ASPEN

A suggested security protocol notation

Anders Andersen
UiT The Arctic University of Norway

2026/01/31 08:59:48
(aspen.sty version 1.25, 2026/01/28 22:11:32)

In security literature, different notations for cryptographic values, functions and protocols have been used and suggested. Three often cited references for such notations are "Kerberos: An Authentication Service for Open Network Systems" [12], "Exploring Kerberos, the Protocol for Distributed Security in Windows 2000" [8], and "A Formal Semantics for Protocol Narrations" [5]. The notation ASPEN presented here is strongly inspired by notations found in these three references, notations found in text books [4], and the notation used in my own publications, teaching and presentations. ASPEN is closely related to what is often called "security protocol notation", "standard protocol engineering notation" [2, 3], "standard protocol notation" [4], or "protocol narrations" [5].

This text documents the ASPEN notation and how this notation can be used in LaTeX documents using the LaTeX package aspen. Since the LaTeX package aspen optionally provides support for the BAN logic notation, we have included the BAN logic notation in the documentation.

ASPEN[1] is *not* a formalism, like BAN (Burrows–Abadi–Needham) logic [7], or a calculus for analysis of cryptographic protocols, like Spi calculus [1]. For a more detailed analysis of cryptographic protocols, more expressive notations like BAN logic, Spi calculus, or something similar should be considered. Other references presenting relevant notations include, but are not limited to: [6, 9, 11].

$$A \longrightarrow S : \{A, B, N'_a\}$$
$$S \longrightarrow A : \{N'_a, B, K_{A,B}, \{K_{A,B}, A\}_{K_{B,S}}\}_{K_{A,S}}$$
$$A \longrightarrow B : \{K_{A,B}, A\}_{K_{B,S}}$$
$$B \longrightarrow A : \{N'_b\}_{K_{A,B}}$$
$$A \longrightarrow B : \{N'_b - 1\}_{K_{A,B}}$$

Figure 1: ASPEN example (what protocol is it?)

Contents

[1] Originally, I had no intention to name the notation presented here. While working on this text, it became clear that it was inconvenient to not be able to refer to the notation with a short name. The name ASPEN *can* be an abbreviation for "A Security Protocol Engineering Notation", but for me it is now short for "Anderson-inspired Standard Protocol Engineering Notation", in memory of the late Professor Ross J. Anderson who has meant so much for the fields of computer security, distributed systems, and, in particular, security engineering [2, 3, 4].

Values:	$true$, $\{m\}$, $H\{m\}$	Values, structured values and typed structured values. In this notation a message is seen as a structured value.
Principals:	A, B, S	Principals in security protocols, including clients, servers, and other participants.
Keys:	$K_{A,B}$, $K'_{C,S}$, K_A^+, K_B^-	Cryptographic keys, used to encrypt, decrypt, sign and verify values and messages.
Nonces:	N'_A, N'_B, N'_S	Nonces are generated to be fresh and commonly include a timestamp or a number that is used only once.
Random:	R_x, R'_1	Random values can be a variant of nonces. The $'$ mark hints about limited useful lifetime (once, or during a session).
Time:	T_S, T_A, L	Timestamps and lifetime are often used, together with nonces, to avoid replay and session keys that are too old.
Strings:	"Hello world!"	Not necessary for the intended notation usage, but text strings are often found in examples in the litterateur.
Variables:	x, y, z, a, b, c	A variable can be assigned a value. Might also be used in the context of "we are not sure about its value".
Functions:	$H(m)$, $Func(x, y) \rightarrow z$	A function can take arguments and produce a value. Some functions are the constructor of typed structured values.
Labels:	M_1, S_1	Labels are used to label steps when a security protocols is presented as a series of steps.
LaTeX code:	`\func{Func}{x,y}`	LaTeX code is shown when documenting the usage of the notation in text using the LaTeX package `aspen`.

Figure 2: In the text, color is used to distinguish different features

1 Introduction

Why Aspen then? One motivation is to have an expressive notation that can be used in publications where security protocols are presented. Another motivation is to have a notation that can be used when teaching security related topics. This text is an attempt to document a notation that have been used and refined over years. The notation should be familiar, but with some new useful refinements and contributions not found in similar notations. It should also be possible to use the notations together with other notations, like BAN logic. A more detailed discussion on the choices made for the Aspen notation is found in Appendix B.1, *Notes on the suggested notation*. The LaTeX package can be downloaded from CTAN or its home:

`https://www.pg12.org/dist/texmf/tex/latex/aspen/`

When using Aspen, colors can be used to distinguish different types of features. Figure 2 illustrates how the different features are colored. Colors are optional when using the notation. They are enabled by the `color` option to the LaTeX package `aspen`. Colors are only added for readability. The LaTeX package `aspen` provides different color profiles for typesetting the notation (see Appendix B.2, *Notes on the typesetting options*).

If the `aspen` package is loaded with the option `ban`, the BAN logic notation from "A Logic of Authentication» [7] is included (see Section 2.2 and 3.2).

2 The notations

In the description of the notation below, notation that might be obvious is included. It is done for completeness and consistency. For some notations, usage examples are provided. These examples might include notations explained later in the text.

2.1 Aspen

Notation	Description
$=, <, \leq, >, \geq$	$=$ means *"is equal"*, either as a statement or a claim (e.g., a claim that can be, or has to be, verified). $<$, $\leq$, $>$ and $\geq$ means *"less than"*, *"less than or equal"*, *"greather than"*, and *"greather than or equal"*, respectively. These notations are typically used to compare counters and timestamps in protocols.
$\oplus$, .	The binary operator $\oplus$ is exclusive or, and the binary operator . is concatenation (used to concatenate two values or strings). The concatenation operator has precedence over the exclusive or operator. In Section B.2, other options for the binary concatenation operator is presented.
$\Rightarrow, \Leftrightarrow$	$x \Rightarrow y$ means *"y, if x"*. $x \Leftrightarrow y$ means *"y, if and only if x"*. This is an example used with the *Verify* function (see below for the description of other notations used in the example): $$Verify\left(K_A^+, \{m\}^{K_x}\right) = true \iff K_x = K_A^-$$
$x \rightsquigarrow y$	We use a leads-to arrow $\rightsquigarrow$ to show more details or to unpack a value or a message. The following example shows that a digital signature is actually a cryptographic hash value of the message encrypted with a private key (see below for the description of other notations used in the example): $$Sig\{m\}^{K_A^-} \rightsquigarrow \left\{H\{m\}\right\}_{K_A^-}$$
true, *false*	The boolean values *true* and *false*. The value *true* will also be used to show that an operation completed with success (if that is important). For example, when we verify a digital signature and it is found valid, the function doing the verification returns the value *true*. Otherwise, the value *false* is returned.
K_X	A shared or secret key, also known as a symmetric encryption and decryption key.
$K_{A,B}$	A shared key for A and B (this notation can be extended, for example with $K_{A,B,C}$ for a key shared between A, B, and C, or $K_{M_1-M_n}$ for a key shared among n group members $M_1, \ldots, M_n$).
$K'_{C,S}$	A session key for C and S (this notation can be extended, for example with $K'_{A,B,S}$ for a key session key shared between A, B, and S, or $K'_{M_1-M_n}$ for a session key shared among n group members $M_1, \ldots, M_n$).
K_A^+	The public key of a public-private key pair of A: (K_A^+, K_A^-).
K_A^-	The private key of a public-private key pair of A: (K_A^+, K_A^-).
$\{m\}$	A structured value or message containing m. A structured value can be nested.

