Aspen

A suggested security protocol notation

Anders Andersen
UiT The Arctic University of Norway

2026/02/23 18:44:35
(aspen.sty version 1.27)

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" [22], "Exploring Kerberos, the Protocol for Distributed Security in Windows 2000" [12], and "A Formal Semantics for Protocol Narrations" [9]. The notation Aspen presented here is strongly inspired by notations found in these three references, notations found in text books [6], 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" [4, 5], "standard protocol notation" [6], or "protocol narrations" [9].

This text documents the Aspen[1] 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.

The Aspen notation is *not* a formalism, like BAN (Burrows–Abadi–Needham) logic [11], 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: [10, 13, 21].

$$\begin{aligned}
A &\longrightarrow S : \{A, B, N'_A\} \\
S &\longrightarrow A : \{N'_A, B, K_{A,B}, \{K_{A,B}, A\}_{K_{B,S}}\}_{K_{A,S}} \\
A &\longrightarrow B : \{K_{A,B}, A\}_{K_{B,S}} \\
B &\longrightarrow A : \{N'_B\}_{K_{A,B}} \\
A &\longrightarrow B : \{N'_B - 1\}_{K_{A,B}}
\end{aligned}$$

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 [4, 5, 6].

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.
Counter:	I_A, C_B	Counter or indexes can be used to identify a session or a number in a sequence.
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 literature.
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 notation 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» [11] is included (see Section 2.2 and 3.2).

2 The notation

In the description of the notation below, notation that might be obvious is included. It is done for completeness and consistency. For some notation constructs, usage examples are provided. These examples might include notation constructs 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 notation constructs 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 parts of the notation used in the example): $$\mathit{Verify}\left(K_A^+, \{m\}^{K_x}\right) = \mathit{true} \quad \Leftrightarrow \quad K_x = K_A^-$$
$x \rightsquigarrow y$	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 parts of the notation used in the example): $$\mathit{Sig}\{m\}^{K_A^-} \rightsquigarrow \{H\{m\}\}_{K_A^-}$$
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^-).

$K_P^{''}$	A secret key generated from the password P.
$\{m\}$	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 parts of the notation used in the example): $Encrypt(K_{A,B}, m) \rightarrow \{m\}_{K_{A,B}}$
$\{m\}_{\Box}$	In general, a subscripted structured value means an encrypted value or message (a ciphertext), where the subscript $\Box$ 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: $\{m\}_K$
$\{m\}^{\Box}$	In general, a superscripted structured value means a signed value or message, where the superscript $\Box$ 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 $c = \{m\}_{K_{A,B}}$: $Decrypt(K_{A,B}, c) \rightsquigarrow Decrypt\left(K_{A,B}, \{m\}_{K_{A,B}}\right) \rightarrow m$
$Encrypt(K_A^+, m)$	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 $c = \{m\}_{K_A^+}$: $Decrypt(K_A^-, c) \rightsquigarrow Decrypt\left(K_A^-, \{m\}_{K_A^+}\right) \rightarrow m$
$H\{m\}$	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 [8] 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: $HMAC(K_A, m) \rightarrow HMAC\{m\}^{K_A} \rightsquigarrow H\{\overline{K}_A \oplus opad \,.\, H\{\overline{K}_A \oplus ipad \,.\, m\}\}$ $\overline{K}_A = \begin{cases} H\{K_A\} & \text{if } K_A \text{ is larger than block size} \\ K_A & \text{otherwise} \end{cases}$ 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 $[B]$): $Sign([A,B], m) \rightarrow \{m\}^{[A,B]} \rightsquigarrow \{m, Sig\{m\}^{[A,B]}\}$

$\mathit{Sign}(K_{A,B}, m)$	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 $K_{A,B}$): $\mathit{Sign}(K_{A,B}, m) \rightarrow \{m\}^{K_{A,B}} \rightsquigarrow \left\{m, \mathit{Sig}\{m\}^{K_{A,B}}\right\} \rightsquigarrow \left\{m, \{H\{m\}\}_{K_{A,B}}\right\}$
$\mathit{PwKey}(P, s)$	Create a secret key from the password P with the salt s. The salt is optional in the notation. *PBKDF2* is an examples of a password-based key derivation function. The function creates a new secret key: $\mathit{PwKey}(P_1) \rightarrow K''_{P_1}$ $\mathit{PwKey}_{\mathit{PBKDF2}}(P_2, s) \rightarrow K''_{P_2}$
$\mathit{DHPubKey}(K_A^-, p)$	Make a public key K_A^+ (a key share) from the private key K_A^- and the public parameters p, typically used in a Diffie–Hellman key exchange protocol [15] (a Diffie–Hellman key share). The public parameters argument is optional: $\mathit{DHPubKey}(K_A^-) \rightarrow K_A^+$ $\mathit{DHPubKey}(K_B^-, p) \rightarrow K_B^+$ In the original implementation of Diffie-Hellman (Finite Field Diffie–Hellman) the public parameters consists of a agreed upon prime and a primitive root.
$\mathit{DHKey}(K_A^-, K_B^+, p)$	Combine private key K_A^- with public key K_B^+ and the public parameters p (optional) to generate a new secret (shared) key $K_{A,B}$, typically the result of a Diffie–Hellman key exchange protocol [15]. The public parameters argument is optional: $\mathit{DHKey}(K_A^-, K_B^+) \rightarrow K_{A,B}$ $\mathit{DHKey}(K_B^-, K_A^+, p) \rightarrow K_{A,B}$ These functions are one-way functions where the private keys K_A^- and K_B^- can not be calculated (in reasonable time) with only the knowledge of the public keys K_A^+ and K_B^+ and the public parameters p. And as a consequence the new shared secret key $K_{A,B}$ is also difficult (impossible in practice) to calculate.
$\mathit{Verify}(K_A^+, s)$	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: $\mathit{Verify}(K_A^+, s) \rightsquigarrow \mathit{Verify}\left(K_A^+, \{m\}^{[x]}\right) \rightsquigarrow \mathit{Verify}\left(K_A^+, \left\{m, \mathit{Sig}\{m\}^{[x]}\right\}\right) \rightsquigarrow \mathit{Verify}\left(K_A^+, \left\{m, \mathit{Sig}\{m\}^{K_x^-}\right\}\right) \rightsquigarrow \mathit{Verify}\left(K_A^+, \left\{m, \{H\{m\}\}_{K_x^-}\right\}\right):$ $\left.\begin{array}{l} \mathit{Decrypt}\left(K_A^+, \mathit{Sig}\{m\}^{[x]}\right) = H\{m\} \rightsquigarrow \\ \mathit{Decrypt}\left(K_A^+, \mathit{Sig}\{m\}^{K_x^-}\right) = H\{m\} \rightsquigarrow \\ \mathit{Decrypt}\left(K_A^+, \{H\{m\}\}_{K_x^-}\right) = H\{m\} \end{array}\right\} \Leftrightarrow K_A^- = K_x^-$ $\mathit{Verify}(K_A^+, s) \rightarrow \mathit{true} \Leftrightarrow K_A^- = K_x^-$

$Verify([C], s)$	Verify that the signed structured value (message) s is signed by principal C.
$Cert\{A, K_A^+\}^{[C]}$	A certificate where certificate authority C binds identity A to public key K_A^+ (in the example, $\ldots$ 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 certificate authority with private key K_C^- binds identity A to public key K_A^+ (in the example, $\ldots$ is other certificate related meta-data): $\dfrac{Cert\{A, K_A^+\}^{K_C^-} \rightsquigarrow \{A, K_A^+, \ldots\}^{K_C^-}}{Verify(K_C^+, Cert\{A, K_A^+\}^{K_C^-}) \rightsquigarrow Verify(K_C^+, \{A, K_A^+, \ldots\}^{K_C^-}) \rightarrow true}$

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" [11].

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.
$A \Mapsto X$	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, is denoted K_A^-.)

$A \overset{X}{\leftrightharpoons} B$	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 interests 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 notation in text

This section explains how to use ASPEN 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]`	`s`	Optional stars (alternative versions) *Examples:* `\cmd`, `\cmd*`
`O{default}`	Optionals with default value *Examples:* `\cmd`, `\cmd[arg]`	`T`	Optional key types: `*`, `-`, `+`, `!`, `'`, or `"` *Examples:* `\cmd`, `\cmd-`, `\cmd!`
`p`	Optionals in parenthesis *Examples:* `\cmd`, `\cmd(arg)`	`x`	Optionals with magic return *Examples:* `\cmd`, `\cmd[*]`, `\cmd[arg]`
`P{default}`	Parenthesis optionals with default *Examples:* `\cmd`, `\cmd(arg)`	`i`	Optional markers (more details) *Examples:* `\cmd`, `\cmd_{arg}`

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 $\rightarrow$ to specify the arguments of a command created by another command. For example, `\mktval` has one optional argument (`o`) and one mandatory argument (`m`)and returns a new command that has one optional star argument (`*`), one optional bracket size argument (`B`), one optional marker (`i`), and one mandatory argument (`m`):

`\mktval`: `om` $\rightarrow$ `sBim`

The optional magic return type `x` is available for some predefined functions and can be be made available for functions created with the commands `\mkfunc` and `\mkkfunc`. The magic return value `*` will create the return value based on what the function does. For example, if magic return is used with the encrypt function, the return value created will be the encrypted value:

`\encrypt`: `TBimmx`
`\encrypt+{A}{m}[*]` $\rightarrow$ $Encrypt(K_A^+, m) \rightarrow \{m\}_{K_A^+}$

3.1 ASPEN

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

Notation	LaTeX code and description
$=, <, \leq, >, \geq$	`=`,`<`,`\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).
$x \leadsto y$	`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}` $\rightarrow n'_0$

Arguments:	`\anonce`: `m`, `\nonce`: `O{N}m`
	`\anonce{<name>}`, `\nonce[<symbol>]{<id>}`

C_A, I_B	`\acounter{C_A}`,`\counter{B}`, used for indexes or counters. The `\counter` command has an optional first argument to change the symbol (the letter): `\counter[i]{0}` $\rightarrow i_0$

Arguments:	`\acounter`: `m`, `\counter`: `O{I}m`
	`\acounter{<name>}`, `\counter[<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}` $\rightarrow r_z$

Arguments:	`\arandom`: `m`, `\random`: `O{R}m`
	`\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}` $\rightarrow t_0$, `\ttl[l]{1}` $\rightarrow l_1$

Arguments:	`\ats`: `m`, `\ts`: `O{T}m`, `\attl`: `m`, `\ttl`: `O{L}m`
	`\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>}`

$^{-}, ^{+}, ^{\prime}, \ldots$	Some command markers (key type markers) are used throughout the ASPEN LaTeX package to specify the key type involved:
	`*` : Means no specific variant (argument is the key, not the label): `\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}`
	`"` : Mark that it is a key generated from the password: `\key"{P}`
	
	`\key*{K}` → K
	`\encrypted"{P}{\key-{B}}` → $\{K_B^-\}_{K_P^{\prime\prime}}$
	`\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)
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>}`

$K^{"}_{P_1}, K^{"}_{P_2}$	`\key"{P_1}`,`\pwkey{P_2}`, used for keys generated from a password (provided by two different LaTeX commands, where the first is a more compact version; use whatever you prefer). The `\pwkey` command has an optional first argument to change the symbol (the letter): `\pwkey[k]{P}` $\rightarrow k^{"}_P$

