Secure Implementations for Typed Session Abstractions

Ricardo Corin, Pierre-Malo Deniélou,
Cédric Fournet, Karthik Bhargavan, James Leifer

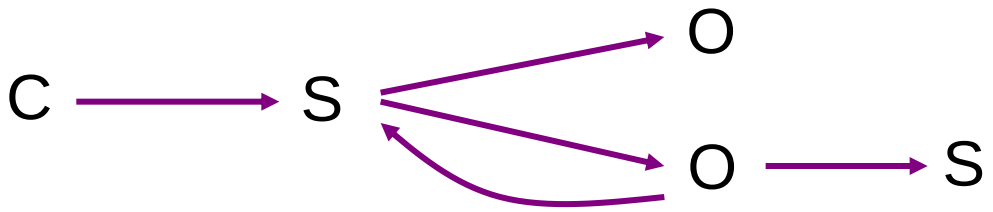
INRIA—Microsoft Research Joint Centre
http://www.msr-inria.inria.fr/projects/sec/sessions/

Distributed applications

- How to program networked independent sites?
  - Each site has its own code & security concerns
  - Sites may interact, but they do not trust one another

- Communication abstractions can help
  - Hide implementation details (message format, routing,…)
  - Basic communication patterns, e.g. RPCs or private channels

  - Sessions, (aka protocols, or contracts, or workflows)



Session types

- Active area for distributed programming
  - From pi calculus to web services, operating systems, ...
  - General strategy: enforce protocol compliance by typing
    If all programs are well-typed, session runs follow their spec

- Secure implementation?
  - Needs protection against network attackers (e.g. SSL)
  - Needs protection from partially-trusted remote parties
  - Defensive implementations must monitor one another, giving up most benefits of abstraction

Compiling session types to protocols

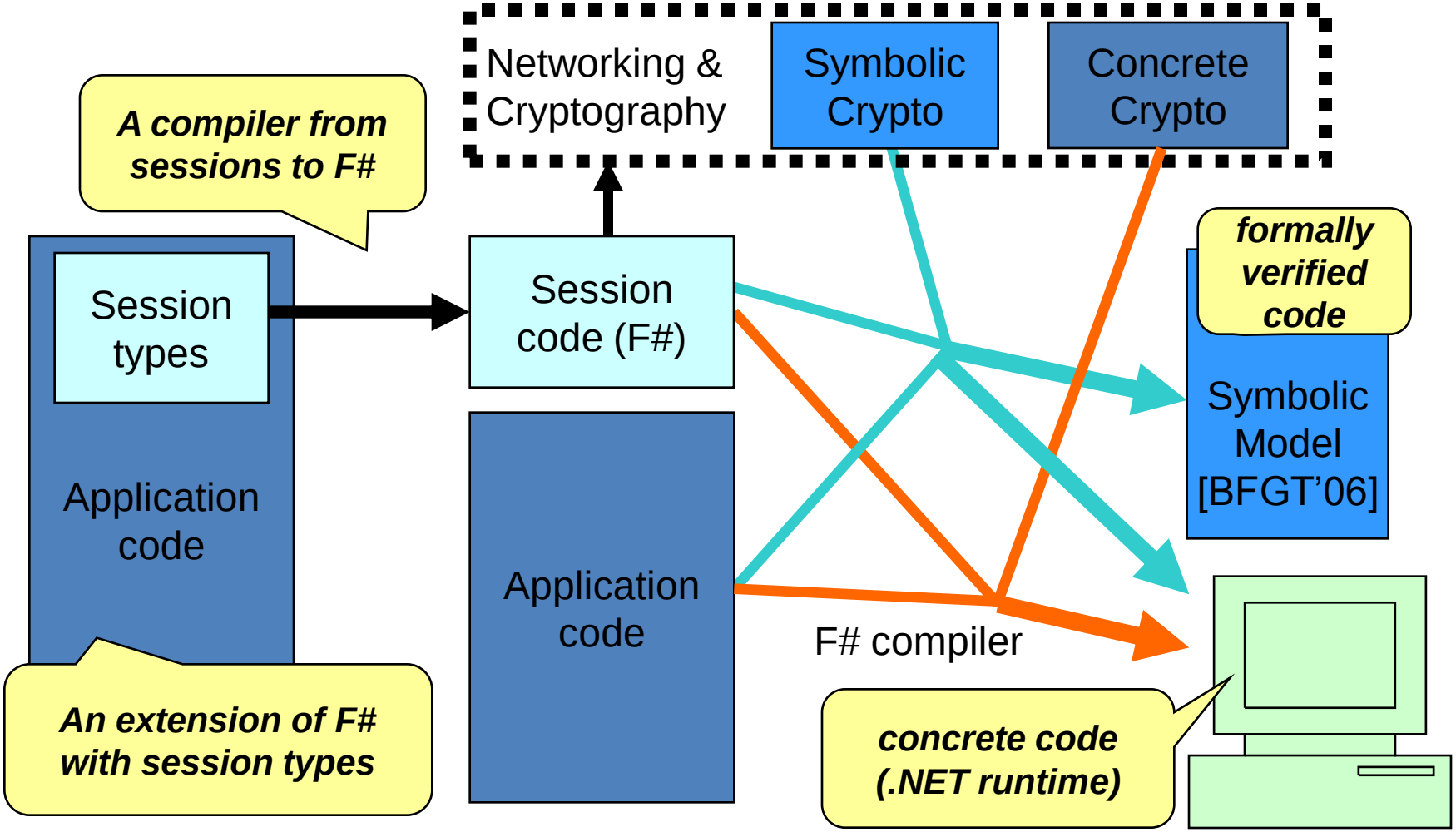
- We extend F# (a variant of ML)
  with session types that express message flows

