# Secure sessions

Karthikeyan Bhargavan Ricardo Corin Pierre-Malo Deniélou

Cédric Fournet James J. Leifer

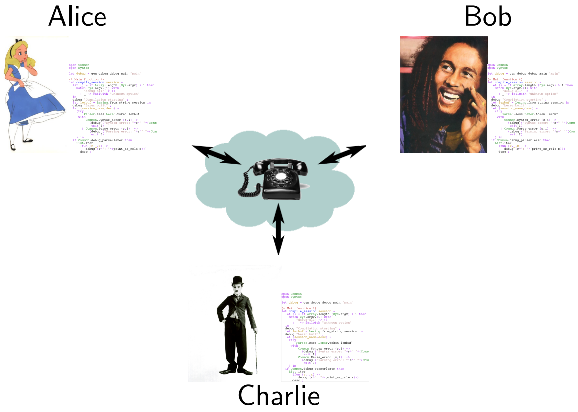
29 April 2010

Joint Institutes Workshop, Orsay

MICROSOFT RESEARCH INRIA JOINT CENTRE

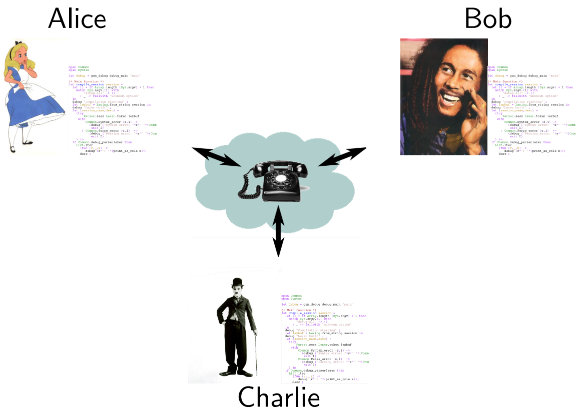
# Goal

Make it simple to write *distributed programs* that engage in *orchestrated patterns* of *secure* communication between *multiple* peers.



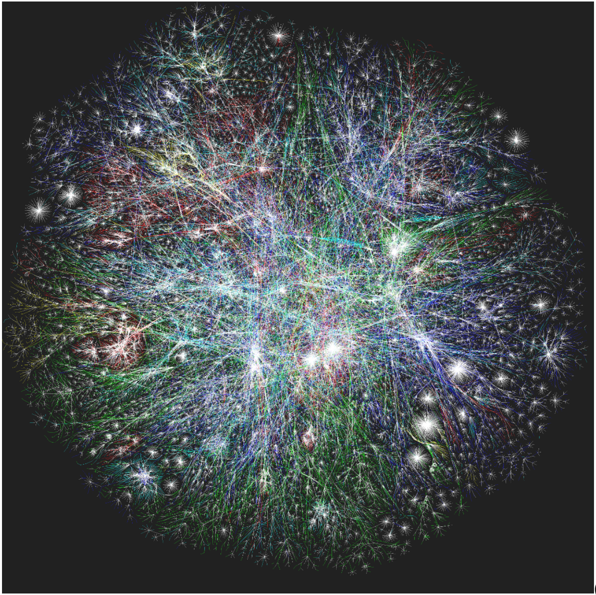
# Goal

Make it simple to write *distributed programs* that engage in *orchestrated patterns* of *secure* communication between *multiple* peers.




Piece of cake! (Assuming we control the network and all the peers.)

# But the network is not under our control...



(The internet circa 2005)

# ...and our peers may not be trustworthy


"On the Internet, nobody knows you're a dog."

# Secure distributed programming

**Only realistic security assumption:**

The network and any coalition of peers are potentially malicious.

# Secure distributed programming

**Only realistic security assumption:**

The network and any coalition of peers are potentially malicious.

**Designing a (correct) security protocol by hand is hard:**

- involves low-level, error-prone coding below communication abstractions,
- depends on global message choreography,
- needs to protect against coalitions of compromised peers.

# Secure distributed programming

**Only realistic security assumption:**

The network and any coalition of peers are potentially malicious.

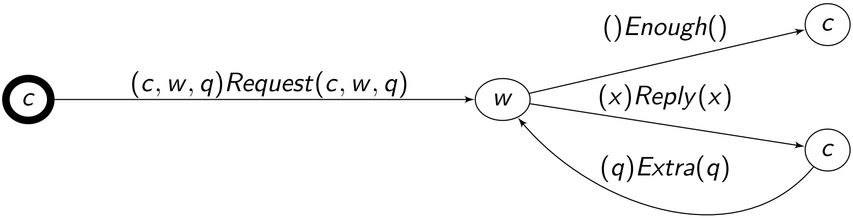
**Designing a (correct) security protocol by hand is hard:**

- involves low-level, error-prone coding below communication abstractions,
- depends on global message choreography,
- needs to protect against coalitions of compromised peers.

