@bb@challengerquery@voffset}{\bbrchallengerqryspace}
}

\newcommand{\bbrchallengerqryto}[1]{
\@bbrchallengerqry{#1}{->}{\@bb@challengerquery@voffset}{\bbrchallengerqryspace}
}

\newcommand*{\bbrchallengerqrytofrom}[2]{%
\bbrchallengerqryto{#1}
\bbrchallengerqryspace{-\@bbr@intermessage@skip}
\bbrchallengerqryspace{\@bbr@intermessage@shortskip}
\bbrchallengerqryfrom{#2}
\bbrchallengerqryspace{\@bbr@intermessage@medskip}
}
```

```
\newcommand*{\bbrchallengerqryfromto}[2]{%
\bbrchallengerqryfrom{#1}
\bbrchallengerqryspace{-\@bbr@intermessage@skip}
\bbrchallengerqryspace{\@bbr@intermessage@shortskip}
\bbrchallengerqryto{#2}
\bbrchallengerqryspace{\@bbr@intermessage@medskip}
}


\newcommand*\bbrcomloopleft{}
\newcommand*\bbrcomloopleftstyle{}
\newcommand*\bbrcomloopright{}
\newcommand*\bbrcomlooprightstyle{}
\newcommand*\bbrcomloopcenter{}
\newcommand*\bbrcomloopcenterstyle{}
\newcommand*\bbrcomloopclockwise{false}
\newcommand*\bbrcomloopangle{50}
\define@key{bbrcomloop}{left}[]{\renewcommand*\bbrcomloopleft{#1}}
\define@key{bbrcomloop}{leftstyle}[]{\renewcommand*\bbrcomloopleftstyle{#1}}
\define@key{bbrcomloop}{right}[]{\renewcommand*\bbrcomloopright{#1}}
\define@key{bbrcomloop}{rightstyle}[]{\renewcommand*\bbrcomlooprightstyle{#1}}
\define@key{bbrcomloop}{center}[]{\renewcommand*\bbrcomloopcenter{#1}}
\define@key{bbrcomloop}{centerstyle}[]{\renewcommand*\bbrcomloopcenterstyle{#1}}
\define@key{bbrcomloop}{angle}[]{\renewcommand*\bbrcomloopangle{#1}}
\define@key{bbrcomloop}{clockwise}[true]{\renewcommand*\bbrcomloopclockwise{#1}}

\newcommand{\bbrloop}[3]{
\begingroup % for local keys
\setkeys{bbrcomloop}{#3}%

\tikzset{BBRLOOP-LEFTSTYLE/.style/.expand once=\bbrcomloopleftstyle}%
\tikzset{BBRLOOP-RIGHTSTYLE/.style/.expand once=\bbrcomlooprightstyle}%
\tikzset{BBRLOOP-CENTERSTYLE/.style/.expand once=\bbrcomloopcenterstyle}%


\ifthenelse{\equal{\bbrcomloopclockwise}{true}}
{
\path[->] (#1) edge[bend left=\bbrcomloopangle] node[midway,left,inner sep=0,outer sep=0,BBRLOOP-LEFTS
\path[->] (#2) edge[bend left=\bbrcomloopangle] node[midway,right,inner sep=0,outer sep=0,BBRLOOP-RIGH
}
{
\path[->] (#1) edge[bend right=\bbrcomloopangle] node[midway,left,inner sep=0,outer sep=0,] (bbrleft) 
\path[->] (#2) edge[bend right=\bbrcomloopangle] node[midway,right,inner sep=0,outer sep=0,] (bbrright
}
\node[at=($(bbrleft.west)!0.5!(bbrright.east)$),anchor=center,BBRLOOP-CENTERSTYLE]() {\bbrcomloopcente

\endgroup
}

\newcommand*\bbrintertexthoffset{1.5cm}
\define@key{bbrintertext}{xshift}[]{\renewcommand*\bbrintertexthoffset{#1}}

