Example

Given statement $Alice$ says $(\mathsf{s}_1 \wedge \mathsf{s}_2)$ how do we conclude that $Alice$ says s_1.

Example

Given statement *Alice* says $(\mathsf{s}_1 \wedge \mathsf{s}_2)$ how do we conclude that *Alice* says s_1.

We use the following steps.

$(\mathsf{s}_1 \wedge \mathsf{s}_2) \longrightarrow \mathsf{s}_1$	by (1)
Alice says $((\mathsf{s}_1 \wedge \mathsf{s}_2) \longrightarrow \mathsf{s}_1)$	by (4)
Alice says s_1	by (3)

Axioms about principals

5 $(P \wedge Q)$ says s $\equiv$ $(P$ says s$) \wedge (Q$ says s$)$

6 $(P \mid Q)$ says s $\equiv$ P says $(Q$ says s$)$

7 $(P = Q) \rightarrow (P$ says s $\equiv$ Q says s$)$

$=$ is equality on principals.

8 $(P_1 \mid (P_2 \mid P_3)) = ((P_1 \mid P_2) \mid P_3)$

Quoting is associative.

9 $(P_1 \mid (P_2 \wedge P_3)) = (P_1 \mid P_2) \wedge (P_1 \mid P_3)$

Quoting distributes over conjunction

10 $(P \Rightarrow Q) \equiv (P = P \wedge Q)$

11 $(P \text{ says } (Q \Rightarrow P)) \rightarrow (Q \Rightarrow P)$

A principal is free to choose a representative.

Example We want to conclude s from the three statements:

– $(Alice \wedge Bob)$ says $(Charlie \Rightarrow (Alice \wedge Bob))$

– $Charlie \mid Alice$ says s

– $(Alice$ says $\mathsf{s}) \rightarrow \mathsf{s}$

$(Alice \wedge Bob)$ says $(Charlie \Rightarrow (Alice \wedge Bob))$	
$\quad \rightarrow (Charlie \Rightarrow (Alice \wedge Bob))$	by (11)
$(Charlie \Rightarrow (Alice \wedge Bob))$	by (2)
$Charlie = (Charlie \wedge Alice \wedge Bob)$	by (10)
$Charlie$ says $(Alice$ says $\mathsf{s})$	by (6)
$(Charlie \wedge Alice \wedge Bob)$ says $(Alice$ says $\mathsf{s})$	by (7,2)

$Alice$ says ($Alice$ says s) by (5,1,2)

$Alice$ says (($Alice$ says s)$\rightarrow$s) by (4)

$Alice$ says s by (3)

s by (2)

Modeling Java stack inspection using ABLP

Wallach, Felten, 1998

Code can be digitally signed by a signer. We treat code, public keys and signers as principals. Stack frames created during execution of code are also treated as principals. Targets (resources to be protected) are also treated as principals.

Modeling Java stack inspection using ABLP

Wallach, Felten, 1998

Code can be digitally signed by a signer. We treat code, public keys and signers as principals. Stack frames created during execution of code are also treated as principals. Targets (resources to be protected) are also treated as principals.

If K is a public key of S then we have the statement

$$K \Rightarrow S \tag{S1}$$

Modeling Java stack inspection using ABLP

Wallach, Felten, 1998

Code can be digitally signed by a signer. We treat code, public keys and signers as principals. Stack frames created during execution of code are also treated as principals. Targets (resources to be protected) are also treated as principals.

If K is a public key of S then we have the statement

$$K \Rightarrow S \tag{S1}$$

If some code C was signed and K is the corresponding public key then we have the statement

$$K \text{ says } (C \Rightarrow K) \tag{S2}$$

If F is the stack frame generated for executing code C then we have the statement

$$F \Rightarrow C \tag{S3}$$

Frame credentials Φ = set of all valid statements of the form S1,S2 and S3.

If F is the stack frame generated for executing code C then we have the statement

$$F \Rightarrow C \qquad \text{(S3)}$$

Frame credentials Φ = set of all valid statements of the form S1,S2 and S3.

