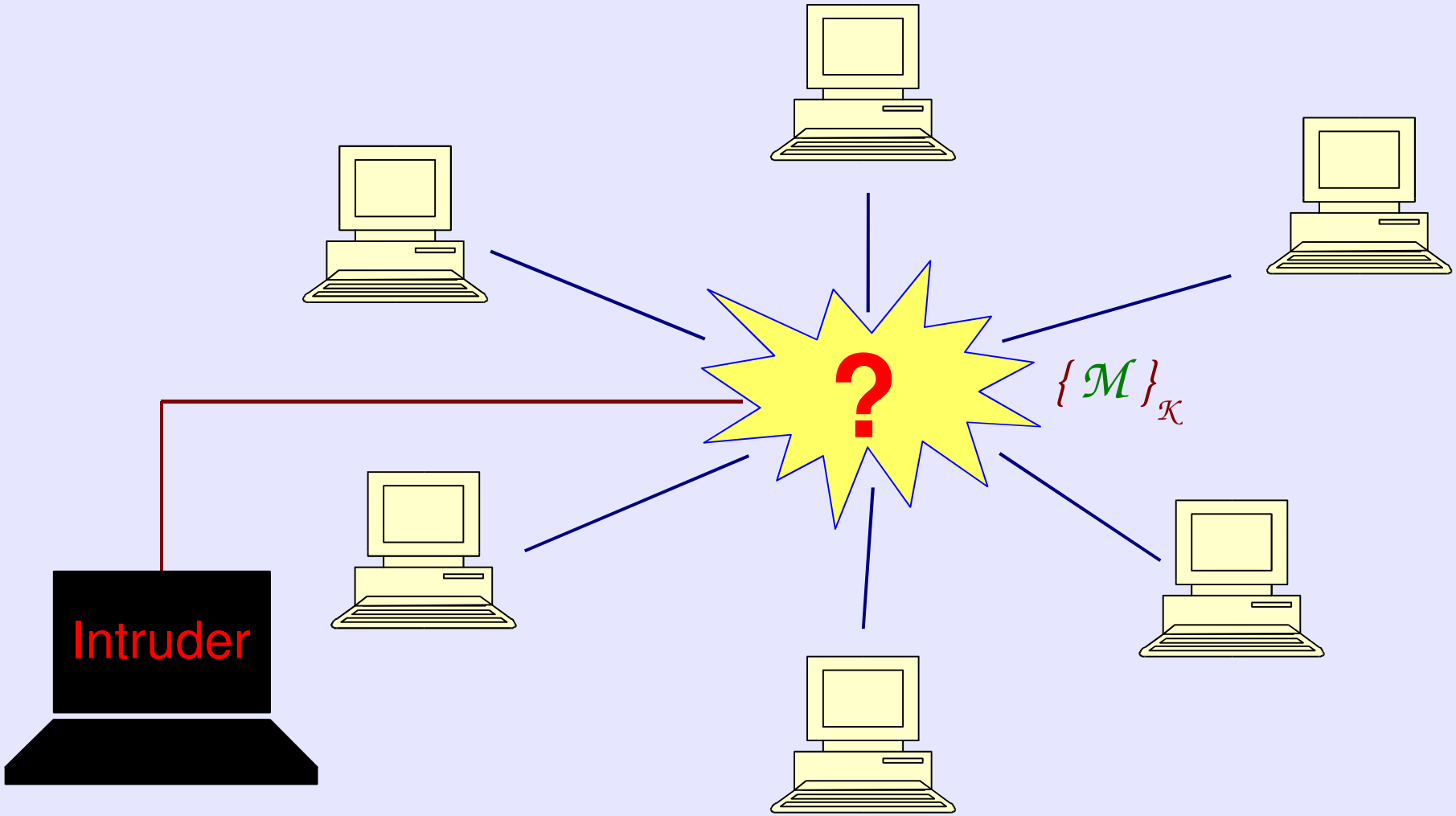
Authenticity by Tagging and Typing

Michele Bugliesi
Riccardo Focardi
Matteo Maffei

Ca' Foscari University
Computer Science Department
Venice (Italy)