\newcommand{\@bb@intertextsetup}[1]{
%load keys
\begingroup % for local keys
```

```
2022
2023 % fix align environment (e.g. for use of pseudocode)
2024 % ^^A https://tex.stackexchange.com/questions/36954/spurious-space-above-align-environment-at-top-of-pa
2025 %\pretocmd\start@align{%
2026   %\if@minipage\kern-0.5\abovedisplayskip\fi
2027 %}{}{}
2028
2029 \setkeys{bbrcom,bbrabase,bbrintertext}{#1}%
2030 \@bbrcom@check@islast{}
2031
2032 \tikzset{BBRBASE-NODESTYLE/.style/.expand once=\@bbrbasenodestyle}%
2033 }
2034
2035 \newcommand{\@bb@intertextfinalize}[1]{
2036 #1{\@bbrcomafterskip}
2037 \endgroup
2038 \def\@bbr@lastskip{\@bbr@intermessage@veryshortskip}
2039 }
```

`\@bbrintertext` 7 -> whether or not this is the first msg/query

```
2040 \newcommand{\@bbrintertext}[7]{
2041 \edef\@tmp@bbr@isfirst{#7}
2042 \renewcommand#7{false}
2043
2044 \@bb@intertextsetup{#1}
2045
2046 % increase space
2047 #5{\@bbrcombeforeskip}
2048 \ifthenelse{\equal{\@bbrcomfixedoffset}{}}
2049 {
2050 \ifthenelse{\equal{\@tmp@bbr@isfirst}{true}}
2051 {}{#5{\@bbr@intermessage@veryshortskip}}
2052
2053 \setlength{\@bb@com@tmpoffset}{#4}%
2054 }
2055 {
2056 \setlength{\@bb@com@tmpoffset}{\@bbrcomfixedoffset}%
2057 }
2058
2059 %
2060 \ifthenelse{\equal{\@bbrcomfixedboffset}{true}}
2061 {
2062 % from bottom
2063 \ifthenelse{\equal{#3}{north east}}{\def\@bbr@tmp@bottomanchor{south east}}{}
2064 \ifthenelse{\equal{#3}{north west}}{\def\@bbr@tmp@bottomanchor{south west}}{}
2065
2066 \node[#2=-\@bb@com@tmpoffset and \bbrintertexthoffset of \@bb@lastbox.\@bbr@tmp@bottomanchor, inner sep
2067 }
2068 {
2069 \node[#2=\@bb@com@tmpoffset and \bbrintertexthoffset of \@bb@lastbox.#3, inner sep=0, outer sep=0, BBR
2070 }
2071 %
2072 % compute height of node
2073 \coordinate (@bbtmpcoord) at (\@bbrbasenodename.north);
2074 \path (@bbtmpcoord);
2075 \pgfgetlastxy{\XCoord}{\YCoordA}
```

```
\coordinate (@bbtmpcoord) at (\@bbrbasenodename.south);
\path (@bbtmpcoord);
\pgfgetlastxy{\XCoord}{\YCoordB}

% update voffset
\setlength{\@bb@tmplength@b}{\YCoordA-\YCoordB}
#5{\the\@bb@tmplength@b}

\@bb@intertextfinalize{#5}
}

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

\newcommand{\bbrqrytxt}[2][]{
\@bbrintertext{#1}{below right}{north east}{\@bb@query@voffset}{\bbrqryspace}{#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}{\@bb@challengerquery@voffset}{\bbrchallengerqryspace}{#2}{\
\endgroup
}

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

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

\newcommand{\bbrqryspace}[1]{
\@pc@globaladdtolength{\@bb@query@voffset}{#1}
}

\newcommand{\bbroracleqryspace}[1]{
\@pc@globaladdtolength{\@bb@oraclequery@voffset}{#1}
}

\newcommand{\bbrchallengerqryspace}[1]{
\@pc@globaladdtolength{\@bb@challengerquery@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]{%
{\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}{}}
{\ensuremath{\pcgame[\thepcgamecounter]\gameprocedurearg}}
{\ensuremath{\pcgame[#1]\gameprocedurearg}}}

```

\gameprocedurearg

