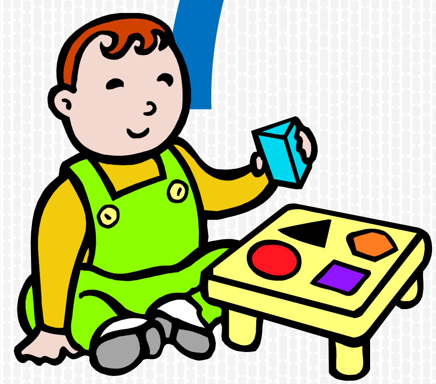
F7



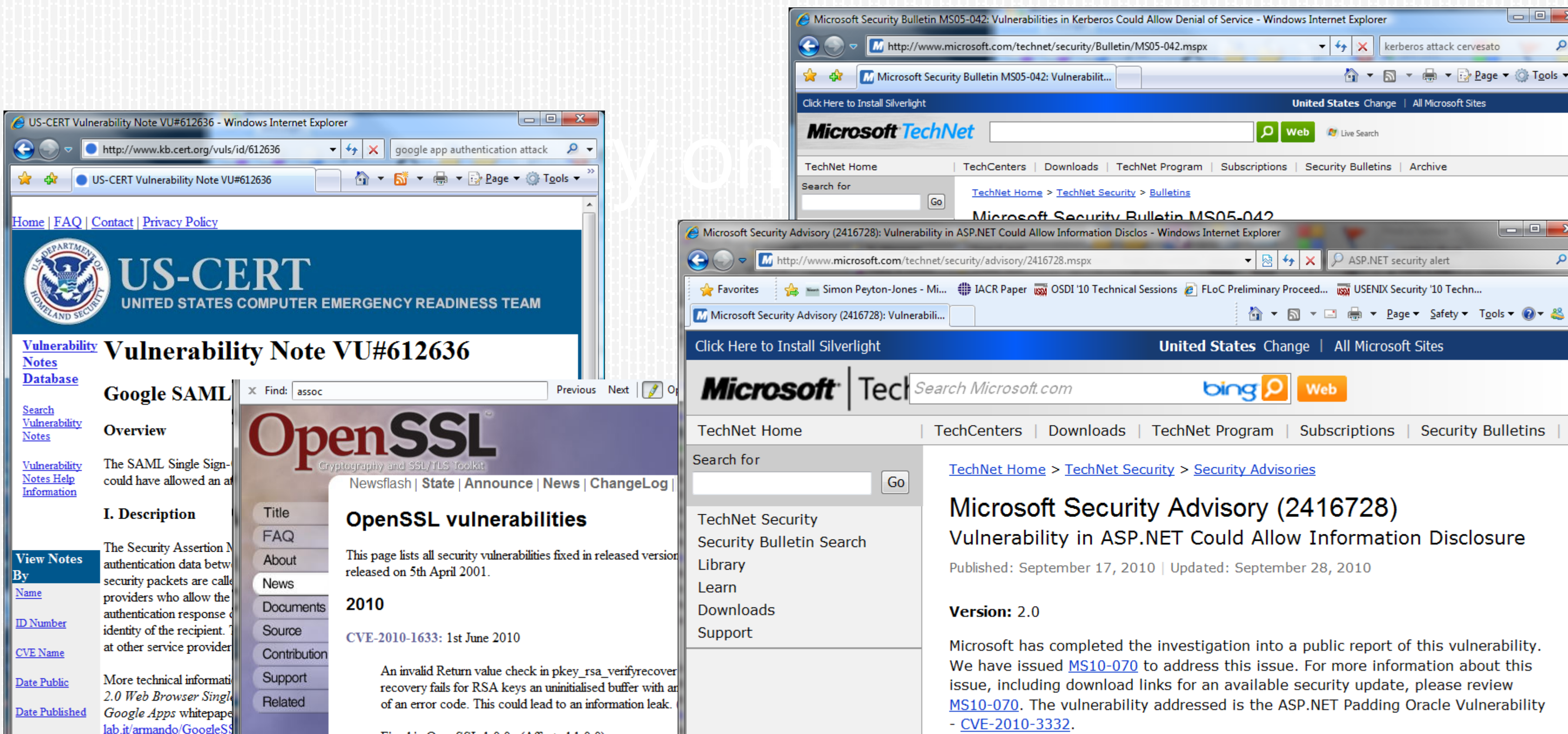
MODULAR CODE-BASED CRYPTOGRAPHIC VERIFICATION

MARKULF KOHLWEISS (MARKULF@MICROSOFT.COM

Joint work with Cédric Fournet and Pierre-Yves Strub

CRYPTO PROTOCOLS (STILL) GO WRONG

- Design & implementation errors lead to vulnerabilities
- Traditional crypto models miss most details
- Production code and design specs differ



THIS TALK

Goal: **Automated verification of protocol code** under **standard cryptographic assumptions** (rather than symbolic verification of protocol models)

Method: **Refinement types & parametricity**
Proofs are by programming, typechecking, and local game-based code rewriting

Outline

- Background / a Mixed Bag
  - A bit of history
  - Type checking for (non-)programmers
  - Goldreich in F#
  - The big picture
- Example Primitive: Authenticated Encryption
- Example Protocol: Remote Procedure Call Protocol

FORMAL COMPUTATIONAL CRYPTOGRAPHY

Two approaches for verifying protocols and programs

Symbolic models (Needham-Schroeder, Dolev-Yao, … late 70's)

- Structural view of protocols, using formal languages and methods
- Many automated verification tools, scales to large systems including full-fledged implementations of protocol standards

Computational models (Yao, Goldwasser, Micali, Rivest, … early 80's)