Note that from K says $(C \Rightarrow K)$ using (11) we can conclude $C \Rightarrow K$.

Further we can show transitivity of $\Rightarrow$: given $A \Rightarrow B$ and $B \Rightarrow C$ we have:
$A = A \wedge B$ by (10)
$B = B \wedge C$ by (10)
Hence $A = A \wedge B \wedge C = A \wedge C$
Hence we have $A \Rightarrow C$

If F is the stack frame generated for executing code C then we have the statement

$$F \Rightarrow C \tag{S3}$$

Frame credentials Φ = set of all valid statements of the form S1,S2 and S3.

Note that from K says $(C \Rightarrow K)$ using (11) we can conclude $C \Rightarrow K$.

Further we can show transitivity of $\Rightarrow$: given $A \Rightarrow B$ and $B \Rightarrow C$ we have:
$A = A \wedge B$ by (10)
$B = B \wedge C$ by (10)
Hence $A = A \wedge B \wedge C = A \wedge C$
Hence we have $A \Rightarrow C$

Hence from S1, S2 and S3 we can conclude $F \Rightarrow S$.

For each target T we treat $\mathsf{Ok}(T)$ as an atomic statement.

It means that access to T is permitted.

We consider the axiom

$$(T \;\mathsf{says}\; \mathsf{Ok}(T)) \rightarrow \mathsf{Ok}(T) \tag{S4}$$

A target is always free to grant permission to itself.

Targets are dummy principals. They never speak, but other (non-dummy) principals representing them may speak for them.

Target credentials $\mathcal{T}$ is the set of such axioms for all targets T.

Policy for a virtual machine M is defined by a set

access credentials $\mathcal{A}_{\mathsf{M}}$ of statements of the form $P \Rightarrow T$ where P is a principal and T is a target.

This rule means that the local policy of virtual machine M allows P to access T.

Stacks

During execution, at any point of time, a stack frame F has a belief set $\mathcal{B}_F$

This is updated as follows.

Stacks

During execution, at any point of time, a stack frame F has a belief set $\mathcal{B}_F$

This is updated as follows.

Starting the program For the initial stack frame F_0

$\mathcal{B}_{F_0} = \{\mathsf{Ok}(T) \mid T \text{ is a target}\}$.

Stacks

During execution, at any point of time, a stack frame F has a belief set $\mathcal{B}_F$

This is updated as follows.

Starting the program For the initial stack frame F_0

$\mathcal{B}_{F_0} = \{\mathsf{Ok}(T) \mid T \text{ is a target}\}$.

Enabling privileges

If stack frame F calls $\mathsf{enablePrivilege}(T)$ then we update: $\mathcal{B}_F := \mathcal{B}_F \cup \{\mathsf{Ok}(T)\}$.

Stacks

During execution, at any point of time, a stack frame F has a belief set $\mathcal{B}_F$

This is updated as follows.

Starting the program For the initial stack frame F_0

$\mathcal{B}_{F_0} = \{\mathsf{Ok}(T) \mid T \text{ is a target}\}$.

Enabling privileges

If stack frame F calls $\mathsf{enablePrivilege}(T)$ then we update: $\mathcal{B}_F := \mathcal{B}_F \cup \{\mathsf{Ok}(T)\}$.

Function calls

Function call from stack frame F creates a new stack frame G.

$\mathcal{B}_G = \{F \ \mathsf{says}\ \mathsf{s} \mid \mathsf{s} \in \mathcal{B}_F\}$.