Arguments:	`\key`: `Tm`, `\pubkey`: `O{K}m`
	`\key"{<password>}`, `\pwkey[<symbol>]{<password>}`

$[A]$	`\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}` $\rightarrow$ $[A]$

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}}` $\rightarrow$ $K_{M_1\text{–}M_n}$ `\agroup[0][s]{M}`, used when th group member indexes are non-standard: `\key{\agroup[0][s]{M}}` $\rightarrow$ $K_{M_0\text{–}M_s}$ `\agroup*{M}`, typically used in a text when referring to a group (with n members): `\key{\agroup*{M}}` $\rightarrow$ $K_{M_1,\ldots,M_n}$ `\agroup*[0][s]{M}`, used when group member indexes are non-standard: `\key{\agroup*[0][s]{M}}` $\rightarrow$ $K_{M_0,\ldots,M_s}$

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

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

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

$\big\{m\big\}$	`\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}` $\rightarrow$ $\{x\}$ `\sval[big]{\sval{x}}` $\rightarrow$ $\big\{\{x\}\big\}$ `\sval[Big]{\sval[big]{\sval{x}}}` $\rightarrow$ $\Big\{\big\{\{x\}\big\}\Big\}$ `\sval[bigg]{\sval[Big]{\sval[big]{\sval{x}}}}` $\rightarrow$ $\bigg\{\Big\{\big\{\{x\}\big\}\Big\}\bigg\}$ We can even use more size specifiers: `big`, `Big`, `bigg`, `Bigg`, `biggg`, `Biggg`, `bigggg`, `Bigggg`, `biggggg`, and `Bigggggg`: $\left\{\left\{\left\{\left\{\left\{\left\{\Bigg\{\bigg\{\Big\{\big\{\{x\}\big\}\Big\}\bigg\}\Bigg\}\right\}\right\}\right\}\right\}\right\}\right\}$

The optional size argument applies for all Aspen LaTeX 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}`	→	$\mathit{Type}\big\{m\big\}$
`\send[big]{A}{B}{m}`	→	$A \longrightarrow B : \big\{m\big\}$
`\func[big]{Func}{x,y}`	→	$\mathit{Func}\big(x, y\big)$
`\encrypted[big]{A,B}{m}`	→	$\big\{m\big\}_{K_{A,B}}$

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

… `_{…}`, 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}`	→	$\mathit{Encrypt}_{RSA}(K_B^+, m)$
`\decrypt'_{AES}{A,B}{c}`	→	$\mathit{Decrypt}_{AES}(K'_{A,B}, c)$
`\sig-_{DSA}{S}{m}`	→	$\mathit{Sig}_{DSA}\{m\}^{K_S^-}$
`\hmac_{SHA-2}{A}{m}`	→	$\mathit{HMAC}_{SHA\text{-}2}\{m\}^{K_A}$

Arguments: `i` (the symbol used for such optional embellishment in argument specifications)

$\mathit{Type}\{m\}$ `\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}}}
```

$$\rightarrow \quad \mathit{KA}\{C, \mathit{Addr}_C, T_t\}_{K_{S,C}}$$

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}}}
```

$$\rightarrow \quad \mathit{KA}_{RSA}\{C, \mathit{Addr}_C, T_t\}_{K_{S,C}}$$

Arguments: `\tval`: `sBmim`
`\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}}}`	→	$A \longrightarrow B : \big\{H\{m\}\big\}_{K_B^+}$

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]` → $Func(x,y) \to z$

Arguments:	`\func`: `Bmimo`
	`\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}` → $\{m\}_{K'_{A,B}}$ `\encrypted"{P}{m}` → $\{m\}_{K''_P}$

Arguments:	`\encrypted`: `TBimm`
	`\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 (and it supports magic return making straightforward to include the return value): `\encrypt+{B}{m}[\encrypted+{B}{m}]` → $Encrypt(K_B^+, m) \to \{m\}_{K_B^+}$ `\encrypt+{B}{m}[*]` → $Encrypt(K_B^+, m) \to \{m\}_{K_B^+}$

Arguments:	`\encrypt`: `TBimmx`
	`\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]` → $Decrypt(K_B^-, \{m\}_{K_B^+}) \to m$

Arguments:	`\decrypt`: `TBimmo`
	`\decrypt<T>[<size>]_{<marker>}{<key>}{<cipher>}[<returns>]`

$H\{m\}$	`\chash{m}`, used for a cryptographic hash value of m.

Arguments:	`\chash`: `Bim`
	`\chash[<size>]_{<marker>}{<value>}`

$MAC\{m\}^{K_A}$	`\mac{A}{m}`, used for message authentication code of m with K_A.

Arguments:	`\mac`: `TBimm`
	`\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`: `TBimm`
	`\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`: `TBimm`
	`\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}[*]` → $H(m) \to H\{m\}$

Arguments:	`\chashf`: `Bimo`
	`\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}[*]` → $MAC(K_A, m) \to MAC\{m\}^{K_A}$

Arguments:	`\macf`: `TBimmo`
	`\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}[*]` → $CMAC(K_A, m) \to CMAC\{m\}^{K_A}$

Arguments:	`\cmacf`: `TBimmo`
	`\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}[*]` → $HMAC(K_A, m) \to HMAC\{m\}^{K_A}$

Arguments:	`\hmacf`: `TBimmo`
	`\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). The other key type markers can also be used with this command.

Arguments:	`\sig`: `TBimm`
	`\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). The other key type markers can also be used with this command. Since `\sigf` is a function, we can include a return value as an optional last argument (and it supports magic return making straightforward to include the return value).

Arguments:	`\sigf`: `TBimmx`
	`\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). The other key type markers can also be used with this command.

Arguments:	`\signed`: `TBimm`
	`\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). The other key type markers can also be used with this command. Since `\sign` is a function, we can include a return value as an optional last argument (and it supports magic return making straightforward to include the return value).

Arguments:	`\sign`: `TBimmx`
	`\sign<T>[<size>]_{<marker>}{<key>}{<value>}[<returns>]`

$PwKey(P, s)$	`\pwkeyf{P}(s)`, used to create a secret key from a password P. Optionally, a salt value s can be provided Since `\pwkeyf` is a function, we can include a return value as an optional last argument (and it supports magic return making straightforward to include the return value): `\pwkeyf{P_1}(s)[*]` $\to$ $PwKey(P_1, s) \to K_{P_1}^{''}$ `\pwkeyf{P_2}[x]` $\to$ $PwKey(P_2) \to x$

Arguments:	`\pwkeyf`: `sBimpx`
	`\pwkeyf<*>[<size>]_{<marker>}{<password>}(<salt>)[<returns>]`

$DHPubKey(K_A^-, p)$	`\dhpubkeyf{A}(p)`, used to create a public key K_A^+ (a key share) from a private key K_A^- and the optional public parameters p, typically used in a Diffie–Hellman key exchange protocol [15]. `\dhpubkeyf{A}(p)[A]` $\to$ $DHPubKey(K_A^-, p) \to K_A^+$ `\dhpubkeyf*{x}[y]` $\to$ $DHPubKey(x) \to y$

Arguments:	`\dhpubkeyf`: `sBimpo`
	`\dhpubkeyf<*>[<size>]_{<marker>}{<key>}(<parms>)[<returns>]`

$DHKey(K_A^-, K_B^+, p)$	`\dhkeyf{A}{B}(p)`, used to combine a public key K_A^+ with another public key K_B^+ and the optional public parameters p to generate a new secret (shared) key $K_{A,B}$, typically used in a Diffie–Hellman key exchange protocol [15]. `\dhkeyf{A}{B}(p)[A,B]` $\to$ $DHKey(K_A^-, K_B^+, p) \to K_{A,B}$ `\dhkeyf*{x}{y}[z]` $\to$ $DHKey(x, y) \to z$

Arguments:	`\dhkeyf`: `sBimmpo`
	`\dhkeyf<*>[<size>]_{<marker>}{<key>}{<key>}(<parms>)[<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. The other key type markers can also be used with this command.

Arguments:	`\verify`: `TBimmo`
	`\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 certificate authority C. The other key type markers can also be used with this command.

Arguments: `\certificate`: `TBimm`
`\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 certificate authority C). The other key type markers can also be used with this command.

Arguments: `\cert`: `TBimm`
`\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` → `sBim`
`\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` → `TBimm`
`\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}
```

$\rightarrow$ $SMsg\{m\}^{K_A^-}$

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

```
\mkstval[smsg]{SMsg}
\smsg-{S}{m}
```

$\rightarrow$ $SMsg\{m\}_{K_S^-}$

Arguments: `\mkstval`: `om` ⟶ `TBimm`
`\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}}
```

$\rightarrow$ $KA(C, Addr_C, T_s)$

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}}
```

$\rightarrow$ $KA(C, Addr_C, T_s)$

The `\mkfunc` has an optional third and last argument to specify the name of the command used for magic return:

```
\mktval[aset]{Set}]
\mkfunc[asetf]{Set}[aset]
\asetf{\aval{x},\aval{y}}[*]
```

$\rightarrow$ $Set(x,y) \rightarrow Set\{x,y\}$

Arguments: `\mkfunc`: `omo` ⟶ `Bimx`
`\mkfunc[<cmd>]{<name>}[<magic-return>]`
⟶ `\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}}` → $KeyF(K'_A, Addr, T_s)$
	The `\mkkfunc` has an optional third and last argument to specify the name of the command used for magic return:
	`\mkstval[sset]{SSet}]`
	`\mkkfunc[ssetf]{SignSet}[sset]` → $SignSet(K^-_A, x, y) \rightarrow SSet\{x, y\}^{K^-_A}$
	`\ssetf-{A}{\aval{x},\aval{y}}[*]`

Arguments:	`\mkkfunc`: `omo` → `TBimmx`
	`\mkkfunc[<cmd>]{<name>}[<magic-return>]`
	→ `\cmd<T>[<size>]_{<marker>}{<key>}{<arguments>}[<returns>]`

3.2 BAN logic

The table below lists the BAN logic notation with the matching LaTeX commands. This notation is 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}` $\rightarrow$ $A \Mapsto X$
$\sharp(X)$	`\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}` $\rightarrow$ $A \overset{K_{A,B}}{\longleftrightarrow} B$
$\overset{K}{\longmapsto}$	`\thepubkey{K}`, used to state that a key is a public key of someone: `\thepubkey{\key+{A}}\apri{A}` $\rightarrow$ $\overset{K_A^+}{\longmapsto} A$
$\overset{X}{\leftrightharpoons}$	`\asecret{X}`, used to state that a secret (X, in this case) is only known to them: `\apri{A}\asecret{X}\apri{B}` $\rightarrow$ $A \overset{X}{\leftrightharpoons} B$
$\{X\}_K$	`\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}}` $\rightarrow$ $\langle X\rangle_Y$

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 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}
```

$$\begin{array}{ll} M_1 & A \longrightarrow B : \{m_1\}_{K_B^+} \\ M_2 & B \longrightarrow A : \{m_2\}_{K_A^+} \end{array}$$

Notice that each step is separated by the `\\` command. Each step is labeled and can be referred to by its name (`\aref{m1}` $\rightarrow$ M_1, and `\aref{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*}
```

$$\begin{array}{l} A \longrightarrow B : \{m_1\}_{K_B^+} \\ B \longrightarrow A : \{m_2\}_{K_A^+} \end{array}$$

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}[*]} \\
  \astep{\sign-[big]{A}{%
      \encrypted+{B}{m_1}}[*]}
\end{steps}
```

S_1 $\quad Encrypt(K_B^+, m_1) \rightarrow \{m_1\}_{K_B^+}$

S_2 $\quad Sign\big(K_A^-, \{m_1\}_{K_B^+}\big) \rightarrow \{\{m_1\}_{K_B^+}\}^{K_A^-}$

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

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

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

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 set the value of each counter using the standard LaTeX command `\setcounter`. For example, if you want the counter of the `S` labels to start with zero, you add this as the first command inside a `steps` environment: `\setcounter{counterS}{-1}` (see B.3.13 for an example).