```
\newcommand*{\gameprocedurearg}{\ensuremath{(\secpar)}}

```

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][]{%
\begingroup%
\setkeys{pcgameproof}{#1}%
\@pc@ensureremember%
\setcounter{pcgamecounter}{\@pcgameproofgamenr}%
\setcounter{pcstartgamecounter}{\@pcgameproofgamenr}\stepcounter{pcstartgamecounter}%
}{\@pc@releaseremember\endgroup}

\newcommand{\setgameproceduredefaultstyle}[1]{%
\PackageWarning{cryptocode}{Deprecated command setgameproceduredefaultstyle. Use pcsetargs instead.}%
\pcsetargs{#1}}

\createpseudocodecommand{gameprocedure}
  {\addtocounter{pcgamecounter}{1}\renewcommand{\@withingame}{true}}
  {\@pc@gametitle}
  {}

```

```
\def\@bxgame@pseudocodeA[#1]#2#3{\setkeys*{pcspace}{#1}\renewcommand{\@bxgameheader}{\@pc@gametitle[#2]
\@pseudocode[head=\@pc@gametitle,#1]{#3}}
\def\@bxgame@pseudocodeB#1#2{\renewcommand{\@bxgameheader}{\@pc@gametitle[#1]}%
\@pseudocode[head=\@pc@gametitle]{#2}}

\newcommand{\bxgameprocedure}{
\begingroup%
\renewcommand{\@withinspaces}{false}%
\renewcommand{\@withingame}{true}%
\renewcommand{\@withinbxgame}{true}%
\stepcounter{pcgamecounter}%
\@ifnextchar[%]
  {\@bxgame@pseudocodeA}
  {\@bxgame@pseudocodeB}%
}

\newcommand{\@pc@secondheader}{}

%tbx top boxed
\createpseudocodecommand{tbxgameprocedure}
  {\addtocounter{pcgamecounter}{1}\renewcommand{\@withingame}{true}%
\renewcommand{\@pc@secondheader}{true}}
{\@pc@gametitle}
{}


\newcommand*\@pcgamehopnodestyle{}
\newcommand*\@pcgamehopedgestyle{bend left}
\newcommand*\@pcgamehoppathestyle{}
\newcommand*\@pcgamehophint{}
\newcommand*\@pcgamehophintbelow{}
\newcommand*\@pcgamehopinhint{}
\newcommand*\@pcgamehoplength{1.5cm}
\define@key{pcgamehop}{nodestyle}[]{\renewcommand*\@pcgamehopnodestyle{#1}}
\define@key{pcgamehop}{edgestyle}[]{\renewcommand*\@pcgamehopedgestyle{#1}}
\define@key{pcgamehop}{pathstyle}[]{\renewcommand*\@pcgamehoppathestyle{#1}}
\define@key{pcgamehop}{hint}[]{\renewcommand*\@pcgamehophint{#1}}
\define@key{pcgamehop}{belowhint}[]{\renewcommand*\@pcgamehophintbelow{#1}}
\define@key{pcgamehop}{inhint}[]{\renewcommand*\@pcgamehopinhint{#1}}
\define@key{pcgamehop}{length}[]{\renewcommand*\@pcgamehoplength{#1}}


\newcommand{\@pc@setupgamehop}[1]{
\begingroup\setkeys{pcgamehop}{#1}%
\tikzset{GAMEHOP-PATH-STYLE/.style/.expand once=\@pcgamehoppathestyle}%
\tikzset{GAMEHOP-NODE-STYLE/.style/.expand once=\@pcgamehopnodestyle}%
\tikzset{GAMEHOP-EDGE-STYLE/.style/.expand once=\@pcgamehopedgestyle}%
}

\newcommand{\@pc@finalizegamehop}{
\endgroup
}

\newcommandx*{\addgamehop}[3]{%
\begingroup%
\ifthenelse{#1<#2}%
```

```
  {\ifthenelse{\equal{\@withingamedescription}{true}}%
   {\renewcommand*\@pcgamehopedgestyle{bend right=20}\renewcommand*\@pcgamehopnodestyle{rotate=90}}{}%
  }%
  {\renewcommand*\@pcgamehopedgestyle{bend right}}%
\@pc@setupgamehop{#3}%
\begin{tikzpicture}[overlay]%
\ifthenelse{#1<#2}{%
