Security protocols

For secure communication over an insecure network.

- Adversary can spy on messages,
- delete messages,
- modify messages,
- impersonate as Alice to Bob,
- deny having sent or received a message
- . . .

Encrypting and decrypting messages

. . . the naive way:

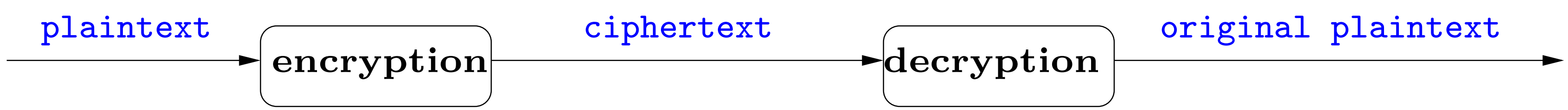
Instead of Alice $\longrightarrow$ Bob:

`This is Alice.  My credit card number is 1234567890123456`

We have Alice $\longrightarrow$ Bob:

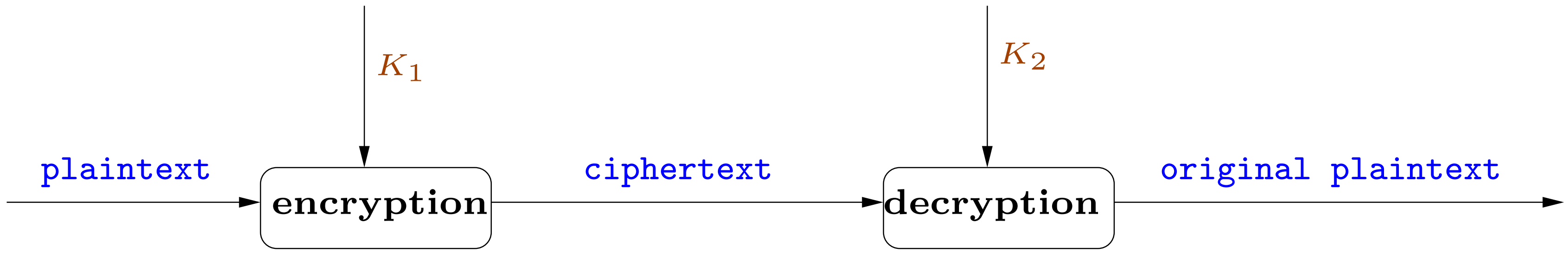
`6543210987654321 si rebmun drac tiderc yM .ecilA si sihT`

Alice and Bob agree on the method of encryption and decryption.



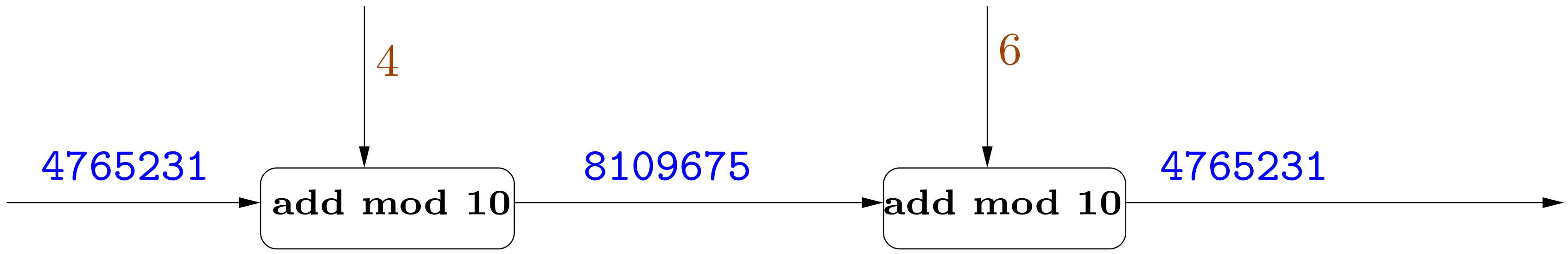
Cryptography with keys

Today we instead have the following picture:



The encryption and decryption algorithms are assumed to be publicly known.

The security lies in the (secret) keys.



Cryptography of the pre-computer age **Substitution ciphers**: each character is mapped to the another character. The famous Caesar cipher: A $\rightarrow$ D, B $\rightarrow$ E, ..., Z $\rightarrow$ C.

