A Security Protocol Engineering Notation

Anders Andersen

UiT The Arctic University of Norway
Anders.Andersen@uit.no

Abstract. ASPEN is a notation for security protocols. Despite many publications and textbooks with security protocols, the notations used are inconsistent. ASPEN aims to provide an expressive, consistent notation that can be used in different settings where security protocols are presented. This text presents and discusses the main concepts of ASPEN.

Keywords: Notation · Security protocols · Cryptography.

1 Introduction

ASPEN [3] is an attempt to create the preferable notation for security protocols (a security protocol engineering notation) that will be used in publications, textbooks, and teaching. The notation presented here is strongly inspired by notations found in security protocol related publications [14, 9, 7, 13], notations found in text books [5], and notations used in my own publications, teaching and presentations.

The main motivation for ASPEN is to provide a consistent and expressive notation for security protocols. The notation should be familiar for those who know the field and it should be possible to use it together with existing notations, like BAN logic. For completeness and consistency, the notation includes refinements and new useful constructs and contributions not found in similar notations.

However, ASPEN is not a formalism, like BAN logic [8], or a calculus for analysis of cryptographic protocols, like Spi calculus [1]. For more detailed analysis of cryptographic protocols, more expressive notations like BAN logic, Spi calculus, or something similar, should be considered.

To ease the adaption of ASPEN, the LaTeX package `aspen`[1] [2] has been created and is available from CTAN[2] and with the main LaTeX distributions.

In this paper we will first present the main concepts of the notation and then compare them with existing notations used in the literature. Finally, we will show some examples and conclude.

[1] https://www.pg12.org/dist/texmf/tex/latex/aspen/
[2] https://ctan.org/pkg/aspen

2 The main concepts

The main concepts of Aspen are inspired by notations found in the literature for security protocols. The goal of Aspen is to make a readable notation that can be used in publications and text books. The concepts of the notation should be familiar, consistent, and expressive. To achieve this, Aspen will reuse concepts from some existing notations, ignore other (sometimes inconsistent) used notations, and extend Aspen with new concepts and constructs not found in existing notations.

In the following, the main concepts of the Aspen notation are presented. They are grouped into six groups: (i) *values and keys*, (ii) *structured values and messages*, (iii) *encrypted and signed*, (iv) *functions and procedures*, (v) *conditionals, repeat, and concurrency*, and some (vi) *extra bits*.

2.1 Values and keys

Values Values and operations on values are nothing special in Aspen. They are presented as straight forward and familiar as possible, including boolean values, numbers, strings, variables (unknown values), mathematical operators and boolean expressions:

- Boolean values: *true*, *false*
- Numbers: 1, 0x3F
- Strings: "`Hello world`"
- Variables (unknown values): x, y
- Mathematical operators and expressions: $i + 1$, $n - 1$
- Boolean expressions: $I_1 > I_2$, $N'_A = N'_B$, $T_S \leq T_C$
- Logical operators and expressions: $\neg\, p_0$, $(N'_x = N'_y \vee T_S \leq T_C) \wedge V_1 = V_2$

Aspen also includes operators for concatenation and binary (bitwise) exclusive or. Concatenation is typically used to concatenate strings and other (byte) sequences. Binary exclusive or is used to perform an binary exclusive or operation of two values. The concatenation operator has precedence over the exclusive or operator. The operator for concatenation is a period and the operator for binary exclusive or is a circled-plus:

- Concatenation: "`Hello `" . "`Bob`"
- Binary exclusive or: $v_0 \oplus \mathit{padding}$

Values are often represented by a name. For example, a random value can be named R_A, where R is the symbol for random values and A is the unique identifier for the given value. In the notation, we use different symbols for different types of values, and the subscript to identify a specific value of the given type:

- Random values: R_0, R_A
- Nonces: N'_C, N'_B
- Counters: I_1, I_S
- Time stamps: T_A, T_1
- Time-to-live: L_K, L_0

The nonces are marked with a $'$ mark (a session mark). The mark hints that nonces have a limited useful lifetime. Other values can also be marked with a session mark to indicate this.

We use other letters (symbols) to identify participants or principals in security protocols, for example A for Alice, B for Bob, S for server, C for client, and $P_1, \ldots, P_n$ for a group of peers.

