

Pattern-Matching Spi-Calculus

A Type System for Cryptographic Protocols

Christian Haack and Alan Jeffrey

DePaul University, Chicago

Types for Cryptographic Protocols

Types for Cryptographic Protocols

- Spi-calculus: A small and abstract domain-specific language for cryptographic protocols:
 - Abadi and Gordon [1997]

Types for Cryptographic Protocols

- Spi-calculus: A small and abstract domain-specific language for cryptographic protocols:
 - Abadi and Gordon [1997]
- Type systems for verifying secrecy or authenticity within the spi-calculus.
 - Abadi [1999]
 - Abadi and Blanchet [2001]
 - Gordon and Jeffrey [2001, 2002]

Types for Cryptographic Protocols

- Spi-calculus: A small and abstract domain-specific language for cryptographic protocols:
 - Abadi and Gordon [1997]
- Type systems for verifying secrecy or authenticity within the spi-calculus.
 - Abadi [1999]
 - Abadi and Blanchet [2001]
 - Gordon and Jeffrey [2001, 2002]
- Advantages of verification by type-checking:
 - Type-checking is easier than proofs from first principles.
 - Type-checking is automatable.

Pattern-Matching Spi: Messages

$$L, M, N ::= n \mid x \mid () \mid (M, N) \mid \{M\}_N \mid \{M\}_{N^{-1}} \\ \mid \text{Enc}(M) \mid \text{Dec}(M)$$

Other constructors by translation to this core language:

Pattern-Matching Spi: Messages

$$L, M, N ::= n \mid x \mid () \mid (M, N) \mid \{M\}_N \mid \{M\}_{N^{-1}} \\ \mid \text{Enc}(M) \mid \text{Dec}(M)$$

Other constructors by translation to this core language:

Symmetric crypto:

$$\{M\}_k \triangleq \{M\}_{\text{Enc}(k)} \text{ where } k \text{ is a secret key pair}$$

Pattern-Matching Spi: Messages

$$L, M, N ::= n \mid x \mid () \mid (M, N) \mid \{M\}_N \mid \{M\}_{N^{-1}} \\ \mid \text{Enc}(M) \mid \text{Dec}(M)$$

Other constructors by translation to this core language:

Symmetric crypto:

$$\{M\}_k \triangleq \{M\}_{\text{Enc}(k)} \text{ where } k \text{ is a secret key pair}$$

Message tagging:

$$l(M) \triangleq \{M\}_{\text{Enc}(l)} \text{ where } l \text{ is a public "key" pair}$$

Pattern-Matching Spi: Messages

$$L, M, N ::= n \mid x \mid () \mid (M, N) \mid \{M\}_N \mid \{M\}_{N^{-1}} \\ \mid \text{Enc}(M) \mid \text{Dec}(M)$$

Other constructors by translation to this core language:

Symmetric crypto:

$$\{M\}_k \triangleq \{M\}_{\text{Enc}(k)} \text{ where } k \text{ is a secret key pair}$$

Message tagging:

$$l(M) \triangleq \{M\}_{\text{Enc}(l)} \text{ where } l \text{ is a public "key" pair}$$

Hashing:

$$\#(M) \triangleq \text{hashtag}(\{M\}_{\text{hashkey}}) \text{ where } \text{hashkey} \text{ is a public encryption key with decryption part unknown to everybody}$$

Pattern-Matching Spi: Processes

$P, Q ::= \text{out } N M \mid \text{inp } N X; P \mid \text{new } n:T; P \mid !P \mid P \mid Q \mid \mathbf{0}$

Pattern-matching input; X is a pattern.

Pattern-Matching Spi: Processes

$P, Q ::= \text{out } N M \mid \text{inp } N X; P \mid \text{new } n:T; P \mid !P \mid P \mid Q \mid \mathbf{0}$

Pattern-matching input; X is a pattern.

$X ::= \{ \vec{x} . M \mid \bar{A} \}$ where \bar{A} is a set of assertions

Pattern-Matching Spi: Processes

$P, Q ::= \text{out } N M \mid \text{inp } N X; P \mid \text{new } n:T; P \mid !P \mid P \mid Q \mid \mathbf{0}$

Pattern-matching input; X is a pattern.

