

# Timed Spi-Calculus with Types for Secrecy and Authenticity

Christian Haack

CTI, DePaul University

`fpl.cs.depaul.edu/chaack`

Alan Jeffrey

Bell Labs, Lucent Technology

`fpl.cs.depaul.edu/ajeffrey`

# Type Systems for Protocol Verification

- Type systems for verifying secrecy and/or authenticity in the spi-calculus:
  - Abadi, Abadi/Blanchet, Gordon/Jeffrey, and more.
- Other methods and systems for protocol verification:
  - model checking (e.g. Casper), BAN logic, proof assistants (e.g. Isabelle), automatic theorem provers (e.g. ProVerif), static analysis, and more.

# Type Systems for Protocol Verification

- Type systems for verifying secrecy and/or authenticity in the spi-calculus:
  - Abadi, Abadi/Blanchet, Gordon/Jeffrey, and more.
- Other methods and systems for protocol verification:
  - model checking (e.g. Casper), BAN logic, proof assistants (e.g. Isabelle), automatic theorem provers (e.g. ProVerif), static analysis, and more.
- Typechecking (our approach):
  - Human help required: **type annotations**.
  - **No finiteness needed**. For instance, arbitrary many session runs are allowed.
  - Our type systems are **incomplete**.

# Why Timed Spi?

- Untimed spi-calculus models perfect cryptography.

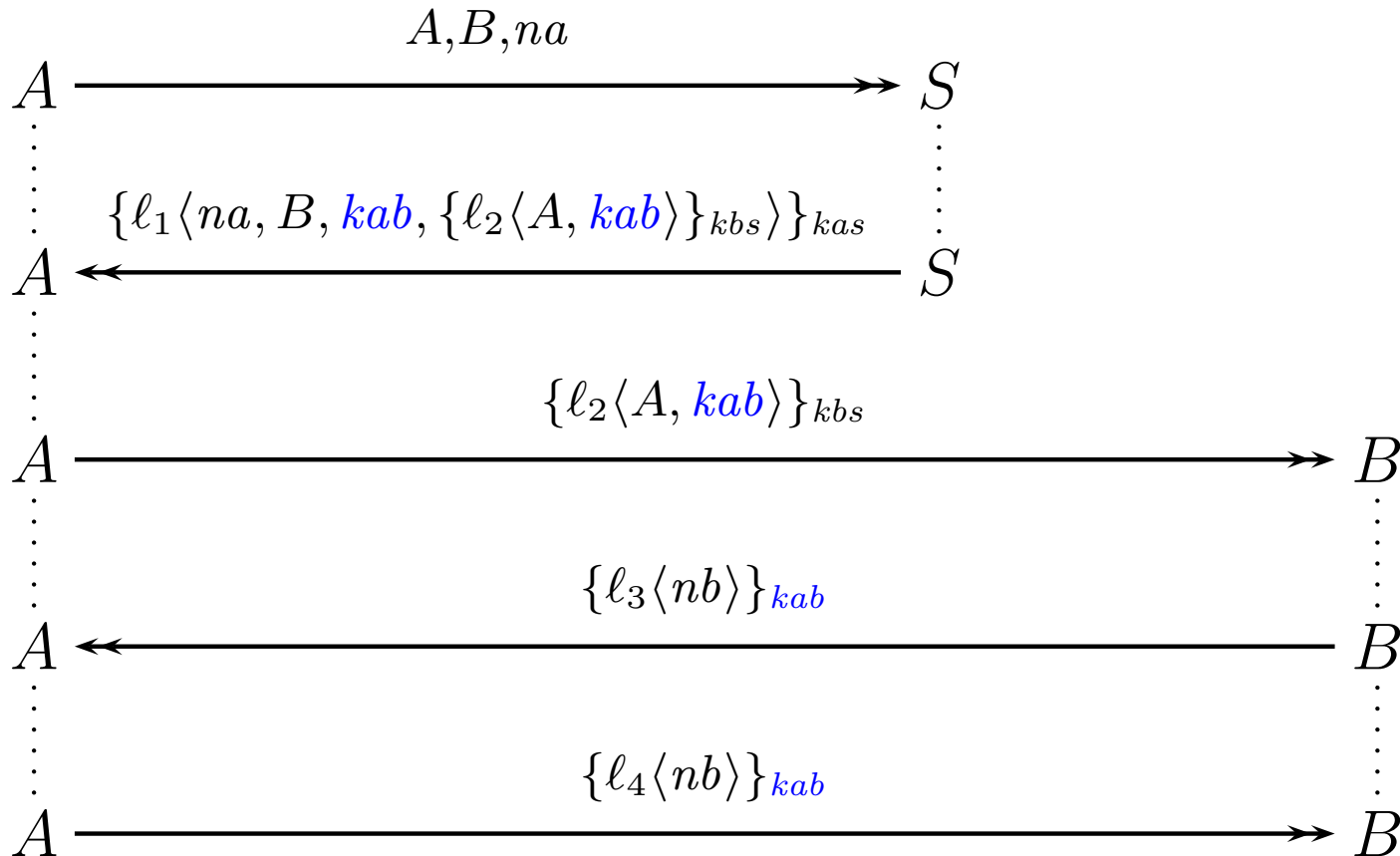
# Why Timed Spi?