Keys One of the most important values in security protocols are the encryption and decryption keys. In the notation, we use the symbol K for such keys. We use markers to distinguish different types of keys:

- Shared/secret keys (symmetric keys): $K_{A,B}$, K_S, $K_{P_1-P_n}$ (no mark)
- Session keys: K'_C (session mark $'$)
- Public keys (in public-private key pairs): K^+_A, K^+_B (public key mark $^+$)
- Private keys (in public-private key pairs): K^-_A, K^-_B (private key mark $^-$)
- Password generated keys: $K^{"}_{P_1}$ (password key mark $"$)

The subscripts of the keys are used as identifiers of specific keys. It is common practice to use the name of principals in such identifiers and by that, give implicit hints about the keys. For example, K^+_A is implicit understood as the public key of principal A, $K'_{C,S}$ is implicit understood as a session key shared by principals C and S, and $K_{P_1-P_n}$ is implicit understood as a secret key shared by the group $P_1, \ldots, P_n$.

Assignments It is sometimes useful to give names to values and be able to change named values when presenting protocols in Aspen. We use the left-arrow to show that a value is assigned to a name:

- Value 10 is assigned to name v_1: $v_1 \leftarrow 10$
- Value v_1 is incremented by one: $v_1 \leftarrow v_1 + 1$

We can also assign values to a named list (or an array). In the following two examples we assign the values x_1, x_2, x_3, and x_4 to the lists $x_\square$:

- Values x_1, x_2, x_3, and x_4 assigned to list $x_{\square}$: $x_{\square} \leftarrow x_1, x_2, x_3, x_4$
- Values x_1, x_2, x_3, and x_4 assigned to list $x_{\square}$: $x_{\square} \leftarrow x_1, \ldots, x_4$

The square subscript on the assigned list name is used to indicate that it represents a list that can be index by a specific subscript for a specific element in the list. For example, x_i should be understood as one specific element from the list, where i is either 1, 2, 3, or 4. If we have a list of principals we call it a group:

- Principals $P_1, \ldots, P_n$ assigned to group $P_{\square}$: $P_{\square} \leftarrow P_1, \ldots, P_n$

When we refer to a group in a subscript as an identifier, the more compact representation of a group should be used. For example in the session key for the group $P_1, \ldots, P_n$: $K'_{P_1-P_n}$

2.2 Structured values and messages

In the notation, structured values (and messages) are typeset with curly brackets. A structured value can consist of multiple values, can be typed, and can be nested:

- A structured value (containing the value v): $\{v\}$
- A structured value (containing a value and a nonce): $\{v, N'_1\}$
- A typed structured value (with the type T): $T\{v\}$
- A nested structured value: $\{A, \{v, N'_A\}\}$

A message is a structured value sent between to principals. We use the arrow key to illustrate the message path (the sender and the receiver) followed by a colon followed by the message:

- The value m is sent from A to B: $A \longrightarrow B : \{m\}$
- A nested structured value is sent from B to S: $B \longrightarrow S : \{A, \{v, N'_A\}\}$

Aspen includes a few predefined typed structured values often used in security protocols, including $H\{m\}$ for cryptographic hash values (in this example, the cryptographic hash value of m). When using Aspen, new typed structured value types might be introduced. This is a feature of the notation.

2.3 Encrypted and signed

Values, structured values and messages can be encrypted and signed. An encrypted or signed value is a structured value. In the notation, we use a key as a subscript to say that something is encrypted with the key, and a key as a superscript to say that something is signed with the key:

- Value v encrypted with session key $K'_{A,B}$: $\{v\}_{K'_{A,B}}$
- Structured value encrypted with public key K_A^+: $\{m, T_A\}_{K_A^+}$
- Structured value signed with private key K_B^-: $\{v, N'_A\}^{K_B^-}$
- Message from A to B encrypted with shared key $K_{A,B}$: $A \longrightarrow B : \{m\}_{K_{A,B}}$
- Message from S to C signed and encrypted: $S \longrightarrow C : \left\{\{A, K_A^+\}^{K_S^-}\right\}_{K'_{S,C}}$
- Structured value with a principal and a key signed by C: $\{B, K_B^+\}^{[C]}$

The last example says that a structured value is signed *by* principal C, and not signed *with* a specific key. Square brackets around a principal name is used to say that something is signed or encrypted by the principal without giving a specific key.

An encrypted or signed value or message is a structure value, and can also be a typed structured value. ASPEN includes a few predefined encrypted or signed typed structured values often used in security protocols:

- Signature of m using private key K_A^-: $Sig\{m\}^{K_A^-}$
- Certificate for public key K_A^+ of principal A signed by C: $Cert\{A, K_A^+\}^{[C]}$
- Message authentication code of m with key K_A: $MAC\{m\}^{K_A}$
- Cipher-based message authentication code of m with key K_A: $CMAC\{m\}^{K_A}$
- HMAC message authentication code of m with key K_A: $HMAC\{m\}^{K_A}$

An unpack (leads-to) arrow $\rightsquigarrow$ is used to show more details or to unpack a value or a message. Three examples where structured values are unpacked:

- Signature: $Sig\{m\}^{K_A^-} \rightsquigarrow \left\{H\{m\}\right\}_{K_A^-}$
- Signed: $\{m\}^{K_A^-} \rightsquigarrow \left\{m, Sig\{m\}^{K_A^-}\right\} \rightsquigarrow \left\{m, \left\{H\{m\}\right\}_{K_A^-}\right\}$
- HMAC: $HMAC\{m\}^{K_A} \rightsquigarrow H\Big\{(\overline{K}_A \oplus opad) \,.\, H\big\{(\overline{K}_A \oplus ipad) \,.\, m\big\}\Big\}$

In the last example, $\overline{K}_A$ is $H\{K_A\}$ if K_A is larger than the block size, otherwise it is K_A; and $opad$ (outer padding) and $ipad$ (inner padding) consists of the repeating byte values $0\text{x}\,5\text{C}$ and $0\text{x}\,36$, respectively [6].

2.4 Functions and procedures

Functions Functions can represent processes producing a value at principals that are parts of a security protocol. Such processes could for example be generating a deterministic encryption key from a password, or they can be each of the processing steps in generating a shared encryption key in the Diffie-Hellman (DH) protocol [10]. When presenting security protocols, such processes should

be used with moderation. For example, when sending an encrypted message between two principals, there is no need to include a step at the sender side where the message is encrypted. This is implicit when it is show that we send an encrypted message.

In ASPEN, a function has a name, take parameters (arguments), and produce a result. The parameters are in a comma separated list inside parenthesis: $Func(u, v)$. If the produced result should be included, it is showed following an arrow: $Func(u, v) \rightarrow w$. ASPEN includes many predefined functions used in security protocols:

- The size of (number of elements in) a list or group: $Size(v_{\square})$
- Encrypt m with key $K_{A,B}$: $Encrypt(K_{A,B}, m) \rightarrow \{m\}_{K_{A,B}}$
- Decrypt $\{m\}_{K_B^+}$ with K_B^-: $Decrypt\big(K_B^-, \{m\}_{K_B^+}\big) \rightarrow m$
- Cryptographic hash value of m: $H(m) \rightarrow H\{m\}$
- Message authentication code of m with K_A: $MAC(K_A, m) \rightarrow MAC\{m\}^{K_A}$
- Cipher-based message authentication code: $CMAC(K_A, m) \rightarrow CMAC\{m\}^{K_A}$
- HMAC message authentication code: $HMAC(K_A, m) \rightarrow HMAC\{m\}^{K_A}$
- Create signature for m with private key K_A^-: $Sig(K_A^-, m) \rightarrow Sig\{m\}^{K_A^-}$
- Sign m with key private K_A^-: $Sign(K_A^-, m) \rightarrow \{m\}^{K_A^-}$
- Create a password generated key from password P_1: $PwKey(P_1) \rightarrow K''_{P_1}$
- Create a password generated key with salt s: $PwKey(P_2, s) \rightarrow K''_{P_2}$
- Create a DH key share from private key K_A^-: $DHPubKey(K_A^-) \rightarrow K_A^+$
- Create a key from a private key and a key share: $DHKey(K_A^-, K_B^+) \rightarrow K_{A,B}$
- Include optional public parameters p: $DHKey(K_B^-, K_A^+, p) \rightarrow K_{A,B}$
- Verify signed message with matching public key: $Verify\big(K_S^+, \{m\}^{K_S^-}\big) \rightarrow true$

When using ASPEN, new functions might be introduced. This is a feature of the notation.

Procedures Procedures are similar to functions, but they don't produce a value. Instead they have an affect on the running protocol, and they are typically used if something fails or some conditions are not satisfied. The name of procedures are underlined and they have only one optional parameter, a comment (a text string). ASPEN has a few predefined procedure. The *Abort* procedure immediately aborts the current protocol, the *Continue* procedure is used to repeat (loop) conditionals (presented below), and the *Break* procedure is used to break (stop) any further execution of the given conditional or to stop waiting for the completion of any further concurrent steps:

- Abort the current protocol: *Abort*
- Abort with comment: *Abort*(“`Signature check failed`”)
- Repeat the current conditional: *Continue*
- Break the current conditional or concurrent steps: *Break*

When using Aspen, new procedures might be introduced. This is a feature of the notation.

2.5 Conditionals, repeat, and concurrency

Conditionals, repeat, and concurrency are currently experimental features of the Aspen notation. These features might evolve and change while more experience using them are gained. Feedback on how to improve and evolve these features are welcome.