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}[*]} \\
  \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 $\quad B : Decrypt\big(K_B^-, \{m_1\}_{K_B^+}\big) \rightarrow m_1$

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*}
```

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

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

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*}
```

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

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

The `\arawstep` command is a low-level command that 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}
```

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

$M_2 \quad$ a – b ; m

The arguments of the commands in the `steps*` environment

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

`\astep`: `sm`
`\astep<*>{<step>}`

`\astepat`: `smm`
`\astepat<*>{<where>}{<step>}`

`\arawstep`: `oooom`
`\arawstep[<sender>][<arrow>][<receiver>][<colon>]{<step>}`

The arguments of the commands in the `steps` environment

`\send`: `sP{M}Bmmmo`
`\send<*>(<type>)[<size>]{<sender>}{<receiver>}{<message>}[<label>]`

`\astep`: `sP{S}mo`
`\astep<*>(<type>){<step>}[<label>]`

`\astepat`: `sP{S}mmo`
`\astepat<*>(<type>){<where>}{<step>}[<label>]`

`\arawstep`: `P{S}oooomo`
`\arawstep(<type>)[<sender>][<arrow>][<receiver>][<colon>]{<step>}[<label>]`

4 Notation usage examples

To illustrate the usability of the notation, we provide a few examples where the notation is used to describe well-known, and not so well-known, security protocols. In the original papers referred to below, you will find the original notations used. The inconsistencies in the notations used in these papers are a major motivation behind Aspen. The following examples are presented here:

• *Original Needham–Schroeder protocol*	LaTeX code: B.3.1
• *Revised Needham–Schroeder protocol*	LaTeX code: B.3.2
• *Otway-Rees protocol*	LaTeX code: B.3.3
• *Kerberos protocol*	LaTeX code: B.3.4
• *Diffie–Hellman key exchange*	LaTeX code: B.3.5
• *Needham–Schroeder public key protocol*	LaTeX code: B.3.6
• *Needham–Schroeder-Lowe public key protocol*	LaTeX code: B.3.7
• *ASW protocol*	LaTeX code: B.3.8
• *Wide-mouthed-frog protocol*	LaTeX code: B.3.9
• *Idealized wide-mouthed-frog protocol*	LaTeX code: B.3.10
• *Wide-mouthed-frog protocol with declarations*	LaTeX code: B.3.11
• *TLS 1.3 Handshake*	LaTeX code: B.3.12
• *SMC: Calculate the mean value*	LaTeX code: B.3.13
• *File sharing with symmetric and public key encryption*	LaTeX code: B.3.14
• *Scalable secure file sharing*	LaTeX code: B.3.15

The three last examples listed are from my own publications. I have included them since they represent the start of my process developing this notation. You will find that my use of notations then was a mixed bag without consistency across publications.

Original Needham–Schroeder protocol

The *Needham–Schroeder protocol* [17] 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 (the *Needham–Schroeder public key protocol* is presented on page 26).

Original Needham–Schroeder protocol

$$\begin{array}{ll}
M_1 & A \longrightarrow S : \{A, B, N'_A\} \\
M_2 & S \longrightarrow A : \left\{N'_A, B, K_{A,B}, \{K_{A,B}, A\}_{K_{B,S}}\right\}_{K_{A,S}} \\
M_3 & A \longrightarrow B : \{K_{A,B}, A\}_{K_{B,S}} \\
M_4 & B \longrightarrow A : \{N'_B\}_{K_{A,B}} \\
M_5 & A \longrightarrow B : \{N'_B - 1\}_{K_{A,B}}
\end{array}$$

N'_A and N'_B are nonces and the shared key $K_{A,B}$ should be fresh, $\sharp(K_{A,B})$. The protocol is vulnerable to a replay attack [14]. See B.3.1 for the LaTeX code.

Revised Needham–Schroeder protocol

The *Revised Needham–Schroeder protocol* [18] addresses a weakness in the protocol related to its vulnerable to a replay attack. A fun fact regarding this suggested revision is its link to my home department, Department of Computer Science at UiT[2]. From [18]:

[2]UiT The Arctic University of Norway, formerly University of Tromsø

In 1986 one of us (RMN) gave a lecture at the University of Tromsø which included the 1978 protocol, the criticism of it, and also a general principle about the use of nonce identifiers. This was that the identifier should always be generated by the party that sought reassurance about the time integrity of a transaction. In discussion Dr Sape J. Mullender of CWI Amsterdam pointed out that this should apply to the reassurance of B against the attack outlined.

The revised version adds two initial messages, M_1 and M_2, between the two principals A and B and the extra nonce N_I' as an identifier of the session:

Revised Needham–Schroeder protocol

$$\begin{array}{ll}
M_1 & A \longrightarrow B : \{A\} \\
M_2 & B \longrightarrow A : \{A, N_I'\}_{K_{B,S}} \\
M_3 & A \longrightarrow S : \big\{A, B, N_A', \{A, N_I'\}_{K_{B,S}}\big\} \\
M_4 & S \longrightarrow A : \big\{N_A', B, K_{A,B}, \{K_{A,B}, A, N_I'\}_{K_{B,S}}\big\}_{K_{A,S}} \\
M_5 & A \longrightarrow B : \{K_{A,B}, A, N_I'\}_{K_{B,S}} \\
M_6 & B \longrightarrow A : \{N_B'\}_{K_{A,B}} \\
M_7 & A \longrightarrow B : \{N_B' - 1\}_{K_{A,B}}
\end{array}$$

The inclusion of the new nonce N_I' prevents any replaying of compromised versions of the message $\{K_{A,B}, A\}_{K_{B,S}}$ since the revised version of the message contains N_I': $\{K_{A,B}, A, N_I'\}_{K_{B,S}}$. This can not be forged since an attacker does not have $K_{B,S}$. See B.3.2 for the LaTeX code of the protocol.

Otway-Rees protocol

The *Otway-Rees protocol* [19] is essential the same as the Revised Needham-Schroeder protocol:

Otway-Rees protocol

$$\begin{array}{ll}
M_1 & A \longrightarrow B : \big\{I_A, A, B, \{N_A', I_A, A, B\}_{K_{A,S}}\big\} \\
M_2 & B \longrightarrow S : \big\{I_A, A, B, \{N_A', I_A, A, B\}_{K_{A,S}}, \{N_B', I_A, A, B\}_{K_{B,S}}\big\} \\
M_3 & S \longrightarrow B : \big\{I_A, \{N_A', K_{A,B}'\}_{K_{A,S}}, \{N_B', K_{A,B}'\}_{K_{B,S}}\big\} \\
M_4 & B \longrightarrow A : \big\{I_A, \{N_A', K_{A,B}'\}_{K_{A,S}}\big\}
\end{array}$$

The identifier I_A prevents the replay attack since an attacker is not able to alter $\{N_A', I_A, A, B\}_{K_{A,S}}$ and $\{N_B', I_A, A, B\}_{K_{B,S}}$. See B.3.3 for the LaTeX code.

Kerberos protocol

The *Kerberos protocol* [22] is based on the Needham-Schroeder protocol, but makes use of timestamps as nonces to remove the problems of the original Needham-Schroeder protocol and to reduce the number of messages needed:

Kerberos protocol

$$\begin{array}{ll}
M_1 & A \longrightarrow S : \{A, B\} \\
M_2 & S \longrightarrow A : \big\{T_s, L, K_{A,B}, B, \{T_s, L, K_{A,B}, A\}_{K_{B,S}}\big\}_{K_{A,S}} \\
M_3 & A \longrightarrow B : \big\{\{T_s, L, K_{A,B}, A\}_{K_{B,S}}, \{A, T_a\}_{K_{A,B}}\big\} \\
M_4 & B \longrightarrow A : \{T_a + 1\}_{K_{A,B}}
\end{array}$$

T_s and T_a are timestamps and L is a lifetime. See B.3.4 for the LaTeX code.

Diffie–Hellman key exchange

The *Diffie–Hellman key exchange* protocol [15] is used to establish a shared symmetric key between two participants over a public channel based a secret (a private key) at each participant and some shared public parameters:

Diffie–Hellman key exchange

$$\begin{array}{lll}
M_1 & A \longrightarrow B : & \{p\} \\
S_1 & A : & \mathit{DHPubKey}(K_A^-, p) \rightarrow K_A^+ \\
S_2 & B : & \mathit{DHPubKey}(K_B^-, p) \rightarrow K_B^+ \\
M_2 & A \longrightarrow B : & \{K_A^+\} \\
M_3 & B \longrightarrow A : & \{K_B^+\} \\
S_3 & A : & \mathit{DHKey}(K_A^-, K_B^+, p) \rightarrow K_{A,B} \\
S_4 & B : & \mathit{DHKey}(K_B^-, K_A^+, p) \rightarrow K_{A,B}
\end{array}$$

The public keys created in step S_1 and S_2 are also called Diffie-Hellman key shares. At the end of this protocol the two participants A and B has a shared secret key $K_{A,B}$. In BAN logic, we can state the following about the outcome $K_{A,B}$:

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

In step M_1 above, participant A send the public parameters p to participant B. Other means of agreeing on these parameters are possible and not important for the protocol itself. See B.3.5 for the LaTeX code of the *Diffie–Hellman key exchange* protocol.

Needham–Schroeder public key protocol

The *Needham–Schroeder public key protocol* [17] intends to provide mutual authentication between two parties:

Needham–Schroeder public key protocol

$$\begin{array}{lll}
M_1 & A \longrightarrow S : & \{A, B\} \\
M_2 & S \longrightarrow A : & \{K_B^+, B\}^{K_S^-} \\
M_3 & A \longrightarrow B : & \{N'_A, A\}_{K_B^+} \\
M_4 & B \longrightarrow S : & \{B, A\} \\
M_5 & S \longrightarrow B : & \{K_A^+, A\}^{K_S^-} \\
M_6 & B \longrightarrow A : & \{N'_A, N'_B\}_{K_A^+} \\
M_7 & A \longrightarrow B : & \{N'_B\}_{K_B^+}
\end{array}$$

This protocol is vulnerable to a man-in-the-middle attack [16]. The fix is however easy.

Needham–Schroeder-Lowe public key protocol

To avoid a man-in-the-middle attack, the *Needham–Schroeder-Lowe public key protocol* [16] includes the identity of the responder in message M_6 of the protocol:

Needham–Schroeder-Lowe public key protocol

M_1 $A \longrightarrow S : \{A, B\}$

M_2 $S \longrightarrow A : \{K_B^+, B\}^{K_S^-}$

M_3 $A \longrightarrow B : \{N_A', A\}_{K_B^+}$

M_4 $B \longrightarrow S : \{B, A\}$

M_5 $S \longrightarrow B : \{K_A^+, A\}^{K_S^-}$

M_6 $B \longrightarrow A : \{N_A', N_B', B\}_{K_A^+}$

M_7 $A \longrightarrow B : \{N_B'\}_{K_B^+}$

See B.3.6 and B.3.7 for the LaTeX code of the *Needham–Schroeder public key protocol* and the *Needham–Schroeder-Lowe public key protocol*.

ASW protocol

The *ASW protocol* is an optimistic fair-exchange protocol for contract signing [7]. This is a good example for Aspen since we in publications find very different (and, if I may say so, hard to read) notations used when presenting the protocol [7, 10]. This is a simplified version of the *exchange* subprotocol (the main part of the protocol) of ASW is shown in Aspen:

ASW exchange protocol

M_1 $O \longrightarrow R : \left\{K_A^+, K_B^+, m, H\{N_O'\}\right\}^{K_O^-}$

M_2 $R \longrightarrow O : \left\{\left\{K_A^+, K_B^+, m, H\{N_O'\}\right\}^{K_O^-}, H\{N_R'\}\right\}^{K_R^-}$

M_3 $O \longrightarrow R : \{N_O'\}$

M_4 $R \longrightarrow O : \{N_R'\}$

Two participants O (originator) and R (recipient) is involved in this subprotocol. In the complete protocol two other subprotocols (*abort* and *resolve*) and a third participant T (third player) is included. See B.3.8 for the LaTeX code.

Wide-mouthed-frog protocol

The *Wide-mouthed-frog protocol* [11] (Section 7, page 25) is a simple protocol that uses shared key cryptography and an authentication server. It transfers a key from A to B via the authentication server S in only two messages by using synchronized clocks and by allowing A to choose the session key:

Wide-mouthed-frog protocol

M_1 $A \longrightarrow S : \left\{A, \{T_A, B, K_{A,B}'\}_{K_{A,S}}\right\}$

M_2 $S \longrightarrow B : \{T_S, A, K_{A,B}'\}_{K_{B,S}}$

A sends a time stamp T_A and session key $K_{A,B}'$ to S. S checks that message M_1 is timely. If it is, it forwards the key $K_{A,B}'$ to B together with its own timestamp T_S in message M_2. B then checks that the timestamp T_S from S is later than any another it has received from S.

Idealized wide-mouthed-frog protocol

The *Idealized wide-mouthed-frog protocol* with BAN logic is shown below:

Idealized wide-mouthed-frog protocol

M_1	$A \longrightarrow S : \{T_A, \{A \overset{K_{A,B}}{\longleftrightarrow} B\}\}_{K_{A,S}}$
M_2	$S \longrightarrow B : \{T_S, A \mid\equiv A \overset{K_{A,B}}{\longleftrightarrow} B\}_{K_{B,S}}$

Wide-mouthed-frog protocol with declarations

In [10], formal declarations as part of protocol narrations is introduced. We can do something similar with BAN logic and ASPEN, where we use BAN logic for the declaration part (D_1–D_2). The following example is similar to their version of the *Wide-mouthed-frog protocol with declarations* (see [10], Table 3, page 487):

Wide-mouthed-frog protocol with declarations

D_1	$A \overset{K_{A,S}}{\longleftrightarrow} S; B \overset{K_{B,S}}{\longleftrightarrow} S$
D_2	$A \triangleleft K'_{A,B}; \sharp(K'_{A,B}); A \triangleleft m$
M_1	$A \longrightarrow S : \{A, \{T_A, B, K'_{A,B}\}_{K_{A,S}}\}$
M_2	$S \longrightarrow B : \{T_S, A, K'_{A,B}\}_{K_{B,S}}$

See B.3.9, B.3.10 and B.3.11 for the LaTeX code of the *Wide-mouthed-frog protocol*, the *Idealized wide-mouthed-frog protocol* and the *Wide-mouthed-frog protocol with declarations*.

TLS 1.3 Handshake

The *TLS 1.3 Handshake* [20] has three stages. Stage one is the key exchange, stage two is the server parameters, and stage three is authentication (we only include server authentication in the example). In the steps below stage one and three are included, the key exchange algorithm used is DHE (Ephemeral Diffie-Hellman) and we ignore PSK (Pre-Shared Key):

TLS 1.3 Handshake

S_1	$C : \sharp(K^-_{C_e}); \mathit{DHPubKey}(K^-_{C_e}) \rightarrow K^+_{C_e}$
M_1	$C \longrightarrow S : \{\ldots, N'_C, K^+_{C_e}, \ldots\}$
S_2	$S : \sharp(K^-_{S_e}); \mathit{DHPubKey}(K^-_{S_e}) \rightarrow K^+_{S_e}$