- Untimed spi-calculus models perfect cryptography.
- A more realistic model:
  - Distinguish between long- and short-term keys.
  - Short-term keys can be cracked given enough time.
  - Such a model allows us to express **key compromise attacks**.

# Why Timed Spi?

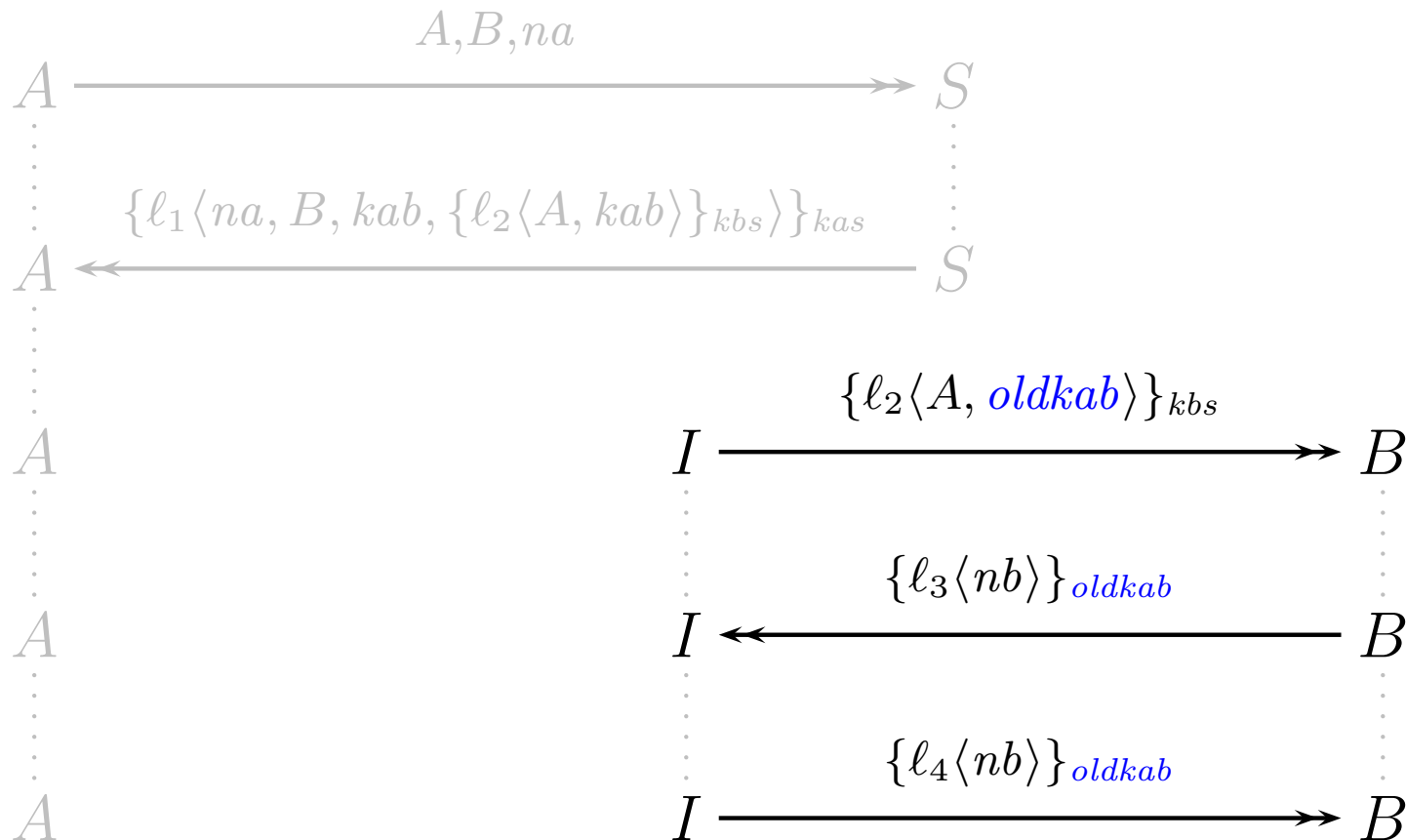
- Untimed spi-calculus models perfect cryptography.
- A more realistic model:
  - Distinguish between long- and short-term keys.
  - Short-term keys can be cracked given enough time.
  - Such a model allows us to express **key compromise attacks**.
- Prime examples: **key distribution protocols (KDPs)**
  - The distributed session keys are short-term.
  - KDPs must make sure that received session keys have been **recently** generated.
  - Previous type systems for untimed spi did not verify recency, our new type system for timed spi does.