Conditionals Conditionals can be used to change the behavior of a protocol based on a condition. A conditional consists of an optional expression followed by question mark **?** followed by conditional steps. Each conditional step consist of a condition followed by a colon **:** followed by the step that should be performed if the condition is satisfied. Conditional steps are separated by a double semicolon **;;**. To summarize, conditionals have the following form

$$\mathit{expression}\ \mathbf{?}\ \ \mathit{cond}_1 : \mathit{step}_1\ \mathbf{;;}\ \ \ldots\ \mathbf{;;}\ \ \mathit{cond}_n : \mathit{step}_n$$

where step_1 is performed if condition cond_1 is satisfied, and so on. Satisfied means that the condition match the expression, or if no expression is given, the condition is true. The optional *expression* should give a value that will be used as a condition at each conditional step:

$$N'_x = N'_y\ \mathbf{?}\ \ \mathit{true} :\ N'_A \leftarrow N'_x\ \mathbf{;;}\ \ \mathit{false} :\ \mathit{Abort}$$

If no expression is given, each condition should be an boolean expression or value. This example, without the optional expression, is semantically the same as the previous example:

$$\mathbf{?}\ \ N'_x = N'_y :\ N'_A \leftarrow N'_x\ \mathbf{;;}\ \ N'_x \neq N'_y :\ \mathit{Abort}$$

You can think about the relation between the optional expression and each condition like this (where v is actually never shown or used):

$$v \leftarrow \mathit{expression}\ \mathbf{?}\ \ v = \mathit{cond}\ 1 :\ \mathit{step}\ 1\ \mathbf{;;}\ \ \ldots\ \mathbf{;;}\ \ v = \mathit{cond}\ n :\ \mathit{step}\ n$$

Each conditional step can consist of several sub-steps. Both a semicolon and a line break can be used to separate steps (and sub-steps) in Aspen. Therefore, is it possible to use line breaks in conditionals. In Figure 1, **1** shows line break for each conditional step, and **2** shows line break for each conditional step *and* for each sub-step.

1:

$$
\begin{array}{l}
\mathit{expression}\ \mathbf{?} \\
\quad \mathit{cond}_1 : (\ \mathit{step}_{1.1}\ ;\ \ \mathit{step}_{1.2}\ )\ \mathbf{;;} \\
\quad \mathit{cond}_2 : (\ \mathit{step}_{2.1}\ ;\ \ \mathit{step}_{2.2}\ )\ \mathbf{;;} \\
\quad \mathit{cond}_3 : (\ \mathit{step}_{3.1}\ ;\ \ \mathit{step}_{3.2}\ )\ \mathbf{;;} \\
\quad \mathit{cond}_4 : (\ \mathit{step}_{4.1}\ ;\ \ \mathit{step}_{4.2}\ )\ \mathbf{;;} \\
\quad \mathit{cond}_5 : (\ \mathit{step}_{5.1}\ ;\ \ \mathit{step}_{5.2}\ )\ \mathbf{;;} \\
\quad \mathit{cond}_6 : (\ \mathit{step}_{6.1}\ ;\ \ \mathit{step}_{6.2}\ )
\end{array}
$$

2:

$$
\begin{array}{l}
\mathit{expression}\ \mathbf{?} \\
\quad \mathit{cond}_1 : \\
\quad\quad \mathit{step}_{1.1} \\
\quad\quad \mathit{step}_{1.2}\ \mathbf{;;} \\
\quad \mathit{cond}_2 : \\
\quad\quad \mathit{step}_{2.1} \\
\quad\quad \mathit{step}_{2.2}
\end{array}
$$

Fig. 1. Conditionals with sub-steps and line breaks

In Aspen, we can use parentheses to group together a sequences of sub-steps. In **1** in Figure 1, we use parentheses to make it explicit that a sequence of sub-steps belongs to the given conditional step. In **2** we instead use indentation to group together a sequence of sub-steps.

A *Break* procedure can be used to ignore the rest of the conditional steps. If the condition is satisfied and the conditional step is performed and ends with a *Break* procedure, the rest of the conditional steps are not evaluated and performed:

$$\mathbf{?}\ \ N'_x = (N'_A + 1) : a \leftarrow \mathit{true}; \mathit{Break}\ \mathbf{;;}\ \ N'_y = (N'_B + 1) : b \leftarrow \mathit{true}$$

If no explicit *Break* procedure is used, the behavior is different; the following conditional steps are evaluated and will be performed if their conditions are satisfied. In the example below, both conditions might be satisfied, and in that case both conditional steps are performed:

$$\mathbf{?}\ \ N'_x = (N'_A + 1) : a \leftarrow \mathit{true}\ \mathbf{;;}\ \ N'_y = (N'_B + 1) : b \leftarrow \mathit{true}$$

The expressive syntax of conditionals might be an invitation to create very complex conditional statements. This is not recommended. Please use this feature with care. Simple conditionals might improve the readability of a protocol. Complex conditionals will make a protocol more difficult to follow and understand.

