Proofs of Security Protocols

Symbolic Methods and Powerful Attackers

Charlie Jacomme,
supervised by Hubert Comon and Steve Kremer

June 30th, 2021

Introduction

Introduction?

Do I really need to introduce security and privacy?

Introduction

One of the biggest lesson of my thesis

Introduction

One of the biggest lesson of my thesis

.

Privacy matters

Privacy

Many people NEED privacy to live:

- Homosexuality is a crime in 69 countries.
- Citizens in authoritarian countries (journalist, political opponents).
- Discrimination (origins, health, religion,. . .) for loans, health insurances, employment. . .
- Uighurs currently tracked in China.

Privacy

Many people NEED privacy to live:

- Homosexuality is a crime in 69 countries.
- Citizens in authoritarian countries (journalist, political opponents).
- Discrimination (origins, health, religion,. . .) for loans, health insurances, employment. . .
- Uighurs currently tracked in China.

But what if I have nothing to hide?

Nothing to hide. . .

1. If we don't have **privacy**, people that need it can't have it.

Nothing to hide...

1. If we don't have **privacy**, people that need it can't have it.
2. Most people actually have something to hide.
   - Should my boss know what I do on my free time? Or what is the global income of my household?
   - Should my government know my political opinions?
   - Should my mail provider know my illness?

Nothing to hide. . .

1. If we don't have **privacy**, people that need it can't have it.
2. Most people actually have something to hide.
   - Should my boss know what I do on my free time? Or what is the global income of my household?
   - Should my government know my political opinions?
   - Should my mail provider know my illness?
3. The simple fact of being watched changes unconsciously our behaviour.
   - Philosophical and sociological theories (Foucault, Deleuze, Guattari,. . .), and fictional examples (Orwell, Damosio,. . .)

Guarantees

Security and Privacy Matter !

Security and Privacy Matter !

For each possible use case, we should know exactly who can access what, whether it is a stranger, a government or a corporation.

Guarantees

Security and Privacy Matter !

Which guarantees, for which attacker?

Security and Privacy Matter !

Which formal guarantees, for which attacker?

The difficulty

Protocols

Primitives Protocols

x^2 ⇄

The difficulty

Implementation | Primitives | Protocols

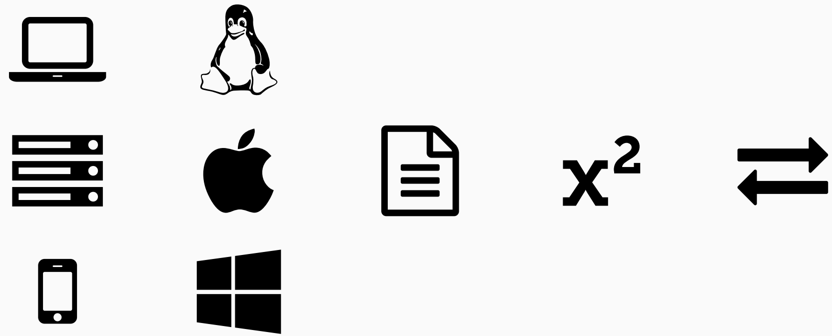
x^2

The difficulty

OS Implementation Primitives Protocols

The difficulty

Hardware OS Implementation Primitives Protocols



The difficulty

Hardware | OS | Implementation | Primitives | Protocols | Users

The difficulty

If any link of the chain is broken, everything is.

Hardware	OS	Implementation	Primitives	Protocols	Users

The difficulty

Hardware | OS | Implementation | Primitives | Protocols | Users

Second difficulty - How to model the attacker ?

Symbolic Model VS Computational Model

Second difficulty - How to model the attacker ?