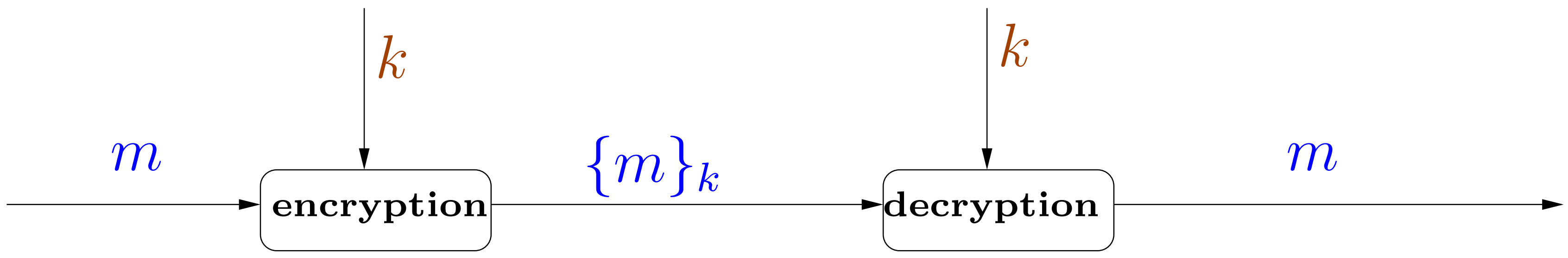
transposition cipher: shuffling around of characters.

Plaintext: `this is alice my credit card number is 1234567890123456`

```
thisisalic
emycreditc
ardnumberi
s123456789
0123456
```

Ciphertext: `teas0 hmr11 iyd22 scn33 iru44 sem55 adb66 lie7i tr8cc i9`

Private key cryptography



- The same key k is used for encryption and decryption
- Given message m and key k, we can compute the encrypted message $\{m\}_k$
- Given the encrypted message $\{m\}_k$ and the key k, we can compute the original message m

Private key cryptography

Suppose K_{ab} is a private key shared between A and B.

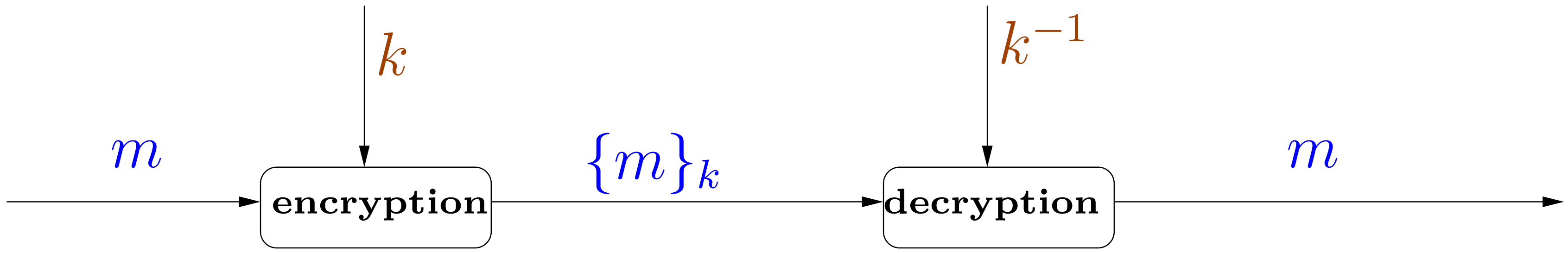
A can send a message m to B using private key cryptography:

$$A \longrightarrow B : \{m\}_{K_{ab}}$$

Only B can get back the message m.

A and B need to agree beforehand on a key K_{ab} which should not be disclosed to any one else

Public key cryptography



- A chooses pair (K_a, K_a^{-1}) of keys such that
  - messages encrypted with K_a can be decrypted with K_a^{-1}
  - K_a^{-1} cannot be calculated from K_a
- A makes K_a public: this is the public key of A
- A keeps K_a^{-1} secret: this is the private key of A

Public key cryptography

Then any B can send a message to A which only A can read:

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

Sometimes we have the additional property: messages encrypted with K_a^{-1} can be decrypted with K_a

Then A can send a message m to B

$$A \longrightarrow B : \{m\}_{K_a^{-1}}$$

and B is sure that the message m was encrypted by A. Hence we have authentication

One way hash functions

Properties of a one way hash function H:

– Given M, it is easy to compute $H(M)$ (called message digest).
– Given $H(M)$ is is difficult to find M' such that $H(M) = H(M')$.

A sends to B the message M together with the encrypted hash value $\{H(M)\}_{K_{ab}}$.

Efficient means of demonstrating authenticity, since $H(M)$ is of a fixed size.

Cryptography is not enough!

Intruder is more clever. He can attack even if the cryptographic algorithms are perfect.

Alice tells Bank to transfer £5000 to Charlie's (intruder) account:

$$A \longrightarrow B : \{A, B, \text{ transfer 5000 euros } \ldots\}_{K_{ab}}$$

- B believes that message comes from A
- Charlie has no way to decrypt the message

Cryptography is not enough!

Intruder is more clever. He can attack even if the cryptographic algorithms are perfect.

Alice tells Bank to transfer £5000 to Charlie's (intruder) account:

$$A \longrightarrow B : \{A, B, \text{ transfer 5000 euros } \ldots\}_{K_{ab}}$$

- B believes that message comes from A
- Charlie has no way to decrypt the message
- But: Charlie can send the same message again to the bank

Intruder can replay known messages (freshness attack)

Solution: use session key

Generate fresh random value (nonce) for each new session and use it as a key for that session.

Solution: use session key

Generate fresh random value (nonce) for each new session and use it as a key for that session.