Repeat The repeat construct is a conditional where we use the procedure *Continue* to restart or reevaluate the conditional. This is used to repeat the conditional as long as a condition is satisfied. The procedure *Break* can be used to skip the rest of the conditional (break out of it). When used in conditionals that are repeated, *Break* can be used to make it explicit when the repeating stops (when to break the loop).

$$c \leq 0\ \mathbf{?}\ \ \mathit{true} : \mathit{Break}\ \mathbf{;;}\ \ \mathit{false} : \big(\ v \leftarrow v + s_c\ ;\ \ c \leftarrow c - 1\ ;\ \ \mathit{Continue}\ \big)$$

If we look closer at the example above, we can see that the *Break* procedure is unnecessary. The following conditional will behave equally since it will stop doing the *Continue* procedure when the expression $c \leq 0$ becomes true:

$$c \leq 0\ \mathbf{?}\ \ \mathit{false} : \big(\ v \leftarrow v + s_c\ ;\ \ c \leftarrow c - 1\ ;\ \ \mathit{Continue}\ \big)$$

We can further simplify the loop by skipping the optional expression:

$$\mathbf{?}\ \ c > 0 : \big(\ v \leftarrow v + s_c\ ;\ \ c \leftarrow c - 1\ ;\ \ \mathit{Continue}\ \big)$$

Concurrency The ASPEN notation supports concurrency. A concurrency statement uses the concurrency symbol || to say that the following steps are done, or can be done, concurrently. Such a statement starts with the concurrency symbol followed by the concurrent steps. Typically, we use lists when doing concurrent steps. The lists can be lists of values or groups of principals. In the following example, a list of values and a group of principals have been assigned the names $v_\square$ and $P_\square$, respectively. The list and the group have the same number of elements ($Size(v_\square) = Size(P_\square)$). In the concurrent steps, messages are concurrently sent from principal S to all principals P_i in the group $P_\square$. Each message contains one value v_i from the list of values $v_\square$ encrypted with the receivers public key:

$$|| \; S \longrightarrow P_i : \{v_i\}_{K^+_{P_i}}$$

We can also use concurrency to state that somethings should happen concurrently at a group of principals. In this example we add the values v_i and w_i at each principal P_i and assign the result to the value s_i:

$$|| \; P_i : \; s_i \leftarrow v_i + w_i$$

It is also useful to be able to perform a step for each completed concurrent step. If a group of principals are involved, the location of such steps should be included. We use an exclamation mark **!** followed by the location followed by a colon followed by the step to be performed, to specify such steps. In the following example, we add each received sample value s_i at principal S to the total value t:

$$|| \; P_i \longrightarrow S : \{s_i\}_{K^+_S} \; ! \; S : \; t \leftarrow t + s_i$$

The example can be extended. In the extended example, we want to calculate the average of the the received sample values $s_\square$ at principal S, but we have decided that when 70% of the values are received, we have enough samples to calculate an estimated average value $\overline{s}$:

$$S : t \leftarrow 0 \; ; \;\; c \leftarrow 0 \; ; \;\; n \leftarrow Size(P_\square)$$
$$|| \; P_i \longrightarrow S : \{s_i\}_{K^+_S} \; ! \; S : (\, t \leftarrow t + s_i \; ; \;\; c \leftarrow c + 1 \; ; \;\; ? \;\; c \geq 0.7n : \; \underline{Break})$$
$$S : \overline{s} \leftarrow t/c$$

When S has received 70% or more of the sample values $s_\square$, it ignores the rest of expected values (does the *Break* procedure). The step that follows calculates the average $\overline{s}$ of the received values. We use t to calculate the total sum of the so-far received sample values, c counts the number of received sample values and n is the number of principals in the group $P_\square$.

2.6 Extra bits

Embellishment ASPEN supports embellishment to provide more details about a structured value or a function. For example, this can be used to illustrate that a cryptographic hash value is an *MD5* value: $H_{MD5}\{m\}$. Embellishment should

be used carefully. If such extra details are not important for the presentation of a security protocol the best option it is often to leave it out.

Embellishment can be used on all types of structured values and functions. It is even possible to use embellishment on non-typed structured values. This is a few example of embellishment usage:

- A *SHA-2*-based cryptographic hash value of m: $H_{SHA\text{-}2}\{m\}$
- Encrypt m with *AES* encryption using key $K'_{A,B}$: $Encrypt_{AES}(K'_{A,B}, m)$
- Decrypt c with *RSA* encryption using key K^-_A: $Decrypt_{RSA}(K^-_A, c)$
- A *DSA*-based signature of m using key K^-_B: $Sig_{DSA}\{m\}^{K^-_B}$
- Create a *PBKDF2* key from password P_1 and salt s: $PwKey_{PBKDF2}(P_1, s)$
- Value v is encrypted with *RSA* encryption using public key K^+_S: ${}_{RSA}\{v\}_{K^+_S}$