Symbolic Model VS Computational Model

Symbolic Model	Computational Model
adversary = fixed set of possible actions	*adversary = any program*

Symbolic Model VS Computational Model

adversary = fixed set of possible actions

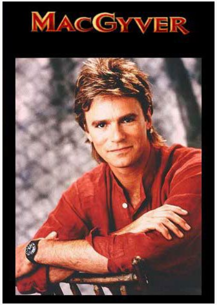
adversary = any program

Second difficulty - How to model the attacker ?

Symbolic Model VS Computational Model

adversary = fixed set of possible actions

adversary = any program



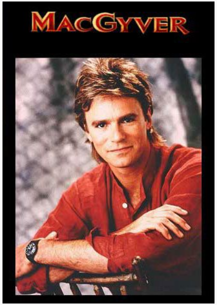
Second difficulty - How to model the attacker ?

Symbolic Model VS Computational Model

adversary = fixed set of possible actions

Automated proofs - strong assumptions

adversary = any program

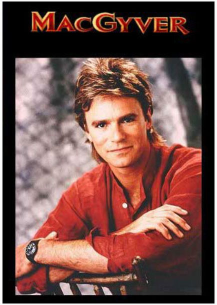


Symbolic Model VS Computational Model

adversary = fixed set of possible actions

Automated proofs - strong assumptions

adversary = any program



Hard automation - strong guarantees

My lines of work

The core of my PhD
Make it easier to prove protocols against attackers as powerful as possible.

My lines of work

The core of my PhD
Make it easier to prove protocols against attackers as powerful as possible.

- Make the symbolic model more precise (detailed threat models);

My lines of work

The core of my PhD
Make it easier to prove protocols against attackers as powerful as possible.

- Make the symbolic model more precise (detailed threat models);
- enable proofs of compound protocols in the computational model:
  - compositional proofs,
  - mechanization,
  - proof automation.

Summary of contributions - outline

1. a methodology to analyze protocols in the symbolic model, but making the attacker as strong as possible, with a case study on multi-factor authentication;

Summary of contributions - outline

1. a methodology to analyze protocols in the symbolic model, but making the attacker as strong as possible, with a case study on multi-factor authentication;

2. a prototype of a mechanized prover in the BC logic;

Summary of contributions - outline

1. a methodology to analyze protocols in the symbolic model, but making the attacker as strong as possible, with a case study on multi-factor authentication;

2. a prototype of a mechanized prover in the BC logic;

3. composition results to allow modular proofs of complex protocols in the computational model;

Summary of contributions - outline

1. a methodology to analyze protocols in the symbolic model, but making the attacker as strong as possible, with a case study on multi-factor authentication;

2. a prototype of a mechanized prover in the BC logic;

3. composition results to allow modular proofs of complex protocols in the computational model;

4. symbolic methods for deciding basic proof steps in computational proofs, formulated as problems on probabilistic programs.

Summary of contributions - outline

1. a methodology to analyze protocols in the symbolic model, but making the attacker as strong as possible, with a case study on multi-factor authentication;
   Part I - just a taste of the methodology
2. a prototype of a mechanized prover in the BC logic;
3. composition results to allow modular proofs of complex protocols in the computational model;
4. symbolic methods for deciding basic proof steps in computational proofs, formulated as problems on probabilistic programs.

Summary of contributions - outline

1. a methodology to analyze protocols in the symbolic model, but making the attacker as strong as possible, with a case study on multi-factor authentication;
   Part I - just a taste of the methodology
2. a prototype of a mechanized prover in the BC logic;
   Part II - big ideas of the BC logic & Squirrel
3. composition results to allow modular proofs of complex protocols in the computational model;
4. symbolic methods for deciding basic proof steps in computational proofs, formulated as problems on probabilistic programs.

Summary of contributions - outline

1. a methodology to analyze protocols in the symbolic model, but making the attacker as strong as possible, with a case study on multi-factor authentication;
   Part I - just a taste of the methodology
2. a prototype of a mechanized prover in the BC logic;
   Part II - big ideas of the BC logic & Squirrel
3. composition results to allow modular proofs of complex protocols in the computational model;
   Part III - presentation of the framework
4. symbolic methods for deciding basic proof steps in computational proofs, formulated as problems on probabilistic programs.