# Example: NSSK



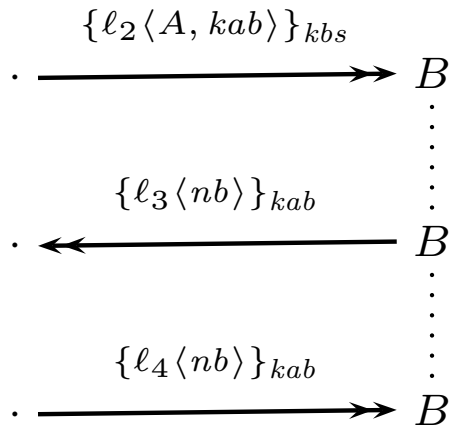
**Goal:** At the end of a complete run both  $A$  and  $B$  want to be sure that  $kab$  is a fresh, secret short-term key shared with the other principal.

# An Attack on NSSK



- $B$  falsely believes that  $oldkab$  is a fresh, secret short-term key shared with  $A$ .
- But really  $oldkab$  is an expired key that has been cracked by  $I$ .

# Expressing NSSK in Typed Spi



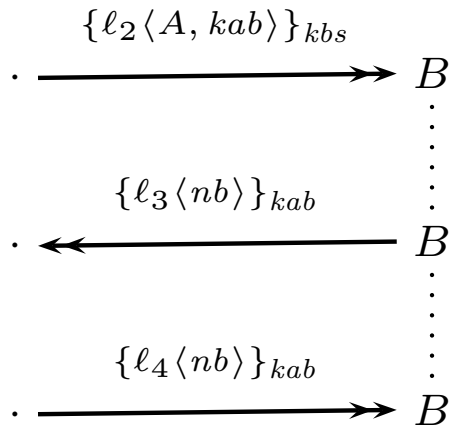
$$P_B(a:\text{Un}, b:\text{Un}, s:\text{Un}, kbs:\text{lt-Key}, net:\text{Un}) \triangleq$$

```

inp net (ctxt : Un);
decrypt ctxt is {x : lt-Auth}_{kbs};
match x is l_2⟨a, kab : ?⟩;
new(nb : Un);
(out net l_3⟨nb⟩ |
inp net (ctxt' : Un);
decrypt ctxt' is {y : ?}_{kbs}; match y is l_4⟨nb⟩;
st-secret(kab))

```

# Expressing NSSK in Typed Spi



$$P_B(a:\text{Un}, b:\text{Un}, s:\text{Un}, kbs:\text{It-Key}, net:\text{Un}) \triangleq$$

```

inp net (ctxt : Un);
decrypt ctxt is {x : It-Auth}_{kbs};
match x is l_2⟨a, kab : ?⟩;
new(nb : Un);
(out net l_3⟨nb⟩ |
inp net (ctxt' : Un);
decrypt ctxt' is {y : ?}_{kbs}; match y is l_4⟨nb⟩;
st-secret(kab))

```

$$\text{System}(a:\text{Un}, b:\text{Un}, s:\text{Un}, net:\text{Un}) \triangleq$$

```

new(kas : It-Key); new(kbs : It-Key);
!P_A(a, b, s, kas, net) | !P_B(a, b, s, kbs, net) | !P_S(a, b, s, kas, kbs, net)

```

# Our Model of Time

- A clock-tick represents the end of an **epoch**, which is the time required for cracking short-term keys.

