

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



US-CERT Vulnerability Note VU#612636 - Windows Internet Explorer

http://www.kb.cert.org/vuls/id/612636 google app authentication attack

US-CERT Vulnerability Note VU#612636

Home | FAQ | Contact | Privacy Policy

US-CERT
UNITED STATES COMPUTER EMERGENCY READINESS TEAM

Vulnerability Notes Database

Vulnerability Note VU#612636

Google SAML

Overview

The SAML Single Sign-on could have allowed an attacker to impersonate a user.

I. Description

The Security Assertion Markup Language (SAML) authentication data between security packets are called assertions. These assertions are sent to service providers who allow the authentication response to be used to identify the recipient. This is done at other service provider.

View Notes By

- Name
- ID Number
- CVE Name
- Date Public
- Date Published

More technical information: [2.0 Web Browser Single Sign-on](#), [Google Apps whitepaper](#), [lab.it/armando/GoogleSS](#)

Find: assoc Previous Next

OpenSSL

Cryptography and SSL/TLS toolkit

Newsflash | State | Announce | News | ChangeLog

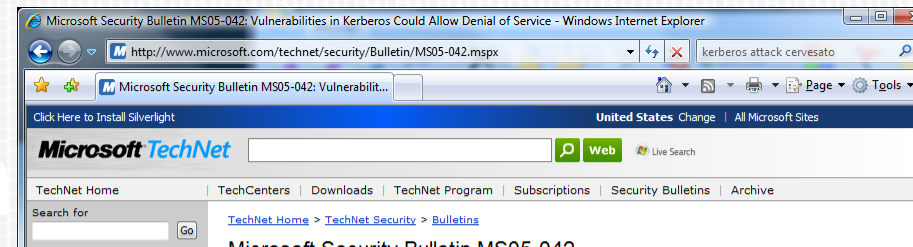
OpenSSL vulnerabilities

This page lists all security vulnerabilities fixed in released versions released on 5th April 2001.

2010

CVE-2010-1633: 1st June 2010

An invalid Return value check in pkey_rsa_verifyrecover recovery fails for RSA keys an uninitialised buffer with an of an error code. This could lead to an information leak.



Microsoft Security Bulletin MS05-042: Vulnerabilities in Kerberos Could Allow Denial of Service - Windows Internet Explorer

http://www.microsoft.com/technet/security/Bulletin/MS05-042.mspx kerberos attack cervasato

Microsoft Security Bulletin MS05-042: Vulnerabil...

Click Here to Install Silverlight

United States Change | All Microsoft Sites

Microsoft TechNet

TechNet Home | TechCenters | Downloads | TechNet Program | Subscriptions | Security Bulletins | Archive

Search for

TechNet Home > TechNet Security > Bulletins

Microsoft Security Bulletin MS05-042



Microsoft Security Advisory (2416728): Vulnerability in ASP.NET Could Allow Information Disclos - Windows Internet Explorer

http://www.microsoft.com/technet/security/advisory/2416728.mspx ASP.NET security alert

Click Here to Install Silverlight

United States Change | All Microsoft Sites

Microsoft | Tech Search Microsoft.com

TechNet Home | TechCenters | Downloads | TechNet Program | Subscriptions | Security Bulletins

Search for

TechNet Home > TechNet Security > Security Advisories

Microsoft Security Advisory (2416728)

Vulnerability in ASP.NET Could Allow Information Disclosure

Published: September 17, 2010 | Updated: September 28, 2010

Version: 2.0

Microsoft has completed the investigation into a public report of this vulnerability. We have issued [MS10-070](#) to address this issue. For more information about this issue, including download links for an available security update, please review [MS10-070](#). The vulnerability addressed is the ASP.NET Padding Oracle Vulnerability - [CVE-2010-3332](#).

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

The screenshot shows a presentation slide with a blue header and footer. The main content area contains a browser window and a search result. The browser window displays a Microsoft Security Advisory (2416728) titled "Vulnerability in ASP.NET Could Allow Information Disclos...". The search result is for "OpenSSL vulnerabilities" and lists a vulnerability fixed in versions of OpenSSL since 0.9.6a, released on 5th April 2001. The search result also mentions "CVE-2010-1633: 1st June 2010" and describes an invalid Return value check in pkey_rsa_verifyrecover.