       \path[->,GAMEHOP-PATH-STYLE] (gamenode#1) edge[GAMEHOP-EDGE-STYLE] node[above,GAMEHOP-NODE-STYLE
       node[below,GAMEHOP-NODE-STYLE] {\@pcgamehophintbelow} (gamenode#2);
}{%
   \path[->,GAMEHOP-PATH-STYLE] (bgamenode#1) edge[GAMEHOP-EDGE-STYLE] node[above,GAMEHOP-NODE-STYLE] 
 node[above,GAMEHOP-NODE-STYLE] {\@pcgamehophintbelow} (bgamenode#2);
}%
\end{tikzpicture}%
\@pc@finalizegamehop%
\endgroup%
}
\newcommandx*{\addstartgamehop}[2][1=\thepcstartgamecounter]{%
\@pc@setupgamehop{#2}
\begin{tikzpicture}[overlay]
       \node[left=\@pcgamehoplength of gamenode#1] (tmpgamenode0) {};
       \path[->,GAMEHOP-PATH-STYLE] (tmpgamenode0) edge[GAMEHOP-EDGE-STYLE] node[above,GAMEHOP-NODE-STY
       node[below,GAMEHOP-NODE-STYLE] {\@pcgamehophintbelow} (gamenode#1);
\end{tikzpicture}
\@pc@finalizegamehop
}
\newcommandx*{\addendgamehop}[2][1=\thepcgamecounter]{%
\@pc@setupgamehop{#2}
\begin{tikzpicture}[overlay]
       \node[right=\@pcgamehoplength of gamenode#1] (tmpgamenode#1) {};
       \path[->,GAMEHOP-PATH-STYLE] (gamenode#1) edge[GAMEHOP-EDGE-STYLE] node[above,GAMEHOP-NODE-STYLE
       node[below,GAMEHOP-NODE-STYLE] {\@pcgamehophintbelow} (tmpgamenode#1);
\end{tikzpicture}
\@pc@finalizegamehop
}
\newcommandx*{\addbxgamehop}[3]{%
\@pc@setupgamehop{#3}
\begin{tikzpicture}[overlay]
       \path[->,GAMEHOP-PATH-STYLE] (bgamenode#1) edge[GAMEHOP-EDGE-STYLE] node[above,GAMEHOP-NODE-STYL
       node[below,GAMEHOP-NODE-STYLE] {\@pcgamehophintbelow} (bgamenode#2);
\end{tikzpicture}
\@pc@finalizegamehop
}
\newcommandx*{\addloopgamehop}[2][1=\thepcgamecounter]{%
\@pc@setupgamehop{#2}
\begin{tikzpicture}[overlay]
       \node (looptemp1)  [right=0.5cm of gamenode#1] {};
       \draw[->,GAMEHOP-PATH-STYLE] (gamenode#1) -- (looptemp1|-gamenode#1) -- node[right,GAMEHOP-NODE
       node[left,GAMEHOP-NODE-STYLE] {\@pcgamehophintbelow} (looptemp1|-bgamenode#1)-- (bgamenode#1);
\end{tikzpicture}
\@pc@finalizegamehop
}
```

9.12.1 Game Descriptions

```
\newenvironment{gamedescription}[1][]{%
\begingroup%
\setkeys{pcgameproof}{#1}
\renewcommand{\@withingamedescription}{true}%
\@pc@ensureremember%
\setcounter{pcgamecounter}{\@pcgameproofgamenr}%
\setcounter{pcstartgamecounter}{\@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) {};
}%
\@pc@gametitle:]%
\begingroup\setkeys{pcgamehop}{#1}%
\ifthenelse{\equal{}{\@pcgamehophint}}
  {}
  {\hspace{-0.7ex}\pcdraw{%the -0.7ex is a horrible hack to fix a whitespace issue with tikz (see http
\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
}}%
\ifthenelse{\equal{}{\@pcgamehopinhint}}
  {}
  {\hspace{-0.7ex}\pcdraw{%the -0.7ex is a horrible hack to fix a whitespace issue with tikz (see http
\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,
}%
}%
\endgroup%
}
%
% \end{macrocode}
%
%
% \iffalse
%    \begin{macrocode}
% \fi
⟨/cryptocode.sty⟩
```

Change History