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]

Advantages of verification by type-checking:
- Type-checking is easier than proofs from first principles.
- Type-checking is automatable.
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 Enc(M) \mid Dec(M) \]

Other constructors by translation to this core language:
Pattern-Matching Spi: Messages

\[ L, M, N ::= \ n \ | \ x \ | \ () \ | \ (M, N) \ | \ \{M\}_N \ | \ \{M\}_{N-1} \]

\[
\ | \ \text{Enc}(M) \ | \ \text{Dec}(M)
\]

Other constructors by translation to this core language:

Symmetric crypto:

\[ \{M\}_k \triangleq \ \{M\}_{\text{Enc}(k)} \] where \( k \) 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}} \]

\[
\mid \operatorname{Enc}(M) \mid \operatorname{Dec}(M)
\]

Other constructors by translation to this core language:

Symmetric crypto:

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

Message tagging:

\[
l(M) \triangleq \{M\}_{\operatorname{Enc}(l)} \text{ where } l \text{ is a public “key” pair}
\]
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

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

Pattern-matching input; \( X \) is a pattern.
Pattern-Matching Spi: Processes

\[ P, Q ::= \text{out } N M | \text{inp } N X; P | \text{new } n:T; P | !P | P | Q | 0 \]

Pattern-matching input; \( X \) is a pattern.

\[ X ::= \{ \vec{x}. M | \vec{A} \} \quad \text{where } \vec{A} \text{ is a set of assertions} \]
Pattern-Matching Spi: Processes

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

Pattern-matching input; \( X \) is a pattern.

\[ X ::= \{ \bar{x} . M \mid \bar{A} \} \quad \text{where} \ \bar{A} \ \text{is a set of assertions} \]

Surface syntax has syntax sugar. For instance:

\[ \text{inp } N \{\{x : T\}_{k-1}; P \quad \overset{\Delta}{=} \quad \text{inp } N \{x . \{x\}_{k-1} \mid x : T\}; P \]
Pattern-Matching Spi: Processes

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

Pattern-matching input; \( X \) is a pattern.

\[ X ::= \{ \bar{x} . M \mid \bar{A} \} \quad \text{where } \bar{A} \text{ is a set of assertions} \]

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 \( \bar{x} \) must have a **witness** in \( M \).
Pattern-Matching Spi: Processes

\[ P, Q ::= \text{out } N M | \text{inp } N X; P | \text{new } n:T; P | !P | P | Q | 0 \]

Pattern-matching input; \( X \) is a pattern.

\[ X ::= \{ \bar{x} . M | \bar{A} \} \quad \text{where } \bar{A} \text{ is a set of assertions} \]

Surface syntax has syntax sugar. For instance:

\[ \text{inp } N \{ x : T \}_{k-1}; P \equiv \text{inp } N \{ x . \{ x \}_{k-1} | x : T \}; P \]

Syntactic restrictions:

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

- Input patterns must be **Dolev-Yao-implementable**. For instance, \( \{ x, k . \{ x \}_{k-1} | \bar{A} \} \) is not D-Y-implementable.
Dynamic semantics.

\[
\text{out } L \ M\{\overline{x} \leftarrow \overline{N}\} \ | \ \text{inp } L \ \{\overline{x} \cdot M \mid A\}; \ P \ \rightarrow \ P\{\overline{x} \leftarrow \overline{N}\}
\]
Semantics of Pattern-Matching

Dynamic semantics.

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

- Dynamic check that input message matches input message pattern \( M \).
Semantics of Pattern-Matching

Dynamic semantics.

\[
\text{out } L M\{\bar{x} \leftarrow \bar{N}\} \mid \text{inp } L \{\bar{x}.M \mid \bar{A}\}; P \rightarrow P\{\bar{x} \leftarrow \bar{N}\}
\]

- Dynamic check that input message matches input message pattern \(M\).
- Dynamic semantics ignores the assertion set \(\bar{A}\).
Semantics of Pattern-Matching

Dynamic semantics.

\[
\text{out } L \ M\{\vec{x} \leftarrow \vec{N}\} \ | \ \text{inp } L \ \{\vec{x}.M \ | \ \vec{A}\}; \ P \ \rightarrow \ P\{\vec{x} \leftarrow \vec{N}\}
\]

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

Static semantics.

\[
E \vdash \vec{A}\{\vec{x} \leftarrow \vec{N}\} \quad E \vdash M\{\vec{x} \leftarrow \vec{N}\} \in \{\vec{x}.M \ | \ \vec{A}\}
\]
Semantics of Pattern-Matching

Dynamic semantics.

\[
\text{out } L \ M\{\bar{x}\leftarrow\bar{N}\} \mid \text{inp } L \ \{\bar{x}. \ M \mid \bar{A}\}; \ P \rightarrow \ P\{\bar{x}\leftarrow\bar{N}\}
\]