Disabling privileges

If stack frame F calls $\mathsf{disablePrivilege}(T)$ then we update

$\mathcal{B}_F := \mathcal{B}_F \setminus \{\mathsf{s} \mid \mathsf{Ok}(T) \text{ occurs in } \mathsf{s}\}$

Disabling privileges

If stack frame F calls $\mathsf{disablePrivilege}(T)$ then we update $\mathcal{B}_F := \mathcal{B}_F \setminus \{\mathsf{s} \mid \mathsf{Ok}(T) \text{ occurs in } \mathsf{s}\}$

Reverting privileges

If stack frame F calls $\mathsf{revertPrivilege}(T)$ then we update $\mathcal{B}_F := \mathcal{B}_F \setminus \{\mathsf{Ok}(T)\}$

Disabling privileges

If stack frame F calls $\mathsf{disablePrivilege}(T)$ then we update

$\mathcal{B}_F := \mathcal{B}_F \setminus \{\mathsf{s} \mid \mathsf{Ok}(T) \text{ occurs in } \mathsf{s}\}$

Reverting privileges

If stack frame F calls $\mathsf{revertPrivilege}(T)$ then we update $\mathcal{B}_F := \mathcal{B}_F \setminus \{\mathsf{Ok}(T)\}$

Checking privileges

When F calls $\mathsf{checkPrivilege}(T)$ then we check that $\mathsf{Ok}(T)$ can be concluded from the set

$\Phi \cup \mathcal{T} \cup \mathcal{A}_{\mathsf{M}} \cup \{F \ \mathsf{says}\ \mathsf{s} \mid \mathsf{s} \in \mathcal{B}_F\}$.

Example Assume at the beginning that $\mathcal{B}_{F_1} = \{\}$.

Now F_1 calls $\mathsf{enablePrivilege}(T_1)$. We have $\mathcal{B}_{F_1} = \{\mathsf{Ok}(T_1)\}$.

F_1 calls $\mathsf{checkPrivilege}(T_1)$.

Hence we take the statement F_1 says $\mathsf{Ok}(T_1)$.

Let S_1 be the signer of the code which produced the frame F_1.

Then we conclude $F_1 \Rightarrow S_1$ from the frame credentials Φ.

If the access credentials set $\mathcal{A}_{\mathsf{M}}$ has a statement $S_1 \Rightarrow T_1$

then using the statement $(T_1 \;\mathsf{says}\; \mathsf{Ok}(T_1)) \rightarrow \mathsf{Ok}(T_1)$ from T

we conclude $\mathsf{Ok}(T_1)$.

Now F_1 makes a function call and the new frame F_2 calls $\mathsf{enablePrivilege}(T_2)$.

We have $\mathcal{B}_{F_2} = \{F_1 \text{ says } \mathsf{Ok}(T_1), \mathsf{Ok}(T_2)\}$

F_2 makes function call and the new frame F_3 calls $\mathsf{disablePrivilege}(T_1)$.

We have $\mathcal{B}_{F_3} = \{F_2 \text{ says } \mathsf{Ok}(T_2)\}$.

F_3 makes function call and the new frame F_4 calls $\mathsf{enablePrivilege}(T_2)$.

We have $\mathcal{B}_{F_4} = \{(F_3 \mid F_2) \text{ says } \mathsf{Ok}(T_2), \mathsf{Ok}(T_2)\}$.

F_4 calls $\mathsf{revertPrivilege}(T_2)$.

We have $\mathcal{B}_{F_4} = \{(F_3 \mid F_2) \text{ says } \mathsf{Ok}(T_2)\}$.

Now F_4 calls $\mathsf{checkPrivilege} T_2$.

We take the statement $(F_4 \mid F_3 \mid F_2)$ says $\mathsf{Ok}(T_2)$ i.e.

F_4 says $(F_3$ says $(F_2$ says $\mathsf{Ok}(T_2)))$.

Suppose from the frame credentials Φ imply that

$F_4 \Rightarrow S_4 \quad F_3 \Rightarrow S_3 \quad F_2 \Rightarrow S_2$

Suppose that $\mathcal{A}_{\mathsf{M}}$ further has statements

$S_4 \Rightarrow T_2 \quad S_3 \Rightarrow T_2 \quad S_2 \Rightarrow T_2$

Then we conclude:

T_2 says $(F_3$ says $(F_2$ says $\mathsf{Ok}(T_2)))$
T_2 says $(T_2$ says $(F_2$ says $\mathsf{Ok}(T_2)))$

T_2 says (T_2 says (T_2 says Ok(T_2)))

Further (T_2 says Ok(T_2))→Ok(T_2) is in $\mathcal{T}$.

