Cryptography and cryptographic protocols

- Cryptography deals with algorithms for encryption, decryption, random number generation, etc. Cryptographic protocols use cryptography for exchanging messages.
- Attacks against cryptographic primitives involves breaking the algorithm for encryption, etc. Attacks against cryptographic protocols may be of completely logical nature.
- Cryptographic protocols may be insecure even if the underlying cryptographic primitives are completely secure.
- Hence we often separate the study of cryptographic protocols from that of cryptographic primitives.

Difficulty in ensuring correctness of cryptographic protocols

- Infinitely many sessions
- Infinitely many participants
- Infinitely many nonces
- Sessions are interleaved
- Adversary can replace messages by any arbitrary message: infinitely branching system

Back to our example

1. $A \longrightarrow B : \{A, N_a\}_{K_b}$
2. $B \longrightarrow A : \{N_a, N_b\}_{K_a}$
3. $A \longrightarrow B : \{N_b\}_{K_b}$

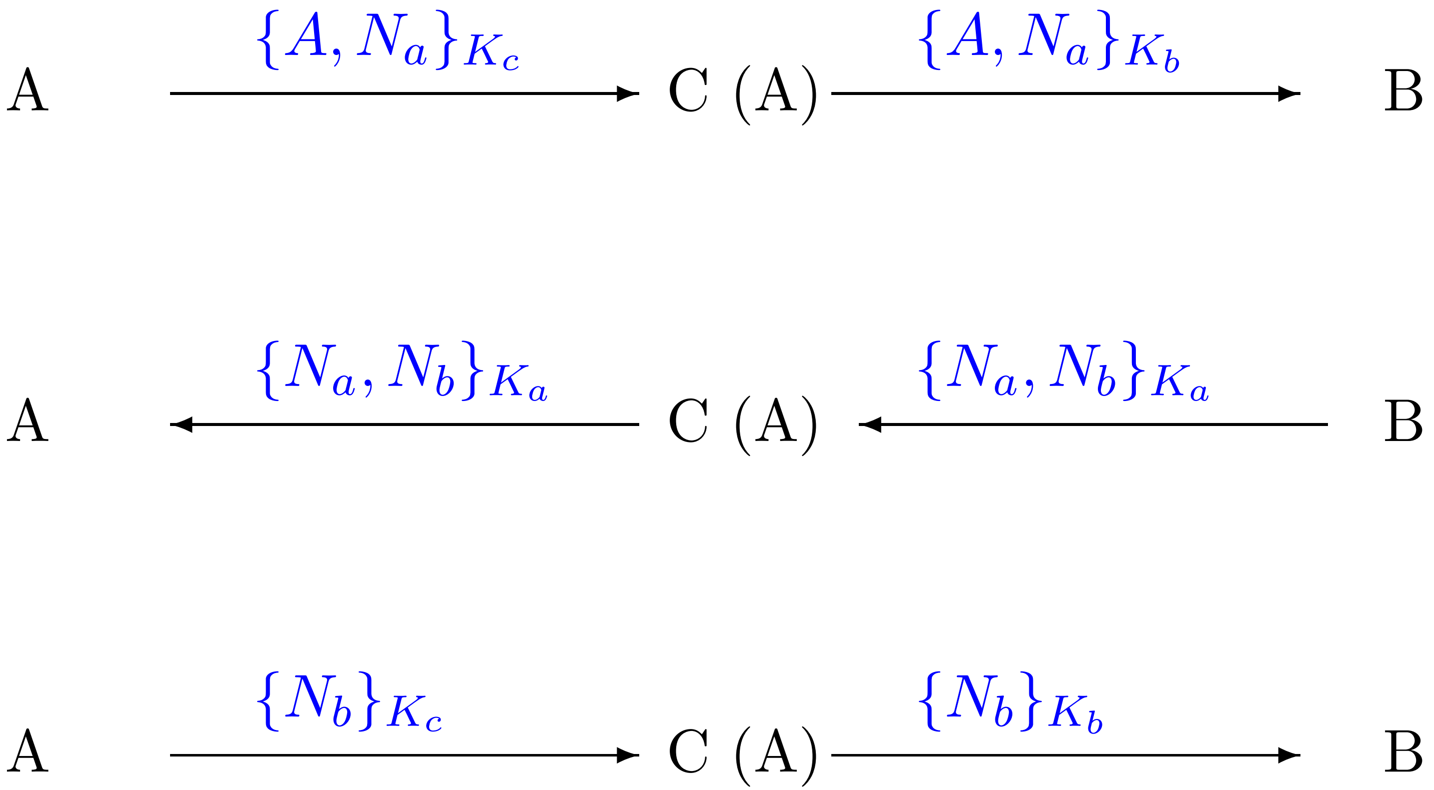
Back to our example

1. $A \longrightarrow B : \{A, N_a\}_{K_b}$
2. $B \longrightarrow A : \{N_a, N_b\}_{K_a}$
3. $A \longrightarrow B : \{N_b\}_{K_b}$

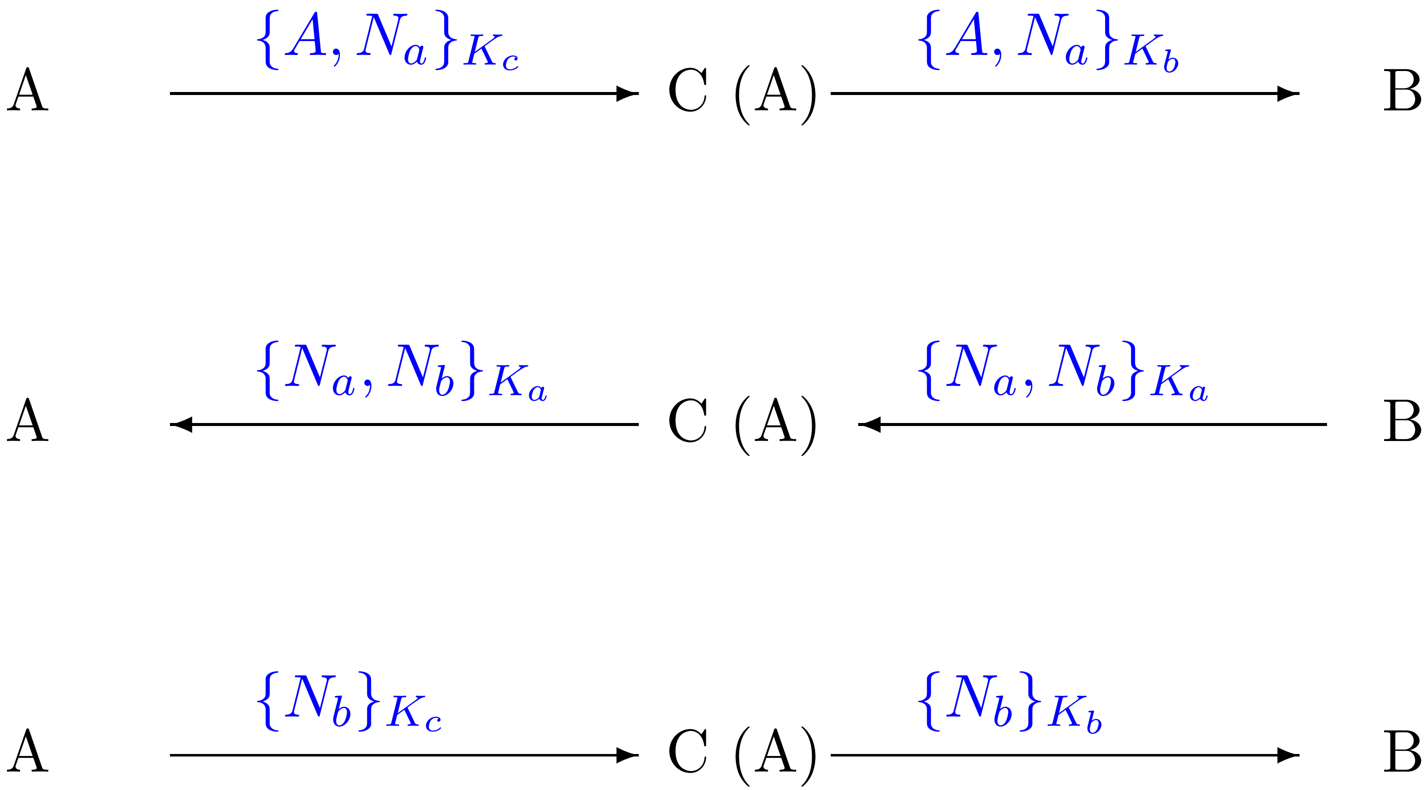
This is the well-known Needham-Schroeder public-key protocol.