  Well-typed programs always play their roles

- We compile session type declarations to crypto protocols
  that shield our programs from any coalitions of remote peers

  Remote sites can be assumed to play their roles
  (without trusting their code)

Compiling session types to protocols

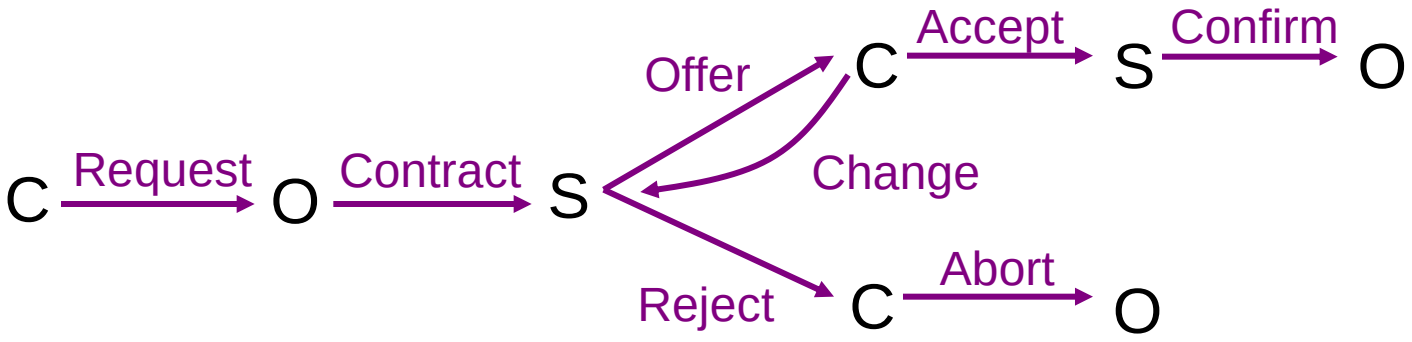


Expressing sessions

- Terminology:
  - Roles: represent session participants
  - Principals: instantiate roles at runtime
  - Messages: consist of labels and payloads

- Two ways to represent sessions:
  - As a graph: useful for global reasoning on sessions
  - As a collection of local roles:
    useful for the language semantics and implementation
  - The two representations are interconvertible

Example



"Customer C negotiates delivery of an item with a store S; the transaction is registered by an officer O."

```
session S3 =
  role store:string =
    ?Contract:string; mu start.
    !( Offer:string;
        ?( Change:string; start
        + Accept; !Confirm )
      + Reject )

  role officer = . . .
  role customer = . . .
```

A small session language

$$
\begin{array}{ll}
\tau ::= & \text{Payload types} \\
\quad \mathrm{int} \mid \mathrm{string} & \quad \text{base types} \\
p ::= & \text{Role processes} \\
\quad !(f_i : \widetilde{\tau}_i \ ; \ p_i)_{i<k} & \quad \text{send} \\
\quad ?(f_i : \widetilde{\tau}_i \ ; \ p_i)_{i<k} & \quad \text{receive} \\
\quad \mu\chi.p & \quad \text{recursive declaration} \\
\quad \chi & \quad \text{recursion} \\
\quad 0 & \quad \text{end} \\
\Sigma ::= & \text{Sessions} \\
\quad (r_i : \widetilde{\tau}_i = p_i)_{i<n} & \quad \text{initial role processes } p_i \text{ for the roles } r_i
\end{array}
$$