Myths Meeting
Venice, 14-6-2003

Data need to be protected



Security Protocols

- Security (cryptographic) protocols exchange encrypted data to achieve security goals
  - Secrecy, non-repudiation, authentication ...
- Security protocols are “small” but ...
  - many attacks reported in the recent literature
- No need to break cryptography: the intruder can combine, duplicate, intercept encrypted data, breaking the protocol logic

Two simple protocols

Two protocols...safe when isolated


A B

n <-

-> $\{A,n,M\}_{k_{AB}}$


A B

n <-

-> $\{B,n,M\}_{k_{AB}}$

Two simple protocols

...unsafe when concurrently executed:

A B

-> n

$\{B,n,M\}_{k_{AB}}$ <-

---

n <-

-> $\{B,n,M\}_{k_{AB}}$

I(A) B

n <-

Two simple protocols

...unsafe when concurrently executed:


A B

-> n

$\{B,n,M\}_{k_{AB}}$ <-

n <-

-> $\{B,n,M\}_{k_{AB}}$


I(A) B

n <-

-> n

Two simple protocols

...unsafe when concurrently executed:

A B

-> n

$\{B,n,M\}_{k_{AB}}$ <-

n <-

-> $\{B,n,M\}_{k_{AB}}$

I(A) B

n <-

-> n

$\{B,n,M\}_{k_{AB}}$ <-

Two simple protocols

...unsafe when concurrently executed:


A B

-> n

$\{B,n,M\}_{k_{AB}}$ <-

n <-

-> $\{B,n,M\}_{k_{AB}}$


I(A) B

n <-

-> n

$\{B,n,M\}_{k_{AB}}$ <-

-> $\{B,n,M\}_{k_{AB}}$

Tagging solves Ambiguity

We tag each significant ciphertext component:

A	B
-> n	
$\{B,n,M\}_k$ <-	$\{Id(B),Claim(n),Auth(M)\}_k$
n <-	
-> $\{B,n,M\}_k$	$\{Id(B),Verif(n),Auth(M)\}_k$

ρ-spi calculus

S::= sequential processes

in(M_1,...,M_m).S

out(M_1,...,M_m).S

encrypt $\{M_1,...,M_m\}_k$ as x.S

decrypt x as $\{M_1,...,M_m\}_k$.S

new(n).S

run(A,B,M).S

commit(A,B,M).S

I,J range over trusted principals (A,B) and enemies (E)

M::= terms

n,C(n)	names
x,C(x)	variables
Pub(n)/Priv(n)	key-pair

C ranges over tags

P::= processes

I ▷S	sequential component
I ▷!S	replicated ...
P|Q	parallel
let k=key(I,J).P	sym-key
let k=key(I).P	asym-key

The Enemy

- Processes do not synchronize directly: instead, they may receive in input an arbitrary message known by the environment
- The environment knows:
  - all the messages sent on the unique channel and all the public keys
  - its own private keys and may share some long-term keys with trusted principals
  - ciphertexts created by its knowledge
  - messages inside ciphertexts whose decryption key is known

Authentication by Correspondence Assertions

Safety definition:

- for a trace *s*: *a* trace *s* is safe if *every* commit (A,B,M) in s is preceded by a *distinct* run(B,A,M)

commit(A,B,M)	unsafe
run(A,B,M)::run(A,B,M)::commit(B,A,M)	safe
run(A,B,M)::commit(A,B,M) bb	unsafe

- for a process P: a process P is safe if every trace generated by P is safe

Types

Types:

- $key_{sym}(I,J)$
- $key_{priv}(I)$
- $key_{pub}(I)$
- Un

Effects:

- in(M)
- dec $\{M_1,\ldots,M_n\}_k$
- fresh(n)
- run(I,J,M)

Subsumption (Enemy Keys):

$key_{sym}(E,J)$, $key_{priv}(E)$, $key_{pub}(E)$ <: Un

Main Judgement

$\Gamma \vdash P:e$: "P can be typed under the hypotheses expressed by the effect e"

Types

Types:

- $key_{sym}(I,J)$
- $key_{priv}(I)$
- $key_{pub}(I)$
- Un

Effects:

- in(M)
- dec $\{M_1,\ldots,M_n\}_k$
- fresh(n)
- run(I,J,M)

Main Judgement

$\Gamma \vdash P:e$: "P can be typed under the hypotheses expressed by the effect e"

An example (Authentication)

$$\frac{\Gamma \vdash \mathcal{S}:e \quad dec\{Id(B),Verif(n),Aut(M)\}_k \in e \quad \Gamma \vdash k:sym\text{-}key(A,B)}{\Gamma \vdash commit(B,A,M).\mathcal{S}:e+fresh(n)}$$

An example

n <-

-> $\{B,n,m\}_k$

let k=sym-key(A,B).(A▷!Res(A,B,k) | B▷ ! In(B,A,k))

Res(A,B,k)=

$\Gamma = \emptyset$

in(x)

e = []

new(m:Un).

run(A,B,m).

encrypt$\{Id(B),Verif(x),Auth(m)\}_k$ as z.

out(z)

An example

n <-

-> $\{B,n,m\}_k$

let k=sym-key(A,B).(A▷!Res(A,B,k) | B▷!In(B,A,k))

Res(A,B,k)=

in(x)
new(m:Un).
run(A,B,m).
encrypt$\{Id(B),Verif(x),Auth(m)\}_k$ as z.
out(z)

Γ =k:key_{sym}(A,B)

e=[]

An example

$n \quad <-$

$-> \quad \{B,n,m\}_k$

let k=sym-key(A,B).(A▷!Res(A,B,k) | B▷ ! In(B,A,k))

Res(A,B,k)=

in(x)
new(m:Un).
run(A,B,m).
encrypt$\{Id(B),Verif(x),Auth(m)\}_k$ as z.
out(z)

Γ=k:key_{sym}(A,B),x:Un
e= [in(x)]

An example

$n \quad <-$

$-> \quad \{B,n,m\}_k$

let k=sym-key(A,B).(A▷!Res(A,B,k) | B▷ ! In(B,A,k))

Res(A,B,k)=

in(x)
new(m:Un).
run(A,B,m).
encrypt$\{Id(B),Verif(x),Auth(m)\}_k$ as z.
out(z)

Γ=k:key_{sym}(A,B),x:Un,m:Un
e= [in(x)]

An example

n <-

-> $\{B,n,m\}_k$

let k=sym-key(A,B).(A▷!Res(A,B,k) | B▷ ! In(B,A,k))

Res(A,B,k)=

in(x)
new(m:Un).
run(A,B,m).
encrypt$\{Id(B),Verif(x),Auth(m)\}_k$ as z.
out(z)

Γ=k:key_{sym}(A,B),x:Un,m:Un
e= [in(x),run(A,B,m)]

An example

n <-

-> $\{B,n,m\}_k$

let k=sym-key(A,B).(A▷!Res(A,B,k) | B▷ ! In(B,A,k))

Res(A,B,k)=

in(x)
new(m:Un).
run(A,B,m).
encrypt$\{Id(B),Verif(x),Auth(m)\}_k$ as z.
out(z)

Γ=k:key_{sym}(A,B),x:Un,m:Un,z:Un
e= [in(x)]

An example

$n \quad <-$

$-> \quad \{B,n,m\}_k$

let k=sym-key(A,B).(A▷!Res(A,B,k) | B▷ ! In(B,A,k))

In(B,A,k)=

$\Gamma = \varnothing$

new(n:Un).

e = []

out(n).

in(z).

decrypt z as $\{Id(B),Verif(n),Auth(x)\}_k$.

commit(B,A,x)

An example

n <-

-> $\{B,n,m\}_k$

let k=sym-key(A,B).(A▷!Res(A,B,k) | B▷!In(B,A,k))

In(B,A,k)=

new(n:Un).
out(n).
in(z).
decrypt z as $\{Id(B),Verif(n),Auth(x)\}_k$.
commit(B,A,x)

$\Gamma = k:key_{sym}(A,B)$

e = []