Published in 1978. Attack found after 17 years in 1995 by Lowe.

Man in the middle attack



Man in the middle attack



Even very simple protocols may have subtle flaws

Consequences

Suppose B is the server of a bank.

C, who can now pretend to be A:

$$C \longrightarrow B : \{N_a, N_b, \text{ transfer £5000 from account of } A \text{ to account of } C\}_{K_b}$$

A fix: the Needham-Schroeder-Lowe protocol [Lowe,1985]

B includes his identity in the message he sends:

1. $A \longrightarrow B : \{A, Na\}_{K_b}$
2. $B \longrightarrow A : \{B, N_a, N_b\}_{K_a}$
3. $A \longrightarrow B : \{N_b\}_{K_b}$

A fix: the Needham-Schroeder-Lowe protocol [Lowe,1985]

B includes his identity in the message he sends:

1. $A \longrightarrow B : \{A, Na\}_{K_b}$
2. $B \longrightarrow A : \{B, N_a, N_b\}_{K_a}$
3. $A \longrightarrow B : \{N_b\}_{K_b}$

Is it secure?

A variant of the Needham-Schroeder-Lowe protocol

Suppose now we change the place of B in the second message:

1. $A \longrightarrow B : \{A, Na\}_{K_b}$
2. $B \longrightarrow A : \{N_a, N_b, B\}_{K_a}$
3. $A \longrightarrow B : \{N_b\}_{K_b}$

A variant of the Needham-Schroeder-Lowe protocol

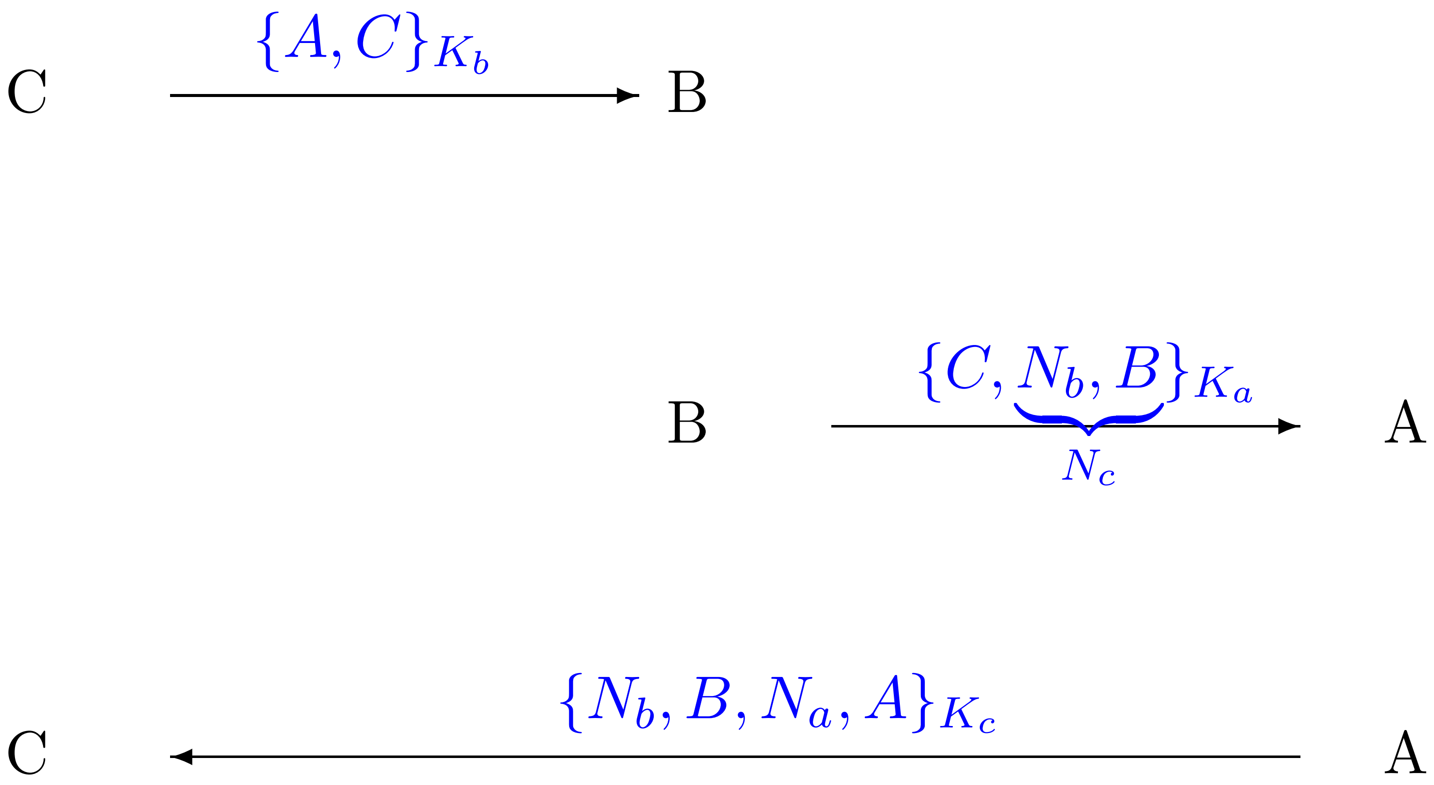
Suppose now we change the place of B in the second message:

1. $A \longrightarrow B : \{A, Na\}_{K_b}$
2. $B \longrightarrow A : \{N_a, N_b, B\}_{K_a}$
3. $A \longrightarrow B : \{N_b\}_{K_b}$

Does this affect security?

Type flaw

An attack on the variant of the Needham-Schroeder-Lowe protocol [Millen]:



The Spi calculus

Abadi, Gordon, 1997

- Extends pi calculus which provides a language for describing processes.
- We treat protocols as processes, where messages sent and received by processes may involve encryption.
- Security is defined as equivalence between processes in the eyes of an arbitrary environment.
- Environment is also a spi calculus process.
- We study information flow to check whether secrets are leaked.

- A process may involve sequences of actions for sending and receiving messages on channels.
- A Processes may contain smaller processes running in parallel.

- A process may involve sequences of actions for sending and receiving messages on channels.
- A Processes may contain smaller processes running in parallel.

Use halt to denote a finished process: it does nothing.

We write $\mathsf{send}_c\langle M\rangle; P$ to denote a process that sends the message M on channel c after which it executes the process P.

$\mathsf{recv}_c(x); Q$ denotes a process that is listening on the channel c.

On receiving some message M on this channel then it executes process $Q[M/x]$.

The process

$$P_1 \triangleq \mathsf{recv}_c(x); \mathsf{send}_d\langle x\rangle; \mathsf{halt}$$

on receiving message M on channel c, sends M on channel d and then halts.

The process

$$P_2 \triangleq \mathsf{send}_c\langle M\rangle; \mathsf{halt}$$

sends M on channel c and halts.

The process

$$P_1 \triangleq \mathsf{recv}_c(x); \mathsf{send}_d\langle x \rangle; \mathsf{halt}$$

on receiving message M on channel c, sends M on channel d and then halts.

The process

$$P_2 \triangleq \mathsf{send}_c\langle M \rangle; \mathsf{halt}$$

sends M on channel c and halts.

Putting them in parallel gives the process

$$P_3 \triangleq P_1 \mid P_2$$

The message sent by P_1 is received by P_2. Hence P_3 as a whole can make a "silent" transition to the process $\mathsf{send}_d\langle M \rangle; \mathsf{halt}$.

Further the process

$$P_5 \triangleq P_3 \mid P_4$$

where

$$P_4 \triangleq \mathsf{recv}_d(x); \mathsf{halt}$$

can halt after making only silent transitions.

Intuitively P_5 represents the protocol

$$P_2 \longrightarrow P_1 : \quad M \quad \text{(on channel } c\text{)}$$

$$P_1 \longrightarrow P_4 : \quad M \quad \text{(on channel } d\text{)}$$

We can restrict access to channels.

The process $\mathsf{new}\ c; P$ creates a fresh channel c and can be used inside process P. No outside process can access c.

(c is like a bound variable whose scope is inside P)

We consider processes to be the same after renaming of bound names.

Consider the process

$$(\mathsf{new}\ c; \mathsf{send}_c\langle M\rangle; \mathsf{halt}) \mid (\mathsf{recv}_c(x); \mathsf{halt})$$

No communication happens between the two smaller processes.