$t\{m\}$	A structured value or message with the type t. An example of a structured value marked with the type *Sig*: $Sig\{m\}^{[A]}$ has the type *Sig* and the superscript $[A]$ (signed by A)
$A \longrightarrow B : \{m\}$	Message $\{m\}$ sent from A to B.
$func(x, y)$	A function *func* with two arguments x and y (*Encrypt* and *Decrypt* described below are examples of such a function).
$func(x, y) \rightarrow z$	We use an arrow $\rightarrow$ to illustrate what a function produces (in this case z). The following example shows that the function *Encrypt* produces a ciphertext encrypted with the given shared key (see below for the description of other notations used in the example): $Encrypt(K_{A,B}, m) \rightarrow \{m\}_{K_{A,B}}$
$\{m\}_{\square}$	In general, a subscripted structured value means an encrypted value or message (a ciphertext), where the subscript $\square$ represents the encryption key (or the holder of the encryption key). This is an example where the plain text m is encrypted with the encryption key K: $\{m\}_K$
$\{m\}^{\square}$	In general, a superscripted structured value means a signed value or message, where the superscript $\square$ represents the key used to sign (or the holder of the key used to sign). This is an example where the plain text m is signed by principal A: $\{m\}^{[A]}$
$Encrypt(K_{A,B}, m)$	Encrypt plain text m with shared key $K_{A,B}$: $Encrypt(K_{A,B}, m) \rightarrow \{m\}_{K_{A,B}}$
$Decrypt(K_{A,B}, c)$	Decrypt cipher text c with shared key $K_{A,B}$, where $c = \{m\}_{K_{A,B}}$: $Decrypt(K_{A,B}, c) \rightsquigarrow Decrypt(K_{A,B}, \{m\}_{K_{A,B}}) \rightarrow m$
$Encrypt(K_A^+, m)$	Encrypt plain text m with public key K_A^+: $Encrypt(K_A^+, m) \rightarrow \{m\}_{K_A^+}$
$Decrypt(K_A^-, c)$	Decrypt cipher text c with private key K_A^-, where $c = \{m\}_{K_A^+}$: $Decrypt(K_A^-, c) \rightsquigarrow Decrypt(K_A^-, \{m\}_{K_A^+}) \rightarrow m$
$H\{m\}$	A cryptographic hash value of m.
$H(m)$	A cryptographic hash function producing the cryptographic hash value of m: $H(m) \rightarrow H\{m\}$

$MAC\{m\}^{K_A}$	The message authentication code of m with key K_A.
$CMAC\{m\}^{K_A}$	The cipher-based message authentication code of m with key K_A.
$HMAC\{m\}^{K_A}$	The HMAC message authentication code of m with key K_A.
$MAC(K_A, m)$	The message authentication code function producing the message authentication code of m with key K_A: $$MAC(K_A, m) \rightarrow MAC\{m\}^{K_A}$$
$CMAC(K_A, m)$	The cipher-based message authentication code function producing the cipher-based message authentication code of m with key K_A: $$CMAC(K_A, m) \rightarrow CMAC\{m\}^{K_A}$$
$HMAC(K_A, m)$	The HMAC message authentication code function producing the HMAC message authentication code of m with key K_A: $$HMAC(K_A, m) \rightarrow HMAC\{m\}^{K_A} \rightsquigarrow H\left\{\overline{K}_A \oplus opad \,.\, H\{\overline{K}_A \oplus ipad \,.\, m\}\right\}$$ $$\overline{K}_A = \begin{cases} H\{K_A\} & \text{if } K_A \text{ is larger than block size} \\ K_A & \text{otherwise} \end{cases}$$ The two block-sized paddings, $opad$ (outer padding) and $ipad$ (inner padding), each consists of a repeating byte value (0x5c and 0x36, respectively).
$Sig\{m\}^{[A]}$	Digital (cryptographic) signature of m signed by A.
$Sig\{m\}^{K_A^-}$	Digital (cryptographic) signature of m signed with private key K_A^-: $$Sig\{m\}^{K_A^-} \rightsquigarrow \{H\{m\}\}_{K_A^-}$$
$Sig\{m\}^{[A,B]}$	Digital (cryptographic) signature of m based on shared secret between A and B.
$Sig\{m\}^{K_{A,B}}$	Digital (cryptographic) signature of m signed with the shared key $K_{A,B}$ (a shared secret between A and B): $$Sig\{m\}^{K_A} \rightsquigarrow \{H\{m\}\}_{K_{A,B}}$$
$Sig(K_A^-, m)$	Function creating a digital (cryptographic) signature of m with private key K_A^-: $$Sig(K_A^-, m) \rightarrow Sig\{m\}^{K_A^-}$$
$Sig([A, B], m)$	Function creating a digital (cryptographic) signature of m based on shared secret between A and B: $$Sig([A, B], m) \rightarrow Sig\{m\}^{[A,B]}$$
$Sig(K_{A,B}, m)$	Function creating a digital (cryptographic) signature of m with the shared key $K_{A,B}$ (a shared secret between A and B): $$Sig(K_{A,B}, m) \rightarrow Sig\{m\}^{K_{A,B}}$$

$\{m\}^{[A]}$	m is signed by A (m signed is a combination of m itself and a digital signature of m, in this case a digital signature signed by A): $$\{m\}^{[A]} \rightsquigarrow \{m, Sig\{m\}^{[A]}\}$$
$\{m\}^{K_A^-}$	m is signed with private key K_A^- (m signed is a combination of m itself and a digital signature of m, in this case a digital signature signed with K_A^- implemented by encrypting the cryptographic hash value of m with K_A^-): $$\{m\}^{K_A^-} \rightsquigarrow \{m, Sig\{m\}^{K_A^-}\} \rightsquigarrow \{m, \{H\{m\}\}_{K_A^-}\}$$
$\{m\}^{[A,B]}$	m is signed with shared secret of A and B (m signed is a combination of m itself and a digital signature of m, in this case a digital signature signed with a shared secret of A and B): $$\{m\}^{[A,B]} \rightsquigarrow \{m, Sig\{m\}^{[A,B]}\}$$
$\{m\}^{K_{A,B}}$	m is signed with a shared secret of A and B; the shared key $K_{A,B}$ (m signed is a combination of m itself and a digital signature of m, in this case a digital signature signed with the shared key $K_{A,B}$ implemented by encrypting the cryptographic hash value of m with $K_{A,B}$): $$\{m\}^{K_{A,B}} \rightsquigarrow \{m, Sig\{m\}^{K_{A,B}}\} \rightsquigarrow \{m, \{H\{m\}\}_{K_{A,B}}\}$$
$Sign([A], m)$	A signs m (m signed is a combination of m itself and a digital signature of m, in this case a digital signature signed by A implemented by $[A]$ encrypting the cryptographic hash value of m): $$Sign([A], m) \rightarrow \{m\}^{[A]} \rightsquigarrow \{m, Sig\{m\}^{[A]}\}$$
$Sign(K_A^-, m)$	Sign m with private key K_A^- (m signed is a combination of m itself and a digital signature of m, in this case a digital signature signed with K_A^- implemented by encrypting the cryptographic hash value of m with K_A^-): $$Sign(K_A^-, m) \rightarrow \{m\}^{K_A^-} \rightsquigarrow \{m, Sig\{m\}^{K_A^-}\} \rightsquigarrow \{m, \{H\{m\}\}_{K_A^-}\}$$
$Sign([A, B], m)$	Sign m with shared secret of A and B (m signed is a combination of m itself and a digital signature of m, in this case a digital signature signed with a shared secret of A and B implemented by encrypting the cryptographic hash value of m with a key based on a shared secret of $[A]$ and $[B]$): $$Sign([A, B], m) \rightarrow \{m\}^{[A,B]} \rightsquigarrow \{m, Sig\{m\}^{[A,B]}\}$$
$Sign(K_{A,B}, m)$	Sign m with a shared secret of A and B; the shared key $K_{A,B}$ (m signed is a combination of m itself and a digital signature of m, in this case a digital signature signed with the shared key $K_{A,B}$ implemented by encrypting the cryptographic hash value of m with $K_{A,B}$): $$Sign(K_{A,B}, m) \rightarrow \{m\}^{K_{A,B}} \rightsquigarrow \{m, Sig\{m\}^{K_{A,B}}\} \rightsquigarrow \{m, \{H\{m\}\}_{K_{A,B}}\}$$