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

Static semantics.

\[
E \vdash \bar{A}\{\bar{x}\leftarrow\bar{N}\}
\]

\[
E \vdash M\{\bar{x}\leftarrow\bar{N}\} \in \{\bar{x}. \ M \mid \bar{A}\}
\]

- Static check that assertion set \( \bar{A} \) holds after input.
Semantics of Pattern-Matching

Dynamic semantics.

\[
\text{out } L M\{\overline{x} \leftarrow \overrightarrow{N}\} | \text{ inp } L \{\overline{x} . M | \overline{A}\}; P \rightarrow P\{\overline{x} \leftarrow \overrightarrow{N}\}
\]

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

Static semantics.

\[
E \vdash \overline{A}\{\overline{x} \leftarrow \overrightarrow{N}\}
\]

\[
E \vdash M\{\overline{x} \leftarrow \overrightarrow{N}\} \in \{\overline{x} . M | \overline{A}\}
\]

- Static check that assertion set \( \overline{A} \) holds after input.
- \( \overline{A} \) may be viewed as checked input post-condition.
Correspondence Assertions

\[ A \rightarrow B \quad (m, A, B) \]

\[ P_A \triangleq \text{new } m : T; \quad \text{out } net \ (m, A, B) \]

\[ P_B \triangleq \text{inp } net \ \{ x, p . (x, p, B) \mid \bar{A}(x, p) \}; \]
Correspondence Assertions

\(A \text{ !begins } "A \text{ sends } m \text{ to } B"\)
\(A \rightarrow B \quad (m, A, B)\)
\(B \text{ ends } "A \text{ sends } m \text{ to } B"\)

\(P_A \triangleq \text{ new } m : T; \begin{align*} \text{begin}(m, A, B); \text{ out net } (m, A, B) \end{align*}\)
\(P_B \triangleq \text{ inp net } \{x, p . (x, p, B) | \bar{A}(x, p)\}; \text{ end}(x, p, B)\)
Correspondence Assertions

\[ A \text{ !begins “} A \text{ sends } m \text{ to } B \text{”} \]

\[ A \rightarrow B \quad (m, A, B) \]

\[ B \text{ ends “} A \text{ sends } m \text{ to } B \text{”} \]

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

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

A process is safe iff in every run every end-assertion is preceeded by a matching begin-assertion.
Correspondence Assertions

A !begins “A sends m to B”
A → B (m, A, B)
B ends “A sends m to B”

\[
P_A \triangleq \text{new } m : T; \ \text{begin!}(m, A, B); \ \text{out } \text{net } (m, A, B)
\]
\[
P_B \triangleq \text{inp net } \{x, p . (x, p, B) \mid \tilde{A}(x, p)\}; \ \text{end}(x, p, B)
\]

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

\[P_A \mid P_B\text{ is safe.}\]
Correspondence Assertions

$A$ !begins “$A$ sends $m$ to $B$”

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

$B$ ends “$A$ sends $m$ to $B$”

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

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

A process is safe iff in every run every end-assertion is preceded by a matching begin-assertion. $P_A \parallel P_B$ is safe.

A process $P$ is robustly safe iff $P \parallel O$ is safe for all opponents $O$. 
Correspondence Assertions

\( A \xrightarrow{!} \text{begins “A sends } m \text{ to } B\)”

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

\( B \text{ ends “A sends } m \text{ to } B\)”

\[
\begin{align*}
  P_A & \triangleq \text{new } m : T; \ \text{begin}!(m, A, B); \ \text{out net} \ (m, A, B) \\
  P_B & \triangleq \text{inp net} \ \{x, p \ . \ (x, p, B) \mid \bar{A}(x, p)\}; \ \text{end}(x, p, B)
\end{align*}
\]

- 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 → B (m, A, B)
B ends “A sends m to B”

\[ P_A \overset{\triangle}{=} \text{new } m : T; \begin{array}{l} \begin{array}{l} \text{begin}!(m, A, B); \text{out} \text{ net } (m, A, B) \end{array} \end{array} \]
\[ P_B \overset{\triangle}{=} \text{inp net } \{x, p \cdot (x, p, B) | \tilde{A}(x, p)\}; \text{end}(x, p, B) \]

A process is safe iff in every run every end-assertion is preceded by a matching begin-assertion.
\[ P_A \mid P_B \text{ is safe.} \]

A process \( P \) is robustly safe iff \( P \mid O \) is safe for all opponents \( O \).
\[ P_A \mid P_B \text{ is not robustly safe.} \]

Theorem: Every well-typed process is robustly safe.
A Well-Typed Protocol

\[ A \text{ !begins "} A \text{ sends } m \text{ to } B\text{"} \]
\[ A \rightarrow B \quad \{m, A, B\}_{esA} \]

\[ B \text{ ends "} A \text{ sends } m \text{ to } B\text{"} \]

\[ esA : ??? \]
\[ dsA : ??? \]