The above process is the same as the following one.

$$(\mathsf{new}\ d; \mathsf{send}_d\langle M\rangle; \mathsf{halt}) \mid (\mathsf{recv}_c(x); \mathsf{halt})$$

Hence new allows us to create channels for secure communication.

Consider the process

$$\mathsf{new}\ c; (\mathsf{send}_c\langle M\rangle; \mathsf{halt} \mid \mathsf{recv}_c(x); P \mid \mathsf{recv}_c(x); Q)$$

Communication can take place between first and second subprocess to create the process $\mathsf{new}\ c; (P[M/x] \mid \mathsf{recv}_c(x); Q)$

Or communication can take place between first and third subprocess to create the process $\mathsf{new}\ c; (\mathsf{recv}_c(x); P \mid Q[M/x])$

Hence new allows us to create channels for secure communication.

Consider the process

$$\mathsf{new}\ c; (\mathsf{send}_c\langle M\rangle; \mathsf{halt} \mid \mathsf{recv}_c(x); P \mid \mathsf{recv}_c(x); Q)$$

Communication can take place between first and second subprocess to create the process $\mathsf{new}\ c; (P[M/x] \mid \mathsf{recv}_c(x); Q)$

Or communication can take place between first and third subprocess to create the process $\mathsf{new}\ c; (\mathsf{recv}_c(x); P \mid Q[M/x])$

However the process

$$(\mathsf{new}\ c; (\mathsf{send}_c\langle M\rangle; \mathsf{halt} \mid \mathsf{recv}_c(x); P)) \mid \mathsf{recv}_c(x); Q$$

can only lead to the process $(\mathsf{new}\ c; P[M/x]) \mid \mathsf{recv}_c(x); Q$

Channels can also be sent as messages. Consider the following protocol where c_{AB} is a freshly created channel whereas c_{AS} and c_{SB} are long term channels.

$$A \longrightarrow S : c_{AB} \text{ on } c_{AS}$$

$$S \longrightarrow B : c_{AB} \text{ on } c_{SB}$$

$$A \longrightarrow B : M \text{ on } c_{AB}$$

can be represented as follows where $F(y)$ is a process involving variable y.

$$\begin{aligned} A &\triangleq \mathsf{new}\ c_{AB}; \mathsf{send}_{c_{AS}}\langle c_{AB}\rangle; \mathsf{send}_{c_{AB}}\langle M\rangle.\mathsf{halt} \\ S &\triangleq \mathsf{recv}_{c_{AS}}(x); \mathsf{send}_{c_{SB}}\langle x\rangle; \mathsf{halt} \\ B &\triangleq \mathsf{recv}_{c_{SB}}(x); \mathsf{recv}_{x}(y); F(y) \\ P &\triangleq \mathsf{new}\ c_{AS}; \mathsf{new}\ c_{SB}; (A \mid S \mid B) \end{aligned}$$

P makes silent transitions to $\mathsf{new}\ c_{AS}; \mathsf{new}\ c_{SB}; F(M)$.

Processes can perform computations like

- encryption, decryption (we will deal with only symmetric key encryption)
- pairing, unpairing
- increments, decrements
- checking equality of messages

Processes can perform computations like

- encryption, decryption (we will deal with only symmetric key encryption)
- pairing, unpairing
- increments, decrements
- checking equality of messages

The process

$$\mathsf{recv}_c(x_1, x_2, x_3); \mathsf{case}\ x_1\ \mathsf{of}$$
$$\{y_1\}_K : \mathsf{check}\ (y_1 == x_2); \mathsf{send}_c\langle y_1, \mathsf{succ}\ (x_3)\rangle; \mathsf{halt}$$

receives an input of the form $\{M\}_K, M, N$ on channel c and sends out $y_1, \mathsf{succ}\ (x_3)$ on channel c.