**Therefore, we propose:**

- to automatically generate tailored cryptographic protocols protecting against the network and compromised peers;
- to hide implementation details and provide mechanised proofs of correctness.

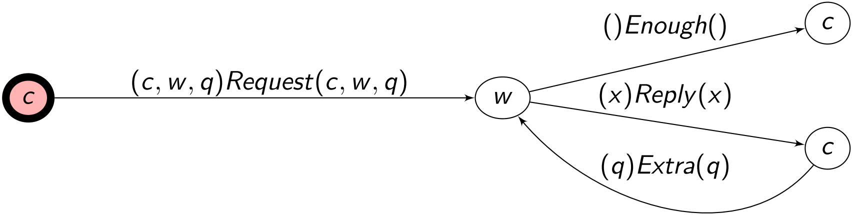
# Sessions (contracts, conversations, workflows, ...)




Text representation:

```
role  c = send Request{c,w,q};
          loop: recv [ Reply{x}  → send Extra{q};loop | Enough ]
role  w = recv Request{c,w,q}  →
          loop: send ( Reply{x}; recv Extra{q}  → loop + Enough )
```

# Sessions (contracts, conversations, workflows, ...)




## Execution

Labels:

Store:

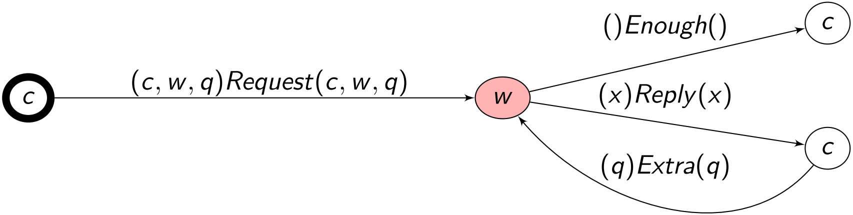
$c$:

$w$:

$q$:

$x$:

# Sessions (contracts, conversations, workflows, ...)



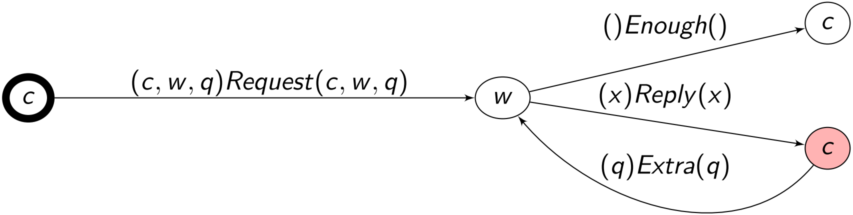

## Execution

Labels: *Request*

Store:

| | |
|---|---|
| $c$: | Alice |
| $w$: | Bob |
| $q$: | "Gone with the wind" |
| $x$: | |

# Sessions (contracts, conversations, workflows, ...)



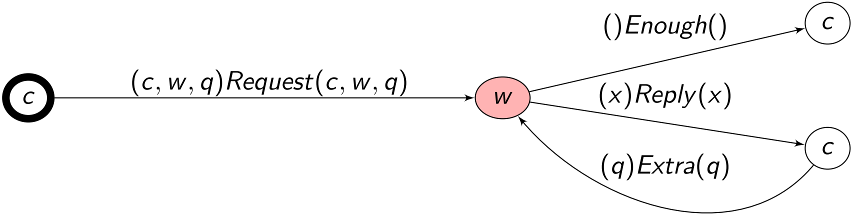

## Execution

Labels: *Request-Reply*

Store:

| | |
|---|---|
| $c$: | Alice |
| $w$: | Bob |
| $q$: | "Gone with the wind" |
| $x$: | "8 euros" |

# Sessions (contracts, conversations, workflows, ...)



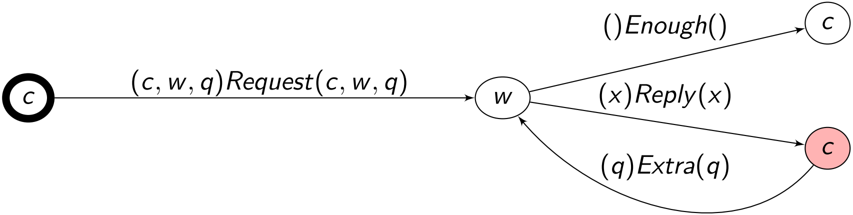

## Execution

Labels: *Request-Reply-Extra*

Store:

| | |
|---|---|
| $c$: | Alice |
| $w$: | Bob |
| $q$: | “In stock?” |
| $x$: | “8 euros” |