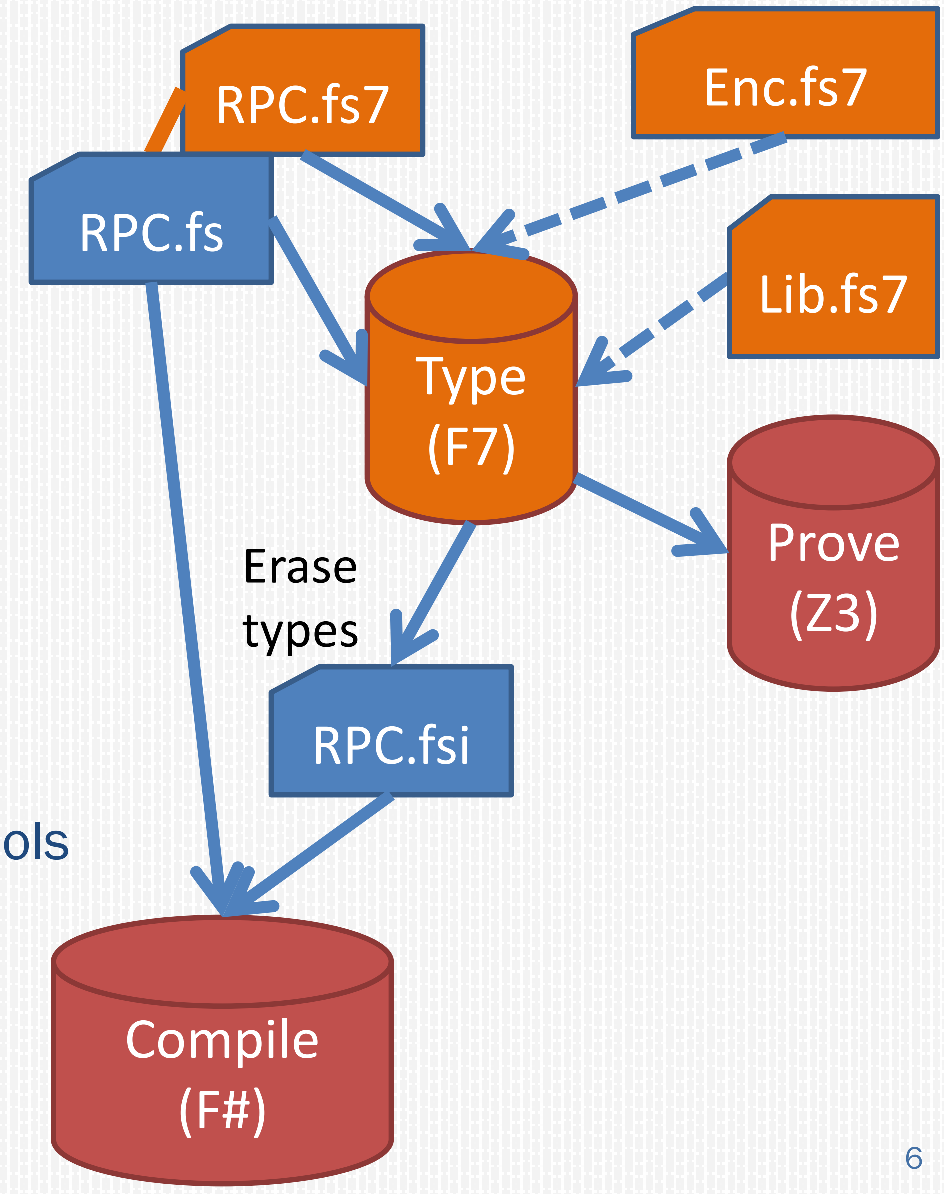
- Concrete, algorithmic view, using probabilistic polynomial-time machines
- New formal tools: CryptoVerif, Certicrypt, Easycrypt

Can we get the best of both worlds?

- Much ongoing work on computational soundness for symbolic cryptography (Abadi Rogaway, Backes Pfitzmann Waidner, Warinschi,… mid 00's)
  - It works… with many mismatches, restrictions, and technicalities
  - At best, one still needs to verify protocols symbolically
- Can we directly verify real-world protocols ? This paper: type-based verification is **more effective** and **more compositional** computationally than symbolically.

F7: REFINEMENT TYPECHECKING FOR F#

- We program in F#
- We specify in F7
  We typecheck programs
  against interfaces
- F7 does some type inference
  & calls Z3, an SMT solver,
  on each logical proof obligation
- In prior work: symbolic crypto
  libraries and verified large protocols
  (e.g. CardSpace at POPL'10)



ASSUME AND ASSERTS; SAFETY BY TYPING

Refinement types $\{x{:}T|C\}$

```
// Sample type and value declarations in F7
type nat  = n:int{ 0 < n }
val read:  n:nat -> b:bytes{ Length(b) <  n }
```

Global set of first-order logical formulas, the log

- **assume** C adds C to the log
- **assert** C succeeds if C logically follows from the logged formulas
- An expression A is *safe* if and only if
  in all evaluations of A, all assertions succeed.
- We use a logic judgement $I \vdash C$ (C follows from refinements in I)

Theorem 1 (Safety by Typing)
If $\emptyset \vdash A{:}T$ then A is safe.

COMPUTATIONAL SECURITY WITH F7

- Use existing F7 typechecker and code base
- Remove non-determinism
- Add probabilistic sampling and native references
- (Prove type safety & parametricity of new extended subset of F7 in Coq)

- We still type protocols and applications against refined typed interfaces that idealize crypto libraries
- We relate two implementations of crypto libraries
  - Ideal, well-typed functionality (replaces symbolic libraries)
  - Concrete implementation (with weaker typing in F7)
- Computational security follows from p.p.t. indistinguishability (a bit similar to universal composability)

COMPLEXITY, PROBABILITY, AND ASYMPTOTICS

- Series $(A_\eta)_{\eta \geq 0}$ of expressions indexed by η. (Short A)
- Define p.p.t. for expressions A such that $I_{Pr} \vdash A : T$ and modules Pr such that $\mathrm{I} \vdash Pr \mapsto I_{Pr}$.
  - Limit ourselves to 1st order interfaces.
  - Top most attacker interface I_{Pr} unrefined,
    ⇒ power of A corresponds to Oracle Turing machine.

- Fair coin tossing primitive with probabilistic semantics $A \rightarrow_p A'$

$$\textbf{sample} \rightarrow_{\frac{1}{2}} \text{true}, \textbf{sample} \rightarrow_{\frac{1}{2}} \text{false}$$

- A is *asymptotically safe* when the series of probabilities of A_η being unsafe is negligible.
- A^0 and A^1 are *asymptotically indistinguishable,* $\mathrm{A}^0 \approx A^1$, when $|\Pr[A^0 \Downarrow M] - \Pr[A^1 \Downarrow M]|$ is negligible for all closed values M.