$Verify(K_A^+, s)$	Verify that the signed structured value (message) s is signed by the matching private key K_A^- of public key K_A^+ and, as a consequence, verify that s is signed by A:

$$Verify(K_A^+, s) \rightsquigarrow Verify\left(K_A^+, \{m\}^{[x]}\right) \rightsquigarrow Verify\left(K_A^+, \left\{m, Sig\{m\}^{[x]}\right\}\right) \rightsquigarrow$$
$$Verify\left(K_A^+, \left\{m, Sig\{m\}^{K_x^-}\right\}\right) \rightsquigarrow Verify\left(K_A^+, \left\{m, \{H\{m\}\}_{K_x^-}\right\}\right):$$

$$\left.\begin{array}{l} Decrypt\left(K_A^+, Sig\{m\}^{[x]}\right) = H\{m\} \rightsquigarrow \\ Decrypt\left(K_A^+, Sig\{m\}^{K_x^-}\right) = H\{m\} \rightsquigarrow \\ Decrypt\left(K_A^+, \{H\{m\}\}_{K_x^-}\right) = H\{m\} \end{array}\right\} \Leftrightarrow K_A^- = K_x^-$$

$$Verify(K_A^+, s) \rightarrow true \quad \Leftrightarrow \quad K_A^- = K_x^-$$

$Cert\{A, K_A^+\}^{[C]}$	A certificate where CA C binds identity A to public key K_A^+ (where $\ldots$ is other certificate related meta-data):

$$Cert\{A, K_A^+\}^{[C]} \rightsquigarrow \{A, K_A^+, \ldots\}^{[C]}$$

$Cert\{A, K_A^+\}^{K_C^-}$	A certificate where a CA with private key K_C^- binds identity A to public key K_A^+ (where $\ldots$ is other certificate related meta-data):

$$\frac{Cert\{A, K_A^+\}^{K_C^-} \rightsquigarrow \{A, K_A^+, \ldots\}^{K_C^-}}{Verify(K_C^+, Cert\{A, K_A^+\}^{K_C^-}) \rightsquigarrow Verify(K_C^+, \{A, K_A^+, \ldots\}^{K_C^-}) \rightarrow true}$$

2.2 BAN logic

The description below of the BAN logic notation is copied directly, with some minor modifications, from the original paper presenting the BAN logic, "A Logic of Authentication" [7].

Notation	Description
$A \mid\equiv X$	A *believes* X, or A would be entitled to believe X. In particular the principal A may act as though X is true. This construct is central to the BAN logic.
$A \triangleleft X$	A *sees* X. Someone has sent a message containing X to A, who can read and repeat X possibly after doing some decryption.
$A \mid\sim X$	A *once said* X. The principal A at some time sent a message including the statement X. It is not known whether the message was sent long ago or during the current run of the protocol, but it is known that A believed X when A sent the message.
$A \Rightarrow X$	A has *jurisdiction* over X (A *controls* X). The principal A is an authority on X and should be trusted on this matter. This construct is used when a principal has delegated authority over some statement. For example, encryption keys need to be generated with some care, and in some protocols certain servers are trusted to do this properly. This may be expressed by the assumption that the principals believe that the server has jurisdiction over statements about the quality of keys.

$\sharp(X)$	The formula X is *fresh*, that is, X has not been sent in a message at any time before the current run of the protocol. This is usually true for nonces, that is expressions generated for the purpose of being fresh (nonce—number used once). Nonces commonly include a timestamp or a number that is used only once, such as a sequence number.
$A \overset{K}{\longleftrightarrow} B$	A and B may use the *shared key* K to communicate. The key K is good, in that it will never be discovered by any principal except A or B, or a principal trusted by either A or B. (In ASPEN, a shared key A and B may use to communicate can be denoted $K_{A,B}$.)
$\overset{K}{\longmapsto} A$	A has K as a *public key*. The matching secret key (the inverse of K, denoted K^{-1}) will never be discovered by any principal except A, or a principal trusted by A. (In ASPEN, a public key of A can be denoted K_A^+, and the inverse of K_A^+, the private key, can be denoted K_A^-.)
$A \overset{X}{\rightleftharpoons} B$	The formula X is a *secret* known only to A and B, and possibly to principals trusted by them. Only A and B may use X to prove their identities to one another. Often X is fresh as well as secret. An example of a shared secret is a password.
$\{X\}_K$	This represents the formula X *encrypted* under the key K. Formally, $\{X\}_K$ is an abbreviation for an expression of the form $\{X\}_K$ *from* A. We make the realistic assumption that each principal is able to recognize and ignore his own messages; the originator of each message is mentioned for this purpose. In the in terests of brevity, we typically omit this in our examples.
$\langle X \rangle_Y$	This represents X *combined* with the formula Y; it is intended that Y be a secret, and that its presence prove the identity of whoever utters $\langle X \rangle_Y$. In implementations, X is simply concatenated with the password Y; our notation highlights that Y plays a special rôle, as proof of origin for X. The notation is intentionally reminiscent of that for encryption, which also guarantees the identity of the source of a message through knowledge of a certain kind of secret.

In the ASPEN notation, when we write $K_{A,B}$, it is implicit that A and B may use $K_{A,B}$ to communicate. We can use the BAN logic notation to make it explicit:

$$A \overset{K_{A,B}}{\longleftrightarrow} B$$

In a similar manner, K_A^+ is in the ASPEN notation implicit a *public key* of A. We can use the BAN logic notation to make it explicit:

$$\overset{K_A^+}{\longmapsto} A$$

Both the BAN logic notation and ASPEN use the notation $\{m\}_K$ for the formula m *encrypted* under the key K (m encrypted with the key K). In this case, ASPEN has adopted the notation used in BAN logic and in a lot of other related publications and text books.

3 Use the notations in text

This section explains how to use this notation in LaTeX documents. The new LaTeX commands and environments used are defined in the LaTeX package `aspen`.

We will in the text include notation examples that might not make sense in a security protocol perspective. However, they are included for completeness. We will in the documentation try to include a wide range of possibilities available from the LaTeX package `aspen`.

For commands with arguments, the argument types are given using a notation inspired by the `xparse` argument specification:

`m`	Mandatory arguments *Examples:* `\cmd{arg}`	`B`	Optional bracket sizes (`big`, `Big`, ...) *Examples:* `\cmd`, `\cmd[Big]`
`o`	Optional arguments *Examples:* `\cmd`, `\cmd[arg]`	`T`	Optional key types: `*`, `-`, `+`, `!`, or `'` *Examples:* `\cmd`, `\cmd-`, `\cmd!`
`O{default}`	Optionals with default value *Examples:* `\cmd`, `\cmd[arg]`	`_{}`	Markers (give more details) *Examples:* `\cmd_{arg}`
`s`	Optional stars (alternative version) *Examples:* `\cmd`, `\cmd*`	→	Makes a command (with arguments) *Examples:* `\mktval`: `om` → `sB_{}m`

We can use this notation to specify the type of the arguments to a command. For example, `om` says that the command takes two arguments where the first one is optional (in square brackets). We use the symbol → to specify the arguments of a command created by another command (for example `\mktval`).

3.1 Aspen

The table below lists the Aspen notation with the matching LaTeX commands. More examples of usage are found in Section 4.

Notation	LaTeX code and description

$^{-}$, $^{+}$, $'$, ...	Some command markers (key type markers) are used throughout the Aspen LaTeX package:
	`*` : Means no specific variant (argument is the key, not the label of it): `\key*{K}`
	`-` : Mark that it is a private key (from a public-private key pair): `\key-{A}`
	`+` : Mark that it is a public key (from a public-private key pair): `\key+{A}`
	`!` : Mark by principal instead of key (the key of): `\key!{A}`
	`'` : Mark that it is temporary (session key or limited lifetime): `\key'{A}`
	`\key*{K}` → K
	`\sig-{A}{m}` → $Sig\{m\}^{K_A^-}$
	`\encrypt+{B}{m}` → $Encrypt(K_B^+, m)$
	`\signed!{S}{m}` → $\{m\}^{[S]}$
	`\decrypt'{A,B}{c}` → $Decrypt(K'_{A,B}, c)$

Arguments:	`T` (the symbol used for these optional markers in argument specifications)