$X ::= \{\vec{x} . M \mid \bar{A}\}$ where \bar{A} is a set of assertions

Surface syntax has syntax sugar. For instance:

$$\text{inp } N \{x : T\}_{k-1}; P \triangleq \text{inp } N \{x . \{x\}_{k-1} \mid x : T\}; P$$

Pattern-Matching Spi: Processes

$P, Q ::= \text{out } N M \mid \text{inp } N X; P \mid \text{new } n:T; P \mid !P \mid P \mid Q \mid \mathbf{0}$

Pattern-matching input; X is a pattern.

$X ::= \{\vec{x} . M \mid \bar{A}\}$ where \bar{A} is a set of assertions

Surface syntax has syntax sugar. For instance:

$$\text{inp } N \{x : T\}_{k-1}; P \triangleq \text{inp } N \{x . \{x\}_{k-1} \mid x : T\}; P$$

Syntactic restrictions:

- Members of binder \vec{x} must have a **witness** in M .

Pattern-Matching Spi: Processes

$P, Q ::= \text{out } N M \mid \text{inp } N X; P \mid \text{new } n:T; P \mid !P \mid P \mid Q \mid \mathbf{0}$

Pattern-matching input; X is a pattern.

$X ::= \{\vec{x} . M \mid \bar{A}\}$ where \bar{A} is a set of assertions

Surface syntax has syntax sugar. For instance:

$\text{inp } N \{x : T\}_{k-1}; P \triangleq \text{inp } N \{x . \{x\}_{k-1} \mid x : T\}; P$

Syntactic restrictions:

- Members of binder \vec{x} must have a **witness** in M .
- Input patterns must be **Dolev-Yao-implementable**. For instance, $\{x, k . \{x\}_{k-1} \mid \bar{A}\}$ is not D-Y-implementable.

Semantics of Pattern-Matching

Dynamic semantics.

$$\text{out } L \ M \{ \vec{x} \leftarrow \vec{N} \} \mid \text{inp } L \{ \vec{x} . M \mid \bar{A} \}; P \quad \rightarrow \quad P \{ \vec{x} \leftarrow \vec{N} \}$$

Semantics of Pattern-Matching

Dynamic semantics.

$$\text{out } L \ M \{ \vec{x} \leftarrow \vec{N} \} \mid \text{inp } L \ { \vec{x} . M \mid \bar{A} }; P \quad \rightarrow \quad P \{ \vec{x} \leftarrow \vec{N} \}$$

- Dynamic check that input message matches input message pattern M .

Semantics of Pattern-Matching

Dynamic semantics.

$$\text{out } L \ M \{ \vec{x} \leftarrow \vec{N} \} \mid \text{inp } L \ \{ \vec{x} . M \mid \bar{A} \}; P \quad \rightarrow \quad P \{ \vec{x} \leftarrow \vec{N} \}$$

- Dynamic check that input message matches input message pattern M .
- Dynamic semantics ignores the assertion set \bar{A} .

Semantics of Pattern-Matching

Dynamic semantics.

$$\text{out } L \ M\{\vec{x} \leftarrow \vec{N}\} \mid \text{inp } L \ \{\vec{x} . M \mid \bar{A}\}; P \quad \rightarrow \quad P\{\vec{x} \leftarrow \vec{N}\}$$

- Dynamic check that input message matches input message pattern M .
- Dynamic semantics ignores the assertion set \bar{A} .

Static semantics.

$$\frac{E \vdash \bar{A}\{\vec{x} \leftarrow \vec{N}\}}{E \vdash M\{\vec{x} \leftarrow \vec{N}\} \in \{\vec{x} . M \mid \bar{A}\}}$$

Semantics of Pattern-Matching

Dynamic semantics.

$$\text{out } L \ M\{\vec{x} \leftarrow \vec{N}\} \mid \text{inp } L \ \{\vec{x} . M \mid \bar{A}\}; P \quad \rightarrow \quad P\{\vec{x} \leftarrow \vec{N}\}$$

- Dynamic check that input message matches input message pattern M .
- Dynamic semantics ignores the assertion set \bar{A} .