CRYPTOGRAPHY USING F7

- $P \cdot G \cdot A$ (oracle systems),
  - P functions describing cryptographic primitives
  - G game programming the oracles made available to attacker
  - A module describing attacker program that tries to win the game
- Auth. Encryption: C_{Enc} defines GEN, ENC, and DEC.
  - p.p.t. adversary A.
  - CTXT security defined as $C_{Enc} \cdot CTXT \cdot A$ asymptotically safe
  - CPA security defined as $C_{Enc} \cdot CPA_0 \cdot A \approx_{\epsilon} C_{ENC} \cdot CPA_1 \cdot A$, where

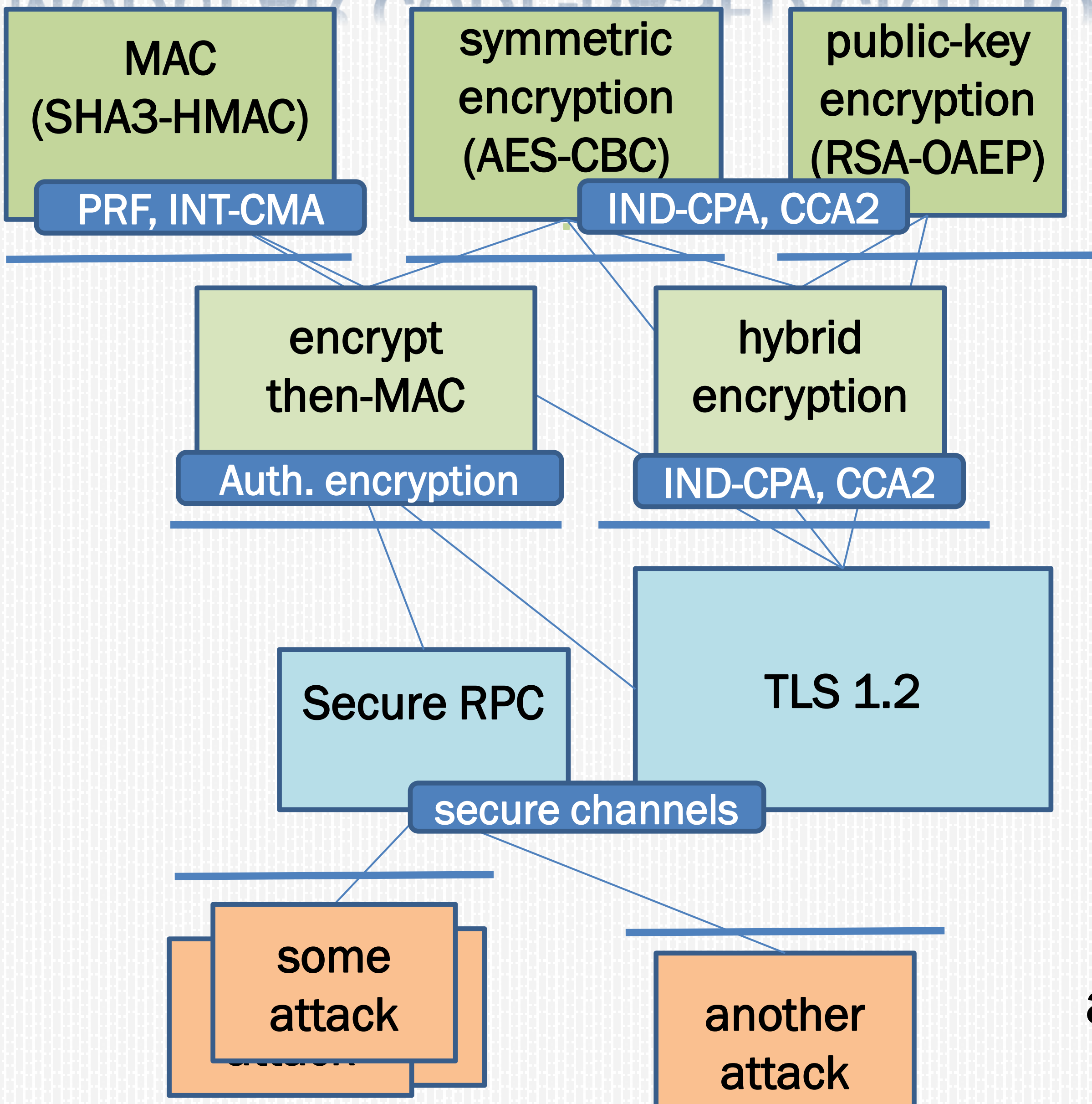
CPA_b

```
let k = GEN ()
 let enc x_0 x_1 =
  let x = x_b in
  let c = ENC k x in
```

CTXT

```
let k = GEN()
let log = ref []
let enc p = let c=ENC k p in log := c::!log; c
let dec c =
 match DEC k c with
 | None -> None
 | Some(x) -> assert(List.mem c !log); x
```

MODULAR CODE-BASED CRYPTO VERIFICATION



cryptographic primitives

typed interfaces
(security guarantees)