# Sessions (contracts, conversations, workflows, ...)



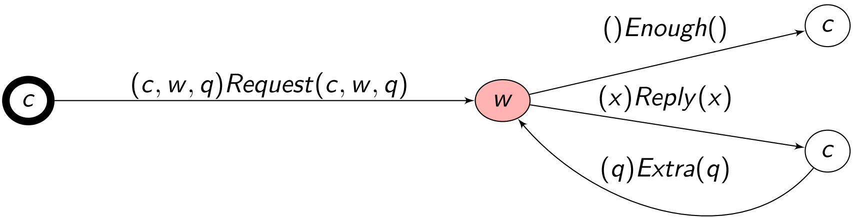

## Execution

Labels: *Request-Reply-Extra-Reply*

Store:

| | |
|---|---|
| $c$: | Alice |
| $w$: | Bob |
| $q$: | “In stock?” |
| $x$: | “yes” |

# Sessions (contracts, conversations, workflows, ...)



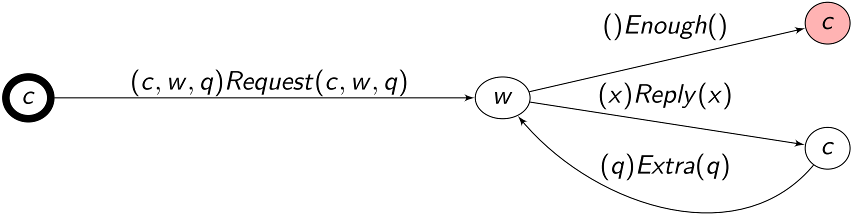

## Execution

Labels: *Request-Reply-Extra-Reply-Extra*

Store:

| | |
|---|---|
| $c$: | Alice |
| $w$: | Bob |
| $q$: | “Delivery date?” |
| $x$: | “yes” |

# Sessions (contracts, conversations, workflows, ...)




### Execution

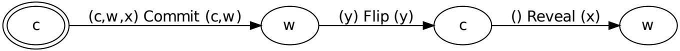
Labels: *Request-Reply-Extra-Reply-Extra-Enough*

Store:

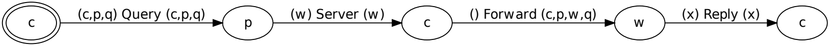
| | |
|---|---|
| $c$: | Alice |
| $w$: | Bob |
| $q$: | "Delivery date?" |
| $x$: | "yes" |

# Expressivity

- Loops, branching, value passing, and value rebinding (as we already saw)
- Committment "coin flips by telephone" ($c$ commits to $x$ without prior knowledge of $y$; likewise, $w$ chooses $y$ without knowledge of $x$)



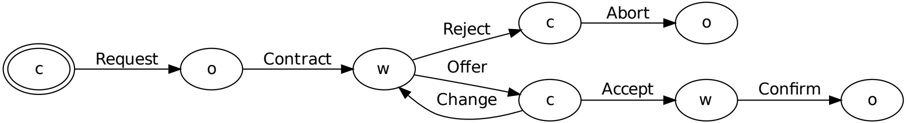

- Dynamic principal binding (the proxy $p$ gets to choose the web server $w$ based on the client $c$ and her login credentials $q$)



# Threats against session integrity

## Powerful Attacker model

- can spy on transmitted messages
- can join a session as any role
- can initiate sessions
- can access the libraries (networking, crypto)
- cannot forge signatures




## Attacks against an insecure implementation

- (Integrity) Rewrite Offer by Reject
- (Replay) Intercept Reject and replay old Offer, triggering a new iteration
- (Sender authentication) send Confirm to *o* without having received an Accept
- ... and many more against the store

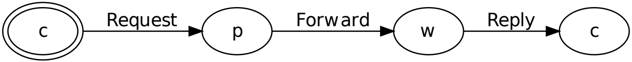
# Protocol outline

Principles of our protocol generation

1. Each edge is implemented by a unique concrete message.
2. We want static message handling for efficiency.

## Against replay attacks

- between session executions: session nonces
- between loop iterations: time stamps
- at session initialisations: anti-replay caches

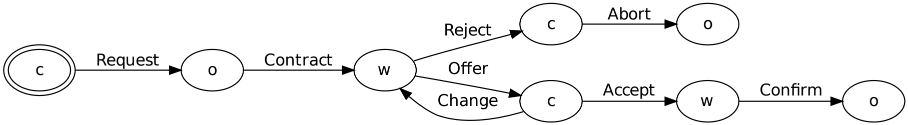



## Against session flow attacks

- Signatures of the entire message history (optimisations possible ...)