# Our Model of Time

- A clock-tick represents the end of an **epoch**, which is the time required for cracking short-term keys.
- New primitive:  $\text{crack } M \text{ is } \{x:\text{Un}\}_{y:\text{Un}}; P$ 
  - Cracking uses up all time of the current epoch.
  - All other actions are instantaneous.
  - Cracking allows us to express key-compromising attackers, for instance, the attack on NSSK.

# Our Model of Time

- A clock-tick represents the end of an **epoch**, which is the time required for cracking short-term keys.
- New primitive:  $\text{crack } M \text{ is } \{x:\text{Un}\}_{y:\text{Un}}; P$ 
  - Cracking uses up all time of the current epoch.
  - All other actions are instantaneous.
  - Cracking allows us to express key-compromising attackers, for instance, the attack on NSSK.
- Specification primitive for short-term secrecy:  $\text{st-secret}(M)$ 
  - A short-term secret is secret within the current epoch.

# Our Model of Time

- A clock-tick represents the end of an **epoch**, which is the time required for cracking short-term keys.
- New primitive:  $\text{crack } M \text{ is } \{x:\text{Un}\}_{y:\text{Un}}; P$ 
  - Cracking uses up all time of the current epoch.
  - All other actions are instantaneous.
  - Cracking allows us to express key-compromising attackers, for instance, the attack on NSSK.
- Specification primitive for short-term secrecy:  $\text{st-secret}(M)$ 
  - A short-term secret is secret within the current epoch.
- Specification primitives for short-term authenticity.

# Our Model of Time

- A clock-tick represents the end of an **epoch**, which is the time required for cracking short-term keys.
- New primitive:  $\text{crack } M \text{ is } \{x:\text{Un}\}_{y:\text{Un}}; P$ 
  - Cracking uses up all time of the current epoch.
  - All other actions are instantaneous.
  - Cracking allows us to express key-compromising attackers, for instance, the attack on NSSK.
- Specification primitive for short-term secrecy:  $\text{st-secret}(M)$ 
  - A short-term secret is secret within the current epoch.
- Specification primitives for short-term authenticity.
- **Input expires** with a clock-tick, **modelling timeout**.

# Operational Semantics

- Instantaneous reductions as usual ...

$$\text{out } N \ M \mid \text{inp } N \ (x:T); P \rightarrow P\{x \leftarrow M\}$$

# Operational Semantics

- Instantaneous reductions as usual ...

$$\text{out } N \ M \mid \text{inp } N \ (x:T); P \rightarrow P\{x \leftarrow M\}$$

- ... plus tick-reductions that use up a clock tick.

$$\frac{P \xrightarrow{\sigma} P' \quad Q \xrightarrow{\sigma} Q'}{P \mid Q \xrightarrow{\sigma} Q' \mid Q'}$$

$$Q = \text{st-secret}(K) \quad \text{or} \quad Q = \text{public}(K)$$

---


$$\text{crack } \{M\}_K \text{ is } \{x:T\}_{y:U}; P \mid Q \xrightarrow{\sigma} P\{x, y \leftarrow M, K\} \mid \text{public}(K)$$

---


$$\text{st-secret}(M) \xrightarrow{\sigma} \text{public}(M)$$

---


$$\text{inp } N \ (x:T); P \xrightarrow{\sigma} \mathbf{0}$$

# Operational Semantics

- Instantaneous reductions as usual ...

$$\text{out } N \ M \mid \text{inp } N \ (x:T); P \rightarrow P\{x \leftarrow M\}$$

- ... plus **tick-reductions** that use up a clock tick.

$$\frac{P \xrightarrow{\sigma} P' \quad Q \xrightarrow{\sigma} Q'}{P \mid Q \xrightarrow{\sigma} Q' \mid Q'}$$

$$Q = \text{st-secret}(K) \quad \text{or} \quad Q = \text{public}(K)$$

---


$$\text{crack } \{M\}_K \text{ is } \{x:T\}_{y:U}; P \mid Q \xrightarrow{\sigma} P\{x, y \leftarrow M, K\} \mid \text{public}(K)$$

---


$$\text{st-secret}(M) \xrightarrow{\sigma} \text{public}(M)$$

---


$$\text{inp } N \ (x:T); P \xrightarrow{\sigma} \mathbf{0}$$

- $\Rightarrow \triangleq (\rightarrow \cup \xrightarrow{\sigma})^*$

# Robust Safety

- An **opponent** is an  $U_n$ -typed process, that does not contain secrecy declarations  $\tau$ -secret( $M$ ).