cryptographic functionalities

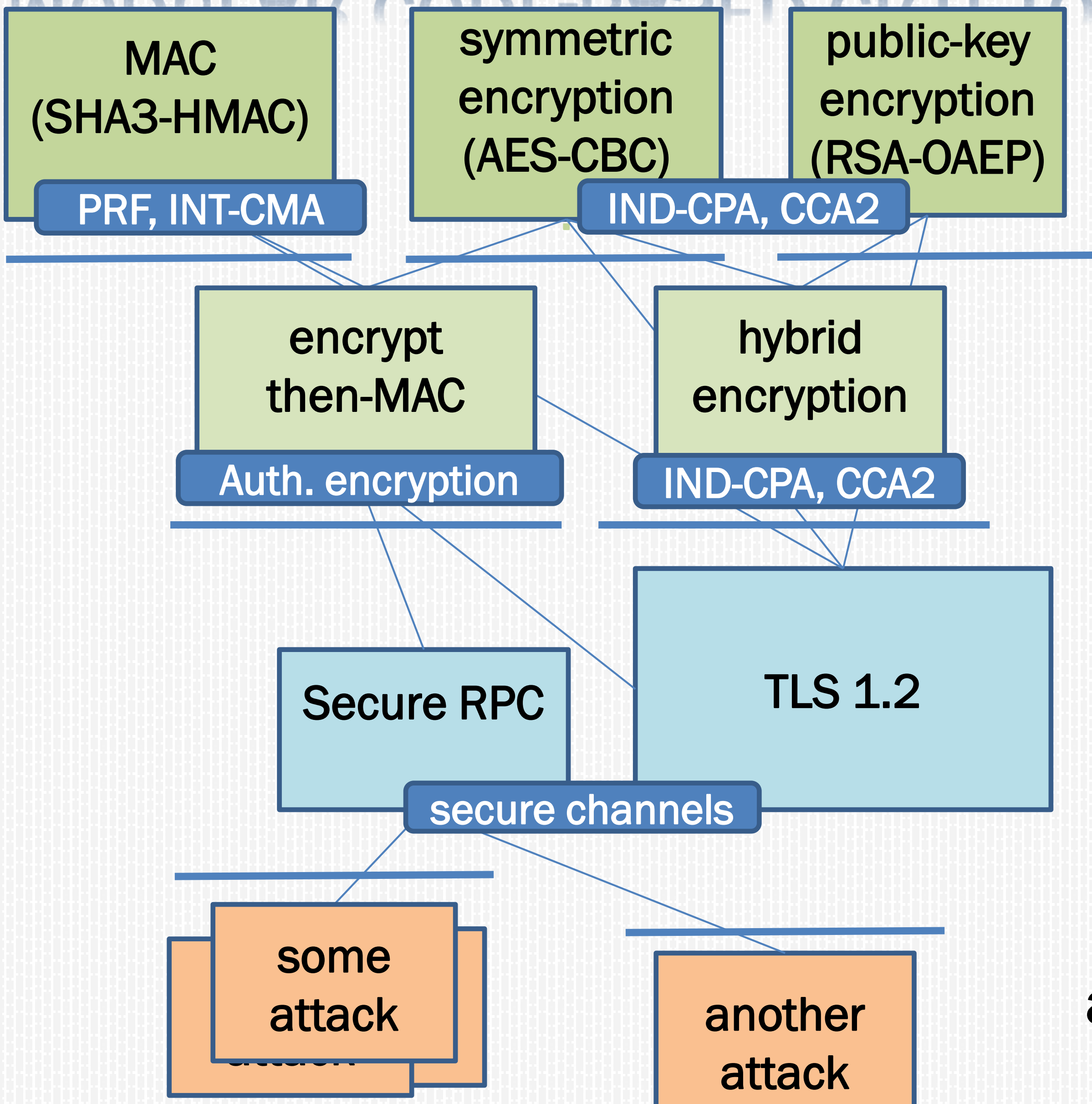
typed interfaces
(security guarantees)

security protocols

typed interfaces
(attacker model)

active adversaries

MODULAR CODE-BASED CRYPTO VERIFICATION



cryptographic primitives

typed interfaces (security guarantees)

cryptographic functionalities

typed interfaces (security guarantees)

security protocols

typed interfaces (attacker model)

active adversaries

Authenticated Encryption

Sample ideal interfaces and functionalities

AUTHENTICATED ENCRYPTION

plain F# interface

```
module Enc

type plain = bytes
type key = bytes
type cipher = bytes


val GEN: unit -> key
val ENC: k:key -> plain  -> cipher
val DEC: k:key -> cipher -> (plain) option
```

This interface says nothing about security of Enc

AUTHENTICATED ENCRYPTION

keys are abstract

Ciphertext has fixed size

ideal F7 interface

```
module Enc        val ciphersize

open Plain
type key                    {Length(b)=ciphersize}
type cipher = b:bytes

predicate Msg of key * plain

val GEN: unit -> key
val ENC: k:key -> t:plain{Msg(k,t)} -> cipher
val DEC: k:key -> t:cipher
                -> (plain{Msg(k,t)}) option
```

Msg is specified by protocols using Enc

"All decrypted messages have been encrypted"

sample protocol using Auth Enc

```
module RPC
definition !k,q. Msg(k,Utf8(q)) <=> Request(q)
```

```
let client q =
  // precondition:
  // Request(q)
 … send ENC k (utf8 q)
```

```
let server q =
  … let m=DEC k (utf8 q)
    if m!=None
    then // we have Request(q)
         process q
```

AUTHENTICATED ENCRYPTION

We express perfect, i.e., information theoretic, properties on interfaces:

$$I_{PLAIN} \vdash C_{Enc} \cdot F_{Enc} \mapsto I^{ae}_{Enc}$$

- Refinements model authenticity properties
- Abstraction in I_{PLAIN} models that other outputs of F_{ENC}, in particular ciphertexts, are independent of abstractly typed plain.

I^{C}_{Plain}

```
type plain
val service: plain → plain
val repr: p:plain →
  b:bytes {Len(b)=plainsize}
val plain:
  b:bytes{Len(b)=plainsize} → p:plain
```

I^{ae}_{Enc}

```
type key
val GEN: unit → key
val ENC: k:key → p:plain {Msg(k,p)}
              → c:cipher
val DEC: k:key → c:cipher
              → (p:plain {Msg(k,p)}) option
```

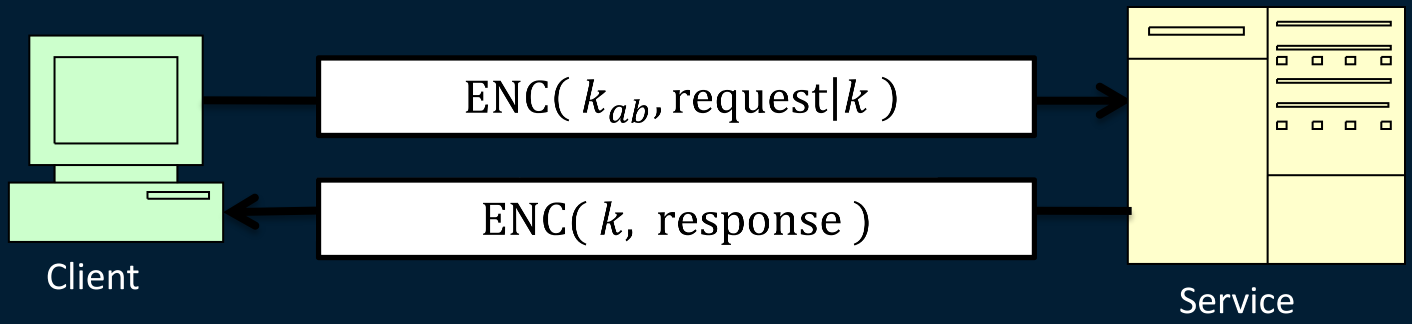
AUTHENTICATED ENCRYPTION

- Real Enc cannot meet this interface, but ideal functionality does

```
let GEN () =
  let kv = Enc.GEN() in
  let log = ref [] in
  Key(kv,log)
let ENC (Key(kv,log)) (x:plain) =
  let c = Enc.ENC kv zero in
  log := (c,x) :: !log;
  c
let DEC (Key(kv,log)) c = assoc kv c !log
```