M_2	$S \longrightarrow C : \{\ldots, N'_S, K^+_{S_e}, \mathit{Cert}\{S, K^+_S\}^{K^-_{CA}}, \{N'_C, N'_S, \ldots\}^{K^-_S}, \ldots\}$
S_3	$C : \mathit{Verify}(K^+_{CA}, \mathit{Cert}\{S, K^+_S\}^{K^-_{CA}}) \rightarrow \mathit{true}; \mathit{Verify}(K^+_S, \{N'_C, N'_S, \ldots\}^{K^-_S}) \rightarrow \mathit{true}$
S_4	$C : \mathit{DHKey}(K^-_{C_e}, K^+_{S_e}) \rightarrow K_{C,S}$
S_5	$S : \mathit{DHKey}(K^-_{S_e}, K^+_{C_e}) \rightarrow K_{C,S}$
M_3	$C \longrightarrow S : \{\ldots\}_{K_{C,S}}$
M_4	$S \longrightarrow C : \{\ldots\}_{K_{C,S}}$

The example above is somewhat simplified to make it easier to follow (some details are hidden and some optional pathways are ignored). After message M_2 and step S_5 the client C and the server S share a new fresh encryption key $K_{C,S}$ and the rest of the TLS handshake is encrypted. Message M_3 and message M_4 are the client handshake finished and the server handshake finished messages, respectively.

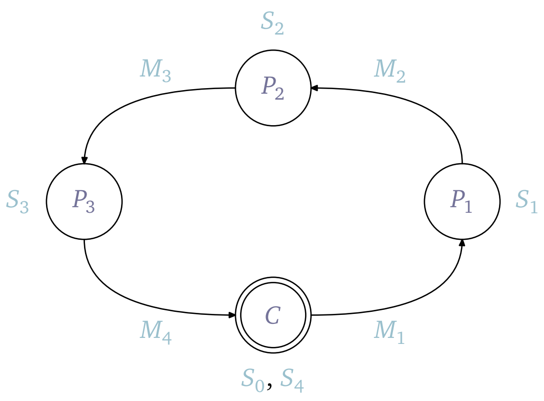

Figure 3: SMC: Calculate the mean value

SMC: Calculate the mean value

In [3], *SMC* (secure multi-party computation) algorithms for analyzing health data were discussed. The first algorithm from this paper is used to calculate the mean value of three values V_1, V_2 and V_3 from three participants P_1, P_2 and P_3 without sharing any knowledge about the individual values. A coordinator C coordinates the calculation. The messages between from P_{i-1} to P_i have the following structure (we can say that participant P_0 and P_4 is the same individual with an alias C):

$$\Big\{\{\{V_i'\}^{K^-_{P_{i-1}}}\}_{K^+_{P_i}}, \{\{P_{i-1}, P_{i+1}\}^{K^-_C}\}_{K^+_{P_i}}, \{\{P_i, P_{i+2}\}^{K^-_C}\}_{K^+_{P_{i+1}}}, \dots\Big\}$$

The first part of the message is the intermediate value received at participant P_i signed by the previous participant in the calculation P_{i-1}. The second part is each path in the calculation (where the input came from and where to send the intermediate result). The current participant P_i can decrypt the first element that says the it should expect the input value from participant P_{i-1} and it should forwards the intermediate result of its calculation to participant P_{i+1}. Each such element is signed by the coordinator and encrypted with the public key of the participant that should be able to read this information. We can now write the protocol used for the calculation of the mean value M using the Aspen notation (we ignore the signature verification steps at each participant):

SMC: Calculate the mean value

$$\begin{array}{lrl}
S_0 & C & : V_0' = R_0';\ \sharp(V_0') \\
M_1 & C \longrightarrow P_1 & : \Big\{\{\{V_0'\}^{K^-_C}\}_{K^+_{P_1}}, \{\{C, P_2\}^{K^-_C}\}_{K^+_{P_1}}, \{\{P_1, P_3\}^{K^-_C}\}_{K^+_{P_2}}, \{\{P_2, C\}^{K^-_C}\}_{K^+_{P_3}}\Big\} \\
S_1 & P_1 & : V_1' = V_0' + V_1 \\
M_2 & P_1 \longrightarrow P_2 & : \Big\{\{\{V_1'\}^{K^-_{P_1}}\}_{K^+_{P_2}}, \{\{P_1, P_3\}^{K^-_C}\}_{K^+_{P_2}}, \{\{P_2, C\}^{K^-_C}\}_{K^+_{P_3}}\Big\} \\
S_2 & P_2 & : V_2' = V_1' + V_2 \\
M_3 & P_2 \longrightarrow P_3 & : \Big\{\{\{V_2'\}^{K^-_{P_2}}\}_{K^+_{P_3}}, \{\{P_2, C\}^{K^-_C}\}_{K^+_{P_3}}\Big\} \\
S_3 & P_3 & : V_3' = V_2' + V_3 \\
M_4 & P_3 \longrightarrow C & : \Big\{\{\{V_3'\}^{K^-_{P_3}}\}_{K^+_C}\Big\} \\
S_4 & C & : M = (V_3' - V_0')/3
\end{array}$$

Figure 3 illustrates the participants and each step and message of the calculation. See B.3.13 for the LaTeX code.

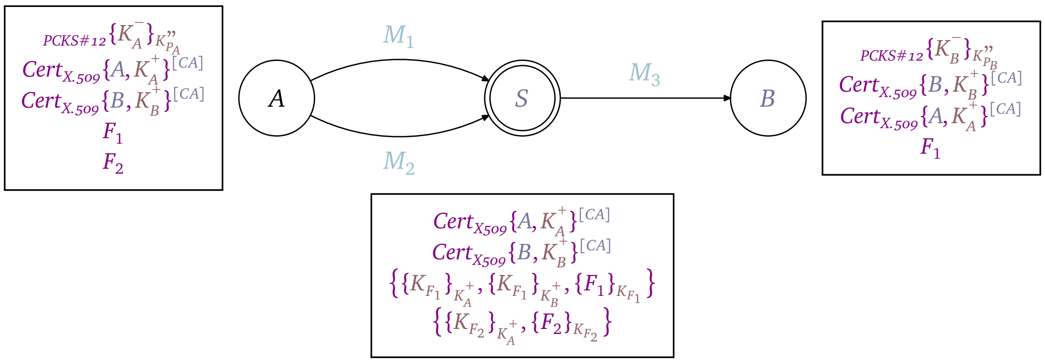
$_{PCKS\#12}\{K_A^-\}_{K_{P_A}^{”}}$
$Cert_{X.509}\{A, K_A^+\}^{[CA]}$
$Cert_{X.509}\{B, K_B^+\}^{[CA]}$
F_1
F_2
M_1
M_2
M_3
A
S
B
$_{PCKS\#12}\{K_B^-\}_{K_{P_B}^{”}}$
$Cert_{X.509}\{B, K_B^+\}^{[CA]}$
$Cert_{X.509}\{A, K_A^+\}^{[CA]}$
F_1
$Cert_{X509}\{A, K_A^+\}^{[CA]}$
$Cert_{X509}\{B, K_B^+\}^{[CA]}$
$\left\{\{K_{F_1}\}_{K_A^+}, \{K_{F_1}\}_{K_B^+}, \{F_1\}_{K_{F_1}}\right\}$
$\left\{\{K_{F_2}\}_{K_A^+}, \{F_2\}_{K_{F_2}}\right\}$

Figure 4: File sharing with symmetric and public key encryption

File sharing with symmetric and public key encryption

Cloud Security Infrastructure (CSI) [2] is a scalable cloud storage architecture with focus on privacy. In [2], a figure (Figure 3) illustrates how a user can share a file with another user using a combination of symmetric (secret) key and public key encryption. In Figure 4 and in the protocol shown below, the intent of original figure in [2]is recreated using Aspen. In the steps below, A uploads two files F_1 and F_2 to a server, where file F_1 is shared with B:

File sharing with symmetric and public key encryption

$$\begin{array}{lrl}
S_1 & & A : \sharp(K_{F_1}); \sharp(K_{F_2}) \\
S_2 & & A : Encrypt(K_{F_1}, F_1) \rightarrow \{F_1\}_{K_{F_1}}; Encrypt(K_{F_2}, F_2) \rightarrow \{F_2\}_{K_{F_2}} \\
S_3 & & A : Encrypt(K_A^+, K_{F_1}) \rightarrow \{K_{F_1}\}_{K_A^+}; Encrypt(K_B^+, K_{F_1}) \rightarrow \{K_{F_1}\}_{K_B^+} \\
S_4 & & A : Encrypt(K_A^+, K_{F_2}) \rightarrow \{K_{F_2}\}_{K_A^+} \\
M_1 & A \longrightarrow S & : \left\{\{K_{F_1}\}_{K_A^+}, \{K_{F_1}\}_{K_B^+}, \{F_1\}_{K_{F_1}}\right\} \\
M_2 & A \longrightarrow S & : \left\{\{K_{F_2}\}_{K_A^+}, \{F_2\}_{K_{F_2}}\right\} \\
M_3 & S \longrightarrow B & : \left\{\{K_{F_1}\}_{K_A^+}, \{K_{F_1}\}_{K_B^+}, \{F_1\}_{K_{F_1}}\right\} \\
S_5 & & B : Decrypt(K_{P_B}^{”}, \{K_B^-\}_{K_{P_B}^{”}}) \rightarrow K_B^-; Decrypt(K_B^-, \{K_{F_1}\}_{K_B^+}) \rightarrow K_{F_1} \\
S_6 & & B : Decrypt(K_{F_1}, \{F_1\}_{K_{F_1}}) \rightarrow F_1
\end{array}$$

In Figure 4, the boxes illustrate what is stored at each principal (data at-rest) and the arrows illustrate the messages sent between the principals (data in-transit). Each messages in the figure is labeled with the same labels used in the steps shown above.

Scalable secure file sharing

The Cloud Security Infrastructure (CSI) [2] has come up with the concept of a *store* and a *share* to achieve scalable secure file sharing. Every principal has a store that is represented by a public and private key pair ($K_{S_A}^+$ and $K_{S_A}^-$ for A in the example below). When a file is shared, a new store called a share is created. In the example below a share represented by the public and private key pair $K_{S_{A,B}}^+$ and $K_{S_{A,B}}^-$ is used to share file F_1 between A and B. Each store (and share) also has a symmetric random encryption key associated with them (K_{S_A} and $K_{S_{A,B}}$ in the example) used to encrypt the private keys of the stores:

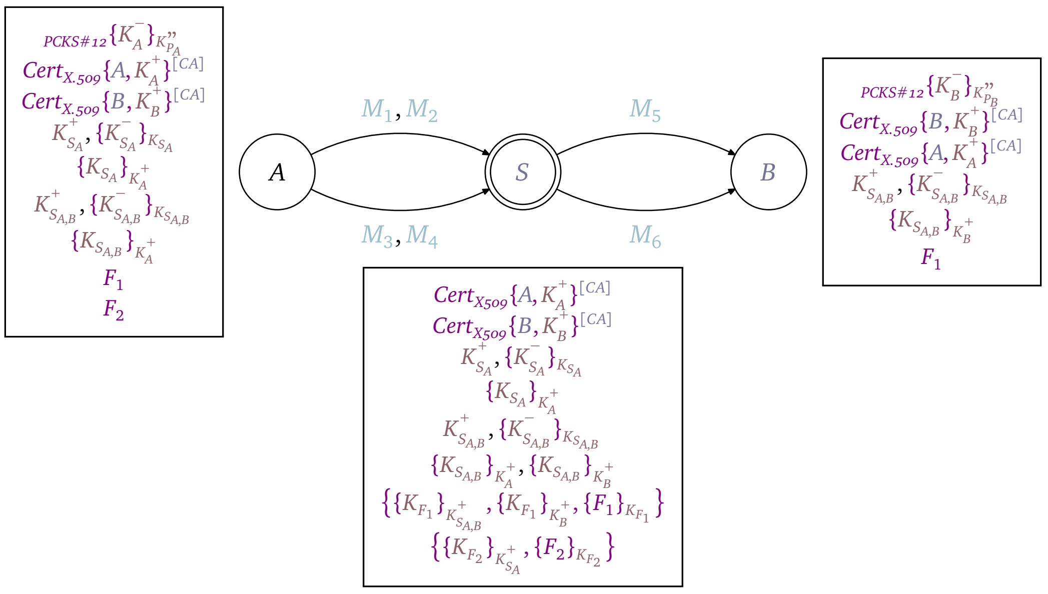
$_{PCKS\#12}\{K_A^-\}_{K_{P_A}^{"}}$
$Cert_{X.509}\{A, K_A^+\}^{[CA]}$
$Cert_{X.509}\{B, K_B^+\}^{[CA]}$
$K_{S_A}^+, \{K_{S_A}^-\}_{K_{S_A}}$
$\{K_{S_A}\}_{K_A^+}$
$K_{S_{A,B}}^+, \{K_{S_{A,B}}^-\}_{K_{S_{A,B}}}$
$\{K_{S_{A,B}}\}_{K_A^+}$
F_1
F_2
M_1, M_2
M_5
A
S
B
M_3, M_4
M_6
$_{PCKS\#12}\{K_B^-\}_{K_{P_B}^{"}}$
$Cert_{X.509}\{B, K_B^+\}^{[CA]}$
$Cert_{X.509}\{A, K_A^+\}^{[CA]}$
$K_{S_{A,B}}^+, \{K_{S_{A,B}}^-\}_{K_{S_{A,B}}}$
$\{K_{S_{A,B}}\}_{K_B^+}$
F_1
$Cert_{X509}\{A, K_A^+\}^{[CA]}$
$Cert_{X509}\{B, K_B^+\}^{[CA]}$
$K_{S_A}^+, \{K_{S_A}^-\}_{K_{S_A}}$
$\{K_{S_A}\}_{K_A^+}$
$K_{S_{A,B}}^+, \{K_{S_{A,B}}^-\}_{K_{S_{A,B}}}$
$\{K_{S_{A,B}}\}_{K_A^+}, \{K_{S_{A,B}}\}_{K_B^+}$
$\{\{K_{F_1}\}_{K_{S_{A,B}}^+}, \{K_{F_1}\}_{K_B^+}, \{F_1\}_{K_{F_1}}\}$
$\{\{K_{F_2}\}_{K_{S_A}^+}, \{F_2\}_{K_{F_2}}\}$

Figure 5: Scalable secure file sharing

Scalable secure file sharing

S_1		$A : \sharp(K_{F_1}); \sharp(K_{F_2}); \sharp(K_{S_A}); \sharp(K_{S_{A,B}}); \sharp(K_{S_A}^-); \sharp(K_{S_A}^+); \sharp(K_{S_{A,B}}^-); \sharp(K_{S_{A,B}}^+)$
S_2		$A : Encrypt(K_{F_1}, F_1) \rightarrow \{F_1\}_{K_{F_1}}; Encrypt(K_{F_2}, F_2) \rightarrow \{F_2\}_{K_{F_2}}$
S_3		$A : Encrypt(K_{S_{A,B}}^+, K_{F_1}) \rightarrow \{K_{F_1}\}_{K_{S_{A,B}}^+}; Encrypt(K_{S_A}^+, K_{F_2}) \rightarrow \{K_{F_2}\}_{K_{S_A}^+}$
S_4		$A : Encrypt(K_{S_A}, K_{S_A}^-) \rightarrow \{K_{S_A}^-\}_{K_{S_A}}; Encrypt(K_{S_{A,B}}, K_{S_{A,B}}^-) \rightarrow \{K_{S_{A,B}}^-\}_{K_{S_{A,B}}}$
S_5		$A : Encrypt(K_A^+, K_{S_A}) \rightarrow \{K_{S_A}\}_{K_A^+}$
S_6		$A : Encrypt(K_A^+, K_{S_{A,B}}) \rightarrow \{K_{S_{A,B}}\}_{K_A^+}; Encrypt(K_B^+, K_{S_{A,B}}) \rightarrow \{K_{S_{A,B}}\}_{K_B^+}$
M_1	$A \longrightarrow S$	$: \{\{K_{S_A}\}_{K_A^+}, \{K_{S_{A,B}}\}_{K_A^+}, \{K_{S_{A,B}}\}_{K_B^+}\}$
M_2	$A \longrightarrow S$	$: \{\{K_{S_A}^-\}_{K_{S_A}}, \{K_{S_{A,B}}^-\}_{K_{S_{A,B}}}\}$
M_3	$A \longrightarrow S$	$: \{\{K_{F_1}\}_{K_{S_{A,B}}^+}, \{F_1\}_{K_{F_1}}\}$
M_4	$A \longrightarrow S$	$: \{\{K_{F_2}\}_{K_{S_A}^+}, \{F_2\}_{K_{F_2}}\}$
M_5	$S \longrightarrow B$	$: \{\{K_{S_{A,B}}\}_{K_B^+}, \{K_{S_{A,B}}^-\}_{K_{S_{A,B}}}\}$
M_6	$S \longrightarrow B$	$: \{\{K_{F_1}\}_{K_{S_{A,B}}^+}, \{F_1\}_{K_{F_1}}\}$
S_7		$B : Decrypt(K_{P_B}^{"}, \{K_B^-\}_{K_{P_B}^{"}}) \rightarrow K_B^-; Decrypt(K_B^-, \{K_{S_{A,B}}\}_{K_B^+}) \rightarrow K_{S_{A,B}}$
S_8		$B : Decrypt(K_{S_{A,B}}, \{K_{S_{A,B}}^-\}_{K_{S_{A,B}}}) \rightarrow K_{S_{A,B}}^-; Decrypt(K_{S_{A,B}}^-, \{K_{F_1}\}_{K_{S_{A,B}}^+}) \rightarrow K_{F_1}$
S_9		$B : Decrypt(K_{F_1}, \{F_1\}_{K_{F_1}}) \rightarrow F_1$

In Figure 5 presenting the example, the boxes illustrate what is stored at each principal (data at-rest) and the arrows illustrate the messages sent between the principals (data in-transit). Each messages in the figure is labeled with the same labels used in the steps shown above. Figure 5 is a recreation of a figure in [2] (Figure 4) illustrating the same example as the one above.

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] Andersen, A., Hardersen, T., Schirmer, N.: Privacy for cloud storage. In: Reimer, H., Pohlmann, N., Schneider, W. (eds.) ISSE 2014 Securing Electronic Business Processes; Highlights of the Information Security Solutions Europe 2014 Conference. Springer-Verlag, Brussels, Belgium (Oct 2014)

[3] Andersen, A., Yigzaw, K.Y., Karlsen, R.: Privacy preserving health data processing. In: Healthcom'14, 16th International Conference on E-health Networking, Application & Services. IEEE, Natal, Brazil (Oct 2014)

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

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

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

[7] Asokan, N., Shoup, V., Waidner, M.: Asynchronous protocols for optimistic fair exchange. In: IEEE Symposium on Security and Privacy. pp. 86–99 (1998). 10.1109/SECPRI.1998.674826

[8] Bellare, M., Canetti, R., Krawczyk, H.: Keying hash functions for message authentication. In: Koblitz, N. (ed.) Advances in Cryptology — CRYPTO'96. Lecture Notes in Computer Science, vol. 1109, pp. 1–15. Springer-Verlag (Aug 1996). 10.1007/3-540-68697-5_1

[9] 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

[10] 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

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

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

[13] 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)

[14] Denning, D.E., Sacco, G.M.: Timestamps in key distribution protocols. Communications of the ACM **24**(8), 533–536 (Aug 1981). doi.org/10.1145/358722.358740

[15] Diffie, W., Hellman, M.E.: New directions in cryptography. IEEE Transactions on Information Theory **22**(6), 644–654 (Nov 1976)

[16] Lowe, G.: An attack on the Needham-Schroeder public-key authentication protocol. Information Processing Letters **56**(3), 131–136 (Nov 1995). 10.1016/0020-0190(95)00144-2

[17] 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

[18] Needham, R., Schroeder, M.: Authentication revisited. Operating Systems Review **21**(1), 7 (Jan 1987). 10.1145/24592.24593

[19] Otway, D., Rees, O.: Efficient and timely mutual authentication. Operating Systems Review **21**(1), 8–10 (Jan 1987). 10.1145/24592.24594

[20] Rescorla, E.: The transport layer security (tls) protocol version 1.3. Request for Comments 8446, Internet Engineering Task Force (IETF), Mozilla (Aug 2018)

[21] 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)

[22] 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. The notation is also influenced by my background as a programmer and not a mathematician or a theoretical computer scientist. 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.

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 [22]
3. A formal semantics for protocol narrations [10]
4. Security Engineering: A Guide to Building Dependable Distributed Systems [6]
5. Current Approaches to Authentication in Wireless and Mobile Communications Networks [21]

Description	1	2	3	4	5	*
Secret key	K_A	K_A	k_A	K	K_A	C
Shared key	$K_{A,B}$	–	$k_{A\,B}$	–	–	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	$A \rightarrow B : \{m\}$	$\textcircled{A} \xrightarrow{m} \textcircled{B}$	$A \rightsquigarrow B : m$	$A \rightarrow B : m$	$A \rightarrow B : m$	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 6 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 6: The color profiles of the `aspen` package

B.3 Notation example listing

In this Section, all notation examples with their LaTeX code is listed without comments and any explanation.

B.3.1 Original Needham–Schroeder protocol