How to agree on a fresh key for each session?

Solution: use session key

Generate fresh random value (nonce) for each new session and use it as a key for that session.

How to agree on a fresh key for each session?

A sends to B the new key K_{ab} at the beginning of the session:

$$A \longrightarrow B : K_{ab}$$

And then uses it during that session.

Solution: use session key

Generate fresh random value (nonce) for each new session and use it as a key for that session.

How to agree on a fresh key for each session?

A sends to B the new key K_{ab} at the beginning of the session:

$$A \longrightarrow B : K_{ab}$$

And then uses it during that session.

Doesn't work. What about

$$A \longrightarrow B : \{K_{ab}\}_{K_{long}}$$

Using a long term key to agree on a session key.

A more complex solution A and B both choose a nonce each.

$$
\begin{array}{ll}
1. & A \longrightarrow B : \{A, N_a\}_{K_b} \\
2. & B \longrightarrow A : \{N_a, N_b\}_{K_a} \\
3. & A \longrightarrow B : \{N_b\}_{K_b}
\end{array}
$$

A more complex solution A and B both choose a nonce each.

$$
\begin{array}{ll}
1. & A \longrightarrow B : \{A, N_a\}_{K_b} \\
2. & B \longrightarrow A : \{N_a, N_b\}_{K_a} \\
3. & A \longrightarrow B : \{N_b\}_{K_b}
\end{array}
$$

The second message is to assure A that B is active and N_b is fresh.
The third message is to assure B that A is active and N_a is fresh.

A more complex solution A and B both choose a nonce each.

$$
\begin{array}{ll}
1. & A \longrightarrow B : \{A, N_a\}_{K_b} \\
2. & B \longrightarrow A : \{N_a, N_b\}_{K_a} \\
3. & A \longrightarrow B : \{N_b\}_{K_b}
\end{array}
$$

The second message is to assure A that B is active and N_b is fresh.
The third message is to assure B that A is active and N_a is fresh.

Expected security property: N_a and N_b are known only to A and B.
Expected authentication property: A and B are assured that they are talking to each other.

$$A \longrightarrow B : \{A, B, N_a, N_b \text{ transfer 5000 euros } \ldots\}_{K_b}$$

A more complex solution A and B both choose a nonce each.

$$
\begin{array}{ll}
1. & A \longrightarrow B : \{A, N_a\}_{K_b} \\
2. & B \longrightarrow A : \{N_a, N_b\}_{K_a} \\
3. & A \longrightarrow B : \{N_b\}_{K_b}
\end{array}
$$

The second message is to assure A that B is active and N_b is fresh.
The third message is to assure B that A is active and N_a is fresh.

Expected security property: N_a and N_b are known only to A and B.
Expected authentication property: A and B are assured that they are talking to each other.

$$A \longrightarrow B : \{A, B, N_a, N_b \text{ transfer 5000 euros } \ldots\}_{K_b}$$

How secure is this ? How to guarantee security ?

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

$$\mathrm{A} \xrightarrow{\{A, N_a\}_{K_c}} \mathrm{C}\ (\mathrm{A}) \xrightarrow{\{A, N_a\}_{K_b}} \mathrm{B}$$

$$\mathrm{A} \xleftarrow{\{N_a, N_b\}_{K_a}} \mathrm{C}\ (\mathrm{A}) \xleftarrow{\{N_a, N_b\}_{K_a}} \mathrm{B}$$

$$\mathrm{A} \xrightarrow{\{N_b\}_{K_c}} \mathrm{C}\ (\mathrm{A}) \xrightarrow{\{N_b\}_{K_b}} \mathrm{B}$$

Man in the middle attack

$$\text{A} \xrightarrow{\quad \{A, N_a\}_{K_c} \quad} \text{C (A)} \xrightarrow{\quad \{A, N_a\}_{K_b} \quad} \text{B}$$

$$\text{A} \xleftarrow{\quad \{N_a, N_b\}_{K_a} \quad} \text{C (A)} \xleftarrow{\quad \{N_a, N_b\}_{K_a} \quad} \text{B}$$

$$\text{A} \xrightarrow{\quad \{N_b\}_{K_c} \quad} \text{C (A)} \xrightarrow{\quad \{N_b\}_{K_b} \quad} \text{B}$$

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]:

$$C \xrightarrow{\{A, C\}_{K_b}} B$$

$$B \xrightarrow{\{C, \underbrace{N_b, B}_{N_c}\}_{K_a}} A$$

$$C \xleftarrow{\{N_b, B, N_a, A\}_{K_c}} A$$

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_2 is received by P_1. 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

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

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

$$\begin{aligned} P &\triangleq \mathsf{recv}_c(x); \mathsf{halt} \\ P_1 &\triangleq \mathsf{send}_c(M_1); \mathsf{halt} \\ P_2 &\triangleq \mathsf{send}_c(M_2); \mathsf{halt} \end{aligned}$$

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 $\mathsf{new}\ K_{AB}; F(M)$.