# Optimisation: visibility

Do we really need to include a complete signed history in every message?

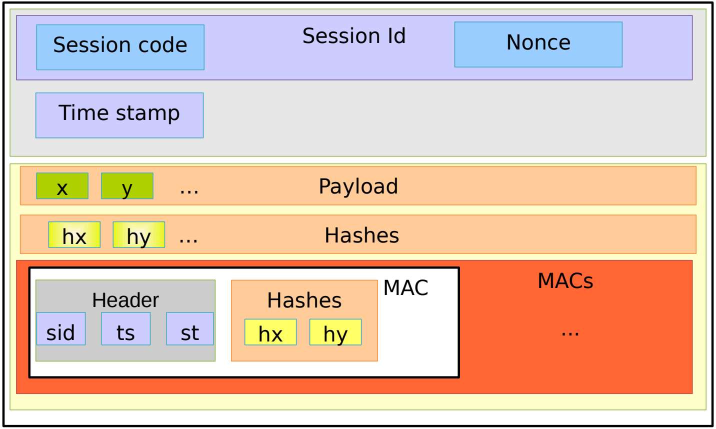



**Execution paths: which signatures to convince the receiver?**

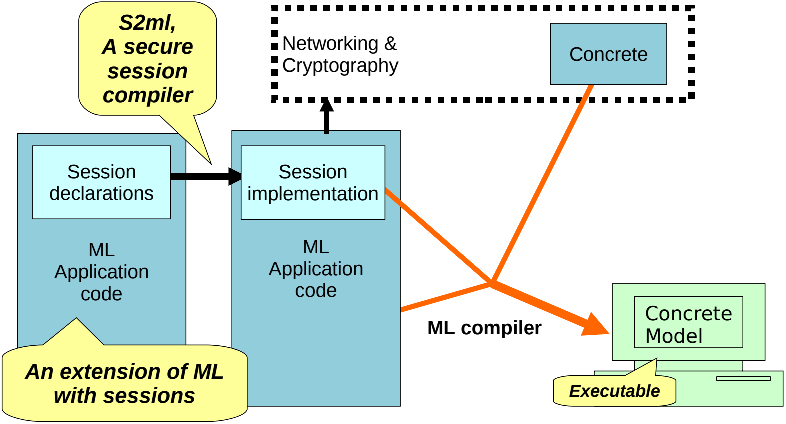
- `Request-Contract-Reject`
- `Request-Contract-Offer-Change-Offer-Change`
- `Request-Contract-(Offer-Change)`$^n$`-Reject-Abort`

Visibility: at most one signature from each of the previous roles is enough.