...	`_{...}`, where the marker is used to provide more details about a structured value or a function. A few examples where the markers are *MD5*, *RSA*, *AES*, *DSA*, and *SHA-2*: `\chash_{MD5}{m}` → $H_{MD5}\{m\}$ `\encrypt+_{RSA}{B}{m}` → $Encrypt_{RSA}(K_B^+, m)$ `\decrypt'_{AES}{A,B}{c}` → $Decrypt_{AES}(K'_{A,B}, c)$ `\sig-_{DSA}{S}{m}` → $Sig_{DSA}\{m\}^{K_S^-}$ `\hmac_{SHA-2}{A}{m}` → $HMAC_{SHA\text{-}2}\{m\}^{K_A}$
Arguments:	`_{}` (the symbol used for such optional embellishment in argument specifications)
$=, <, \leq, >, \geq$	`=`,`<`,`\leq`,`>`,`\geq`, used to compare values.
$\oplus$, .	`\axor`, `\aconcat`, used as binary operators for *exclusive or* and *concatenation* (used to concatenate two values or strings), respectively.
$\Rightarrow, \Leftrightarrow$	`\aifthen`,`\aiffthen`, used to reason about protocols and protocol steps (meaning, *"if, then"* and *"if, and only if, then"*, respectively).
$x \rightsquigarrow y$	`x \leadsto y`, used to unpack more details.
1, 2	`\aval{1}`,`\aval{2}`, used for values.
Arguments:	`\aval`: `m` `\aval{<value>}`
true,*false*	`\atrue`,`\afalse`, used for the boolean values.
A, B, S	`\apri{A}`,`\apri{B}`,`\apri{S}`, used for principals.
Arguments:	`\apri`: `m` `\apri{<principal>}`
N'_A, N'_S	`\anonce{N_A}`,`\nonce{S}`, used for nonces. The `\nonce` command has an optional first argument to change the symbol (the letter): `\nonce[n]{0}` → n'_0
Arguments:	`\anonce`: `m`, `\nonce`: `om` `\anonce{<name>}`, `\nonce[<symbol>]{<id>}`
R_x, R_y, R'_1	`\arandom{R_x}`,`\random{y}`,`\random'{1}`, used for random values (the $'$ hints about limited useful lifetime). The `\random` command has an optional first argument to change the symbol (the letter): `\random[r]{z}` → r_z
Arguments:	`\arandom`: `m`, `\random`: `om` `\arandom{<name>}`, `\random[<symbol>]{<id>}`
T_A, T_S, L, L_1	`\ats{T_A}`,`\ts{S}`,`\attl{L}`,`\ttl{1}`, used for time related values, like time stamps and lifetime (time to live). Both the `\ts` and `\ttl` command have an optional first argument to change the symbol (the letter): `\ts[t]{0}` → t_0, `\ttl[l]{1}` → l_1
Arguments:	`\ats`: `m`, `\ts`: `om`, `\attl`: `m`, `\ttl`: `om` `\ats{<name>}`, `\ts[<symbol>]{<id>}`, `\attl{<name>}`, `\ttl[<symbol>]{<id>}`
"Hello"	`\astr{Hello}`, used for text strings.
Arguments:	`\astr`: `m` `\astr{<str>}`

x, y	`\avar{x}`,`\avar{y}`, used for variables.
Arguments:	`\avar`: `m` `\avar{<variable>}`
K	`\akey{K}`, used for (non-specific) encryption keys. If you mark the `key` command with a `*`, the produced output is the same: `\key*{K}` → K
Arguments:	`\akey`: `m`, `\key`: `Tm` `\akey{<key>}`, `\key<T>{<key>}`
K_A, $K_{A,B}$	`\key{A}`,`\sharedkey{A,B}`, used for shared (secret/symmetric) keys (provided by two different LaTeX commands, where the first is a more compact version; use whatever you prefer). The `\sharedkey` command has an optional first argument to change the symbol (the letter): `\sharedkey[k]{B}` → k_B
Arguments:	`\key`: `Tm`, `\sharedkey`: `O{K}m` `\key{<id>}`, `\sharedkey[<symbol>]{<id>}`
$K'_{C,S}$, $K'_{A,B,S}$	`\key'{C,S}`,`\sessionkey{A,B,S}`, used for session keys (provided by two different LaTeX commands, where the first is a more compact version; use whatever you prefer). The `\sessionkey` command has an optional first argument to change the symbol (the letter): `\sessionkey[k]{A,B}` → $k'_{A,B}$
Arguments:	`\key`: `Tm`, `\sessionkey`: `O{K}m` `\key'{<id>}`, `\sessionkey[<symbol>]{<id>}`
K^+_A, K^+_B	`\key+{A}`,`\pubkey{B}`, used for public keys (provided by two different LaTeX commands, where the first is a more compact version; use whatever you prefer). The `\pubkey` command has an optional first argument to change the symbol (the letter): `\pubkey[k]{S}` → k^+_S
Arguments:	`\key`: `Tm`, `\pubkey`: `O{K}m` `\key+{<id>}`, `\pubkey[<symbol>]{<id>}`
K^-_A, K^-_B	`\key-{A}`,`\privkey{B}`, used for private keys (provided by two different LaTeX commands, where the first is a more compact version; use whatever you prefer). The `\privkey` command has an optional first argument to change the symbol (the letter): `\privkey[k]{A}` → k^-_A
Arguments:	`\key`: `Tm`, `\privkey`: `O{K}m` `\key-{<id>}`, `\privkey[<symbol>]{<id>}`
$[A]$	`\aname{A}`, typically used to indicate who signed (or encrypted) a message, but no specific key is given, known or relevant. If you mark a key with a `!`, the produced output is the same: `\key!{A}` → $[A]$
Arguments:	`\aname`: `m`, `\key`: `Tm` `\aname{<id>}`, `\key!{<id>}`
M_1–M_n	`\agroup{M}`, specifies a group and is typically used as a label for a shared key shared within a group with n members: `\key{\agroup{M}}` → $K_{M_1\text{-}M_n}$ `\agroup[O][s]{M}`, used when th group member indexes are non-standard: `\key{\agroup[O][s]{M}}` → $K_{M_0\text{-}M_s}$ `\agroup*{M}`, typically used in a text when referring to a group (with n members): `\key{\agroup*{M}}` → $K_{M_1,\ldots,M_n}$

`\agroup*[O][s]{M}`, used when group member indexes are non-standard:

```
\key{\agroup*[O][s]{M}}  →  K_{M_0,...,M_s}
```

Arguments: `\agroup`: `sO{1}O{n}m`
`\agroup<*>[<first>][<last>]{<id>}`

$\{m\}, \{A, B\}$ `\sval{m}`,`\msg{\apri{A},\apri{B}}`, used to type a structured value or message (a message can be seen as structured value).

Arguments: `\sval`: `Bm`, `\msg`: `Bm`
`\sval[<size>]{<value>}`, `\msg[<size>]{<message>}`

$\{m\}$ `\sval[big]{m}`, or `\msg[big]{m}`, where first optional size argument can be `big`, `Big`, `bigg`, or `Bigg` for increased size of parenthesis (typically used with nested structured values and/or functions):

```
\sval{x}  →  {x}
\sval[big]{\sval{x}}  →  {{x}}
\sval[Big]{\sval[big]{\sval{x}}}  →  {{{x}}}
\sval[bigg]{\sval[Big]{\sval[big]{\sval{x}}}}  →  {{{{x}}}}
```

We can even use more size specifiers: `big`, `Big`, `bigg`, `Bigg`, `biggg`, `Biggg`, `bigggg`, `Bigggg`, `biggggg`, and `Biggggg`:

$$\{\{\{\{\{\{\{\{\{\{x\}\}\}\}\}\}\}\}\}\}$$

The optional size argument applies for all AspenTeX commands that produces a pair of parenthesis, both ordinary parenthesis and curly brackets. A few examples (see below for more details on these commands):

```
\tval[big]{Type}{m}        →  Type{m}
\send[big]{A}{B}{m}        →  A ⟶ B : {m}
\func[big]{Func}{x,y}      →  Func(x,y)
\encrypted[big]{A,B}{m}    →  {m}_{K_{A,B}}
```

Arguments: `\sval`: `Bm`, `\msg`: `Bm`
`\sval[<size>]{<value>}`, `\msg[<size>]{<message>}`

$Type\{m\}$ `\tval{Type}{m}`, used for a typed structured value where the first argument is the type. The `\tval*` variant is used for a typed structured value where the first argument is the type, but the value is not wrapped with curly brackets. This is typically used when the value is already wrapped as a structured value (e.g., encrypted or signed data). This is an example with a Kerberos Authenticator as a typed structured value:

```
\tval*{KA}{\encrypted{S,C}{\apri{C},\textit{Addr}_C,\ts{t}}}
                →  KA{C,Addr_C,T_t}_{K_{S,C}}
```

The marker (`_{RSA}` in the example below) can be used to give more details about the typed structured value:

```
\tval*{KA}_{RSA}{\encrypted{S,C}{\apri{C},\textit{Addr}_C,\ts{t}}}
```
→ $KA_{RSA}\{C, Addr_C, T_t\}_{K_{S,C}}$