Hence T_2 says (T_2 says ((T_2 says Ok(T_2))→Ok(T_2))).

Hence T_2 says (T_2 says Ok(T_2)).

Similarly T_2 says Ok(T_2).

Hence Ok(T_2).

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:

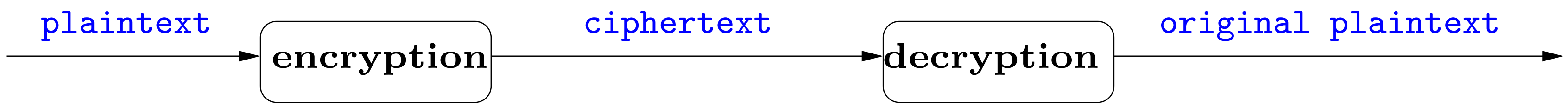
Instead of Alice ⟶ Bob:

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

We have Alice ⟶ Bob:

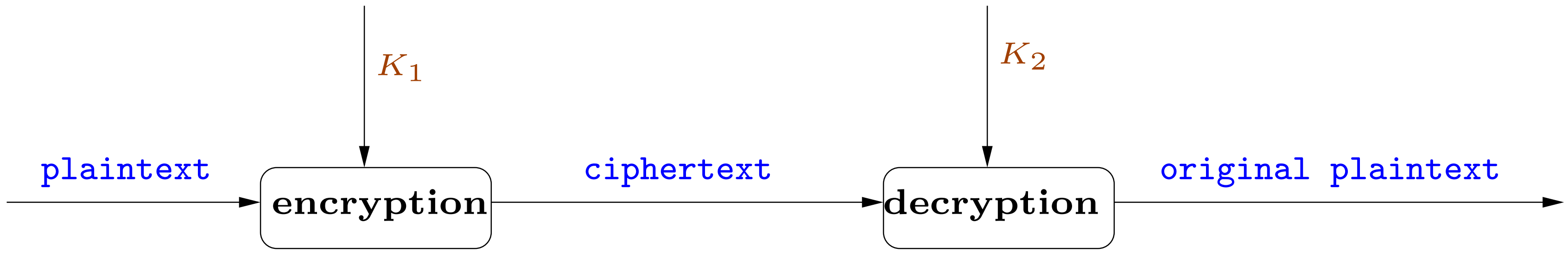
`6543210987654321 si rebmun drac tiderc yM .ecilA si sihT`

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



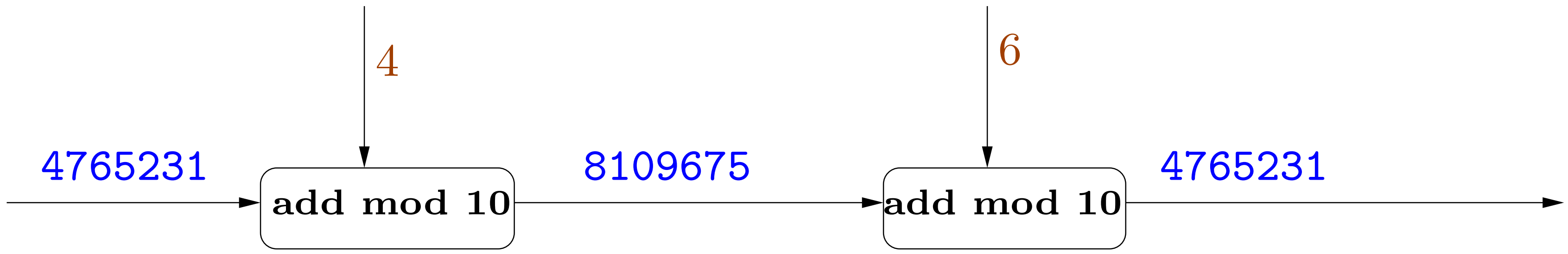
Cryptography with keys

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 $\to$ D, B $\to$ E, ..., Z $\to$ C.