- Check using typing that $\mathrm{I}_{\mathrm{Plain}} \vdash \mathrm{C}_{\mathrm{Enc}} \cdot \mathrm{F}^{ae}_{\mathrm{Enc}} \mapsto I^{ae}_{Enc}$
- Prove that ∀p.p.t. P, A, s.t., $\vdash P \mapsto I^{c}_{Plain}$ and $I^{c}_{Plain}, I^{ae}_{Enc} \vdash A$.

$$P \cdot \mathrm{C}_{\mathrm{Enc}} \cdot A \approx_{\epsilon} P \cdot C_{Enc} \cdot F^{ae}_{Enc} \cdot A$$



Encrypting Session Keys

AUTHENTICATED Encrypted RPC Sample Protocol

We obtain no guarantee of request/response correlation:

Client sends request1, request2 awaits replies
Service computes and sends response1, response2
Opponent swaps response1, response2
Client successfully checks MACs, and acts on the swapped responses

MULTI SESSION RPC PROTOCOL

1. $a \rightarrow b : Enc0.ENC\ k_{ae}\ (concat\ s\ k)$
2. $b \rightarrow a : Enc.ENC\ k\ t$

RPC

```
let keygen (a:pri) (b:pri) =
  let k0 = Enc0.GEN() in assume(KeyAB(k0,a,b)); k0  (* for encryption of requests *)

let client (a:pri) (b:pri) (k0:key{KeyAB(k0,a,b)})  s =
  let k= Enc.GEN()   (* for response *)
  assume (Request(a,b,s,k));
  let p = concat s k
  send (Enc0.ENC k0 p);
  recv ( fun msg ->
    if length msg = Enc.ciphersize then
      let res = Enc.DEC k msg
      match res with
      | Some t -> assert (Response(a,b,s,t))
      | None -> ();
      res
```

```
let server a b (k0:key {KeyAB(k0,a,b)})  =
  recv (fun msg ->
   if length msg = Enc0.ciphersize then
    match Enc0.DEC k0 msg with
    | Some sk ->
       let (s,k) = split Enc.keysize sk in
       assert (Request(a,b,s,k));
       let t = service s in
       assume (Response(a,b,s,t));
       send (Enc.ENC k t)
    | None -> ())
```

ADVERSARY INTERFACE

– A ‘trusted’ with message transfer and scheduling

I_{NET}

```
send: bytes -> unit
recv: (bytes -> unit) -> unit
```

I_{NET}^{A}

```
A_check_send: unit -> bytes
A_check_recv: unit -> handle
A_continue_recv: handle -> bytes  -> unit
```

– Uses only unrefined 1st order interface I_{RPC}^{A}:

I_{RPC}^{A}

```
val keygen: principal -> principal -> unit
val client: principal -> principal -> bytes-> unit
val server: principal -> principal -> unit
```

– $C_{RPC} \triangleq \text{RPC} \cdot C_{RPC}^{A}$

C_{RPC}^{A}

```
let keys = ref []
let keygen a b = let k = RPC.keygen() in keys:=((a,b),k) :: !keys
let client a b s = let k = List.assoc !keys (a,b) in RPC.client a b k plain(s); ()
val server a b = let k = List.assoc !keys (a,b) in RPC.server a b k
```

SAMPLE SECURITY THEOREM

If C_{ENC} securely emulate F_{Enc}^{ae} and if $Net \cdot\ C_{RPC}$ is p.p.t. such that $\vdash\ Net \mapsto I_{NET}$, $\vdash\ Net \mapsto I_{NET}^{A}$,

then for any p.p.t. A such that $I_{NET}^{A}, I_{RPC}^{A} \vdash A{:}\,bool$:

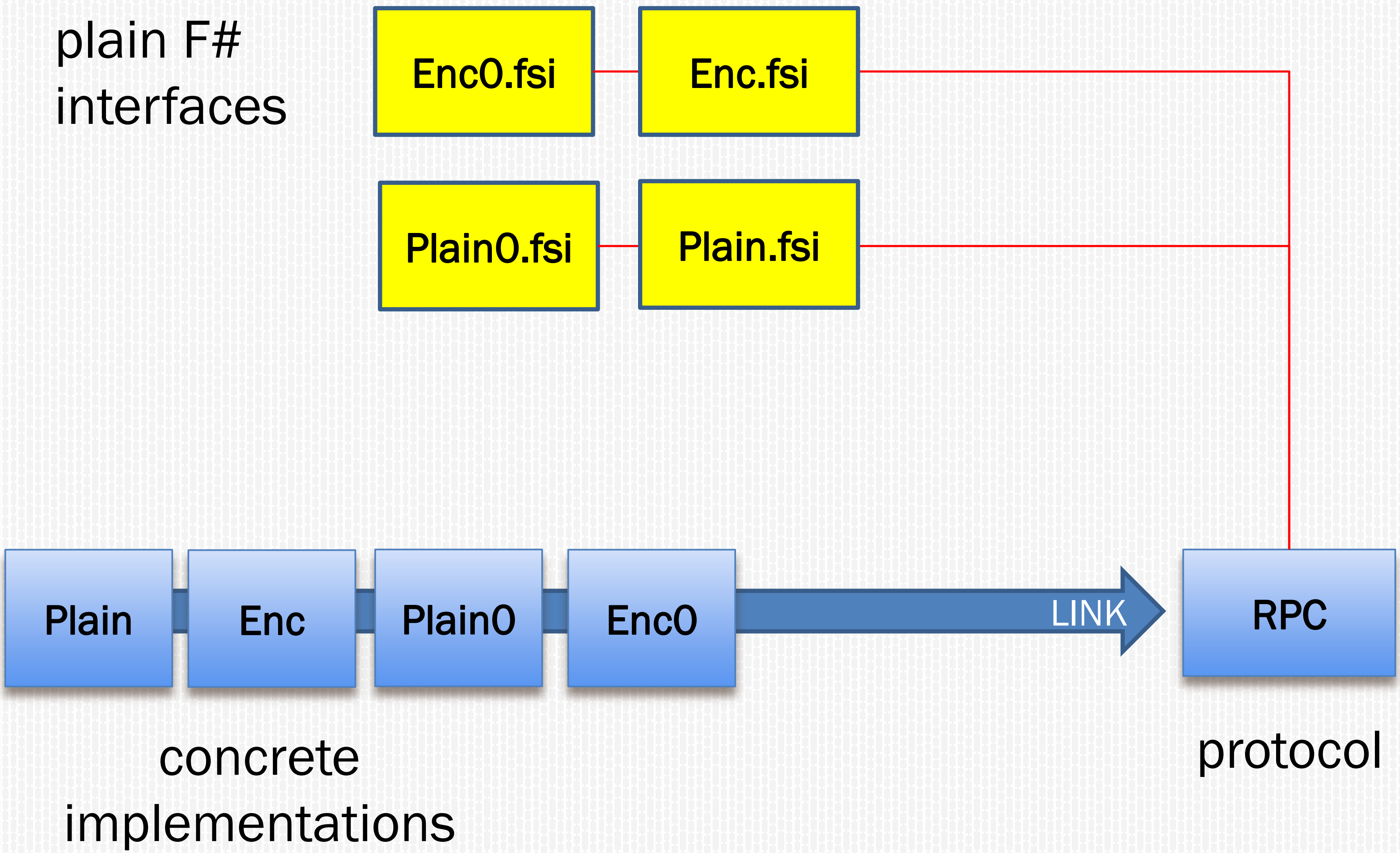
(We abbreviate $A' \triangleq C_{Enc} \cdot P_0 \cdot C_{Enc0} \cdot Net \cdot\ C_{RPC} \cdot A$)