US-CERT
UNITED STATES COMPUTER EMERGENCY READINESS TEAM

Vulnerability Note VU#612636
Google SAML Single Sign on vulnerability

Overview

Microsoft Security Advisory (2416728): Vulnerability in ASP.NET Could Allow Information Disclos - Windows Internet Explorer
http://www.microsoft.com/technet/security/advisory/2416728.mspx

Find: assoc

OpenSSL
Cryptography and SSL/TLS Toolkit

Newsflash | State | Announce | News | ChangeLog | Vulnerabilities | Internet

Title
FAQ
About
News
Documents
Source
Contribution
Support

OpenSSL vulnerabilities

This page lists all security vulnerabilities fixed in released versions of OpenSSL since 0.9.6a was released on 5th April 2001.

2010

CVE-2010-1633: 1st June 2010

An invalid Return value check in pkey_rsa_verifyrecover was discovered. When verification recovery fails for RSA keys an uninitialised buffer with an undefined length is returned instead

United States Change | All Microsoft Sites

bing Web

loads | TechNet Program | Subscriptions | Security Bulletin

hNet Security > Security Advisories

Security Advisory (2416728)
an ASP.NET Could Allow Information Disclosure
r 17, 2010 | Updated: September 28, 2010

3

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)

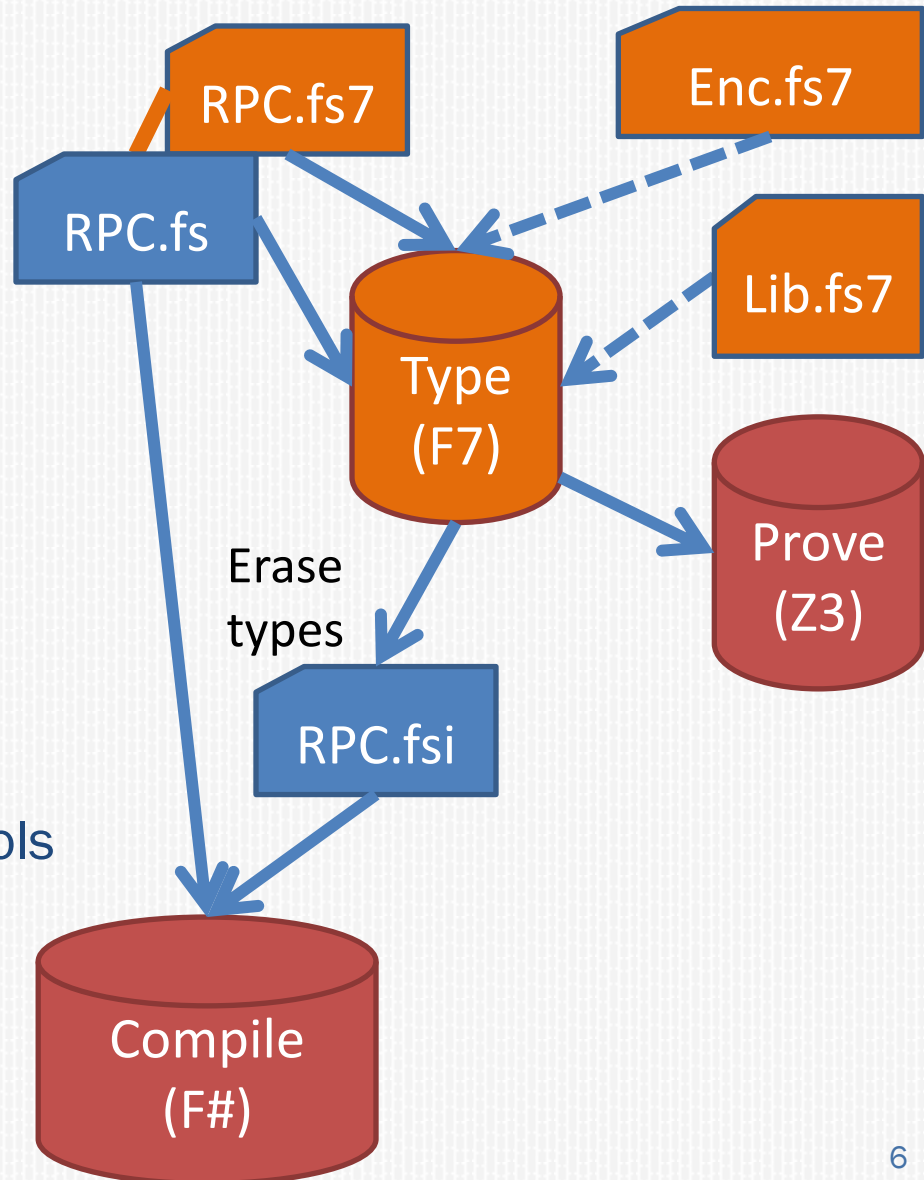
- ◆ Concrete, algorithmic view, using probabilistic polynomial-time machines
- ◆ New formal tools: CryptoVerif, Certicrypt, EasyCrypt