Arguments:
`\tval`: `sBm_{}m`
`\tval<*>[<size>]{<type>}_{<marker>}{<value>}`

$A \longrightarrow B : \{m\}$

`\send{A}{B}{m}`, used to specify that a message $\{m\}$ is sent from A to B. The `\send*` variant is used to specify that the message is not wrapped as a structured value or message. This is typically used when what-is-sent is already wrapped as a structured value (e.g., encrypted or signed data):

```
\send*{A}{B}{\encrypted+{B}{m}}
```
→ $A \longrightarrow B : \{m\}_{K_B^+}$

```
\send*{A}{B}{\encrypted+[big]{B}{\chash{m}}}
```
→ $A \longrightarrow B : \left\{H\{m\}\right\}_{K_B^+}$

Arguments:
`\send`: `sBmmm`
`\send<*>[<size>]{<sender>}{<receiver>}{<message>}`

$Func(x, y)$

`\func{Func}{x,y}`, used for any functions. An optional last argument is used if a return value of the function is given:

```
\func{Func}{x,y}[z]
```
→ $Func(x, y) \rightarrow z$

Arguments:
`\func`: `Bm_{}mo`
`\func[<size>]{<name>}_{<marker>}{<arguments>}[<returns>]`

$\{m\}_{K_{A,B}}$

`\encrypted{A,B}{m}`, where the message is encrypted with an shared secret encryption key (in this case, the shared key $K_{A,B}$ of A and B). The other options (with markers) are:

```
\encrypted*{K}{m}
```
→ $\{m\}_{K}$

```
\encrypted+{A}{m}
```
→ $\{m\}_{K_A^+}$

```
\encrypted-{A}{m}
```
→ $\{m\}_{K_A^-}$

```
\encrypted!{A}{m}
```
→ $\{m\}_{[A]}$

```
\encrypted'{A,B}{m}
```
→ $\{m\}_{K'_{A,B}}$

Arguments:
`\encrypted`: `TB_{}mm`
`\encrypted<T>[<size>]_{<marker>}{<key>}{<plain>}`

$Encrypt(K_{A,B}, m)$

`\encrypt{A,B}{m}`, where the value m is encrypted with a secret shared encryption key (in this case, a shared key of A and B). Other options are `*`, `+`, `-`, `!`, and `'` (see above for explanation). Since `\encrypt` is a function, we can include a return value as an optional last argument:

```
\encrypt+{B}{m}[\encrypted+{B}{m}]
```
→ $Encrypt(K_B^+, m) \rightarrow \{m\}_{K_B^+}$

Arguments:
`\encrypt`: `TB_{}mmo`
`\encrypt<T>[<size>]_{<marker>}{<key>}{<plain>}[<returns>]`

$Decrypt(K_{A,B}, c)$	`\decrypt{A,B}{\avar{c}}`, where the cipher text c is decrypted with an secret shared encryption key (in this case, a shared key between A and B). Other options are `*`, `+`, `-`, `!`, and `'` (see above for explanation). Since `\decrypt` is a function, we can include a return value as an optional last argument: `\decrypt-{B}{\encrypted+{B}{m}}[m]` → $Decrypt(K_B^-, \{m\}_{K_B^+}) \rightarrow m$
Arguments:	`\decrypt`: `TB_{}mmo` `\decrypt<T>[<size>]_{<marker>}{<key>}{<cipher>}[<returns>]`
$H\{m\}$	`\chash{m}`, used for a cryptographic hash value of m.
Arguments:	`\chash`: `B_{}m` `\chash[<size>]_{<marker>}{<value>}`
$MAC\{m\}^{K_A}$	`\mac{A}{m}`, used for message authentication code of m with K_A.
Arguments:	`\mac`: `TB_{}mm` `\mac<T>[<size>]_{<marker>}{<key>}{<value>}`
$CMAC\{m\}^{K_A}$	`\cmac{A}{m}`, used for cipher-based message authentication code of m with K_A.
Arguments:	`\cmac`: `TB_{}mm` `\cmac<T>[<size>]_{<marker>}{<key>}{<value>}`
$HMAC\{m\}^{K_A}$	`\hmac{A}{m}`, used for HMAC message authentication code of m with K_A.
Arguments:	`\hmac`: `TB_{}mm` `\hmac<T>[<size>]_{<marker>}{<key>}{<value>}`
$H(m)$	`\chashf{m}`, used for a cryptographic hash value function producing the cryptographic hash value of m. Since `\chashf` is a function, we can include a return value as an optional last argument: `\chashf{m}[\chash{m}]` → $H(m) \rightarrow H\{m\}$
Arguments:	`\chashf`: `B_{}mo` `\chasf[<size>]_{<marker>}{<value>}[<returns>]`
$MAC(K_A, m)$	`\macf{A}{m}`, used for a message authentication code function with the arguments K_A and m. Since `\macf` is a function, we can include a return value as an optional last argument: `\macf{A}{m}[\mac{A}{m}]` → $MAC(K_A, m) \rightarrow MAC\{m\}^{K_A}$
Arguments:	`\macf`: `TB_{}mmo` `\macf<T>[<size>]_{<marker>}{<key>}{<value>}[<returns>]`
$CMAC(K_A, m)$	`\cmacf{A}{m}`, used for a cipher-based message authentication code with the arguments K_A and m. Since `\cmacf` is a function, we can include a return value as an optional last argument: `\cmacf{A}{m}[\cmac{A}{m}]` → $CMAC(K_A, m) \rightarrow CMAC\{m\}^{K_A}$
Arguments:	`\cmacf`: `TB_{}mmo` `\cmacf<T>[<size>]_{<marker>}{<key>}{<value>}[<returns>]`

$HMAC(K_A, m)$	`\hmacf{A}{m}`, used for a HMAC message authentication code with the arguments K_A and m. Since `\hmacf` is a function, we can include a return value as an optional last argument: `\hmacf{A}{m}[\hmac{A}{m}]` → $HMAC(K_A, m) \rightarrow HMAC\{m\}^{K_A}$
Arguments:	`\hmacf`: `TB_{}mmo` `\hmacf<T>[<size>]_{<marker>}{<key>}{<value>}[<returns>]`
$Sig\{m\}^{K_A^-}$	`\sig-{A}{m}`, used for the signature of A on m, where the `-` says that the signature is signed with a private key (in this case, the private key of A).
Arguments:	`\sig`: `TB_{}mm` `\sig<T>[<size>]_{<marker>}{<key>}{<value>}`
$Sig(K_A^-, m)$	`\sigf-{A}{m}`, used to create a signature of A on m, where the `-` says that the signature is signed with a private key (in this case, the private key of A).
Arguments:	`\sigf`: `TB_{}mmo` `\sigf<T>[<size>]_{<marker>}{<key>}{<value>}[<returns>]`
$\{m\}^{K_A^-}$	`\signed-{A}{m}`, used for m signed, where the `-` says that the signature is signed with a private key (in this case, the private key of A).
Arguments:	`\signed`: `TB_{}mm` `\signed<T>[<size>]_{<marker>}{<key>}{<value>}`
$Sign(K_A^-, m)$	`\sign-{A}{m}`, used to sign m, where the `-` says that the signature is signed with a private key (in this case, the private key of A).
Arguments:	`\sign`: `TB_{}mmo` `\sign<T>[<size>]_{<marker>}{<key>}{<value>}[<returns>]`
$Verify(K_A^+, s)$	`\verify+{A}{\avar{s}}`, used to verify the signed data s, where the `+` says that the signed data is verified towards the public key of A.
Arguments:	`\verify`: `TB_{}mmo` `\verify<T>[<size>]_{<marker>}{<key>}{<value>}[<returns>]`
$Cert\{B, K_B^+\}^{[C]}$	`\certificate!{C}{\apri{B},\key+{B}}`, used for a certificate binding the public key K_B^+ (public key of B) to the principal B, where `!` says that the signature is signed by the CA C.
Arguments:	`\certificate`: `TB_{}mm` `\certificate<T>[<size>]_{<marker>}{<key>}{<content>}`
$Cert\{A, K_A^+\}^{K_C^-}$	`\cert-{C}{A}`, used for a certificate binding the public key +A (public key of A) to the principal A, where the `-` says that the signature is signed with a private key (in this case, the private key of the CA C).
Arguments:	`\cert`: `TB_{}mm` `\certificate<T>[<size>]_{<marker>}{<key>}{<principal>}`
$X\{m\}$	`\mktval{X}`, used to create a new typed structured value type where the argument is the type. In this example, the result is a new LaTeX command `\tvalX` (created combining the prefix `tval` and the given name). We can for example use this to create a new typed structured type for a specific message type: `\mktval{ReqMsg}` `\tvalReqMsg{\apri{A},m}` → $ReqMsg\{A, m\}$