1. The expression $P \cdot A'$ is asymptotically safe
2. $P^0 \cdot A' \approx_\epsilon P^1 \cdot A'$ where

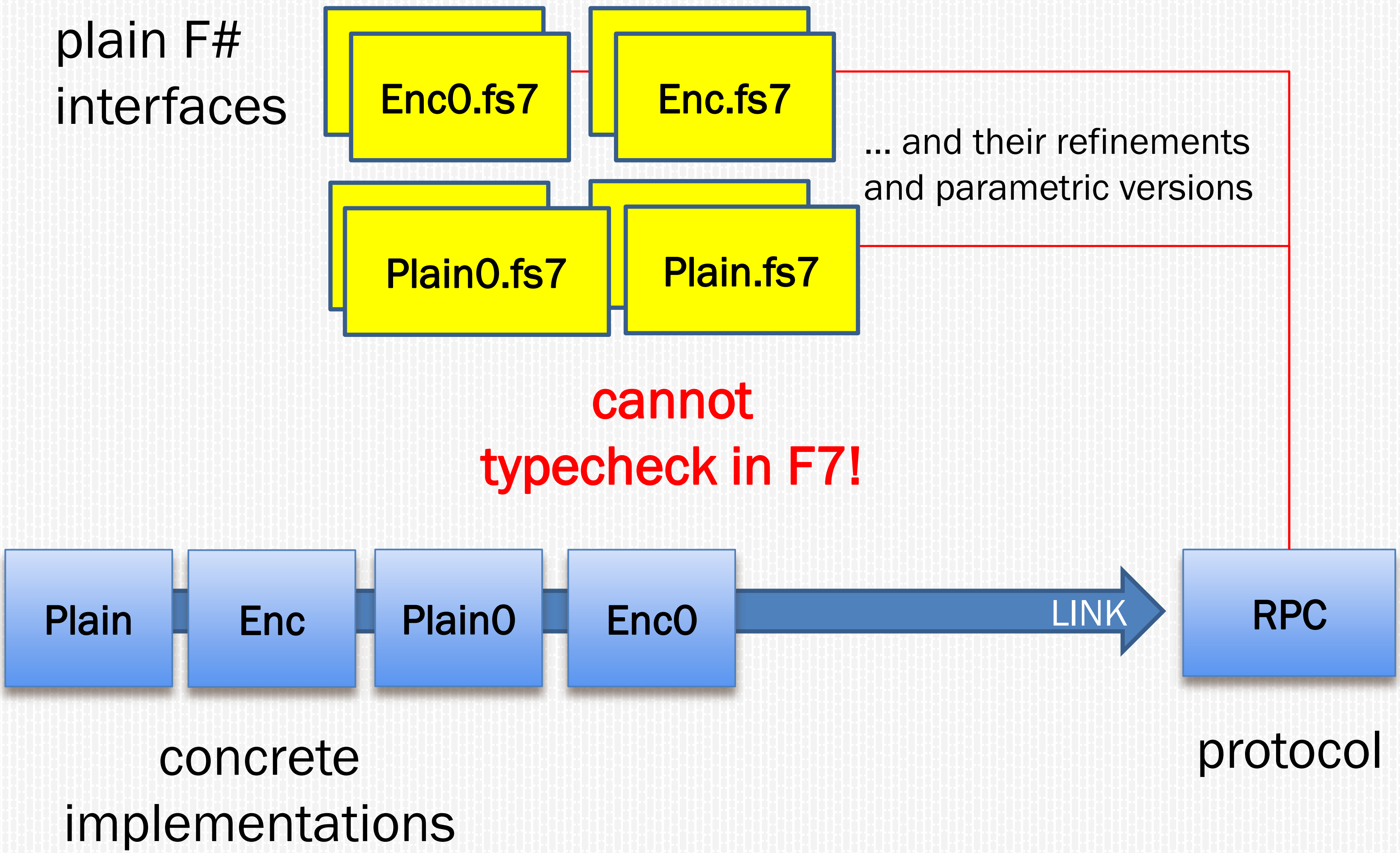
   $\vdash P^0 \mapsto I_{Plain}$ and $\vdash P^1 \mapsto I_{Plain}$

Note, P^0 and P^1 may implement different service functions.