Logical implication and equivalence Logical implication ($\Rightarrow$) and logical equivalence ($\Leftrightarrow$) can be used when describing security protocols, for example when describing the steps of verifying a digital signature:

- Logical implication; y, if x: $x \Rightarrow y$
- Logical equivalence; y, if and only if x: $x \Leftrightarrow y$

3 Discussion

Aspen is based on notations used in publications and textbooks, but developed further to make a more consistent and complete notation for security protocols. The notation will evolve further, but the concept with different *value types*, *encryption key types*, *structured values* and *typed structured values*, *encrypted* and *signed* (structured) values, and *functions* and *procedures*, is set. For more complex protocols, the recent additions of *conditionals*, *repeat*, and *concurrency* have been valuable. These features are still considered experimental and might continue to evolve over time.

Aspen is based on existing notations. The main inspiration is found in the following four publications: (i) *Kerberos: An Authentication Service for Open Network Systems* [14], (ii) *A formal Semantics for Protocol Narrations* [7], (iii) *Security Engineering: A Guide to Building Dependable Distributed Systems* [5], and (iv) *Current Approaches to Authentication in Wireless and Mobile Communications Networks* [13]. In Table 1, the relevant notations used in these publications are shown. From the table we can see two things; the inspiring sources of Aspen, and the inconsistencies in the used notations.

In the table, the notations used for keys are similar to Aspen, but not as expressive. In (iv), the use of plus and minus for public and private keys are introduced, but in Aspen the plus and minus on the keys are superscripts. The

Table 1. Existing notations used in articles and textbooks

Description	(i)	(ii)	(iii)	(iv)
Secret key	K_A	k_A	K	K_A
Shared key	–	$k_{A\,B}$	–	–
Session key	$K_{A,B}$	–	–	–
Public key	–	$\mathrm{pub}(k_A)$	KR	$+K_A$
Private key	–	$\mathrm{priv}(k_A)$	KR^{-1}	$-K_A$
Encrypted	$\{m\}K$	$\{m\}_K$	$\{m\}_K$	$\{m\}_K$
Signed with	–	–	$\mathrm{sig}_K\{m\}$	–
Signed by	–	–	–	$A\,[m]$
Send	$A \xrightarrow{m} B$	$A \rightsquigarrow B : m$	$A \rightarrow B : m$	$A \rightarrow B : m$
Hash value	–	$\mathrm{H}(m)$	$h(m)$	$H(m)$
MAC	–	–	$\mathrm{MAC}_K(m)$	–
HMAC	–	–	$\mathrm{HMAC}_K(m)$	–
Certificate	–	–	$Cert_{KC^{-1}}(A, KR)$	$Cert_{-K_{CA}}(+K_A)$
Certificate by	–	–	–	$CA\,\langle\langle\, A \,\rangle\rangle$

use of superscripts in Aspen is inspired by (iii) and other publications using the negative exponent -1 (inverse) for the private key in a public-private key pair. In Aspen, the superscript marker on keys is used for more than distinguishing public and private keys. Other symbols are introduced, and the use of only the plus and minus symbols for public and private keys is more in line with this. For example, Aspen also has markers for session keys and password generated keys.

The notation for an encrypted value where the value is enclosed in curly brackets and marked with the encryption key as a subscript, $\{m\}_K$, has become common in the literature. The usage of the key as a superscript for signing a value, $\{m\}^K$, is novel for Aspen. This way of distinguishing encrypted and signed values makes the Aspen notation expressive, readable and consistent.

The notation for sending messages, with an arrow from the sender to the receiver followed by a colon followed by the message sent, has also become common in the literature. In Aspen, we have the concept of structured values written as curly brackets enclosing the value (the content), and since a message always is a structured value, all messages are written with curly brackets in Aspen: $A \longrightarrow B : \{m\}$.

All structured values in Aspen are written with curly brackets. Optionally they can have a type (a type prefix) and they can be encrypted (subscripted by encryption key) or signed (superscripted by key used to sign). Examples includes encrypted messages, cryptographic hash values, message authentication codes and certificates:

$$\{m\}_K, \quad H\{m\}, \quad MAC\{m\}^K, \quad HMAC\{m\}^K, \quad Cert\{A, K_A^+\}^{K_{CA}^-}$$

Distinguishing structured values and functions is a feature of Aspen. In protocols, this can be important. For example, when it is significant when and where a value is created. Functions are written with ordinary parentheses enclosing the arguments (if any). Examples of two predefined functions in Aspen and the structured value they produce:

$$H(m) \to H\{m\}, \quad HMAC(K, m) \to HMAC\{m\}^K$$

In addition to what is commonly used in the literature, Aspen provides many more features presented in the previous section. The Aspen notation is consistent and expressive, making it a suitable tool to describe a wide variations of security protocols. We claim that the Aspen notation is readable, even for more complex security protocols.