```
\begin{steps}
  \send{A}{S}{\apri{A}, \apri{B}, \nonce{A}}[ons:1] \\
  \send*{S}{A}{\encrypted[big]{A,S}{\nonce{A}, \apri{B},
      \key{A,B}, \encrypted{B,S}{\key{A,B}, \apri{A}}}}[ons:2] \\
  \send*{A}{B}{\encrypted{B,S}{\key{A,B}, \apri{A}}}[ons:3] \\
  \send*{B}{A}{\encrypted{A,B}{\nonce{B}}}[ons:4] \\
  \send*{A}{B}{\encrypted{A,B}{\nonce{B} - 1}}[ons:5]
\end{steps}
```

$$\begin{array}{ll}
M_1 & A \longrightarrow S : \{A, B, N'_A\} \\
M_2 & S \longrightarrow A : \big\{N'_A, B, K_{A,B}, \{K_{A,B}, A\}_{K_{B,S}}\big\}_{K_{A,S}} \\
M_3 & A \longrightarrow B : \{K_{A,B}, A\}_{K_{B,S}} \\
M_4 & B \longrightarrow A : \{N'_B\}_{K_{A,B}} \\
M_5 & A \longrightarrow B : \{N'_B - 1\}_{K_{A,B}}
\end{array}$$

B.3.2 Revised Needham–Schroeder protocol