The new command will have a * version similar to the \tval* command (the value is not wrapped with curly brackets). The \mktval has an optional first argument to specify the name of the command created:

```
\mktval[reqmsg]{RMsg}
\reqmsg{m}
```
→ *RMsg*{*m*}

Arguments:
\mktval: om → sB_{}m
\mktval[*<cmd>*]{*<type>*}
→ *cmd<*>*[*<size>*]_{*<marker>*}{*<value>*}

$X\{m\}_{K_A}$

\mketval{X}, used to create a new *encrypted* typed structured value type where the argument is the type. In this example, the result is a new LaTeX command \etvalX (created combining the prefix etval and the given name). We can for example use this to create a new typed structured type for an encrypted message type:

```
\mketval{EMsg}
\etvalEMsg'{C,S}{m}
```
→ $EMsg\{m\}_{K'_{C,S}}$

The \mketval has an optional first argument to specify the name of the command created (here we define the command \aka for Kerberos Authenticators):

```
\mketval[aka]{KA}
\aka'{S,C}{\apri{C},\textit{Addr}_C,\ts{s}}
```
→ $KA\{C, Addr_C, T_s\}_{K'_{S,C}}$

In this case, it might be a good idea to create a new LaTeX command \ka implemented with \aka and the proper arguments (implementation details not shown):

```
\newcommand{\ka}[3]{\aka'{...}}
\ka{S,C}{C}{s}
```
→ $KA\{C, Addr_C, T_s\}_{K'_{S,C}}$

Arguments:
\mketval: om → TB_{}mm
\mketval[*<cmd>*]{*<type>*}
→ *cmd<T>*[*<size>*]_{*<marker>*}{*<key>*}{*<value>*}

$X\{m\}^{K_A}$

\mkstval{X}, used to create a new *signed* typed structured value type where the argument is the type. In this example, the result is a new LaTeX command \stvalX (created combining the prefix stval and the given name). We can for example use this to create a new typed structured type for an signed message type:

```
\mkstval{SMsg}
\stvalSMsg-{A}{m}
```
→ $SMsg\{m\}^{K_A^-}$

The \mkstval has an optional first argument to specify the name of the command created:

```
\mkstval[smsg]{SMsg}
\smsg-{S}{m}
```
→ $SMsg\{m\}_{K_S^-}$

Arguments:
\mkstval: om → TB_{}mm
\mkstval[*<cmd>*]{*<type>*}
→ *cmd<T>*[*<size>*]_{*<marker>*}{*<key>*}{*<value>*}

$X(x, y)$

`\mkfunc{X}`, used when creating a new function type where the argument is the name of the function type. In this example the result is a new LaTeX command `\funcX` (created combining the prefix `func` and the given name). We can for example use this to create a new function type for a creating a Kerberos Authenticator:

```
\mkfunc{KA}
\funcKA{\apri{C},\textit{Addr}_C,\ts{s}}
```
→ $KA(C, Addr_C, T_s)$

The `\mkfunc` has an optional first argument to specify the name of the command created:

```
\mkfunc[kaf]{KA}
\kaf{\apri{C},\textit{Addr}_C,\ts{s}}
```
→ $KA(C, Addr_C, T_s)$

Arguments:

`\mkfunc`: `om` → `B_{}mo`
`\mkfunc[<cmd>]{<name>}`
→ `\cmd[<size>]_{<marker>}{<arguments>}[<returns>]`

$X(K_A, x, y)$

`\mkkfunk{X}`, used when creating a new function type for functions where the first argument is an encryption key. The argument is the name of the function type. In this example the result is a new LaTeX command `\funcX` (created combining the prefix `func` and the given name) with two arguments; the first argument is an encryption key and the second argument is a comma separated list of the rest of the function arguments. We can for example use this to create this new function type with an encryption key as the first argument (a session key in this example):

```
\mkkfunc{KeyF}
\funcKeyF'{A}{\textit{Addr},\ts{s}}
```
→ $KeyF(K'_A, Addr, T_s)$

The `\mkkfunc` has an optional first argument to specify the name of the command created:

```
\mkkfunc[kf]{KeyF}
\kf'{A}{\textit{Addr},\ts{s}}
```
→ $KeyF(K'_A, Addr, T_s)$

Arguments:

`\mkkfunc`: `om` → `TB_{}mmo`
`\mkkfunc[<cmd>]{<name>}`
→ `\cmd<T>[<size>]_{<marker>}{<key>}{<arguments>}[<returns>]`

3.2 BAN logic

The table below lists the BAN logic notation with the matching LaTeX commands. These notations are available when the LaTeX package `aspen` is loaded with the option `ban`.

Notation	LaTeX code and description
$\mid\!\equiv$	`\believes`, used to state that someone *believes* something (and acts as it is true): `\apri{A}\believes\aval{X}` → $A \mid\!\equiv X$
$\triangleleft$	`\sees`, used to state that someone sees something (Someone has sent a message to someone and they have bee able to read it): `\apri{A}\sees\aval{X}` → $A \triangleleft X$
$\mid\!\sim$	`\oncesaid`, used to state to someone at some time said something (someone some time sent a message including the statement): `\apri{A}\oncesaid\aval{X}` → $A \mid\!\sim X$
$\Mapsto$	`\controls`, used to state that someone has *jurisdiction* (controls) over something: `\apri{A}\controls\aval{X}` → $A \Mapsto X$
$\sharp(X)$	`\fresh{X}`, used to state that something is fresh (X has not been sent in a message at any time before in the current run of the protocol).
$\overset{K}{\longleftrightarrow}$	`\asharedkey{K}`, used to state that a key is shared: `\apri{A}\asharedkey{\key{A,B}}\apri{B}` → $A \overset{K_{A,B}}{\longleftrightarrow} B$
$\overset{K}{\longmapsto}$	`\thepubkey{K}`, used to state that a key is a public key of someone: `\thepubkey{\key+{A}}\apri{A}` → $\overset{K_A^+}{\longmapsto} A$
$\overset{X}{\leftrightharpoons}$	`\asecret{X}`, used to state that a secret (X, in this case) is only known to them: `\apri{A}\asecret{X}\apri{B}` → $A \overset{X}{\leftrightharpoons} B$
$\{X\}_K$	`\encryptedwith{K}{X}`, used to state that something is encrypted with the key (X is encrypted with the key K).
$\langle x \rangle_y$	`\combine{x}{y}`, used to state that x is combined with y: `\combine{\aval{X}}{\aval{Y}}` → $\langle X \rangle_Y$

3.3 Series of steps

The LaTeX package `aspen` provides support for presenting a security protocol as a series of messages and steps with the `steps` environment. A message between to principals is in the `steps` environment is typeset with the familiar `\send` command. With `\send` commands, the `steps` environment can be used like this:

```
\begin{steps}
  \send*{A}{B}{\encrypted+{B}{m_1}}[m1] \\
  \send*{B}{A}{\encrypted+{A}{m_2}}[m2]
\end{steps}
```

$$M_1 \quad A \longrightarrow B : \{m_1\}_{K_B^+}$$
$$M_2 \quad B \longrightarrow A : \{m_2\}_{K_A^+}$$

Notice that each step is separated by the `\\` command. Each step is labeled and can be refereed to by its name (`\ref{m1}` $\rightarrow M_1$, and `\ref{m2}` $\rightarrow M_2$). The `steps*` version of the environment is without the labels:

```
\begin{steps*}
  \send*{A}{B}{\encrypted+{B}{m_1}} \\
  \send*{B}{A}{\encrypted+{A}{m_2}}
\end{steps*}
```

$$A \longrightarrow B : \{m_1\}_{K_B^+}$$
$$B \longrightarrow A : \{m_2\}_{K_A^+}$$