(SEND) $!(f_i : \widetilde{\tau}_i \ ; \ p_i)_{i<k} \xrightarrow{\overline{f}_i}_r p_i$ (RECEIVE) $?(f_i : \widetilde{\tau}_i \ ; \ p_i)_{i<k} \xrightarrow{f_i}_r p_i$

Our formal subset of F# with sessions

$T ::=$		Type expressions
	t	type variable
	int, string, unit	base types
	T chan	channel types
	$T_1 \to T_2$	arrow type
$v ::=$		Values (also used as Patterns)
	x	variable
	$0, 1, \ldots,$ Alice, Bob, $\ldots, ()$	constants for base types
	$l, c, n, \ldots$	names for functions, channels, nonces
	$f(v_1, \ldots, v_k)$	constructed term (when f has arity k)
$e ::=$		Expressions
	v	value
	$l\ v_1 \ldots v_k$	function application
F	match v with $(\lvert v_i \to e_i)_{i<k}$	value matching
	let $x = e_1$ in e_2	value definition
	let $(l_i\ x_0 \ldots x_{k_i} = e_i)_{i<k}$ in e	mutually-recursive function definition
+	type $(t_i = (\lvert f_{j_i}$ of $\widetilde{T}_{j_i})_{j_i<k_i})_{i<k}$ in	mutually-recursive datatype definition
	session $S = \Sigma$ in e	session type definition
S	$S.r^b\ \widetilde{v}\ (v)$	session entry
	$s.p(e)$	session role (run-time only)
$E[\cdot] ::=$		Evaluation contexts
	$[\cdot]$	top level
	let $x = E[\cdot]$ in e_2	sequential evaluation
	$s.p(E[\cdot])$	in-session evaluation (run-time only)
$P ::=$		Processes
	e	running thread
	$P \mid P$	parallel composition
	$\mathbf{0}$	inert process

F+S semantics

- The source F+S semantics models a centralized session monitor
  - layered semantics

    →roles

    $$(\text{SEND}) \quad !(f_i : \widetilde{\tau}_i \,;\, p_i)_{i<k} \xrightarrow{\overline{f}_i}_r p_i \qquad (\text{RECEIVE}) \; ?(f_i : \widetilde{\tau}_i \,;\, p_i)_{i<k} \xrightarrow{f_i}_r p_i$$

    →sessions

    $$(\text{STEP}) \frac{p \xrightarrow{\eta}_r p'}{\rho, s.p \xrightarrow{\eta}_s \rho, s.p'} \qquad (\text{SENDS}) \frac{\rho, s.p \xrightarrow{\overline{g}}_s \rho', s.p'}{\rho, s.p \; (g(\widetilde{v}), w) \xrightarrow{s\overline{g}\;\widetilde{v}}_e \rho', s.p' \; (w)}$$

    →expressions (…)

    →configurations (…)

  - constitutes our global specification for sessions
  - does not exist in F, our target language

Global session integrity

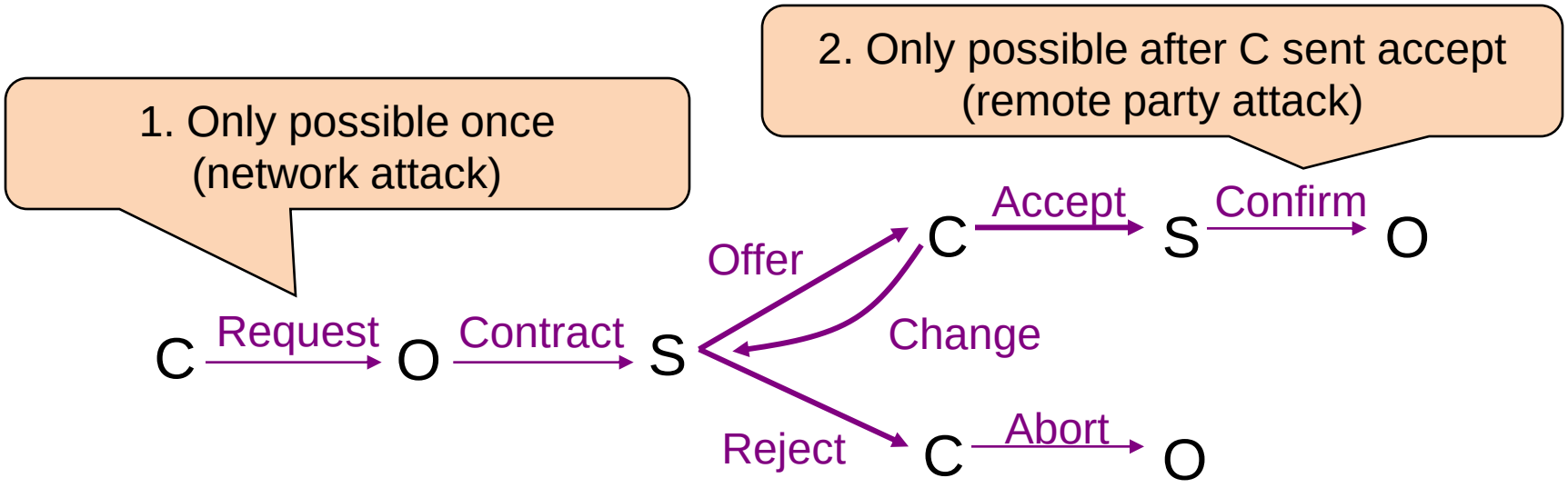
- For any run,
  - for any choice of good and bad principals,
    for any session:
  - there exists a valid path in the session graph
  - that is consistent with all the messages
    sent and received by the good principals