```
\begin{steps}
  \send{A}{B}{\apri{A}}[rns:1] \\
  \send*{B}{A}{\encrypted{B,S}{\apri{A},\nonce{I}}}[rns:2] \\
  \send[big]{A}{S}{\apri{A}, \apri{B}, \nonce{A},
    \encrypted{B,S}{\apri{A},\nonce{I}}}[rns:3] \\
  \send*{S}{A}{\encrypted[big]{A,S}{\nonce{A}, \apri{B},
      \key{A,B}, \encrypted{B,S}{\key{A,B}, \apri{A}, \nonce{I}}}}[rns:4] \\
  \send*{A}{B}{\encrypted{B,S}{\key{A,B}, \apri{A}, \nonce{I}}}[rns:5] \\
  \send*{B}{A}{\encrypted{A,B}{\nonce{B}}}[rns:6] \\
  \send*{A}{B}{\encrypted{A,B}{\nonce{B} - 1}}[rns:7]
\end{steps}
```

$$\begin{array}{ll}
M_1 & A \longrightarrow B : \{A\} \\
M_2 & B \longrightarrow A : \{A, N'_I\}_{K_{B,S}} \\
M_3 & A \longrightarrow S : \big\{A, B, N'_A, \{A, N'_I\}_{K_{B,S}}\big\} \\
M_4 & S \longrightarrow A : \big\{N'_A, B, K_{A,B}, \{K_{A,B}, A, N'_I\}_{K_{B,S}}\big\}_{K_{A,S}} \\
M_5 & A \longrightarrow B : \{K_{A,B}, A, N'_I\}_{K_{B,S}} \\
M_6 & B \longrightarrow A : \{N'_B\}_{K_{A,B}} \\
M_7 & A \longrightarrow B : \{N'_B - 1\}_{K_{A,B}}
\end{array}$$

B.3.3 Otway-Rees protocol

```
\begin{steps}
  \send[big]{A}{B}{\counter{A}, \apri{A}, \apri{B},
    \encrypted{A,S}{\nonce{A}, \counter{A}, \apri{A}, \apri{B}}}[or:1] \\
  \send[big]{B}{S}{\counter{A}, \apri{A}, \apri{B},
    \encrypted{A,S}{\nonce{A}, \counter{A}, \apri{A}, \apri{B}},
    \encrypted{B,S}{\nonce{B}, \counter{A}, \apri{A}, \apri{B}}}[or:2] \\
  \send[big]{S}{B}{\counter{A},
    \encrypted{A,S}{\nonce{A}, \key'{A,B}},
    \encrypted{B,S}{\nonce{B}, \key'{A,B}}}[or:3] \\
  \send[big]{B}{A}{\counter{A},
    \encrypted{A,S}{\nonce{A}, \key'{A,B}}}[or:4]
\end{steps}
```

$$\begin{array}{ll}
M_1 & A \longrightarrow B : \left\{I_A, A, B, \{N'_A, I_A, A, B\}_{K_{A,S}}\right\} \\
M_2 & B \longrightarrow S : \left\{I_A, A, B, \{N'_A, I_A, A, B\}_{K_{A,S}}, \{N'_B, I_A, A, B\}_{K_{B,S}}\right\} \\
M_3 & S \longrightarrow B : \left\{I_A, \{N'_A, K'_{A,B}\}_{K_{A,S}}, \{N'_B, K'_{A,B}\}_{K_{B,S}}\right\} \\
M_4 & B \longrightarrow A : \left\{I_A, \{N'_A, K'_{A,B}\}_{K_{A,S}}\right\}
\end{array}$$

B.3.4 Kerberos protocol

```
\begin{steps}
  \send{A}{S}{\apri{A}, \apri{B}} \\
  \send*{S}{A}{\encrypted[big]{A,S}{\ts{s}, \ttl{}, \key{A,B},
      \apri{B}, \encrypted{B,S}{\ts{s}, \ttl{}, \key{A,B}, \apri{A}}}} \\
  \send[big]{A}{B}{\encrypted{B,S}{\ts{s}, \ttl{}, \key{A,B},
      \apri{A}}, \encrypted{A,B}{\apri{A}, \ts{a}}} \\
  \send*{B}{A}{\encrypted{A,B}{\ts{a} + 1}}
\end{steps}
```

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

B.3.5 Diffie–Hellman key exchange

```
\begin{steps}
  \send{A}{B}{\aval{p}}[dh-ex:1] \\
  \astepat*{A}{\dhpubkeyf{A}(p)[A]}[dh-ex:2] \\
  \astepat*{B}{\dhpubkeyf{B}(p)[B]}[dh-ex:3] \\
  \send{A}{B}{\key+{A}}[dh-ex:4] \\
  \send{B}{A}{\key+{B}}[dh-ex:5] \\
  \astepat*{A}{\dhkeyf{A}{B}(p)[A,B]}[dh-ex:6] \\
  \astepat*{B}{\dhkeyf{B}{A}(p)[A,B]}[dh-ex:7]
\end{steps}
```

M_1	$A \longrightarrow B : \{p\}$
S_1	$A : \mathit{DHPubKey}(K_A^-, p) \rightarrow K_A^+$
S_2	$B : \mathit{DHPubKey}(K_B^-, p) \rightarrow K_B^+$
M_2	$A \longrightarrow B : \{K_A^+\}$
M_3	$B \longrightarrow A : \{K_B^+\}$
S_3	$A : \mathit{DHKey}(K_A^-, K_B^+, p) \rightarrow K_{A,B}$
S_4	$B : \mathit{DHKey}(K_B^-, K_A^+, p) \rightarrow K_{A,B}$

B.3.6 Needham–Schroeder public key protocol