# Robust Safety

- An **opponent** is an  $U_n$ -typed process, that does not contain secrecy declarations  $\tau$ -secret( $M$ ).
- ... but opponents may declare public( $M$ ).

# Robust Safety

- An **opponent** is an  $U_n$ -typed process, that does not contain secrecy declarations  $\tau$ -secret( $M$ ).
- ... but opponents may declare public( $M$ ).
- $P$  is **safe** iff  $P \not\Rightarrow \text{public}(M) \mid \tau\text{-secret}(M) \mid Q$ .

# Robust Safety

- An **opponent** is an  $U_n$ -typed process, that does not contain secrecy declarations  $\tau$ -secret( $M$ ).
  - ... but opponents may declare public( $M$ ).
- $P$  is **safe** iff  $P \not\Rightarrow \text{public}(M) \mid \tau\text{-secret}(M) \mid Q$ .
- $P$  is **robustly safe** if  $P \mid O$  is safe for all opponents  $O$ .

# Robust Safety

- An **opponent** is an  $U_n$ -typed process, that does not contain secrecy declarations  $\tau$ -secret( $M$ ).
  - ... but opponents may declare public( $M$ ).
- $P$  is **safe** iff  $P \not\Rightarrow \text{public}(M) \mid \tau\text{-secret}(M) \mid Q$ .
- $P$  is **robustly safe** if  $P \mid O$  is safe for all opponents  $O$ .
- For instance:
  - NSSK is safe.
  - NSSK it is not robustly safe.

# Robust Safety

- An **opponent** is an Un-typed process, that does not contain secrecy declarations  $\tau\text{-secret}(M)$ .
  - ... but opponents may declare  $\text{public}(M)$ .
- $P$  is **safe** iff  $P \not\Rightarrow \text{public}(M) \mid \tau\text{-secret}(M) \mid Q$ .
- $P$  is **robustly safe** if  $P \mid O$  is safe for all opponents  $O$ .
- For instance:
  - NSSK is safe.
  - NSSK it is not robustly safe.

## Theorem (Robust Safety of the Type System):

If  $\vec{n}:\vec{T} \vdash \text{public}(\vec{n})$  and  $\vec{n}:\vec{T} \vdash P$ , then  $P$  is robustly safe.

# Sample Rules: Encryption/Decryption

- Introduction and elimination rules for honest agents ...

(Encrypt)

$$\frac{E \vdash K : \tau\text{-Key}, M : \tau\text{-Auth}}{E \vdash \{M\}_K : \text{Un}} \quad (\text{where } \tau \in \{\text{lt}, \text{st}\})$$

(Decrypt)

$$\frac{E \vdash M : \text{Un}, K : \tau\text{-Key} \quad E, x : \tau\text{-Auth} \vdash P}{E \vdash \text{decrypt } M \text{ is } \{x : \tau\text{-Auth}\}_K; P}$$

# Sample Rules: Encryption/Decryption

- Introduction and elimination rules for honest agents ...

(Encrypt)

$$\frac{E \vdash K : \tau\text{-Key}, M : \tau\text{-Auth}}{E \vdash \{M\}_K : \text{Un}} \quad (\text{where } \tau \in \{\text{lt}, \text{st}\})$$

(Decrypt)

$$\frac{E \vdash M : \text{Un}, K : \tau\text{-Key} \quad E, x : \tau\text{-Auth} \vdash P}{E \vdash \text{decrypt } M \text{ is } \{x : \tau\text{-Auth}\}_K; P}$$

- ... plus intro- and elim-rules for opponents.

(Encrypt Un)

$$\frac{E \vdash K : \text{Un}, M : \text{Un}}{E \vdash \{M\}_K : \text{Un}}$$

(Decrypt Un)

$$\frac{E \vdash M : \text{Un}, K : \text{Un} \quad E, x:\text{Un} \vdash P}{E \vdash \text{decrypt } M \text{ is } \{x:T\}_K; P}$$

# Tag Types

$$\ell : T \rightarrow \tau\text{-Auth}$$

$$M : \tau\text{-Auth} \triangleq \text{“}M \text{ requires authentication by a } \tau\text{-key”}$$

# Tag Types

$$\ell : T \rightarrow \tau\text{-Auth}$$

$M : \tau\text{-Auth} \triangleq$  “ $M$  requires authentication by a  $\tau$ -key”

• Restrictions on tag types:

• where  $step(T)$  is the type that  $T$  turns into in the next epoch:

•  $step(\text{st-Secret}) \triangleq \text{Un},$

•  $step(\text{lt-Secret}) \triangleq \text{lt-Secret},$

•  $step(\text{Un}) \triangleq \text{Un},$

•  $step(\text{Top}) \triangleq \text{Top},$  and more cases.