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

\[
P_B \triangleq \text{inp net } \{x, p . \{x, p, B\}_{dsA^{-1}} \mid ???\}; \]
\[ \text{end}(x, p, B) \]
A Well-Typed Protocol

\[ A \text{ !begins} \text{“} A \text{ sends } m \text{ to } B \text{”} \]
\[ A \to B \quad \{m, A, B\}_{esA} \]
\[ B \text{ ends} \text{“} A \text{ sends } m \text{ to } B \text{”} \]

\[ esA : \ \text{SignEncKey}(X) \]
\[ dsA : \ \text{SignDecKey}(X) \]
\[ X \triangleq \{x, p, q . \ (x, p, q) \mid \text{!begun}(x, p, q)\} \]

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

\[ P_B \triangleq \text{inp net} \ \{x, p . \ \{x, p, B\}_{dsA^{-1}} \mid \text{??}\}; \]
\[ \text{end}(x, p, B) \]
A Well-Typed Protocol

\[ A \text{ !begins "A sends } m \text{ to } B" \]

\[ A \rightarrow B \quad \{m, A, B\} \_{esA} \]

\[ B \text{ ends "A sends } m \text{ to } B" \]

\[ esA : \text{SignEncKey}(X) \]

\[ dsA : \text{SignDecKey}(X) \]

\[ X \triangleq \{x, p, q . \ (x, p, q) \mid !\text{begun}(x, p, q)\} \]

\[ P_A \triangleq \text{new } m : \text{Public}; \]

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

\[ \text{out } \text{net} \quad \{m, A, B\} \_{esA} \]

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

\[ \text{end}(x, p, B) \]
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”

\[ esA : \text{SignEncKey}(X) \]
\[ dsA : \text{SignDecKey}(X) \]
\[ X \trianglerighteq \{ x, p, q . (x, p, q) \mid \text{!begun}(x, p, q) \} \]

\[ P_A \trianglerighteq \text{new m : Public; begin!(m, A, B); out net } \{ \{ m, A, B \} \}_{esA} \]

\[ P_B \trianglerighteq \text{inp net } \{ x, p . \{ x, p, B \} \}_{dsA^{-1}} \mid \text{!begun}(x, p, B) \}; \text{end}(x, p, B) \]
Protocol-Independent Key Types

\[
\text{out net } \{m, A, B\}_{esA} \text{ type-checks with } \\
esA : \text{SignEncKey}(\{x, p, q . (x, p, q) \mid \!\text{begun}(x, p, q)\}).
\]

Problems.
Protocol-Independent Key Types

\[
\text{out net } \{m, A, B \}_{esA} \quad \text{type-checks with} \\
\quad esA : \text{SignEncKey}(\{x, p, q . (x, p, q) \mid \text{!begun}(x, p, q)\}).
\]

Problems.

- The type of \( esA \) is specific to this particular protocol.
Protocol-Independent Key Types

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

type-checks with

\[ esA : \text{SignEncKey}(\{x, p, q . (x, p, q) | \![\text{begun}(x, p, q)]\}) \].

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

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

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

\[ \text{out net } \{ \ell(m, B) \} \}_{esA} \]
Tag and Authorization Types

\[
\text{out } net \ {\ell(m, B)} \}_{esA} \\
\text{type-checks with} \\
esA : \text{SignEncKey}(\emptyset \text{Auth}(A))
\]
Tag and Authorization Types

\[
\text{out } net \{\ell(m, B)\}_{esA} \text{ type-checks with } esA : \text{SignEncKey}(\emptyset \text{Auth}(A))
\]

- 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

\[ \text{out net } \{ \ell(m, B) \} \] \_\text{esA} \]

\text{type-checks with}

\text{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:

```
\ell : \forall p . X(p) \rightarrow \text{Auth}(p)
```

```
X(p) \triangleq \{ x, q . (x, q) \mid !\text{begun}(x, p, q) \}
```
Tag and Authorization Types

\[
\text{out net } \{\ell(m, B)\}_{esA} \]

\text{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:

\[
\ell : \forall p . X(p) \rightarrow \text{Auth}(p)
\]

\[X(p) \triangleq \{x, q . (x, q) \mid !\text{begun}(x, p, q)\}\]

- Compare to \(X \triangleq \{x, p, q . (x, p, q) \mid !\text{begun}(x, p, q)\}\).
Pattern-Matching Spi: Types

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

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: \( K, H \subseteq \{\text{Public, Tainted}\} \)

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: \( K, H \subseteq \{ \text{Public, Tainted} \} \)

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: \( K, H \subseteq \{ \text{Public, Tainted} \} \)

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: \( K, H \subseteq \{ \text{Public, Tainted} \} \)

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

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.
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.
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.