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 $I \vdash Pr \mapsto I_{Pr}$.
 - ♦ Limit ourselves to 1st order interfaces.
 - ♦ Top most attacker interface I_{Pr} unrefined,
 \Rightarrow power of A corresponds to Oracle Turing machine.
- Fair coin tossing primitive with probabilistic semantics $A \rightarrow_p A'$
 $\text{sample} \xrightarrow{\frac{1}{2}} \text{true}, \text{sample} \xrightarrow{\frac{1}{2}} \text{false}$
- A is *asymptotically safe* when the series of probabilities of A_η being unsafe is negligible.
- A^0 and A^1 are *asymptotically indistinguishable*, $A^0 \approx A^1$, 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

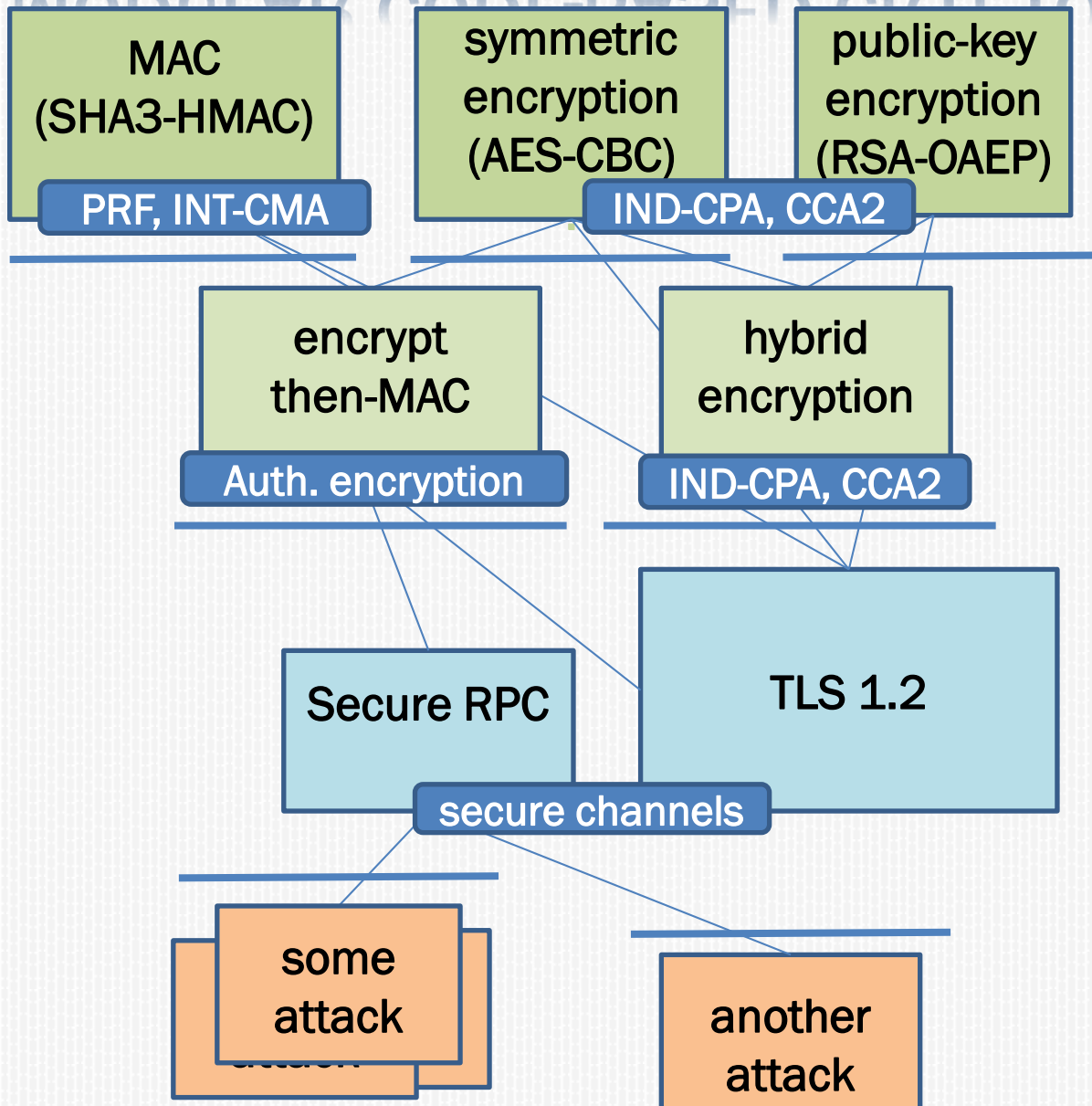
```
let k = GEN ()
let enc x0 x1 =
  let x = xb in
  let c = ENC k x in
```