The syntax

$M ::=$	term
n	name
(M, N)	pair
0	zero
$\mathsf{succ}\ (M)$	successor
$\{M_1, \ldots, M_k\}_N$	encryption
x	variable

$P ::=$	process
$\mathsf{send}_M\langle N_1, \ldots, N_k\rangle; P$	output
$\mathsf{recv}_M(x_1, \ldots, x_k); P$	input
halt	halt
$P \mid Q$	parallel composition
$\mathsf{repeat}\ P$	replication
$\mathsf{new}\ n; P$	restriction
$\mathsf{check}\ (M == N); P$	comparison
$\mathsf{let}\ (x, y) = M; P$	unpairing
$\mathsf{case}\ M\ \mathsf{of}\ 0 : P,\ \mathsf{succ}\ (x) : Q$	integer case analysis
$\mathsf{case}\ M\ \mathsf{of}\ \{x_1, \ldots, x_k\}_N : P$	decryption

Intuitively, $\mathsf{repeat}\ P$ represents infinitely many copies of P running in parallel.

In other words we can consider $\mathsf{repeat}\ P$ to represent $P \mid P \mid P \mid \ldots$

Consider

$$P \triangleq \mathsf{recv}_c(x); \mathsf{halt}$$

$$P_1 \triangleq \mathsf{send}_c(M_1); \mathsf{halt}$$

$$P_2 \triangleq \mathsf{send}_c(M_2); \mathsf{halt}$$

The process

$$P_1 \mid P_2 \mid \mathsf{repeat}\ P$$

can make silent transitions (internal communication) to create the process

$$\mathsf{repeat}\ P$$

A one message protocol using cryptography, where K_{AB} is a symmetric key shared between A and B for private communication.

$$A \longrightarrow B : \{M\}_{K_{AB}} \text{ on } c_{AB}$$

This can be represented as

$$\begin{aligned} A &\triangleq \mathsf{send}_{c_{AB}}\langle\{M\}_{K_{AB}}\rangle; \mathsf{halt} \\ B &\triangleq \mathsf{recv}_{c_{AB}}(x); \mathsf{case}\ x\ \mathsf{of}\ \{y\}_{K_{AB}} : F(y) \\ P &\triangleq \mathsf{new}\ K_{AB}; (A \mid B) \end{aligned}$$

The key K_{AB} is restricted, only A and B can use it.

The channel c_{AB} is public. Other principals may send messages on it or listen on it.

P can make silent transitions to $\mathsf{new}\ K_{AB}; F(M)$.

Formal semantics

We now need to define how processes execute.

For example we would like

$$\mathsf{send}_c\langle M\rangle; P \mid \mathsf{recv}_c(x); Q \xrightarrow{\tau} P \mid Q[M/x]$$

where τ denotes a silent action (internal communication).

Let $fn(M)$ and $fn(P)$ be the set of free names in term M and process P respectively.

Let $fv(M)$ and $fv(P)$ be the set of free variables in term M and process P respectively.

Closed processes are processes without any free variables.

Let $P \triangleq \mathsf{new}\ c; \mathsf{new}\ K; \mathsf{recv}_d(x); \mathsf{case}\ x\ \mathsf{of}\ \{y\}_{K'} : \mathsf{send}_d\langle\{y\}_K, z, c\rangle; \mathsf{halt}$.

We have

$fn(\mathsf{send}_d\langle\{y\}_K, z, c\rangle; \mathsf{halt}) = \{c, d, K\}$

$fv(\mathsf{send}_d\langle\{y\}_K, z, c\rangle; \mathsf{halt}) = \{y, z\}$

$fn(\mathsf{case}\ x\ \mathsf{of}\ \{y\}_{K'} : \mathsf{send}_d\langle\{y\}_K, z, c\rangle; \mathsf{halt}) = \{c, d, K, K'\}$

$fv(\mathsf{case}\ x\ \mathsf{of}\ \{y\}_{K'} : \mathsf{send}_d\langle\{y\}_K, z, c\rangle; \mathsf{halt}) = \{x, z\}$

$fn(P) = \{d, K'\}$

$fv(P) = \{z\}$

$fn(\{y\}_K) = \{K\}$

$fv(\{y\}_K) = \{y\}$

First we define reduction relation $>$ on closed processes:

$$
\begin{array}{rll}
\mathsf{repeat}\ P & > P \mid \mathsf{repeat}\ P & \text{(R-Repeat)} \\
\mathsf{check}\ (M == M); P & > P & \text{(R-Check)} \\
\mathsf{let}\ (x, y) = (M, N); P & > P[M/x, N/y] & \text{(R-Let)} \\
\mathsf{case}\ 0\ \mathsf{of}\ 0 : P,\ \mathsf{succ}\ (x) : Q & > P & \text{(R-Zero)} \\
\mathsf{case}\ \mathsf{succ}\ (M)\ \mathsf{of}\ 0 : P,\ \mathsf{succ}\ (x) : Q & > Q[M/x] & \text{(R-Succ)} \\
\mathsf{case}\ \{M\}_N\ \mathsf{of}\ \{x\}_N : P & > P[M/x] & \text{(R-decrypt)}
\end{array}
$$

When these rules cannot be applied, it means that the process cannot be simplified.