Static semantics.

$$\frac{E \vdash \bar{A}\{\vec{x} \leftarrow \vec{N}\}}{E \vdash M\{\vec{x} \leftarrow \vec{N}\} \in \{\vec{x} . M \mid \bar{A}\}}$$

- Static check that assertion set \bar{A} holds after input.

Semantics of Pattern-Matching

Dynamic semantics.

$$\text{out } L \ M \{ \vec{x} \leftarrow \vec{N} \} \mid \text{inp } L \ \{ \vec{x} . M \mid \bar{A} \}; P \quad \rightarrow \quad P \{ \vec{x} \leftarrow \vec{N} \}$$

- Dynamic check that input message matches input message pattern M .
- Dynamic semantics ignores the assertion set \bar{A} .

Static semantics.

$$\frac{E \vdash \bar{A} \{ \vec{x} \leftarrow \vec{N} \}}{E \vdash M \{ \vec{x} \leftarrow \vec{N} \} \in \{ \vec{x} . M \mid \bar{A} \}}$$

- Static check that assertion set \bar{A} holds after input.
- \bar{A} may be viewed as checked input post-condition.

Correspondence Assertions

$$A \rightarrow B \quad (m, A, B)$$

$$P_A \stackrel{\Delta}{=} \text{new } m : T; \quad \text{out } net \ (m, A, B)$$

$$P_B \stackrel{\Delta}{=} \text{inp } net \ \{x, p . (x, p, B) \mid \bar{A}(x, p)\};$$

Correspondence Assertions

A !begins “*A* sends *m* to *B*”

$A \rightarrow B \quad (m, A, B)$

B ends “*A* sends *m* to *B*”

$P_A \triangleq \text{new } m : T; \text{begin!}(m, A, B); \text{out } net (m, A, B)$

$P_B \triangleq \text{inp } net \{x, p . (x, p, B) \mid \bar{A}(x, p)\}; \text{end}(x, p, B)$

Correspondence Assertions

A !begins “*A* sends *m* to *B*”

$A \rightarrow B \quad (m, A, B)$

B ends “*A* sends *m* to *B*”

$P_A \triangleq \text{new } m : T; \text{begin!}(m, A, B); \text{out } net (m, A, B)$

$P_B \triangleq \text{inp } net \{x, p . (x, p, B) \mid \bar{A}(x, p)\}; \text{end}(x, p, B)$

- A process is **safe** iff in every run every end-assertion is preceded by a matching begin-assertion.

Correspondence Assertions

A !begins “*A* sends *m* to *B*”

$A \rightarrow B \quad (m, A, B)$

B ends “*A* sends *m* to *B*”

$P_A \triangleq \text{new } m : T; \text{ begin!}(m, A, B); \text{ out } net (m, A, B)$

$P_B \triangleq \text{inp } net \{x, p . (x, p, B) \mid \bar{A}(x, p)\}; \text{ end}(x, p, B)$

- A process is **safe** iff in every run every end-assertion is preceded by a matching begin-assertion.

$P_A \mid P_B$ is safe.

Correspondence Assertions

A !begins “*A* sends *m* to *B*”

$A \rightarrow B \quad (m, A, B)$

B ends “*A* sends *m* to *B*”

$P_A \triangleq \text{new } m : T; \text{begin!}(m, A, B); \text{out } net (m, A, B)$

$P_B \triangleq \text{inp } net \{x, p . (x, p, B) \mid \bar{A}(x, p)\}; \text{end}(x, p, B)$

- A process is **safe** iff in every run every end-assertion is preceded by a matching begin-assertion.

$P_A \mid P_B$ is safe.

- A process P is **robustly safe** iff $P \mid O$ is safe for all opponents O .