CPA_b

```
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
```

CTXT

MODULAR CODE-BASED CRYPTO VERIFICATION



cryptographic
primitives

typed interfaces
(security guarantees)

cryptographic
functionalities

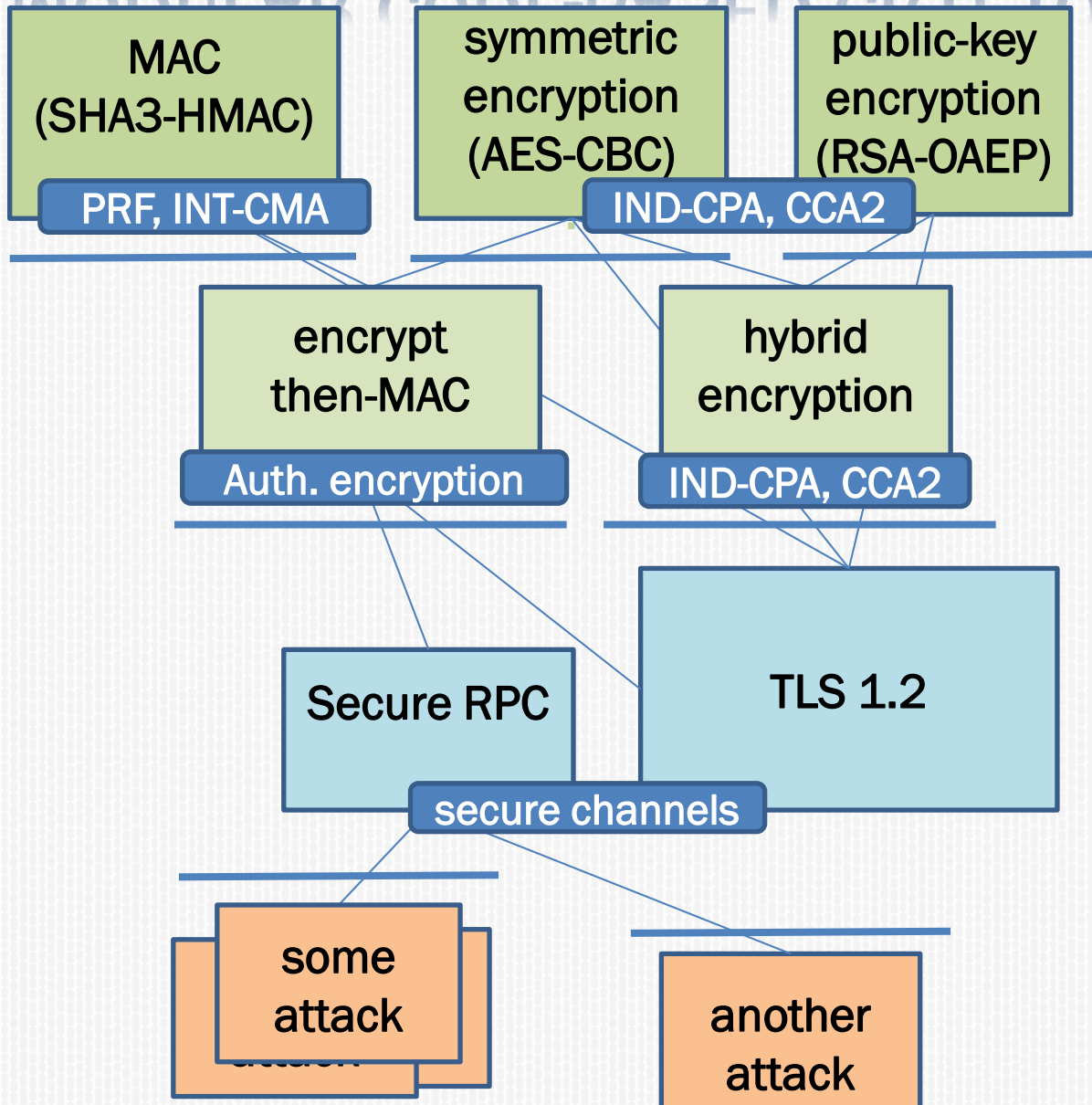
typed interfaces
(security guarantees)