An example

$n \quad <-$

$-> \quad \{B,n,m\}_k$

let k=sym-key(A,B).(A▷!Res(A,B,k) | B▷ ! In(B,A,k))

In(B,A,k)=

new(n:Un).
out(n).
in(z).
decrypt z as $\{Id(B),Verif(n),Auth(x)\}_k$.
commit(B,A,x)

Γ=k:key_{sym}(A,B),n:Un
e = [fresh(n)]

An example

n <-

-> $\{B,n,m\}_k$

let k=sym-key(A,B).(A▷!Res(A,B,k) | B▷ ! In(B,A,k))

In(B,A,k)=

new(n:Un).
out(n).
in(z).
decrypt z as $\{Id(B),Verif(n),Auth(x)\}_k$.
commit(B,A,x)

Γ=k:key_{sym}(A,B),n:Un
e = [fresh(n)]

An example

n <-

-> $\{B,n,m\}_k$

let k=sym-key(A,B).(A▷!Res(A,B,k) | B▷ ! In(B,A,k))

In(B,A,k)=

new(n:Un).
out(n).
in(z).
decrypt z as $\{Id(B),Verif(n),Auth(x)\}_k$.
commit(B,A,x)

Γ=k:key_{sym}(A,B),n:Un,z:Un

e = [fresh(n),in(z)]

An example

n <-

-> $\{B,n,m\}_k$

let k=sym-key(A,B).(A▷!Res(A,B,k) | B▷ ! In(B,A,k))

In(B,A,k)=

new(n:Un).
out(n).
in(z).
decrypt z as $\{Id(B),Verif(n),Auth(x)\}_k$.
commit(B,A,x)

Γ=k:key_{sym}(A,B),n:Un,z:Un

e = [fresh(n),in(z),
dec $\{Id(B),Verif(n),Auth(m)\}_k$]

An example

n <-

-> $\{B,n,m\}_k$

let k=sym-key(A,B).(A▷!Res(A,B,k) | B▷ ! In(B,A,k))

In(B,A,k)=

new(n:Un).
out(n).
in(z).
decrypt z as $\{Id(B),Verif(n),Auth(x)\}_k$.
commit(B,A,x)

Γ=k:key_{sym}(A,B),n:Un,z:Un

e = [in(z), dec $\{Id(B),Verif(n),Auth(m)\}_k$]

Results

A system is formalized as the parallel composition of, possibly replicated, sequential components

$$P = keys_{\{k_1,\ldots,k_n\}}.(I_1 \triangleright !S_1 \mid \ldots \mid I_m \triangleright !S_m)$$

The typing of each sequential component is local and the anysis fully compositional:

- *Theorem(Safety): If* $\emptyset \vdash P : []$ then

  every trace generated by P is safe

- *Theorem(Strong Compositionality):* $\emptyset \vdash P : []$ iff

  $\emptyset \vdash keys_{\{k_1,\ldots,k_n\}}.I_i \triangleright !S_i : []$, for every i in [1,n]

Conclusion and Future Work

What we have done:

- a uniform, role-based tagging of message components in authentication protocols
- a type-and effect system for analyzing authentication
- The analysis if fully compositional and requires limited human effort.

Work in progress

- develop a tool providing both tags and type inference

To be done:

- study a formal relationship between our results, the type system by Gordon and Jeffrey and the Strand Spaces approach by Guttman and others.

Related Work

Types for Secrecy

- [Abadi, JACM 1999]
- [Abadi-Blanchet, FoSSaCS'01, POPL'02]

Control-Flow analysis for authentication of origin:

- [Bodei-Buckholtz-Degano -F. Nielson-H. Riis Nielson, CSFW'03]

Types for Authentication

- [Gordon-Jeffrey, CSFW'01-02]

  "These are weird types; you would not want to give these types to the average programmer" (Alan Jeffrey)

With respect to the types by Gordon and Jeffrey

- Protocols proved safe even in presence of enemies provided with "trusted" keys
- Limited human effort
- Tags, instead of types, convey the authentication guarantees
- Compositionality: trusted principals share only keys, not effects describing their behaviour
- Future work: trying to compare the two analyses and to infer their types and effects

THANK YOU