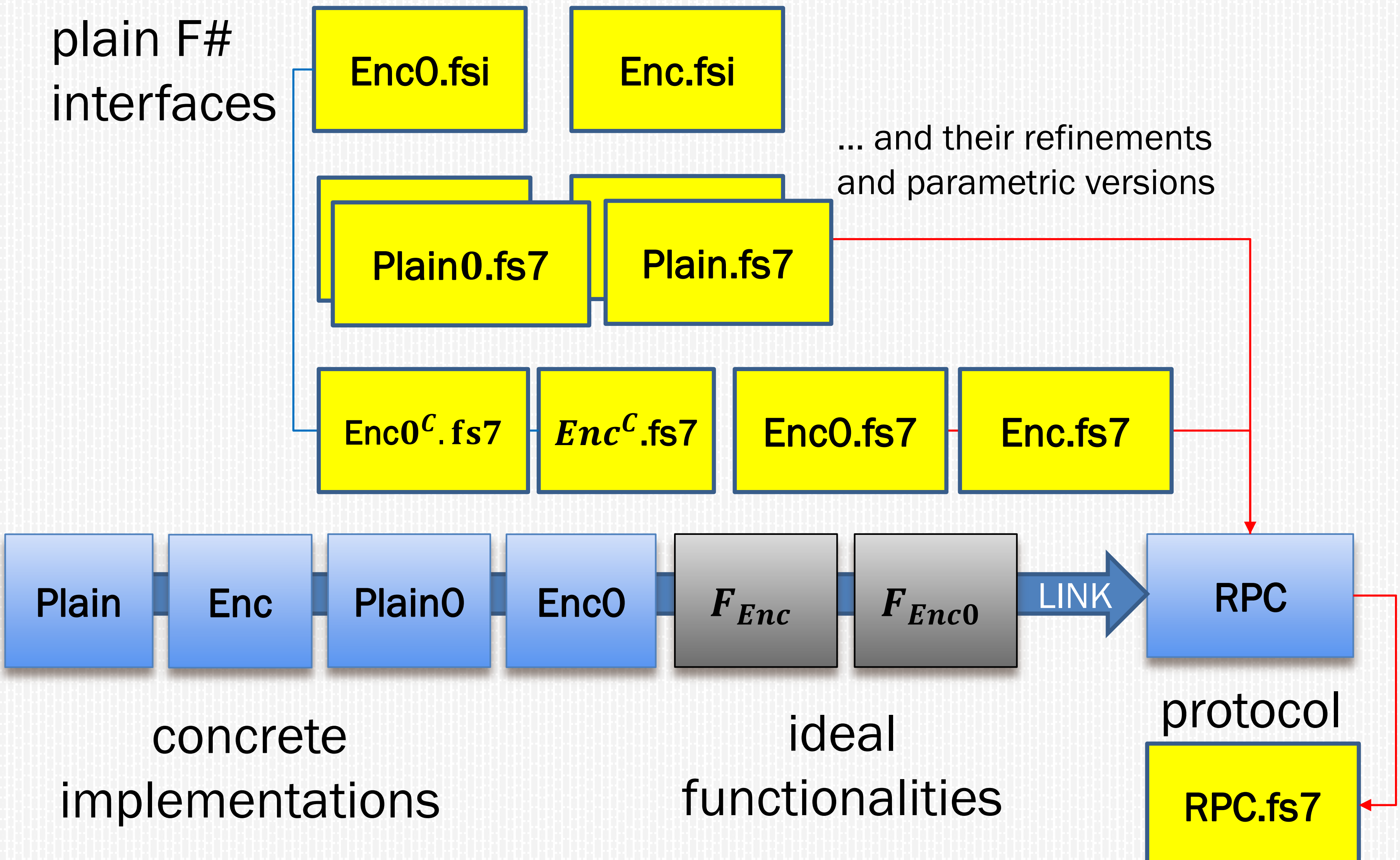
RPC: INTERFACES AND IMPLEMENTATIONS



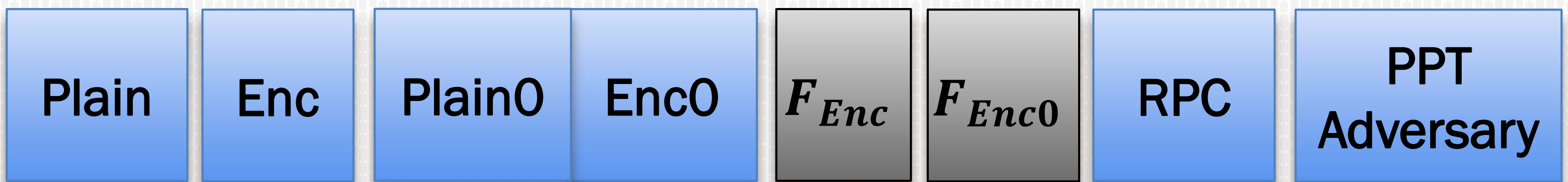
RPC: INTERFACES AND IMPLEMENTATIONS



RPC: INTERFACES AND IMPLEMENTATIONS

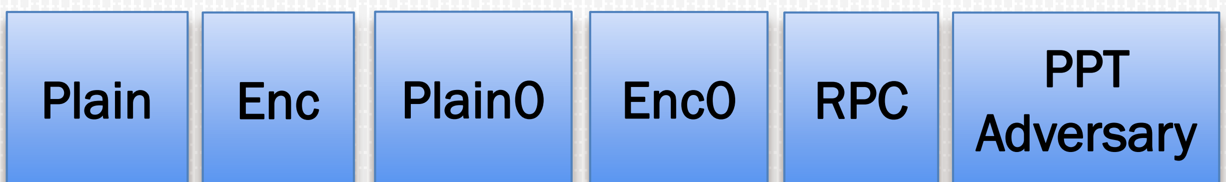


RPC: INTERFACES AND IMPLEMENTATIONS



is always safe (by typing)

is indistinguishable from



is safe too, with overwhelming probability

PROOF SKETCH

To prove:

$$P \cdot C_{Enc} \cdot P_0 \cdot C_{Enc0} \cdot Net \cdot C_{RPC} \cdot A \approx_\epsilon \quad (1)$$
$$P \cdot C_{Enc} \cdot P_0 \cdot C_{Enc0} \cdot F^{ae}_{Enc} \cdot F^{ae}_{Enc0} \cdot Net \cdot C_{RPC} \cdot A \quad (2)$$

Game 0:

$$(1) \approx P_0 \cdot C_{Enc0} \cdot P \cdot C_{ENC} \cdot Net \cdot C_{RPC} \cdot A$$

Typecheck:
$I^{C}_{Plain0}, I^{ae}_{Enc0}, I^{C}_{Plain}, I^{C,ae}_{Enc0}, I_{NET} \vdash C_{RPC} \mapsto I^{A}_{RPC}$

Game 1:

$$\approx_\epsilon P_0 \cdot C_{Enc0} \cdot F^{ae}_{Enc0} \cdot P \cdot C_{Enc} \cdot Net \cdot C_{RPC} \cdot A$$