A LaTeX package has been developed to ease the adaption of the platform [2]. In this package the option `color` can be used to make the presentation of the notation more colorful (as used in this document). The colors are not part of the notation itself and are only provided to make the notation even more readable.

4 Examples

To demonstrate the expressiveness and suitability of Aspen to describe security protocols, a few examples of using Aspen to describe well-known and not so well-known protocols are shown.

In the examples that follows, each protocol is presented as a series of labeled steps. The label for each step is found on the left-hand side. Labels are useful to easily reference every single step when a protocol is discussed. Messages are usually labeled with the letter M and a number. Other steps are usually labeled with the letter S and a number. More examples where Aspen is used to describe protocols can be found in the latest Aspen documentation.

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

Fig. 2. The original Needham-Schroeder protocol

Original Needham-Schroeder protocol The original *Needham-Schroeder protocol* [12] aims to establish a session key between two parties A and B on a network, typically to protect further communication. The protocol involves a third-party server S. The protocol is presented in Figure 2. It is based on a symmetric encryption and it forms the basis for the Kerberos protocol. The protocol expect that two secret shared keys exists; $K_{A,S}$ shared by A and S and $K_{B,S}$

shared by B and S. The server creates the new secret key $K_{A,B}$. The protocol uses the two nonces N'_A and N'_B. When the protocol terminates, A and B both have the shared secret key $K_{A,B}$ and, if they trust S, are ensured that this key is intended to protect further communication between them.

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

Fig. 3. The Diffie-Hellman key exchange protocol

Diffie-Hellman key exchange The *Diffie-Hellman key exchange protocol* [10] is used to establish a shared symmetric key between two participants over a public channel based on a secret (a private key) at each participant and some shared public parameters. The protocol is presented in Figure 3. The public keys K_A^+ and K_B^+ created in step S_1 and S_2 are called Diffie-Hellman key shares. At the end of this protocol the two participants A and B have a shared secret key $K_{A,B}$. In step M_1, participant A sends the public parameters p to participant B. Other means of agreeing on these parameters are possible and not important for the protocol itself.

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

Fig. 4. The Needham-Schroeder-Lowe public key protocol

Needham-Schroeder-Lowe public key protocol The *Needham-Schroeder-Lowe public key protocol* [11] is an improvement of the original *Needham-Schroeder public key protocol* [12]. The original protocols is vulnerable to a man-in-the-middle attack. However, the fix is easy; message M_6 in the improved protocol

includes the identity of the responder. The improved protocol is presented in Figure 4.

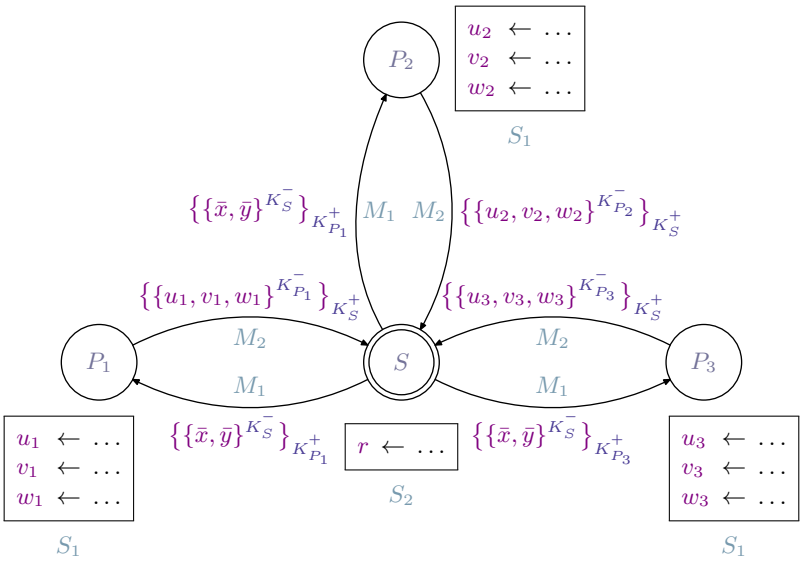

Fig. 5. An updated version of Figure 2 "Calculating Pearson's r of n samples of two variables x and y at 3 health institutions" from the article *Privacy preserving health data processing* [4], including the four labeled steps M_1, S_1, M_2 and S_2 from Figure 6, and with signed and encrypted messages using the current ASPEN notation