```
\begin{steps}
  \send{A}{S}{\apri{A}, \apri{B}}[ns-pk:1] \\
  \send*{S}{A}{\signed-{S}{\key+{B},\apri{B}}}[ns-pk:2] \\
  \send*{A}{B}{\encrypted+{B}{\nonce{A},\apri{A}}}[ns-pk:3] \\
  \send{B}{S}{\apri{B}, \apri{A}}[ns-pk:4] \\
  \send*{S}{B}{\signed-{S}{\key+{A},\apri{A}}}[ns-pk:5] \\
  \send*{B}{A}{\encrypted+{A}{\nonce{A},\nonce{B}}}[ns-pk:6] \\
  \send*{A}{B}{\encrypted+{B}{\nonce{B}}}[ns-pk:7]
\end{steps}
```

M_1	$A \longrightarrow S : \{A, B\}$
M_2	$S \longrightarrow A : \{K_B^+, B\}^{K_S^-}$
M_3	$A \longrightarrow B : \{N_A', A\}_{K_B^+}$
M_4	$B \longrightarrow S : \{B, A\}$
M_5	$S \longrightarrow B : \{K_A^+, A\}^{K_S^-}$
M_6	$B \longrightarrow A : \{N_A', N_B'\}_{K_A^+}$
M_7	$A \longrightarrow B : \{N_B'\}_{K_B^+}$

B.3.7 Needham–Schroeder-Lowe public key protocol

```
\begin{steps}
  \send{A}{S}{\apri{A}, \apri{B}}[nsl-pk:1] \\
  \send*{S}{A}{\signed-{S}{\key+{B},\apri{B}}}[nsl-pk:2] \\
  \send*{A}{B}{\encrypted+{B}{\nonce{A},\apri{A}}}[nsl-pk:3] \\
  \send{B}{S}{\apri{B}, \apri{A}}[nsl-pk:4] \\
  \send*{S}{B}{\signed-{S}{\key+{A},\apri{A}}}[nsl-pk:5] \\
  \send*{B}{A}{\encrypted+{A}{\nonce{A},\nonce{B},\apri{B}}}[nsl-pk:6] \\
  \send*{A}{B}{\encrypted+{B}{\nonce{B}}}[nsl-pk:7]
\end{steps}
```

$M_1 \quad A \longrightarrow S : \{A, B\}$

$M_2 \quad S \longrightarrow A : \{K_B^+, B\}^{K_S^-}$

$M_3 \quad A \longrightarrow B : \{N'_A, A\}_{K_B^+}$

$M_4 \quad B \longrightarrow S : \{B, A\}$

$M_5 \quad S \longrightarrow B : \{K_A^+, A\}^{K_S^-}$

$M_6 \quad B \longrightarrow A : \{N'_A, N'_B, B\}_{K_A^+}$

$M_7 \quad A \longrightarrow B : \{N'_B\}_{K_B^+}$

B.3.8 ASW exchange protocol

```
\begin{steps}
  \send*{O}{R}{\signed-[big]{O}{%
      \key+{A},\key+{B},m,\chash{\nonce{O}}}}[asW:1] \\
  \send*{R}{O}{\signed-[Big]{R}{%
      \signed-[big]{O}{\key+{A},\key+{B},m,\chash{\nonce{O}}},
      \chash{\nonce{R}}}}[asw:2] \\
  \send{O}{R}{\nonce{O}}[asw:3] \\
  \send{R}{O}{\nonce{R}}[asw:4]
\end{steps}
```

$M_1 \quad O \longrightarrow R : \big\{K_A^+, K_B^+, m, H\{N'_O\}\big\}^{K_O^-}$

$M_2 \quad R \longrightarrow O : \Big\{\big\{K_A^+, K_B^+, m, H\{N'_O\}\big\}^{K_O^-}, H\{N'_R\}\Big\}^{K_R^-}$

$M_3 \quad O \longrightarrow R : \{N'_O\}$

$M_4 \quad R \longrightarrow O : \{N'_R\}$

B.3.9 Wide-mouthed-frog protocol

```
\begin{steps}
  \send[big]{A}{S}{A, \encrypted{A,S}{\ts{A}, \apri{B}, \key'{A,B}}}[wmf:1] \\
  \send*[big]{S}{B}{\encrypted{B,S}{\ts{S}, \apri{A}, \key'{A,B}}}[wmf:2]
\end{steps}
```

$M_1 \quad A \longrightarrow S : \big\{A, \{T_A, B, K'_{A,B}\}_{K_{A,S}}\big\}$

$M_2 \quad S \longrightarrow B : \{T_S, A, K'_{A,B}\}_{K_{B,S}}$

B.3.10 Idealized wide-mouthed-frog protocol

```
\begin{steps}
  \send*{A}{S}{\encrypted[big]{A,S}{\ts{A},
      \{\apri{A}\asharedkey{\key{A,B}}\apri{B}\}}}[iwmf:1] \\
  \send*[big]{S}{B}{\encrypted{B,S}{\ts{S},
      \apri{A}\believes\apri{A}\asharedkey{\key{A,B}}\apri{B}}}[iwmf:2]
\end{steps}
```

$M_1 \quad A \longrightarrow S : \big\{T_A, \{A \overset{K_{A,B}}{\longleftrightarrow} B\}\big\}_{K_{A,S}}$

$M_2 \quad S \longrightarrow B : \{T_S, A \mid\equiv A \overset{K_{A,B}}{\longleftrightarrow} B\}_{K_{B,S}}$

B.3.11 Wide-mouthed-frog protocol with declarations

```
\begin{steps}[labels=D]
  \astep(D){\apri{A}\asharedkey{\key{A,S}}\apri{S};
    \apri{B}\asharedkey{\key{B,S}}\apri{S}}[dwmf:1] \\
  \astep(D){\apri{A}\sees\key'{A,B}; \fresh{\key'{A,B}};
    \apri{A}\sees\aval{m}}[dwmf:3] \\
  \send[big]{A}{S}{A,\encrypted{A,S}{\ts{A},\apri{B},\key'{A,B}}}[dwmf:4] \\
  \send*[big]{S}{B}{\encrypted{B,S}{\ts{S},\apri{A},\key'{A,B}}}[dwmf:5]
\end{steps}
```

$$
\begin{array}{ll}
D_1 & A \overset{K_{A,S}}{\longleftrightarrow} S;\ B \overset{K_{B,S}}{\longleftrightarrow} S \\
D_2 & A \triangleleft K'_{A,B};\ \sharp(K'_{A,B});\ A \triangleleft m \\
M_1 & A \longrightarrow S : \big\{A, \{T_A, B, K'_{A,B}\}_{K_{A,S}}\big\} \\
M_2 & S \longrightarrow B : \{T_S, A, K'_{A,B}\}_{K_{B,S}}
\end{array}
$$

B.3.12 TLS 1.3 Handshake

```
\begin{steps}
  \astepat*{C}{\fresh{\key-{C_e}}; \dhpubkeyf{C_e}[C_e]}[tls-hs:1] \\
  \send{C}{S}{\ldots, \nonce{C}, \key+{C_e}, \ldots}[tls-hs:2] \\
  \astepat*{S}{\fresh{\key-{S_e}}; \dhpubkeyf{S_e}[S_e]}[tls-hs:3] \\
  \send[big]{S}{C}{\ldots, \nonce{S}, \key+{S_e}, \cert-{CA}{S},
    \signed{S}{\nonce{C}, \nonce{S}, \ldots}, \ldots}[tls-hs:4] \\
  \astepat*{C}{\verify+{CA}{\cert-{CA}{S}}[\atrue];
    \verify+{S}{\signed{S}{\nonce{C}, \nonce{S},
        \ldots}}[\atrue]}[tls-hs:5a] \\
  \astepat*{C}{\dhkeyf{C_e}{S_e}[C,S]}[tls-hs:5b] \\
  \astepat*{S}{\dhkeyf{S_e}{C_e}[C,S]}[tls-hs:6] \\
  \send*{C}{S}{\encrypted{C,S}{\ldots}}[tls-hs:7] \\
  \send*{S}{C}{\encrypted{C,S}{\ldots}}[tls-hs:8]
\end{steps}
```

$$
\begin{array}{lrl}
S_1 & C & : \sharp(K^-_{C_e});\ \mathit{DHPubKey}(K^-_{C_e}) \rightarrow K^+_{C_e} \\
M_1 & C \longrightarrow S & : \{\ldots, N'_C, K^+_{C_e}, \ldots\} \\
S_2 & S & : \sharp(K^-_{S_e});\ \mathit{DHPubKey}(K^-_{S_e}) \rightarrow K^+_{S_e} \\
M_2 & S \longrightarrow C & : \big\{\ldots, N'_S, K^+_{S_e}, \mathit{Cert}\{S, K^+_S\}^{K^-_{CA}}, \{N'_C, N'_S, \ldots\}^{K_S}, \ldots\big\} \\
S_3 & C & : \mathit{Verify}(K^+_{CA}, \mathit{Cert}\{S, K^+_S\}^{K^-_{CA}}) \rightarrow \mathit{true};\ \mathit{Verify}(K^+_S, \{N'_C, N'_S, \ldots\}^{K_S}) \rightarrow \mathit{true} \\
S_4 & C & : \mathit{DHKey}(K^-_{C_e}, K^+_{S_e}) \rightarrow K_{C,S} \\
S_5 & S & : \mathit{DHKey}(K^-_{S_e}, K^+_{C_e}) \rightarrow K_{C,S} \\
M_3 & C \longrightarrow S & : \{\ldots\}_{K_{C,S}} \\
M_4 & S \longrightarrow C & : \{\ldots\}_{K_{C,S}}
\end{array}
$$

B.3.13 SMC: Calculate the mean value

```
\begin{steps}\setcounter{counterS}{-1}%
```

```
  \astepat*{C}{$\aval{V'_0} = \random'{0}$; \fresh{\aval{V'_0}}}\\
  \send[Big]{C}{P_1}{%
    \encrypted+[big]{P_1}{\signed-{C}{\aval{V'_0}}},
    \encrypted+[big]{P_1}{\signed-{C}{\apri{C},\apri{P_2}}},
    \encrypted+[big]{P_2}{\signed-{C}{\apri{P_1},\apri{P_3}}},
    \encrypted+[big]{P_3}{\signed-{C}{\apri{P_2},\apri{C}}}} \\
  \astepat*{P_1}{$\aval{V'_1} = \aval{V'_0} + \aval{V_1}$} \\
  \send[Big]{P_1}{P_2}{%
    \encrypted+[big]{P_2}{\signed-{P_1}{\aval{V'_1}}},
    \encrypted+[big]{P_2}{\signed-{C}{\apri{P_1},\apri{P_3}}},
    \encrypted+[big]{P_3}{\signed-{C}{\apri{P_2},\apri{C}}}} \\
  \astepat*{P_2}{$\aval{V'_2} = \aval{V'_1} + \aval{V_2}$} \\
  \send[Big]{P_2}{P_3}{%
    \encrypted+[big]{P_3}{\signed-{P_2}{\aval{V'_2}}},
    \encrypted+[big]{P_3}{\signed-{C}{\apri{P_2},\apri{C}}}} \\
  \astepat*{P_3}{$\aval{V'_3} = \aval{V'_2} + \aval{V_3}$} \\
  \send[Big]{P_3}{C}{%
    \encrypted+[big]{C}{\signed-{P_3}{\aval{V'_3}}}} \\
  \astepat*{C}{$\aval{M} = (\aval{V'_3}-\aval{V'_0}) / \aval{3}$}
\end{steps}
```

$$
\begin{array}{llcl}
S_0 & & C & : V'_0 = R'_0;\ \sharp(V'_0) \\
M_1 & C \longrightarrow & P_1 & : \Big\{\big\{\{V'_0\}^{K_C^-}\big\}_{K_{P_1}^+}, \big\{\{C, P_2\}^{K_C^-}\big\}_{K_{P_1}^+}, \big\{\{P_1, P_3\}^{K_C^-}\big\}_{K_{P_2}^+}, \big\{\{P_2, C\}^{K_C^-}\big\}_{K_{P_3}^+}\Big\} \\
S_1 & & P_1 & : V'_1 = V'_0 + V_1 \\
M_2 & P_1 \longrightarrow & P_2 & : \Big\{\big\{\{V'_1\}^{K_{P_1}^-}\big\}_{K_{P_2}^+}, \big\{\{P_1, P_3\}^{K_C^-}\big\}_{K_{P_2}^+}, \big\{\{P_2, C\}^{K_C^-}\big\}_{K_{P_3}^+}\Big\} \\
S_2 & & P_2 & : V'_2 = V'_1 + V_2 \\
M_3 & P_2 \longrightarrow & P_3 & : \Big\{\big\{\{V'_2\}^{K_{P_2}^-}\big\}_{K_{P_3}^+}, \big\{\{P_2, C\}^{K_C^-}\big\}_{K_{P_3}^+}\Big\} \\
S_3 & & P_3 & : V'_3 = V'_2 + V_3 \\
M_4 & P_3 \longrightarrow & C & : \Big\{\big\{\{V'_3\}^{K_{P_3}^-}\big\}_{K_C^+}\Big\} \\
S_4 & & C & : M = (V'_3 - V'_0)/3
\end{array}
$$

B.3.14 File sharing with symmetric and public key encryption