Correspondence Assertions

A !begins “*A* sends *m* to *B*”

$A \rightarrow B \quad (m, A, B)$

B ends “*A* sends *m* to *B*”

$P_A \triangleq \text{new } m : T; \text{begin!}(m, A, B); \text{out } net \ (m, A, B)$

$P_B \triangleq \text{inp } net \ \{x, p . (x, p, B) \mid \bar{A}(x, p)\}; \text{end}(x, p, B)$

- A process is **safe** iff in every run every end-assertion is preceded by a matching begin-assertion.

$P_A \mid P_B$ is safe.

- A process *P* is **robustly safe** iff $P \mid O$ is safe for all opponents *O*.

$P_A \mid P_B$ is not robustly safe.

Correspondence Assertions

A !begins “*A* sends *m* to *B*”

$A \rightarrow B \quad (m, A, B)$

B ends “*A* sends *m* to *B*”

$P_A \triangleq \text{new } m : T; \text{begin!}(m, A, B); \text{out } net (m, A, B)$

$P_B \triangleq \text{inp } net \{x, p . (x, p, B) \mid \bar{A}(x, p)\}; \text{end}(x, p, B)$

- A process is **safe** iff in every run every end-assertion is preceded by a matching begin-assertion.

$P_A \mid P_B$ is safe.

- A process P is **robustly safe** iff $P \mid O$ is safe for all opponents O .

$P_A \mid P_B$ is not robustly safe.

- Theorem: Every well-typed process is robustly safe.

A Well-Typed Protocol

A !begins “*A* sends *m* to *B*”

$A \rightarrow B \quad \{m, A, B\}_{esA}$

B ends “*A* sends *m* to *B*”

$esA \quad : \quad ???$

$dsA \quad : \quad ???$

$P_A \triangleq$ new $m : ???;$
begin!(m, A, B);
out net $\{m, A, B\}_{esA}$

$P_B \triangleq$ inp net $\{x, p . \{x, p, B\}_{dsA^{-1}} \mid ???\};$
end(x, p, B)

A Well-Typed Protocol

A !begins “*A* sends *m* to *B*”

$A \rightarrow B \quad \{m, A, B\}_{esA}$

B ends “*A* sends *m* to *B*”

$esA \quad :$ SignEncKey(X)

$dsA \quad :$ SignDecKey(X)

$X \quad \triangleq \{x, p, q . (x, p, q) \mid !begun(x, p, q)\}$

$P_A \quad \triangleq \text{new } m : ???;$

$\text{begin!}(m, A, B);$

$\text{out net } \{m, A, B\}_{esA}$

$P_B \quad \triangleq \text{inp net } \{x, p . \{x, p, B\}_{dsA^{-1}} \mid ???\};$

$\text{end}(x, p, B)$

A Well-Typed Protocol

A !begins “*A* sends *m* to *B*”

$A \rightarrow B \quad \{m, A, B\}_{esA}$

B ends “*A* sends *m* to *B*”

$esA \quad :$ SignEncKey(*X*)

$dsA \quad :$ SignDecKey(*X*)

$X \quad \triangleq \{x, p, q . (x, p, q) \mid !begun(x, p, q)\}$

$P_A \quad \triangleq$ new *m* : Public;

begin!(*m*, *A*, *B*);

out net $\{m, A, B\}_{esA}$

$P_B \quad \triangleq$ inp net $\{x, p . \{x, p, B\}_{dsA^{-1}} \mid ???\};$

end(*x*, *p*, *B*)

A Well-Typed Protocol

A !begins “*A* sends *m* to *B*”

$A \rightarrow B \quad \{m, A, B\}_{esA}$

B ends “*A* sends *m* to *B*”

$esA \quad :$ SignEncKey(*X*)

$dsA \quad :$ SignDecKey(*X*)

$X \quad \triangleq \{x, p, q . (x, p, q) \mid !begun(x, p, q)\}$

$P_A \quad \triangleq$ new *m* : Public;

begin!(*m*, *A*, *B*);

out net $\{m, A, B\}_{esA}$

$P_B \quad \triangleq$ inp net $\{x, p . \{x, p, B\}_{dsA^{-1}} \mid !begun(x, p, B)\};$

end(*x*, *p*, *B*)

Protocol-Independent Key Types

out *net* $\{m, A, B\}_{esA}$

type-checks with

$esA : \text{SignEncKey}(\{x, p, q . (x, p, q) \mid !\text{begun}(x, p, q)\})$.