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
```

This interface says nothing
about security of Enc

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

AUTHENTICATED ENCRYPTION

keys are abstract

```
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
```

Ciphertext has fixed
size

ideal F7
interface

Msg is specified by
protocols using Enc

“All decrypted
messages
have been
encrypted”

```
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
```

sample
protocol
using
Auth Enc

AUTHENTICATED ENCRYPTION

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

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

- ◆ Refinements model authenticity properties
- ◆ Abstraction in I_{PLAIN} models that other outputs of F_{ENC} , in particular ciphertexts, are independent of abstractly typed 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_{Plain}^C

```
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
```

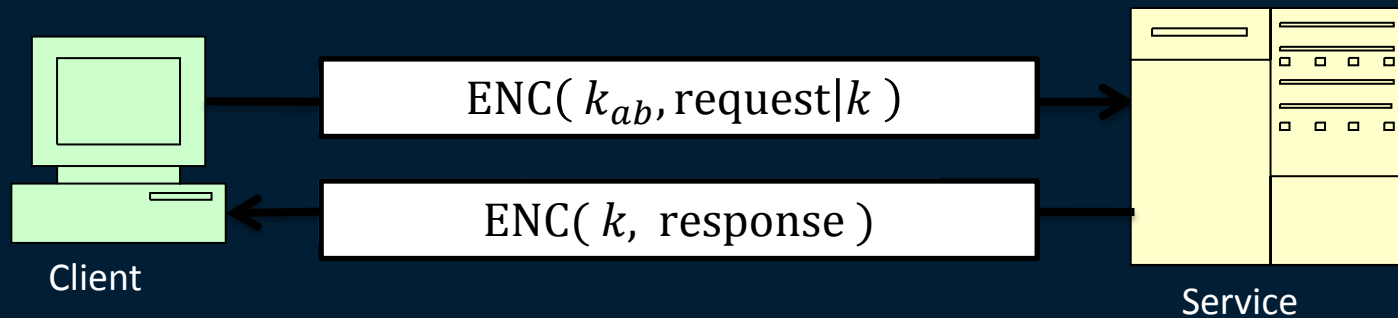
I_{Enc}^{ae}

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 $I_{\text{Plain}} \vdash C_{\text{Enc}} \cdot F_{\text{Enc}}^{ae} \mapsto I_{\text{Enc}}^{ae}$
- Prove that $\forall p.p.t. P, A, \text{s.t.}, \vdash P \mapsto I_{\text{Plain}}^c$ and $I_{\text{Plain}}^c, I_{\text{Enc}}^{ae} \vdash A$.
$$P \cdot C_{\text{Enc}} \cdot A \approx_{\epsilon} P \cdot C_{\text{Enc}} \cdot F_{\text{Enc}}^{ae} \cdot A$$



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

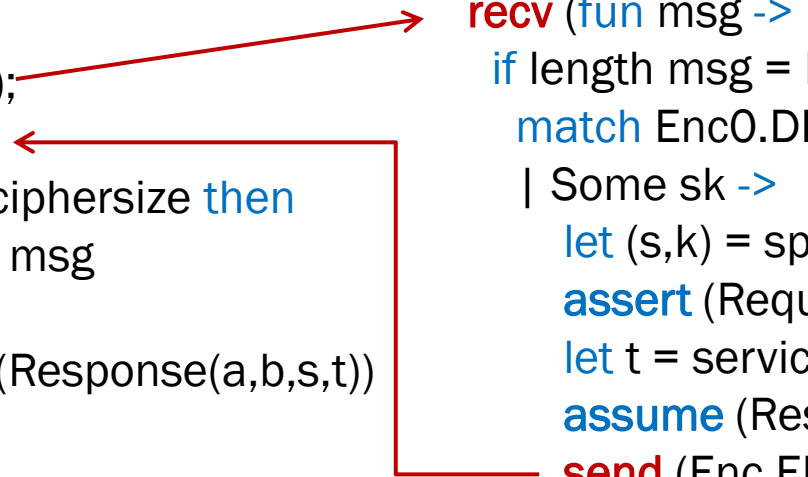
Opponent swaps response1, response2

Client successfully checks MACs, and acts on the swapped responses

MULTI SESSION RPC PROTOCOL

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