By default, the `steps` environment has two types of labels; `M` for messages and `S` for other steps. In the example above only messages (`\send` commands) are used. Other steps are given with the `\astep` or the `\astepat` commands. In the following example the `\astep` command is used and the space between the label and the step is adjusted with the optional key-value argument `lspace` (the default value is `1.5em`):

```
\begin{steps}[lspace=1em]
  \astep{\encrypt+{B}{m_1}%
    [\encrypted+{B}{m_1}]} \\
  \astep{\sign-[big]{A}{%
      \encrypted+{B}{m_1}}%
    [{\signed-[big]{A}{%
        \encrypted+{B}{m_1}}}]}
\end{steps}
```

$$S_1 \quad Encrypt(K_B^+, m_1) \rightarrow \{m_1\}_{K_B^+}$$
$$S_2 \quad Sign\left(K_A^-, \{m_1\}_{K_B^+}\right) \rightarrow \left\{\{m_1\}_{K_B^+}\right\}^{K_A^-}$$

The optional key-value argument `rmarg` sets the right margin width of the steps environment and `lmarg` sets the left margin width of the steps environment. The default margin widths are `\tabcolsep`. In the following example the margins are removed:

```
\begin{steps*}[lmarg=0pt,rmarg=0pt]
  \astep{No margins}
\end{steps*}
```

No margins

We can also change the margins, and the space between the label and the step, by adjusting the lengths `\stepsleftmargin`, `\stepsrightmargin` and `\stepslabelspace`. To change these values for the whole document we can place these commands at the beginning of the LaTeX file (after the LaTeX package `aspen` is loaded):

```
\setlength{\stepsleftmargin}{0pt}
\setlength{\stepsrightmargin}{0pt}
\setlength{\stepslabelspace}{1em}
```

The `\astepat` command can be used to specified *where* a step is performed. The command has an extra first argument where this is specified (in this example, at principal A):

```
\begin{steps}*
  \astepat{A}{\sign-{A}{m_1}%
    [\signed-{A}{m_1}]} \\
  \astepat{A}{\encrypt+[big]{B}{%
      \signed-{A}{m_1}}%
    [{\encrypted+[big]{B}{%
        \signed-{A}{m_1}}}]}
\end{steps}
```

$S_3 \quad A : Sign(K_A^-, m_1) \rightarrow \{m_1\}^{K_A^-}$

$S_4 \quad A : Encrypt\left(K_B^+, \{m_1\}^{K_A^-}\right) \rightarrow \left\{\{m_1\}^{K_A^-}\right\}_{K_B^+}$

The `*` marker of the `steps` environment (not to be confused with the `steps*` version of the environment) means that the counters of the labels are *not* reset (the counting continues from the previous `steps` environment).

It is also possible to add new types of labels with the optional key-value argument `labels`. In this example, new label types `A` and `B` are introduced and the `\astepat` commands are labeled with the new label types by using the optional first argument to the command:

```
\begin{steps}[labels={A,B}]
  \astepat[A]{A}{\encrypt+{B}{m_1}%
    [\encrypted+{B}{m_1}]} \\
  \send*{A}{B}{\encrypted+{B}{m_1}} \\
  \astepat[B]{B}{\decrypt-[big]{B}{%
      \encrypted+{B}{m_1}}[m_1]}
\end{steps}
```

$A_1 \quad A : Encrypt(K_B^+, m_1) \rightarrow \{m_1\}_{K_B^+}$

$M_1 \quad A \longrightarrow B : \{m_1\}_{K_B^+}$

$B_1 \quad B : Decrypt\left(K_B^-, \{m_1\}_{K_B^+}\right) \rightarrow m_1$

The `\astepat*` version of the `\astepat` command changes the horizontal position of the text of such steps so the colons are aligned:

```
\begin{steps*}
  \send*{A}{B}{\signed-{A}{m_1}} \\
  \astepat*{B}{\verify+{A}{\signed-{A}{m_1}}}
\end{steps*}
```

$A \longrightarrow B : \{m_1\}^{K_A^-}$

$B : Verify(K_A^+, \{m_1\}^{K_A^-})$

The `\astep*` version of the `\astep` command changes the horizontal position of the text of the step in a similar way:

```
\begin{steps*}
  \send*{A}{B}{\signed-{A}{m_1}} \\
  \astep*{\verify+{A}{\signed-{A}{m_1}}}
\end{steps*}
```

$A \longrightarrow B : \{m_1\}^{K_A^-}$

$Verify(K_A^+, \{m_1\}^{K_A^-})$

The `\arawstep` command is a low-level command that we usually is not necessary. In a `steps*` environment it has four optional arguments followed by one non-optional argument. In a `steps` environment another optional first argument and an optional last argument is added related to the labels of the step. To better understand the command we show it here used together with a `\send` command in a `steps` environment:

```
\begin{steps}
  \send{A}{B}{m}[s1] \\
  \arawstep[M][a][--][b][;]{m}[s2]
\end{steps}
```

$M_1 \quad A \longrightarrow B : \{m\}$

$M_2 \quad \mathrm{a} \; - \; \mathrm{b} \; ; \; \mathrm{m}$

$$
\begin{aligned}
M_1 \quad & A \longrightarrow S : \{A, B, N'_a\} \\
M_2 \quad & S \longrightarrow A : \left\{N'_a, B, K_{A,B}, \{K_{A,B}, A\}_{K_{B,S}}\right\}_{K_{A,S}} \\
M_3 \quad & A \longrightarrow B : \{K_{A,B}, A\}_{K_{B,S}} \\
M_4 \quad & B \longrightarrow A : \{N'_b\}_{K_{A,B}} \\
M_5 \quad & A \longrightarrow B : \{N'_b - 1\}_{K_{A,B}}
\end{aligned}
$$

Figure 3: The original Needham-Schroeder protocol

$$
\begin{aligned}
M_1 \quad & A \longrightarrow S : \{A, B\} \\
M_2 \quad & S \longrightarrow A : \left\{T_s, L, K_{A,B}, B, \{T_s, L, K_{A,B}, A\}_{K_{B,S}}\right\}_{K_{A,S}} \\
M_3 \quad & A \longrightarrow B : \left\{\{T_s, L, K_{A,B}, A\}_{K_{B,S}}, \{A, T_a\}_{K_{A,B}}\right\} \\
M_4 \quad & B \longrightarrow A : \{T_a + 1\}_{K_{A,B}}
\end{aligned}
$$

Figure 4: The Kerberos protocol

4 Notation usage examples

To illustrate the usability of the notation, a few examples where the notation is used to describe well-known, and not so well-known, security protocols.

The *Needham–Schroeder protocol* [10] aims to establish a session key between two parties on a network, typically to protect further communication. The protocol is based on a symmetric encryption and it forms the basis for the Kerberos protocol. In Aspen, the original Needham–Schroeder protocol is written like seen in Figure 3. N'_a and N'_b are nonces and the shared key $K_{A,B}$ should be fresh, $\sharp(K_{A,B})$.

The *Kerberos protocol* [12] is based on the Needham-Schroeder protocol, but makes use of timestamps as nonces to remove the problems of the Needham-Schroeder protocol and to reduce the number of messages needed. Figure 4 shows the Kerberos protocol in Aspen. T_s and T_a are timestamps and L is a lifetime.