- Session integrity holds by design in F+S
  - Generalizes correspondence properties (=path properties)

- Our compiler generates cryptographic code to enforce this in F

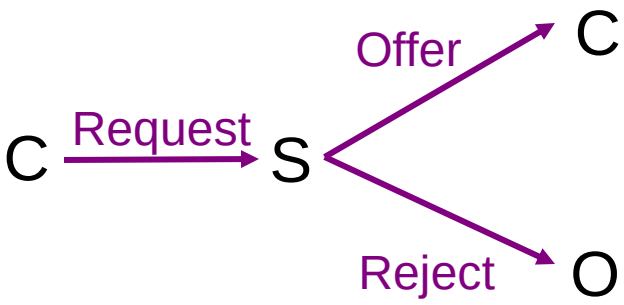
Global session integrity

- Examples that could break integrity:



Implementability conditions

- Some sessions are always vulnerable



- We detect them and rule them out
  - They can be turned into safe sessions but only with extra messages

Security protocol

- We combine standard mechanisms
  - X509 digital signatures
  - Logical timestamps for loop control
  - Anti-replay cache
    - Per principal, based on session identifier Hash(S, a, N) + role
- Which evidence to sign & forward?

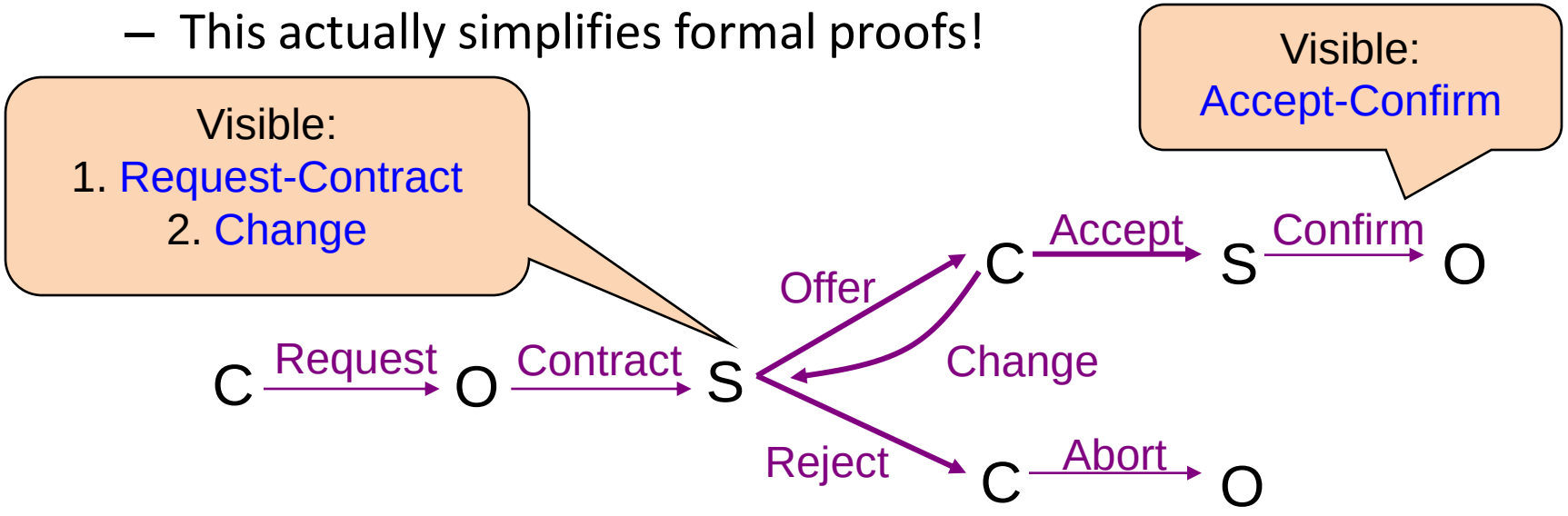
Forwarding history

- Complete history
  - Every sender countersigns the whole history so far
  - Every receiver checks signatures and simulates the history vs. session spec
  - Large overhead (unbounded crypto processing)

- We can do much better

Visibility

- Visibility = minimum information needed to update local role
  - The sequence of last labels from all peers since last message send
  - Any less information would break integrity
- Can be computed statically from the session graph
  - More work for the compiler = less runtime tests
  - This actually simplifies formal proofs!



Our session compiler

- Generates interface (types for all messages)
- Generates specific sending and receiving code for each visible sequence
  - Checks exactly what is expected
  - Zero dynamic graph computation

- 5000 lines in F# + dual F# libraries

Dual libraries [BFGT'06]

- Crypto library:

```
type bytes
type keybytes
val nonce: name → bytes
val hash: bytes → bytes
val genskey: name → keybytes
val genvkey: keybytes → keybytes
val sign: bytes → keybytes → bytes
val verify: bytes → bytes → keybytes → bool
```

- Principals library:

```
val skey : principal → keybytes
val vkey : principal → keybytes
val psend : principal → bytes → unit
val precv : principal → bytes
val safe : principal → bool
val psend• : (principal ∗ bytes) chan