# Tag Types

$$\ell : T \rightarrow \tau\text{-Auth}$$

$M : \tau\text{-Auth} \triangleq$  “ $M$  requires authentication by a  $\tau$ -key”

- Restrictions on tag types:

- If  $\tau = \text{st}$ , then  $\text{step}(T) \leq \text{Un}$ .

- where  $\text{step}(T)$  is the type that  $T$  turns into in the next epoch:

- $\text{step}(\text{st-Secret}) \triangleq \text{Un}$ ,

- $\text{step}(\text{lt-Secret}) \triangleq \text{lt-Secret}$ ,

- $\text{step}(\text{Un}) \triangleq \text{Un}$ ,

- $\text{step}(\text{Top}) \triangleq \text{Top}$ , and more cases.

# Tag Types

$$\ell : T \rightarrow \tau\text{-Auth}$$

$M : \tau\text{-Auth} \triangleq$  “ $M$  requires authentication by a  $\tau$ -key”

- Restrictions on tag types:
  - If  $\tau = \text{st}$ , then  $\text{step}(T) \leq \text{Un}$ .
  - If  $\tau = \text{lt}$ , then  $\text{step}(T) = T$ .
- where  $\text{step}(T)$  is the type that  $T$  turns into in the next epoch:
  - $\text{step}(\text{st-Secret}) \triangleq \text{Un}$ ,
  - $\text{step}(\text{lt-Secret}) \triangleq \text{lt-Secret}$ ,
  - $\text{step}(\text{Un}) \triangleq \text{Un}$ ,
  - $\text{step}(\text{Top}) \triangleq \text{Top}$ , and more cases.

# Tag Types for NSSK

$$\begin{array}{ccc} \vdots & & \\ \vdots & \{ \ell_2 \langle A, kab \rangle \}_{kbs} & \\ A & \longrightarrow & B \\ \vdots & & \vdots \end{array}$$

# Tag Types for NSSK

$$\begin{array}{ccc} \vdots & & \\ \vdots & \{ \ell_2 \langle A, kab \rangle \}_{kbs} & \\ A & \longrightarrow & B \\ \vdots & & \vdots \end{array}$$

$$\ell_2 : \langle \text{Un}, \text{st-Secret} \rangle \rightarrow \text{It-Auth}$$

# Tag Types for NSSK

$$\begin{array}{ccc} \vdots & & \\ \vdots & \{ \ell_2 \langle A, kab \rangle \}_{kbs} & \\ A & \longrightarrow & B \\ \vdots & & \vdots \end{array}$$

$$\ell_2 : \langle \text{Un}, \text{st-Secret} \rangle \rightarrow \text{It-Auth}$$

- ... but this is not a legal tag type, because  $\text{step}(\text{st-Secret}) \neq \text{st-Secret}$ .

# Tag Types for NSSK

$$\begin{array}{ccc} \vdots & & \\ \vdots & \{ \ell_2 \langle A, kab \rangle \}_{kbs} & \\ A & \longrightarrow & B \\ \vdots & & \vdots \end{array}$$

$$\ell_2 : \langle \text{Un}, \text{st-Secret} \rangle \rightarrow \text{It-Auth}$$

- ... but this is not a legal tag type, because  $\text{step}(\text{st-Secret}) \neq \text{st-Secret}$ .
- So NSSK does not typecheck ...