```
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.keysizesize 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 -> ())
```



RPC

ADVERSARY INTERFACE

- A ‘trusted’ with message transfer and scheduling

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

I_{NET}

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

I_{NET}^A

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

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

I_{RPC}^A

- $C_{RPC} \triangleq RPC \cdot C_{RPC}^A$

```
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
```

C_{RPC}^A

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 $I_{NET}^A, I_{RPC}^A \vdash A: bool$:

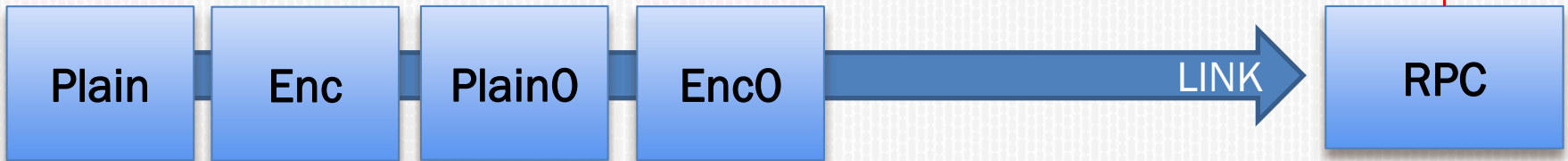
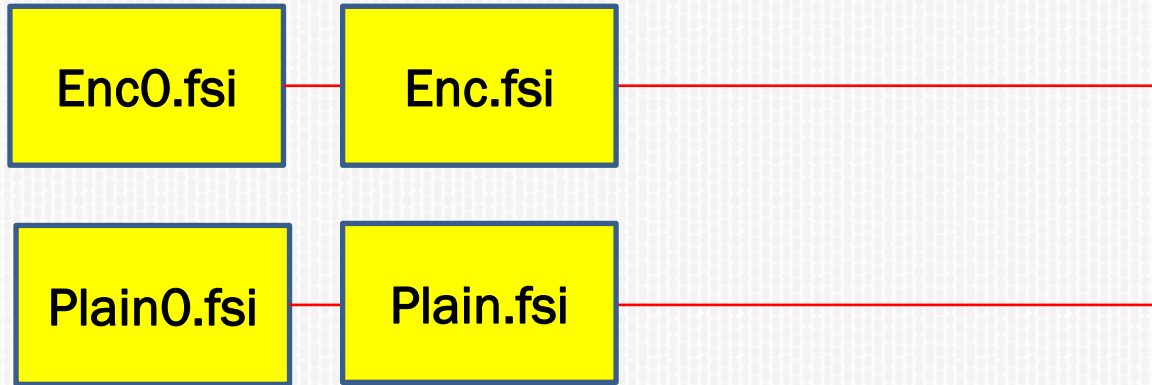
(We abbreviate $A' \triangleq C_{Enc} \cdot P_0 \cdot C_{Enc0} \cdot Net \cdot C_{RPC} \cdot A$)

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

plain F#
interfaces

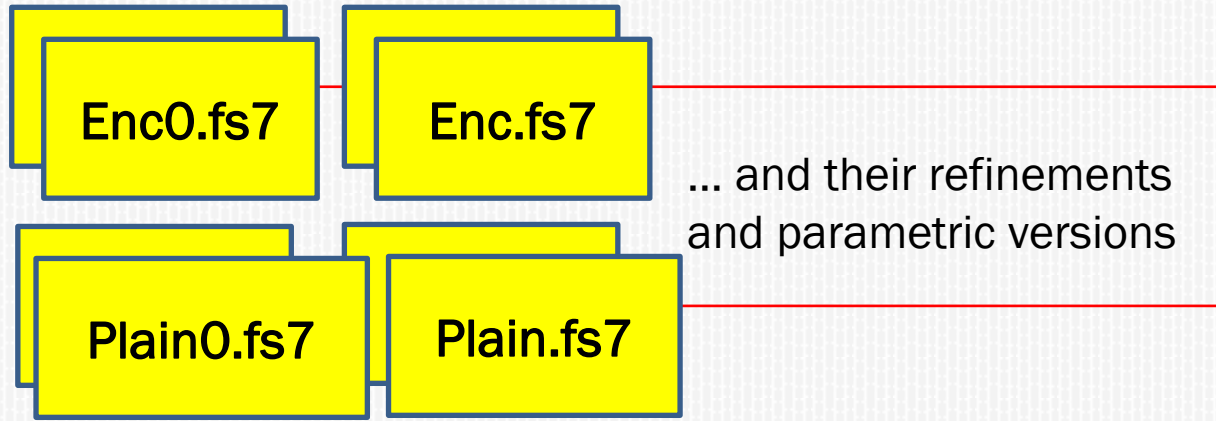


concrete
implementations

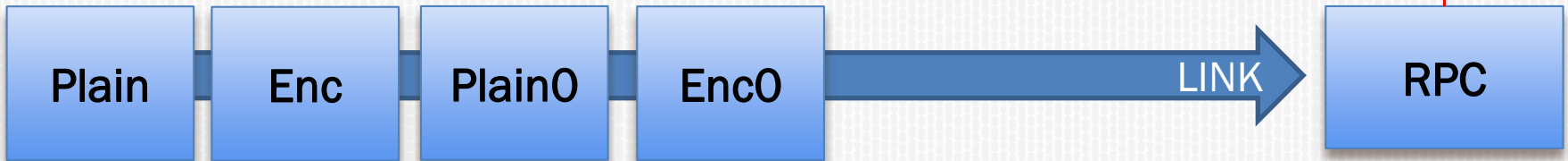
protocol

RPC: INTERFACES AND IMPLEMENTATIONS

plain F#
interfaces



**cannot
typecheck in F7!**



concrete
implementations

protocol

RPC: INTERFACES AND IMPLEMENTATIONS

plain F#
interfaces

Enc0.fsi

Enc.fsi

... and their refinements
and parametric versions

Plain0.fs7

Plain.fs7

Enc0^c.fs7

Enc^c.fs7

Enc0.fs7

Enc.fs7

Plain

Enc

Plain0

Enc0

F_{Enc}