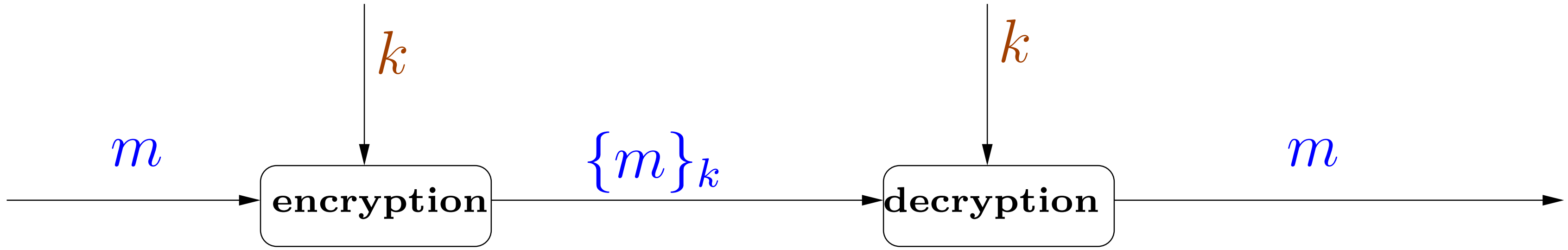
transposition cipher: shuffling around of characters.

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.

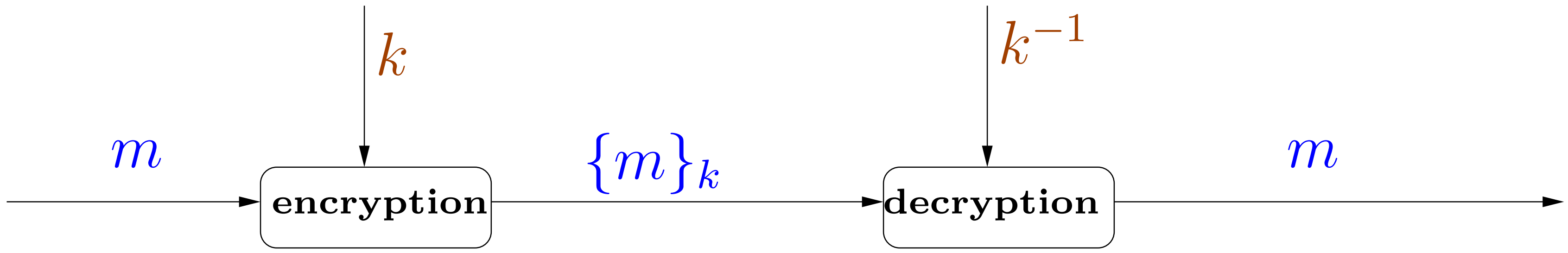
A can send a message m to B using private key cryptography:

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

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.

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

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

$$1. \quad A \longrightarrow B : \{A, N_a\}_{K_b}$$

$$2. \quad B \longrightarrow A : \{N_a, N_b\}_{K_a}$$

$$3. \quad A \longrightarrow B : \{N_b\}_{K_b}$$

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.

$$1. \quad A \longrightarrow B : \{A, N_a\}_{K_b}$$

$$2. \quad B \longrightarrow A : \{N_a, N_b\}_{K_a}$$

$$3. \quad A \longrightarrow B : \{N_b\}_{K_b}$$

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.

$$1. \quad A \longrightarrow B : \{A, N_a\}_{K_b}$$

$$2. \quad B \longrightarrow A : \{N_a, N_b\}_{K_a}$$

$$3. \quad A \longrightarrow B : \{N_b\}_{K_b}$$

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 ?