The following processes cannot be simplified, hence cannot be executed further.

$\mathsf{check}\ (0 == \mathsf{succ}\ (0); P$ (comparison fails).

$\mathsf{let}\ (x, y) = 0; P$ (unpairing fails)

$\mathsf{case}\ (M, N)\ \mathsf{of}\ 0 : P,\ \mathsf{succ}\ (x) : Q$ (not an integer)

$\mathsf{case}\ (M, N)\ \mathsf{of}\ \{x, y\}_K : P$ (not an encrypted message)

$\mathsf{case}\ \{M, N\}_{K'}\ \mathsf{of}\ \{x, y\}_K : P$ where $K \neq K'$

When these rules cannot be applied, it means that the process cannot be simplified.

The following processes cannot be simplified, hence cannot be executed further.

$\mathsf{check}\ (0 == \mathsf{succ}\ (0); P$ (comparison fails).

$\mathsf{let}\ (x, y) = 0; P$ (unpairing fails)

$\mathsf{case}\ (M, N)\ \mathsf{of}\ 0 : P,\ \mathsf{succ}\ (x) : Q$ (not an integer)

$\mathsf{case}\ (M, N)\ \mathsf{of}\ \{x, y\}_K : P$ (not an encrypted message)

$\mathsf{case}\ \{M, N\}_{K'}\ \mathsf{of}\ \{x, y\}_K : P$ where $K \neq K'$

This is also based on the perfect cryptography assumption: distinct terms represent distinct messages.

A barb β is either

- a name n (representing input on channel n), or
- a co-name $\overline{n}$ (representing output on channel n)

An action is either

- a barb (representing input or output to the outside world), or
- τ (representing a silent action i.e. internal communication)

We write $P \xrightarrow{\alpha} Q$ to mean that P makes action α after which Q is the remaining process that is left to be executed.

Commitment relation Consider again $\mathsf{send}_c\langle M\rangle; P \mid \mathsf{recv}_c(x); Q$

Commitment relation Consider again $\mathsf{send}_c\langle M\rangle; P \mid \mathsf{recv}_c(x); Q$

The first subprocess makes an output action on channel c.

We will represent it as $\mathsf{send}_c\langle M\rangle; P \xrightarrow{\overline{c}} \langle M\rangle P$.

$\langle M\rangle P$ is called a concretion: it represents a commitment to output message M after which P will be executed.

Commitment relation Consider again $\mathsf{send}_c\langle M\rangle; P \mid \mathsf{recv}_c(x); Q$

The first subprocess makes an output action on channel c.

We will represent it as $\mathsf{send}_c\langle M\rangle; P \xrightarrow{\overline{c}} \langle M\rangle P$.

$\langle M\rangle P$ is called a concretion: it represents a commitment to output message M after which P will be executed.

The second subprocess makes an input action on channel c.

We will represent it as $\mathsf{recv}_c(x); Q \xrightarrow{c} (x)Q$.

$(x)Q$ is called an abstraction:it represents a commitment to input some x after which P will be executed.

Commitment relation Consider again $\mathsf{send}_c\langle M\rangle; P \mid \mathsf{recv}_c(x); Q$

The first subprocess makes an output action on channel c.

We will represent it as $\mathsf{send}_c\langle M\rangle; P \xrightarrow{\overline{c}} \langle M\rangle P$.

$\langle M\rangle P$ is called a concretion: it represents a commitment to output message M after which P will be executed.

The second subprocess makes an input action on channel c.

We will represent it as $\mathsf{recv}_c(x); Q \xrightarrow{c} (x)Q$.

$(x)Q$ is called an abstraction:it represents a commitment to input some x after which P will be executed.

Abstractions and concretions can be combined:

$\langle M\rangle P \;@\; (x)Q = P \mid Q[M/x]$