F_{Enc0}

LINK

RPC

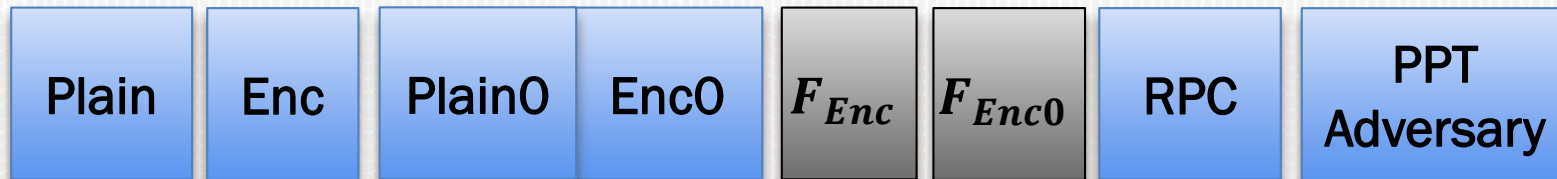
concrete
implementations

ideal
functionalities

protocol

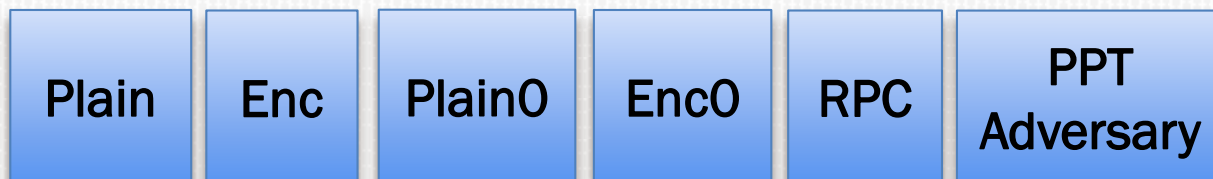
RPC.fs7

RPC: INTERFACES AND IMPLEMENTATIONS



is always safe (by typing)

is indistinguishable from



is safe too, with overwhelming probability

PROOF SKETCH

To prove:

$$P \cdot C_{Enc} \cdot P_0 \cdot C_{Enc0} \cdot Net \cdot C_{RPC} \cdot A \approx_{\epsilon} \quad (1)$$

$$P \cdot C_{Enc} \cdot P_0 \cdot C_{Enc0} \cdot F_{Enc}^{ae} \cdot F_{Enc0}^{ae} \cdot Net \cdot C_{RPC} \cdot A \quad (2)$$

Game 0:

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

Typecheck:

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

Game 1:

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

Game 2:

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

Typecheck:

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

Game 3

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

AUTHENTICITY BY TYPING

Safety:

- $\text{Msg}(k,m)$ is the logical payload of an AE of bytes m with key k
- $\text{KeyAB}(k,a,b)$ means k is shared between a and b for this specific protocol
- **assume** $\forall a, b, k_0, p. \text{KeyAB}(k_0, a, b) \Rightarrow$
 $\text{Enc}_0. \text{Msg}(k_0, p) \Leftrightarrow \exists k, s. (p = s \mid k \wedge$
 $\text{Length}(s) = \text{plainsize} \wedge \text{Request}(a, b, s, k))$