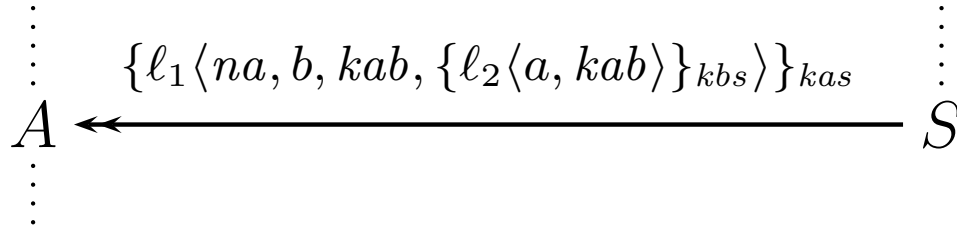
# Tag Types for NSSK

$$\begin{array}{ccc} \vdots & & \\ \vdots & \{ \ell_2 \langle A, kab \rangle \}_{kbs} & \\ A & \longrightarrow & B \\ \vdots & & \vdots \end{array}$$

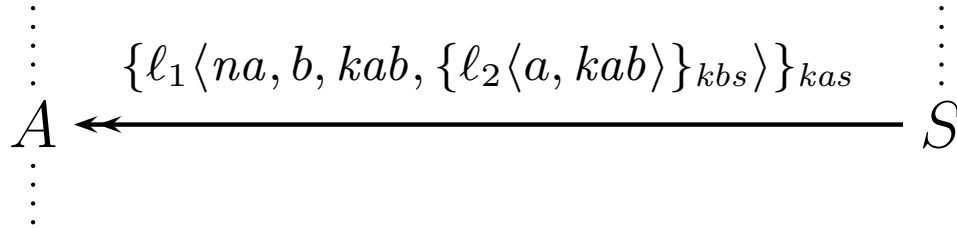
$$\ell_2 : \langle \text{Un}, \text{st-Secret} \rangle \rightarrow \text{It-Auth}$$

- ... but this is not a legal tag type, because  $\text{step}(\text{st-Secret}) \neq \text{st-Secret}$ .
- So NSSK does not typecheck ...
- ... because  $B$  gets no guarantee that  $kab$  is fresh.

# Tag Types for NSSK



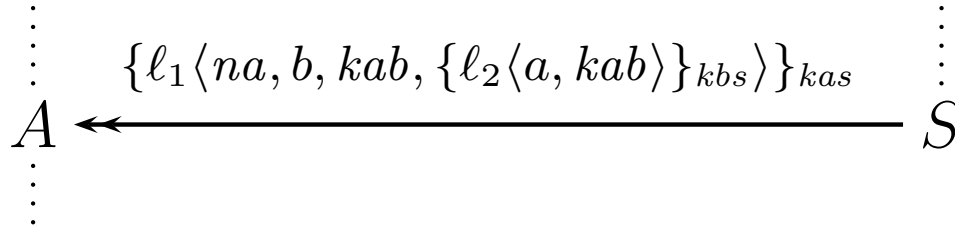
# Tag Types for NSSK



$\ell_1 : \langle na:Un, Un, kab:Top, Un \rangle [A(na, kab)] \rightarrow \text{It-Auth}$

where  $A(na, kab) =$

# Tag Types for NSSK

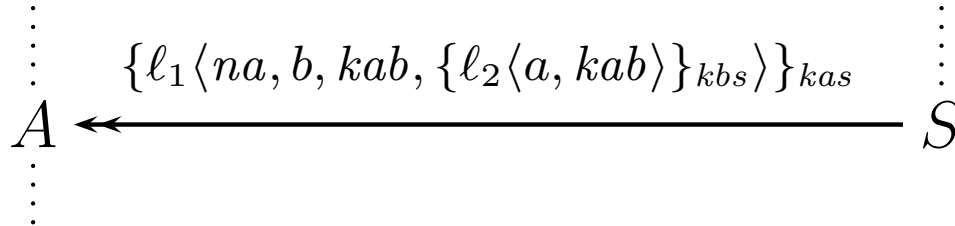


$\ell_1 : \langle na:Un, Un, kab:Top, Un \rangle [A(na, kab)] \rightarrow \text{It-Auth}$

where  $A(na, kab) =$

- The assertion  $A(na, kab)$  is a pre-condition for tagging and a post-condition for untagging.