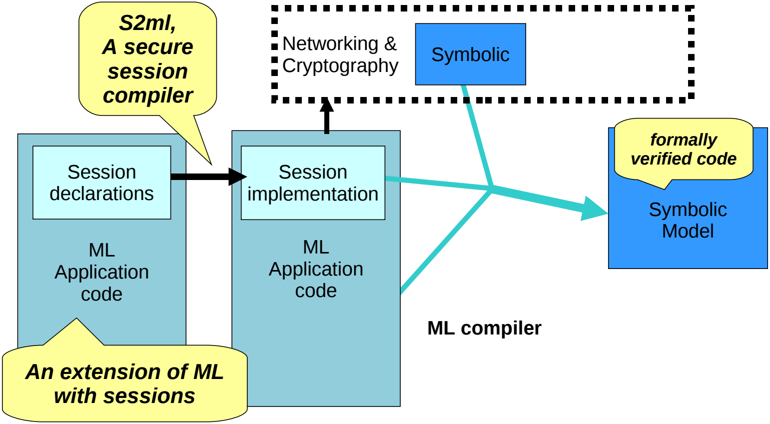
# Message format



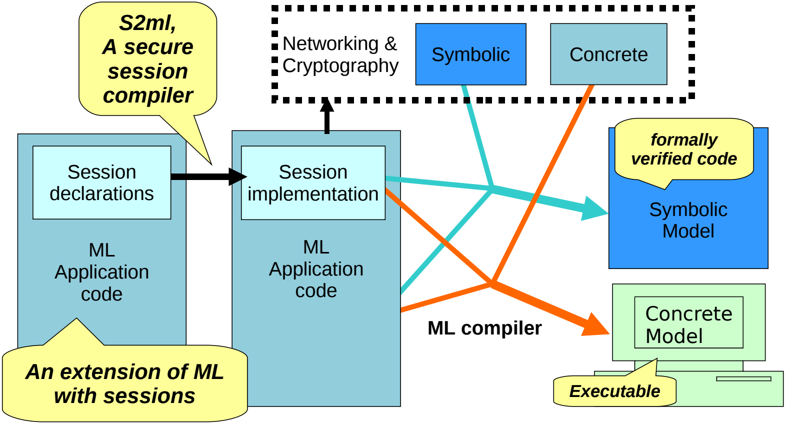
# Architecture



# Architecture



# Architecture



# Security result

**Theorem (Session Integrity)**

For any run of a $S_1 \ldots . S_n$-system, there is a partition of the compliant events such that each equivalence class coincides with a compliant subtrace of a session $S_i$ from from $S_1 \ldots S_n$.

# Security result

**Theorem (Session Integrity)**

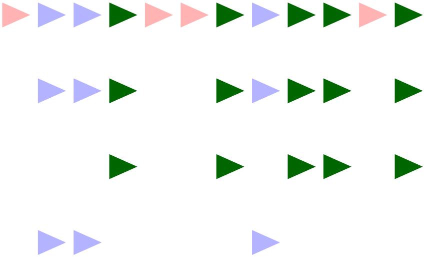
For any run of a $S_1 \ldots S_n$-system, there is a partition of the compliant events such that each equivalence class coincides with a compliant subtrace of a session $S_i$ from from $S_1 \ldots S_n$.

All events:

Compliant events:

...corresponding to $S_1$ events:

...and $S_2$ events:



# Performance evaluation

## Performance of the code generation

| Session `S` | Roles | File `.session` (loc) | Application (loc) | Local graph (loc) | Graph (loc) | `S.mli` (loc) | `S.ml` (loc) | Compilation (s) |
|---|---|---|---|---|---|---|---|---|
| `Single` | 2 | 5 | 21 | 8 | 12 | 19 | 247 | 1.26 |
| `Rpc` | 2 | 7 | 25 | 10 | 18 | 23 | 377 | 1.35 |
| `Forward` | 3 | 10 | 33 | 12 | 25 | 34 | 632 | 1.66 |
| `Auth` | 4 | 15 | 45 | 16 | 38 | 49 | 1 070 | 1.86 |
| `Ws` | 2 | 7 | 33 | 12 | 24 | 25 | 481 | 1.36 |
| `Wsn` | 2 | 15 | 44 | 13 | 42 | 29 | 782 | 1.50 |
| `Wsne` | 2 | 19 | 45 | 15 | 48 | 31 | 881 | 1.90 |
| `Shopping` | 3 | 29 | 70 | 21 | 85 | 49 | 1 780 | 2.43 |
| `Conf` | 3 | 48 | 86 | 37 | 181 | 78 | 3 451 | 3.32 |
| `Loi` | 6 | 101 | 189 | 57 | 310 | 141 | 7 267 | 6.29 |

## Performance of the generated code for Conf (10 000 messages)

| | Time | Overhead |
|---|---|---|
| Unprotected (no key establishment) | 1.31 s | 0 % |
| Don't sign but do cache checking | 1.43 s | 9 % |
| Sign but don't verify | 1.66 s | 27 % |
| Fully protected | 1.77 s | 35 % |

# Conclusion

- Security protocols are hard to write by hand. They are long, complicated, difficult to verify, and fragile in the face of specification change.
- Automatic generation with mechanised verification is the future!

# Conclusion

- Security protocols are hard to write by hand. They are long, complicated, difficult to verify, and fragile in the face of specification change.
- Automatic generation with mechanised verification is the future!

We have:

- designed a high-level session language,
- built a compiler for generating secure implementations from session specifications,
- mechanised the verification of the resulting security protocols (executable code not just models!)

`http://www.msr-inria.inria.fr/projects/sec/sessions/`

# Conclusion

- Security protocols are hard to write by hand. They are long, complicated, difficult to verify, and fragile in the face of specification change.
- Automatic generation with mechanised verification is the future!

We have:

- designed a high-level session language,
- built a compiler for generating secure implementations from session specifications,
- mechanised the verification of the resulting security protocols (executable code not just models!)

`http://www.msr-inria.inria.fr/projects/sec/sessions/`

Thank you and *bon appétit*!