Problems.

Protocol-Independent Key Types

out $net \{m, A, B\}_{esA}$

type-checks with

$esA : \text{SignEncKey}(\{x, p, q . (x, p, q) \mid !\text{begun}(x, p, q)\})$.

Problems.

- The type of esA is specific to this particular protocol.

Protocol-Independent Key Types

out $net \{m, A, B\}_{esA}$

type-checks with

$esA : \text{SignEncKey}(\{x, p, q . (x, p, q) \mid !\text{begun}(x, p, q)\})$.

Problems.

- The type of esA is specific to this particular protocol.
- The inclusion of principal name A is redundant, because A 's signature already authenticates A .

Protocol-Independent Key Types

out $net \{m, A, B\}_{esA}$

type-checks with

$esA : \text{SignEncKey}(\{x, p, q . (x, p, q) \mid !\text{begun}(x, p, q)\})$.

Problems.

- The type of esA is specific to this particular protocol.
- The inclusion of principal name A is redundant, because A 's signature already authenticates A .

A Solution.

- Typed message tagging and “authorization” types.

Tag and Authorization Types

$\text{out } \textit{net} \{ \ell(m, B) \}_{esA}$

Tag and Authorization Types

out *net* $\{\ell(m, B)\}_{esA}$

type-checks with

$esA : \text{SignEncKey}(\emptyset \text{Auth}(A))$

Tag and Authorization Types

out *net* $\{\ell(m, B)\}_{esA}$

type-checks with

$esA : \text{SignEncKey}(\emptyset \text{Auth}(A))$

- From a sender's point of view, $\emptyset \text{Auth}(A)$ is a type of messages that require authorization by A .

Tag and Authorization Types

out $net \{ \ell(m, B) \}_{esA}$

type-checks with

$esA : \text{SignEncKey}(\emptyset \text{Auth}(A))$

- From a sender's point of view, $\emptyset \text{Auth}(A)$ is a type of messages that require authorization by A .
- From a receiver's point of view, $\emptyset \text{Auth}(A)$ is a type of messages that have been authorized by A .

Tag and Authorization Types

out $net \ \{\ell(m, B)\}_{esA}$

type-checks with

$esA : \text{SignEncKey}(\emptyset \text{Auth}(A))$

- From a sender's point of view, $\emptyset \text{Auth}(A)$ is a type of messages that require authorization by A .
- From a receiver's point of view, $\emptyset \text{Auth}(A)$ is a type of messages that have been authorized by A .
- Tag type:

$$\begin{aligned} \ell & : \quad \forall p . X(p) \rightarrow \text{Auth}(p) \\ X(p) & \triangleq \{x, q . (x, q) \mid !\text{begun}(x, p, q)\} \end{aligned}$$

Tag and Authorization Types

out $net \ \{\ell(m, B)\}_{esA}$

type-checks with

$esA : \text{SignEncKey}(\emptyset \text{Auth}(A))$

- From a sender's point of view, $\emptyset \text{Auth}(A)$ is a type of messages that require authorization by A .
- From a receiver's point of view, $\emptyset \text{Auth}(A)$ is a type of messages that have been authorized by A .
- Tag type:

$$\ell \quad : \quad \forall p . X(p) \rightarrow \text{Auth}(p)$$
$$X(p) \quad \triangleq \quad \{x, q . (x, q) \mid !\text{begun}(x, p, q)\}$$

- Compare to $X \triangleq \{x, p, q . (x, p, q) \mid !\text{begun}(x, p, q)\}$.

Pattern-Matching Spi: Types

Kinds: $K, H \subseteq \{\text{Public}, \text{Tainted}\}$

Types: $T, U ::= (K, H) KT(X) \mid K \text{ Top} \mid K \text{ Auth}(M)$

$KT ::= \text{EncKey} \mid \text{DecKey} \mid \text{KeyPair}$

Pattern-Matching Spi: Types

Kinds: $K, H \subseteq \{\text{Public}, \text{Tainted}\}$

Types: $T, U ::= (K, H) KT(X) \mid K \text{ Top} \mid K \text{ Auth}(M)$

$KT ::= \text{EncKey} \mid \text{DecKey} \mid \text{KeyPair}$

- In key types $(K, H) KT(X)$, K is the kind of the encryption key and H the kind of the decryption key.