```
\begin{steps}
  \astepat*{A}{\fresh{\key{F_1}}; \fresh{\key{F_2}}} \\
  \astepat*{A}{%
    \encrypt{F_1}{F_1}[*];
    \encrypt{F_2}{F_2}[*]} \\
  \astepat*{A}{%
    \encrypt+{A}{\key{F_1}}[*];
    \encrypt+{B}{\key{F_1}}[*]} \\
  \astepat*{A}{\encrypt+{A}{\key{F_2}}[*]} \\
  \send[big]{A}{S}{\encrypted+{A}{\key{F_1}},
    \encrypted+{B}{\key{F_1}}, \encrypted{F_1}{F_1}} \\
  \send[big]{A}{S}{\encrypted+{A}{\key{F_2}},
      \encrypted{F_2}{F_2}} \\
  \send[big]{S}{B}{\encrypted+{A}{\key{F_1}},
    \encrypted+{B}{\key{F_1}}, \encrypted{F_1}{F_1}} \\
  \astepat*{B}{%
```

```
    \decrypt"{P_B}{\encrypted"{P_B}{\key-{B}}}[\key-{B}];
    \decrypt-{B}{\encrypted+{B}{\key{F_1}}}[\key{F_1}]} \\
  \astepat*{B}{\decrypt{F_1}{\encrypted{F_1}{F_1}}[\aval{F_1}]}
\end{steps}
```

$$
\begin{array}{lrl}
S_1 & & A : \sharp(K_{F_1}); \sharp(K_{F_2}) \\
S_2 & & A : \mathit{Encrypt}(K_{F_1}, F_1) \rightarrow \{F_1\}_{K_{F_1}}; \mathit{Encrypt}(K_{F_2}, F_2) \rightarrow \{F_2\}_{K_{F_2}} \\
S_3 & & A : \mathit{Encrypt}(K_A^+, K_{F_1}) \rightarrow \{K_{F_1}\}_{K_A^+}; \mathit{Encrypt}(K_B^+, K_{F_1}) \rightarrow \{K_{F_1}\}_{K_B^+} \\
S_4 & & A : \mathit{Encrypt}(K_A^+, K_{F_2}) \rightarrow \{K_{F_2}\}_{K_A^+} \\
M_1 & A \longrightarrow & S : \left\{\{K_{F_1}\}_{K_A^+}, \{K_{F_1}\}_{K_B^+}, \{F_1\}_{K_{F_1}}\right\} \\
M_2 & A \longrightarrow & S : \left\{\{K_{F_2}\}_{K_A^+}, \{F_2\}_{K_{F_2}}\right\} \\
M_3 & S \longrightarrow & B : \left\{\{K_{F_1}\}_{K_A^+}, \{K_{F_1}\}_{K_B^+}, \{F_1\}_{K_{F_1}}\right\} \\
S_5 & & B : \mathit{Decrypt}(K_{P_B}^{"}, \{K_B^-\}_{K_{P_B}^{"}}) \rightarrow K_B^-; \mathit{Decrypt}(K_B^-, \{K_{F_1}\}_{K_B^+}) \rightarrow K_{F_1} \\
S_6 & & B : \mathit{Decrypt}(K_{F_1}, \{F_1\}_{K_{F_1}}) \rightarrow F_1
\end{array}
$$

B.3.15 Scalable secure file sharing

```
\begin{steps}
  \astepat*{A}{%
    \fresh{\key{F_1}}; \fresh{\key{F_2}};
    \fresh{\key{S_A}}; \fresh{\key{S_{A,B}}};
    \fresh{\key-{S_A}}; \fresh{\key+{S_A}};
    \fresh{\key-{S_{A,B}}}; \fresh{\key+{S_{A,B}}}} \\
  \astepat*{A}{%
    \encrypt{F_1}{F_1}[*];
    \encrypt{F_2}{F_2}[*]} \\
  \astepat*{A}{%
    \encrypt+{S_{A,B}}{\key{F_1}}[*];
    \encrypt+{S_A}{\key{F_2}}[*]} \\
  \astepat*{A}{%
    \encrypt{S_A}{\key-{S_A}}[*];
    \encrypt{S_{A,B}}{\key-{S_{A,B}}}[*]} \\
  \astepat*{A}{%
    \encrypt+{A}{\key{S_A}}[*]} \\
  \astepat*{A}{%
    \encrypt+{A}{\key{S_{A,B}}}[*];
    \encrypt+{B}{\key{S_{A,B}}}[*]} \\
  \send[big]{A}{S}{\encrypted+{A}{\key{S_A}}, \encrypted+{A}{\key{S_{A,B}}},
    \encrypted+{B}{\key{S_{A,B}}}} \\
  \send[big]{A}{S}{\encrypted{S_A}{\key-{S_A}},
    \encrypted{S_{A,B}}{\key-{S_{A,B}}}} \\
  \send[big]{A}{S}{\encrypted+{S_{A,B}}{\key{F_1}}, \encrypted{F_1}{F_1}} \\
  \send[big]{A}{S}{\encrypted+{S_A}{\key{F_2}}, \encrypted{F_2}{F_2}} \\
  \send[big]{S}{B}{\encrypted+{B}{\key{S_{A,B}}},
    \encrypted{S_{A,B}}{\key-{S_{A,B}}}} \\
  \send[big]{S}{B}{\encrypted+{S_{A,B}}{\key{F_1}}, \encrypted{F_1}{F_1}} \\
  \astepat*{B}{%
    \decrypt"{P_B}{\encrypted"{P_B}{\key-{B}}}[\key-{B}];
    \decrypt-{B}{\encrypted+{B}{\key{S_{A,B}}}}[\key{S_{A,B}}]} \\
  \astepat*{B}{%
    \decrypt{S_{A,B}}{\encrypted{S_{A,B}}{\key-{S_{A,B}}}}[\key-{S_{A,B}}];
    \decrypt-{S_{A,B}}{\encrypted+{S_{A,B}}{\key{F_1}}}[\key{F_1}]} \\
  \astepat*{B}{\decrypt{F_1}{\encrypted{F_1}{F_1}}[\aval{F_1}]}
\end{steps}
```

$$
\begin{array}{lll}
S_1 & & A : \sharp(K_{F_1}); \sharp(K_{F_2}); \sharp(K_{S_A}); \sharp(K_{S_{A,B}}); \sharp(K^-_{S_A}); \sharp(K^+_{S_A}); \sharp(K^-_{S_{A,B}}); \sharp(K^+_{S_{A,B}}) \\
S_2 & & A : Encrypt(K_{F_1}, F_1) \rightarrow \{F_1\}_{K_{F_1}}; Encrypt(K_{F_2}, F_2) \rightarrow \{F_2\}_{K_{F_2}} \\
S_3 & & A : Encrypt(K^+_{S_{A,B}}, K_{F_1}) \rightarrow \{K_{F_1}\}_{K^+_{S_{A,B}}}; Encrypt(K^+_{S_A}, K_{F_2}) \rightarrow \{K_{F_2}\}_{K^+_{S_A}} \\
S_4 & & A : Encrypt(K_{S_A}, K^-_{S_A}) \rightarrow \{K^-_{S_A}\}_{K_{S_A}}; Encrypt(K_{S_{A,B}}, K^-_{S_{A,B}}) \rightarrow \{K^-_{S_{A,B}}\}_{K_{S_{A,B}}} \\
S_5 & & A : Encrypt(K^+_A, K_{S_A}) \rightarrow \{K_{S_A}\}_{K^+_A} \\
S_6 & & A : Encrypt(K^+_A, K_{S_{A,B}}) \rightarrow \{K_{S_{A,B}}\}_{K^+_A}; Encrypt(K^+_B, K_{S_{A,B}}) \rightarrow \{K_{S_{A,B}}\}_{K^+_B} \\
M_1 & A \longrightarrow S : & \left\{\{K_{S_A}\}_{K^+_A}, \{K_{S_{A,B}}\}_{K^+_A}, \{K_{S_{A,B}}\}_{K^+_B}\right\} \\
M_2 & A \longrightarrow S : & \left\{\{K^-_{S_A}\}_{K_{S_A}}, \{K^-_{S_{A,B}}\}_{K_{S_{A,B}}}\right\} \\
M_3 & A \longrightarrow S : & \left\{\{K_{F_1}\}_{K^+_{S_{A,B}}}, \{F_1\}_{K_{F_1}}\right\} \\
M_4 & A \longrightarrow S : & \left\{\{K_{F_2}\}_{K^+_{S_A}}, \{F_2\}_{K_{F_2}}\right\} \\
M_5 & S \longrightarrow B : & \left\{\{K_{S_{A,B}}\}_{K^+_B}, \{K^-_{S_{A,B}}\}_{K_{S_{A,B}}}\right\} \\
M_6 & S \longrightarrow B : & \left\{\{K_{F_1}\}_{K^+_{S_{A,B}}}, \{F_1\}_{K_{F_1}}\right\} \\
S_7 & & B : Decrypt(K''_{P_B}, \{K^-_B\}_{K''_{P_B}}) \rightarrow K^-_B; Decrypt(K^-_B, \{K_{S_{A,B}}\}_{K^+_B}) \rightarrow K_{S_{A,B}} \\
S_8 & & B : Decrypt(K_{S_{A,B}}, \{K^-_{S_{A,B}}\}_{K_{S_{A,B}}}) \rightarrow K^-_{S_{A,B}}; Decrypt(K^-_{S_{A,B}}, \{K_{F_1}\}_{K^+_{S_{A,B}}}) \rightarrow K_{F_1} \\
S_9 & & B : Decrypt(K_{F_1}, \{F_1\}_{K_{F_1}}) \rightarrow F_1
\end{array}
$$