A References

[1] Abadi, M., Gordon, A.D.: A calculus for cryptographic protocols: The Spi calculus. Information and Computation **148**, 1–70 (1999). 10.1006/inco.1998.2740

[2] Anderson, R.: Security Engineering: A Guide to Building Dependable Distributed Systems. Wiley Computer Publishing, John Wiley & Sons (2001)

[3] Anderson, R.: Security Engineering: A Guide to Building Dependable Distributed Systems. John Wiley & Sons, 2 edn. (2008)

[4] Anderson, R.: Security Engineering: A Guide to Building Dependable Distributed Systems. John Wiley & Sons, 3 edn. (2020)

[5] Briais, S., Nestmann, U.: A formal semantics for protocol narrations. In: De Nicola, R., Sangiorgi, D. (eds.) Trustworthy Global Computing, International Symposium, TGC 2005. Lecture Notes in Computer Science, vol. 3705, pp. 163–181. Springer-Verlag, Edinburgh, UK (Apr 2005). 10.1007/11580850_10

[6] Briais, S., Nestmann, U.: A formal semantics for protocol narrations. Theoretical Computer Science **389**(3), 484–511 (Dec 2007). 10.1016/j.tcs.2007.09.005

[7] Burrows, M., Abadi, M., Needham, R.: A logic of authentication. SRC Research Reports 39, DEC's System Research Center (Feb 1989)

[8] Chappell, D.: Exploring Kerberos, the protocol for distributed security in Windows 2000. Microsoft System Journal (Aug 1999)

[9] Davis, D., Swick, R.: Workstation services and Kerberos authentication at project Athena. LCS Technical Memos MIT-LCS-TM-424, Massachusetts Institute of Technology, Laboratory for Computer Science (Mar 1989)

[10] Needham, R., Schroeder, M.: Using encryption for authentication in large networks of computers. Communications of the ACM **21**(12), 993–999 (Dec 1978). 10.1145/359657.359659

[11] Schäfer, G., Festag, A., Karl, H., Wolisz, A.: Current approaches to authentication in wireless and mobile communications networks. TKN Technical Report TKN-01-002, Technical University Berlin, Telecommunication Networks Group (Mar 2001)

[12] Steiner, J.G., Neuman, C., Schiller, J.: Kerberos: An authentication service for open networks systems. In: Proceedings of Usenix Winter Conference 1988. pp. 191–202 (Feb 1988)

B Notes

B.1 Notes on the suggested notation

The notations used for security protocols in different articles and textbooks is not consistent. ASPEN is an attempt to create one consistent notation. Mostly, for my own usage, but if the suggested notation is found useful for others, it is a nice bonus. In the following, the choices of ASPEN will be discussed and compared with similar notations used in articles and textbooks. This is not an attempt to provide a complete overview over existing notations and how they compare to ASPEN. It is more a discussion of notations that inspired ASPEN and the choices made in the suggested notation. Feedback on the notation are welcome.

The notation used in the original Kerberos documentation includes secret keys (called private keys, but they are not the private key of a public-private key pair), session keys and encrypted messages.

Below, ASPEN is compared with the notation used in litterateur. The following sources of different notations are used:

1. ASPEN
2. Kerberos: An Authentication Service for Open Network Systems [12]
3. A formal semantics for protocol narrations [6]
4. Security Engineering: A Guide to Building Dependable Distributed Systems [4]
5. Current Approaches to Authentication in Wireless and Mobile Communications Networks [11]

Description	**1**	**2**	**3**	**4**	**5**	*
Secret key	K_A	K_A	k_A	K	K_A	C
Shared key	$K_{A,B}$	–	k_{AB}	–	–	C
Session key	$K'_{A,B}$	$K_{A,B}$	–	–	–	B
Public key	K_A^+	–	$\mathrm{pub}(k_A)$	KR	$+K_A$	A
Private key	K_A^-	–	$\mathrm{priv}(k_A)$	KR^{-1}	$-K_A$	A
Encrypted	$\{m\}_K$	$\{m\}K$	$\{m\}_K$	$\{m\}_K$	$\{m\}_K$	C
Signed with	$\{m\}^K$	–	–	$\mathrm{sig}_K\{m\}$	–	A
Signed by	$\{m\}^{[A]}$	–	–	–	$A[m]$	A
Send	$A \rightarrow B : \{m\}$	$A \xrightarrow{m} B$	$A \rightsquigarrow B : m$	$A \rightarrow B : m$	$A \rightarrow B : m$	B
Hash value	$H\{m\}$	–	$\mathrm{H}(m)$	$h(m)$	$H(m)$	B
MAC	$MAC\{m\}^K$	–	–	$\mathrm{MAC}_K(m)$	–	B
HMAC	$HMAC\{m\}^K$	–	–	$\mathrm{HMAC}_K(m)$	–	B
Signature	$Sig\{m\}^K$	–	–	–	–	A
Certificate	$Cert\{A, K_A^+\}^{K_{CA}^-}$	–	–	$Cert_{KC^{-1}}(A, KR)$	$Cert_{-K_{CA}}(+K_A)$	B
Certificate by	$Cert\{A, K_A^+\}^{[CA]}$	–	–	–	$CA\langle\langle A\rangle\rangle$	A

In the table, the rightmost column classifies the notation in these groups:

C: The notation is commonly used in textbooks and other publications

B: The notation (or similar) is found in textbooks and other publications

A: The notation is believed to be unique for ASPEN (invented here)

B.2 Notes on the typesetting options

Colors

The LaTeX package `aspen` provides the option `color`:

```
\usepackage[color]{aspen}
```

The package provides different color profiles. The default color profile is called `aspen`. Other color profiles are loaded by assigning a color profile to the `color` option. The following statement will load the same default color profile as the example above:

```
\usepackage[color=aspen]{aspen}
```

In addition, a few color profiles from Pygments are available: `autumn`, `colorful`, `default` (the default profile of Pygments), `emacs`, `friendly`, `gruvboxlight` (called `gruvbox-light` in Pygments), `manni`, and `staroffice`. Figure 5 shows the colors of all the color profiles of the Aspen package.

Other typesetting options

The default way of typesetting the public and the private key of the public-private key pair of A in Aspen is with a + superscript and a − superscript, like K_A^+ and K_A^- respectively. This behavior can be changed with the to package options `tradpubkey` and `tradprivkey`:

```
\usepackage[tradpubkey,tradprivkey]{aspen}
```

The result is that the public key of A will be typeset K_A and the private key of A will be typeset K_A^{-1}.

The default way of typesetting concatenation in Aspen is with the binary operator " . " (used to typeset concatenation of two values or strings). The Aspen package provides three options for typesetting concatenation: " . ", " || ", or " + ". This can be changed by passing a value to the `concat` option of the package. The valid values are `dot`, `dblbar`, and `plus`. The default is " . ". In this example " || " is chosen to be the concatenation operator:

```
\usepackage[concat=dblbar]{aspen}
```

aspen

Value	no	1, *true*
Principle	na	A
Key	kt	K_A
Nonce	nn	N_1
Timestamp	nt	T_s
String	sc	"Hello"
Variable	nv	x, y, z
Function	nf	$H(m)$
Code	go	\key{A}
Label	nl	M_1

autumn

Value	no	1, *true*
Principle	na	A
Key	kt	K_A
Nonce	nn	N_1
Timestamp	nt	T_s
String	sc	"Hello"
Variable	nv	x, y, z
Function	nf	$H(m)$
Code	go	\key{A}
Label	nl	M_1

colorful

Value	no	1, *true*
Principle	na	A
Key	kt	K_A
Nonce	nn	N_1
Timestamp	nt	T_s
String	sc	"Hello"
Variable	nv	x, y, z
Function	nf	$H(m)$
Code	go	\key{A}
Label	nl	M_1

default

Value	no	1, *true*
Principle	na	A
Key	kt	K_A
Nonce	nn	N_1
Timestamp	nt	T_s
String	sc	"Hello"
Variable	nv	x, y, z
Function	nf	$H(m)$
Code	go	\key{A}
Label	nl	M_1

emacs

Value	no	1, *true*
Principle	na	A
Key	kt	K_A
Nonce	nn	N_1
Timestamp	nt	T_s
String	sc	"Hello"
Variable	nv	x, y, z
Function	nf	$H(m)$
Code	go	\key{A}
Label	nl	M_1

friendly

Value	no	1, *true*
Principle	na	A
Key	kt	K_A
Nonce	nn	N_1
Timestamp	nt	T_s
String	sc	"Hello"
Variable	nv	x, y, z
Function	nf	$H(m)$
Code	go	\key{A}
Label	nl	M_1

gruvboxlight

Value	no	1, *true*
Principle	na	A
Key	kt	K_A
Nonce	nn	N_1
Timestamp	nt	T_s
String	sc	"Hello"
Variable	nv	x, y, z
Function	nf	$H(m)$
Code	go	\key{A}
Label	nl	M_1

manni

Value	no	1, *true*
Principle	na	A
Key	kt	K_A
Nonce	nn	N_1
Timestamp	nt	T_s
String	sc	"Hello"
Variable	nv	x, y, z
Function	nf	$H(m)$
Code	go	\key{A}
Label	nl	M_1

staroffice

Value	no	1, *true*
Principle	na	A
Key	kt	K_A
Nonce	nn	N_1
Timestamp	nt	T_s
String	sc	"Hello"
Variable	nv	x, y, z
Function	nf	$H(m)$
Code	go	\key{A}
Label	nl	M_1

Figure 5: The color profiles of the aspen package