Game 2:

$$\approx P \cdot C_{Enc} \cdot P_0 \cdot C_{Enc0} \cdot F^{ae}_{Enc0} \cdot Net \cdot C_{RPC} \cdot A$$

Typecheck:
$I^{C}_{Plain}, I^{ae}_{Enc}, I_{Plain0}, I^{ae}_{Enc0}, I_{NET}, \vdash C_{RPC} \mapsto I^{A}_{RPC}$

Game 3

$$\approx_\epsilon P \cdot C_{Enc} \cdot F^{ae}_{Enc} \cdot P_0 \cdot C_{Enc0} \cdot F^{ae}_{Enc0} \cdot Net \cdot C_{RPC} \cdot A \approx (2)$$

AUTHENTICITY BY TYPING

Safety:

- Msg(k,m) is the logical payload of an AE of bytes m with key k
- KeyAB(k,a,b) means k is shared between a and b for this specific protocol

- **assume** $\forall\, a, b, k0, p. KeyAB(k0, a, b) \Rightarrow$ $\text{Enc0}.Msg(k0, p) \;\Leftrightarrow\; \exists k, s. (p \;=\; s \mid k \land Length(s) = plainsize \land Request(a, b, s, k))$
- **assume** $\forall\, a, b, s, k. Request(a, b, s, k) \Rightarrow$ $\forall\, t.\; Enc.Msg(k, t) \Leftrightarrow Response(a, b, s, t)$

SECRECY BY TYPING

Parametricity:

- A "secret module" P_α operates on secrets
- A programs A uses P_α via an interface I_α that gives type α to secrets, but does not directly access their representation.
- Different implementations of I_α are equivalent for A.

Secret Interface: $I_\alpha \triangleq \alpha, x_1 : T_{\alpha,1}, \ldots, x_n : T_{\alpha,n}$ where

$$T_\alpha = \alpha \mid T \to T_\alpha$$

Theorem (Secrecy by Typing).
Let A such that $I_\alpha \vdash A : bool$.
For all pure $\vdash P_\alpha^0 \mapsto I_\alpha$ and $\vdash P_\alpha^1 \mapsto I_\alpha$, we have

$$P_\alpha^0 \cdot A \approx P_\alpha^1 \cdot A.$$

Strong Secrecy:

$$I_{Plain} \vdash C_{Enc} \cdot Net \cdot P_0 \cdot F_{Enc0}^{ae} \cdot C_{Enc} \cdot F_{Enc}^{ae} \cdot C_{RPC} \cdot A$$

CONCLUSION

F7

- Code based analysis through and through
  - verification of programs
  - formal (as proposed by Bellare et al.)
- Efficient
  - We pay only for crypto we need (CPA, AE)
  - Types guarantee that cryptography is used appropriately
- Modular
  - We verify one module at a time.
  - Do cryptographic reasoning at the right place (little overhead)
- Powerful
  - We support trace and indistinguishability properties
  - We can encrypt key

http://research.microsoft.com/~fournet/comp-f7/

- We support different corruption models
- More ideal functionalities: e.g., public-key cryptography, CCA encryption

CONCLUSION

F7

- Code based analysis through and through.
  - Clean and general purpose programming language: ML, F#,
  - General purpose automated program verification tool: F7 refinement types typechecker for F#.
  - We support both
    - formal theorem proving (Coq): (type safety, parametricity)
    - automated protocol verification: (wiring, ordering, spec)
    - manual code-based reasoning: (for justifying abstractions)
  - Combine all three in single language framework

ENCRYPTION

- $\vdash P \mapsto I^{C}_{PLAIN}$ *and* $I^{C}_{PLAIN} <: I_{PLAIN}$
  - $I^{C}_{PLAIN} \vdash C_{ENC} \mapsto I^{C}_{ENC}$
  - $I_{PLAIN}, I^{C}_{ENC} \vdash F_{\mathrm{Enc}} \mapsto I_{ENC}$

- **Theorem (Ideal Functionality for CCA2).** If C_{ENC} is CCA2 secure and A is a p.p.t. expression such that $I^{C}_{PLAIN}, I_{ENC} \vdash A$ then

$$P \cdot C_{ENC} \cdot A \approx_{\epsilon} P \cdot C_{ENC} \cdot F_{ENC} \cdot A$$

- **Theorem (Asymptotic Secrecy).** If C_{ENC} is CCA2 secure and A is a p.p.t. expression such that $I_{PLAIN}, I_{ENC} \vdash A$ then for any two pure P^{b} of I_{PLAIN}

$$P^{0} \cdot C_{ENC} \cdot A \approx_{\epsilon} P^{1} \cdot C_{ENC} \cdot A.$$

MAC

- $\vdash C_{MAC} \mapsto I_{MAC}^{C}$
  $I_{MAC}^{C} \vdash F_{MAC} \mapsto I_{MAC}$

- **Theorem (Ideal Functionality for MAC).** If C_{MAC} is CMA secure and A is a p.p.t. expression such that $I_{MAC} \vdash A$ then

$$C_{MAC} \cdot A \approx_{\epsilon} C_{MAC} \cdot F_{MAC} \cdot A$$

- **Theorem (Asymptotic Safety).** If C_{MAC} is CMA secure and A is a p.p.t. expression such that $I_{MAC} \vdash A\colon bool$ then

$C_{MAC} \cdot A$ is asymptotically safe.

COMPUTATIONAL COMPLEXITY

- Asymptotic notions consider series $(A_\eta)_{\eta \geq 0}$ of expressions indexed by integer constant η.
  - We write A instead of $(A_\eta)_{\eta \geq 0}$
- Closed expression series E is *p.p.t.* when $\exists p \in Poly_\eta. \forall \eta \geq 0. E_\eta$ terminates in at most $p(\eta)$ steps
- Closed first-order functional value is p.p.t. when its runtime is bounded by a polynomial in the size of its parameters.
- Let B be module of such values.
  - Open expression A such that $I \vdash A : T$ is p.p.t. when for every $\vdash B \mapsto I$, the closed expression $B \cdot A$ **is p.p.t.**
  - A module F such that $\mathrm{I} \vdash F \mapsto I_F$ is p.p.t. when, for every $\vdash B \mapsto I$ and p.p.t. expression A such that $I_F \vdash A$, the closed expression $B \cdot F \cdot A$ is p.p.t.