- **assume** $\forall a, b, s, k. \text{Request}(a, b, s, k) \Rightarrow$
 $\forall t. \text{Enc}. \text{Msg}(k, t) \Leftrightarrow \text{Response}(a, b, s, t)$

SECRECY BY TYPING

Parametricity:

- A "secret module" P_α operates on secrets
- A program 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: $I_\alpha \triangleq \alpha, x_1:T_{\alpha,1}, \dots, x_n:T_{\alpha,n}$ where

$$T_\alpha = \alpha \mid T \rightarrow T_\alpha$$

Theorem (Secrecy by Typing).

Let A such that $I_\alpha \vdash A: \text{bool}$.

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:

$$I_{\text{Plain}} \vdash C_{\text{Enc}} \cdot \text{Net} \cdot P_0 \cdot F_{\text{Enc}0}^{ae} \cdot C_{\text{Enc}} \cdot F_{\text{Enc}}^{ae} \cdot C_{\text{RPC}} \cdot A$$

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

– $\vdash P \mapsto I_{PLAIN}^C$ and $I_{PLAIN}^C \leq I_{PLAIN}$

♦ $I_{PLAIN}^C \vdash C_{ENC} \mapsto I_{ENC}^C$

♦ $I_{PLAIN}, I_{ENC}^C \vdash F_{ENC} \mapsto I_{ENC}$

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

$$\begin{aligned} - & \vdash C_{MAC} \mapsto I_{MAC}^C \\ & I_{MAC}^C \vdash F_{MAC} \mapsto I_{MAC} \end{aligned}$$

- **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 $I_{MAC} \vdash A: bool$ then

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

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 $I \vdash F \mapsto I_F$ 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.