val chans• : (principal ∗ bytes chan) list
val skeys• : (principal ∗ bytes) list
```

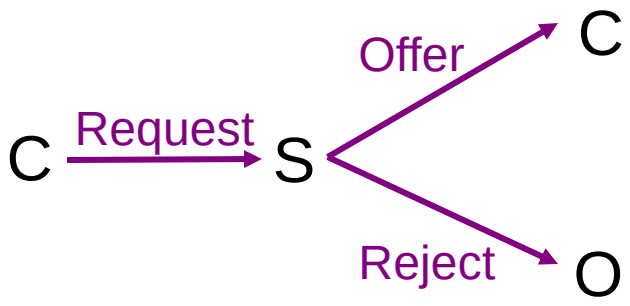
- Dual implementations
  - Symbolic: using algebraic datatypes and type abstraction
  - Concrete: using actual system (.NET) operations

Integrity theorems

- Configuration =
  Libraries + Session Declarations + User Code + Opponent Code

Theorem 1 (Security, reduction-based). If $L\ M_{\widetilde{S}}\ U\ O'$ may fail in F for some O' where ω does not occur, then $L\ \widetilde{S}\ U\ O$ may fail in F+S for some O where ω does not occur.

counter-example if
we allowed session forks:



Theorem 2 (Security, labelled-transition based). Let W be a valid implementation of H. For all transitions $W \overset{\varphi}{\Rightarrow}_{\mathsf{K}} W'$ in F, where φ represents the observable trace of those transitions, there exists W° valid implementation of H°, such that $W \overset{\varphi}{\Rightarrow}_{\mathsf{K}} W^\circ \rightarrow^*_{\mathsf{KD}} W''$ and $W' \rightarrow^*_{\mathsf{KD}} W''$ and $H \overset{\psi}{\Rightarrow}_{\mathsf{K}} H^\circ$ with φ the translation of ψ.

Discussion

- Session types are an active area of study
  - we address their secure implementation
- Protocol verification:
  - We verify our implementation code—not just a simplified model
  - Our results hold for any number of (concurrent) sessions
  - Even for a single session, this is beyond automated verification tools (loops and branching)
  - Crypto is Dolev-Yao but not far from computational model
  - Integrity, not liveness (so no progress or global termination)
- Related work on secure implementations of process calculi, on automated protocol transformations

Conclusion

- Cryptographic protocols can sometimes be derived (and verified) from application security requirements
  - Strong, simple security model
  - Safer, more efficient than ad hoc design & code

- Future work?
  - Data binding and correlation
  - More dynamic principals
  - Secure marshalling for richer types

- Try it out today!
  http://www.msr-inria.inria.fr/projects/sec/sessions/