Calculating Pearson's *r* In the article *Privacy preserving health data processing* [4] from 2014, a secure approach to calculating Pearson's r with samples for multiple participants is discussed. In Figure 5, the calculation of Pearson's r of two variables x and y from three participants, when the mean values of variables x and y are known, is illustrated. The mean value of variable x is $\overline{x}$ and the mean variable y is $\overline{y}$. The participants are group $P_\square$ (with the members P_1, P_2 and P_3 in the illustration) and the coordinator is S. Each participant has a list of sample values for the variables x and y, $x_{i_\square}$, and $y_{i_\square}$, respectively (where the subscript i represents the participants and x_{i_j} and y_{i_j} represent each sample values of x and y at participant P_i). The list $s_\square$ contains the number of samples at each participant. The actual calculations of u_i, v_i, w_i at each participant in Step S_1 are calculated by the functions $P1$, $P2$, and $P3$, respectively:

$$P1(\overline{x}, x_{i_j}, \overline{y}, y_{i_j}) \rightarrow \sum_{i=1}^{s_i} (x_{i_j} - \overline{x})(y_{i_j} - \overline{y})$$

$$P2(\overline{x}, x_{i_j}) \rightarrow \sum_{i=1}^{s_i} (x_{i_j} - \overline{x})$$

$$P3(\overline{y}, y_{i_j}) \rightarrow \sum_{i=1}^{s_i} (y_{i_j} - \overline{y})$$

In Step S_2 the function *PFinal* calculates Pearson's r for n participants using the values calculated in the functions *P1*, *P2*, and *P3* at each participants:

$$PFinal(u_i, v_i, w_i) \rightarrow \frac{\sum_{i=1}^{n} u_i}{\sqrt{\sum_{i=1}^{n} v_i \sum_{i=1}^{n} w_i}}$$

Figure 6 presents this example for calculating Pearson's r in the ASPEN notation with one update compared with the example in the original article from 2014; the messages are signed and encrypted.

$$\begin{aligned}
M_1 \quad & || \quad S \longrightarrow P_i : \left\{ \{\overline{x}, \overline{y}\}^{K_S^-} \right\}_{K_{P_i}^+} \\
S_1 \quad & || \quad P_i : \left(u_i \leftarrow P1(\overline{x}, x_{i_j}, \overline{y}, y_{i_j}); v_i \leftarrow P2(\overline{x}, x_{i_j}); w_i \leftarrow P3(\overline{y}, y_{i_j}) \right) \\
M_2 \quad & || \quad P_i \longrightarrow S : \left\{ \{u_i, v_i, w_i\}^{K_{P_i}^-} \right\}_{K_S^+} \\
S_2 \quad & \quad S : r \leftarrow PFinal(u_i, v_i, w_i)
\end{aligned}$$

Fig. 6. Calculating Pearson's r of two variables x and y in the ASPEN notation

5 Conclusion

The ASPEN security protocol notation is an expressive and consistent notation for security protocols. The notation has been developed over years, but recent additions have broaden its scope, in particular the introduction of conditionals and concurrency. While these new additions will be developed further, they have already demonstrated their usefulness, for example in secure multiparty computation protocols.

References

1. Abadi, M., Gordon, A.D.: A calculus for cryptographic protocols: The Spi calculus. Information and Computation **148**, 1–70 (1999)
2. Andersen, A.: A LaTeX package for the Aspen security protocol notation and BAN logic. TUGboat **47**(1), 75–80 (2026)
3. Andersen, A.: Aspen: A security protocol notation. The complete documentation, UiT The Arctic University of Norway (2026)
4. 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)
5. Anderson, R.: Security Engineering: A Guide to Building Dependable Distributed Systems. John Wiley & Sons, 3 edn. (2020)
6. Bellare, M., Canetti, R., Krawczyk, H.: Keying hash functions for message authentication. In: Koblitz, N. (ed.) Advances in Cryptology — CRYPTO96. Lecture Notes in Computer Science, vol. 1109, pp. 1–15. Springer-Verlag (Aug 1996)

7. 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)
8. Burrows, M., Abadi, M., Needham, R.: A logic of authentication. ACM Transactions on Computer Systems **8**(1), 18–36 (Feb 1990)
9. Chappell, D.: Exploring Kerberos, the protocol for distributed security in Windows 2000. Microsoft System Journal (Aug 1999)
10. Diffie, W., Hellman, M.E.: New directions in cryptography. IEEE Transactions on Information Theory **22**(6), 644–654 (Nov 1976)
11. Lowe, G.: An attack on the Needham-Schroeder public-key authentication protocol. Information Processing Letters **56**(3), 131–136 (Nov 1995)
12. Needham, R., Schroeder, M.: Using encryption for authentication in large networks of computers. Communications of the ACM **21**(12), 993–999 (Dec 1978)
13. 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)
14. 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)