Pattern-Matching Spi: Types

Kinds: $K, H \subseteq \{\text{Public}, \text{Tainted}\}$

Types: $T, U ::= (K, H) KT(X) \mid K \text{ Top} \mid K \text{ Auth}(M)$

$KT ::= \text{EncKey} \mid \text{DecKey} \mid \text{KeyPair}$

- In key types $(K, H) KT(X)$, K is the kind of the encryption key and H the kind of the decryption key.
- $K \text{ Top}$ is the greatest type for messages of kind K .

Pattern-Matching Spi: Types

Kinds: $K, H \subseteq \{\text{Public}, \text{Tainted}\}$

Types: $T, U ::= (K, H) KT(X) \mid K \text{ Top} \mid K \text{ Auth}(M)$

$KT ::= \text{EncKey} \mid \text{DecKey} \mid \text{KeyPair}$

- In key types $(K, H) KT(X)$, K is the kind of the encryption key and H the kind of the decryption key.
- $K \text{ Top}$ is the greatest type for messages of kind K .
- $K \text{ Auth}(M)$ is a type of messages authorized by M and of kind K .

Pattern-Matching Spi: Types

Kinds: $K, H \subseteq \{\text{Public}, \text{Tainted}\}$

Types: $T, U ::= (K, H) KT(X) \mid K \text{ Top} \mid K \text{ Auth}(M)$

$KT ::= \text{EncKey} \mid \text{DecKey} \mid \text{KeyPair}$

- In key types $(K, H) KT(X)$, K is the kind of the encryption key and H the kind of the decryption key.
- $K \text{ Top}$ is the greatest type for messages of kind K .
- $K \text{ Auth}(M)$ is a type of messages authorized by M and of kind K .
- The types from the previous examples translate to this core language of types.

Summary and Contributions

Summary and Contributions

- Pattern-matching input instead of message destructors and equality checks.

Summary and Contributions

- Pattern-matching input instead of message destructors and equality checks.
- Static pattern matching instead of dependent types results in more flexible scoping rules.

Summary and Contributions

- Pattern-matching input instead of message destructors and equality checks.
- Static pattern matching instead of dependent types results in more flexible scoping rules.
 - Can now type-check hashing and nested encryption.

Summary and Contributions

- Pattern-matching input instead of message destructors and equality checks.
- Static pattern matching instead of dependent types results in more flexible scoping rules.
 - Can now type-check hashing and nested encryption.
- Authorization types and tag types instead of tagged union types.

Summary and Contributions

- Pattern-matching input instead of message destructors and equality checks.
- Static pattern matching instead of dependent types results in more flexible scoping rules.
 - Can now type-check hashing and nested encryption.
- Authorization types and tag types instead of tagged union types.
 - Protocol-independent key types.

Summary and Contributions

- Pattern-matching input instead of message destructors and equality checks.
- Static pattern matching instead of dependent types results in more flexible scoping rules.
 - Can now type-check hashing and nested encryption.
- Authorization types and tag types instead of tagged union types.
 - Protocol-independent key types.
 - Authentication by signature.

Summary and Contributions

- Pattern-matching input instead of message destructors and equality checks.
- Static pattern matching instead of dependent types results in more flexible scoping rules.
 - Can now type-check hashing and nested encryption.
- Authorization types and tag types instead of tagged union types.
 - Protocol-independent key types.
 - Authentication by signature.
- Small core language.

Summary and Contributions

- Pattern-matching input instead of message destructors and equality checks.
- Static pattern matching instead of dependent types results in more flexible scoping rules.
 - Can now type-check hashing and nested encryption.
- Authorization types and tag types instead of tagged union types.
 - Protocol-independent key types.
 - Authentication by signature.
- Small core language.
 - The rule system, its correctness proofs and its implementation remain tractable.