# Tag Types for NSSK

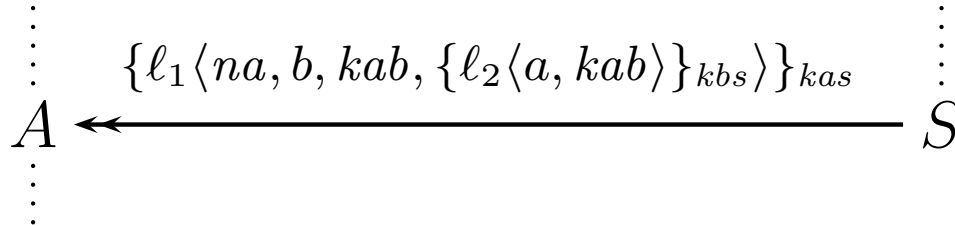


$\ell_1 : \langle na:Un, Un, kab:Top, Un \rangle [A(na, kab)] \rightarrow \text{It-Auth}$

where  $A(na, kab) = na\text{-stamped}(kab : st\text{-Secret})$

- The assertion  $A(na, kab)$  is a pre-condition for tagging and a post-condition for untagging.

# Tag Types for NSSK

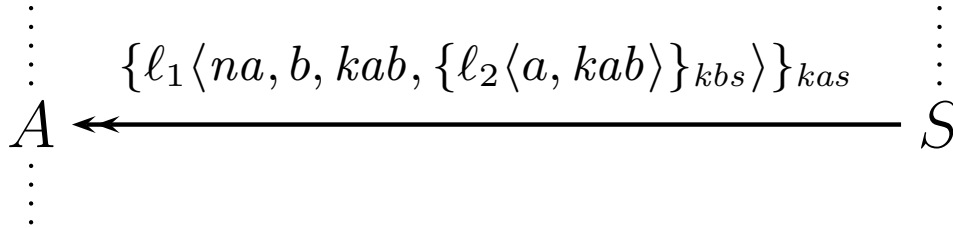


$\ell_1 : \langle na:Un, Un, kab:Top, Un \rangle [A(na, kab)] \rightarrow \text{It-Auth}$

where  $A(na, kab) = na\text{-stamped}(kab : st\text{-Secret})$

- The assertion  $A(na, kab)$  is a pre-condition for tagging and a post-condition for untagging.
- This type is legal:  $step(M\text{-stamped}(A)) \triangleq M\text{-stamped}(A)$ .

# Tag Types for NSSK



$\ell_1 : \langle na:Un, Un, kab:Top, Un \rangle [A(na, kab)] \rightarrow \text{It-Auth}$

where  $A(na, kab) = na\text{-stamped}(kab : st\text{-Secret})$

- The assertion  $A(na, kab)$  is a pre-condition for tagging and a post-condition for untagging.
- This type is legal:  $step(M\text{-stamped}(A)) \triangleq M\text{-stamped}(A)$ .
- Typechecker may unstamp only if stamp is fresh.

# Nonce- and Time-Stamps

$A ::= M:T \mid M\text{-stamped}(A) \mid \text{fresh}(M) \quad (\text{assertions})$

- Short-term can be turned into long-term assertions by stamping them with a nonce:

(Nonce Stamp)

$$\frac{E \vdash M : \text{Top}, A}{E \vdash M\text{-stamped}(A)}$$

- Stamped assertions can be unstamped provided the stamp is fresh:

(Nonce Unstamp)

$$\frac{E \vdash M\text{-stamped}(A), \text{fresh}(M) \quad E, A \vdash P}{E \vdash P}$$

# Summary

- We have extended the spi-calculus by an abstract notion of discrete time.
- This allows us to:
  - distinguish between long- and short-term secrets,
  - model key compromise attacks,
  - express protocols with timestamps.
- We have proposed a type system for verifying short-term secrecy and short-term authenticity.

# Related Work

- Timeliness without explicit model of time:
  - BAN logic [89], Guttman [MFPS 01] (strand space model), Paulson [98] (Isabelle proof assistant, inductive method)
- With explicit models of time:
  - Evans and Schneider [ESORICS 00] (tock-CSP, PVS, rank functions)
  - Lowe (timed CSP, Casper model checker)
  - Gorrieri, Locatelli, Martinelli [ESOP 03] (tCryptoSpa)
  - Bozga, Ene, Lakhnech [CONCUR 04] (real time, symbolic decision procedure, bounded number of sessions required)
  - Delzanno, Ganty [TACAS 04] (real time, symbolic decision procedure, possibly non-terminating)

# Related Work

- Type systems for cryptographic protocol verification:
  - Abadi [JACM 99] (strong secrecy for symmetric crypto)
  - Abadi, Blanchet [FOSSACS 01] (secrecy for asymmetric crypto)
  - Gordon, Jeffrey [JCS 03] (authenticity for symmetric crypto)
  - Gordon, Jeffrey [JCS 03] (authenticity for asymmetric crypto)