Summary of contributions - outline

1. a methodology to analyze protocols in the symbolic model, but making the attacker as strong as possible, with a case study on multi-factor authentication;
   Part I - just a taste of the methodology
2. a prototype of a mechanized prover in the BC logic;
   Part II - big ideas of the BC logic & Squirrel
3. composition results to allow modular proofs of complex protocols in the computational model;
   Part III - presentation of the framework
4. symbolic methods for deciding basic proof steps in computational proofs, formulated as problems on probabilistic programs.
   Not presented. (decidability of universal equivalence between programs over finite fields; library integrated into EasyCrypt and MaskVerif)

Make the symbolic model more precise

Second factor authentication

A case study : Second Factor authentication

How to improve passwords (which are weak)
Use a second factor to confirm login, either a smartphone or a dedicated token.

A case study : Second Factor authentication

How to improve passwords (which are weak)
Use a second factor to confirm login, either a smartphone or a dedicated token.

Considered protocols:

- Google 2 Step (Verification code, Single Tap, Double Tap)
- FIDO's U2F (Google, Facebook, Github, Dropbox,...)

Main ideas

A case study[1] of Google 2 Step and FIDO's U2F.

- Many different detailed threat models;
    - malware on the phone,
    - keylogger on the computer,
    - weak SMS channel,
    - ...
- model the full authentication system;
- completely automated analysis of all scenarios;
- simple, small modifications (adding info to display) that enhance security.

[1] C. Jacomme and S. Kremer, CSF'18 & ACM TOPS

Main ideas

A case study[1] of Google 2 Step and FIDO's U2F.

- Many different detailed threat models;
  - malware on the phone,
  - keylogger on the computer,
  - weak SMS channel,
  - ...
- model the full authentication system;
- completely automated analysis of all scenarios;
- simple, small modifications (adding info to display) that enhance security.

[1] C. Jacomme and S. Kremer, CSF'18 & ACM TOPS

Main ideas

A case study[1] of Google 2 Step and FIDO's U2F.

- Many different detailed threat models;
  - malware on the phone,
  - keylogger on the computer,
  - weak SMS channel,
  - ...
- model the full authentication system;
- completely automated analysis of all scenarios;
- simple, small modifications (adding info to display) that enhance security.

→ 6 172 (non-redundant) scenarios analysed by PROVERIF in 8 minutes

[1] C. Jacomme and S. Kremer, CSF'18 & ACM TOPS

Threat Scenarios				g2V$^{\text{fpr}}$	g2V$^{\text{dis}}$	g2OT$^{\text{dis}}$	g2DT$^{\text{dis}}$
PH				✔	✔	✔	✔
PH	FS			✖	✗✓✓	✖	✗✓✓
PH	FS	$\mathcal{M}^{\textbf{t-tls}}_{\textbf{io}:\mathcal{RO}}$		✖	✖	✖	✗✓✗
PH	FS	$\mathcal{M}^{\textbf{t-usb}}_{\textbf{in}:\mathcal{RO}}$		✖	✖	✖	✗✓✓
PH	FS	$\mathcal{M}^{\textbf{t-dis}}_{\textbf{io}:\mathcal{RW}}$		✖	✗✓✓	✖	✖
		$\mathcal{M}^{\textbf{t-tls}}_{\textbf{io}:\mathcal{RO}}$		✔	✔	✓✓✓	✔
		$\mathcal{M}^{\textbf{t-usb}}_{\textbf{in}:\mathcal{RO}}$		✔	✔	✓✓✓	✔
		$\mathcal{M}^{\textbf{t-tls}}_{\textbf{io}:\mathcal{RW}}$		✗✓✗	✓✓✗	✓✓✗	✓✓✗
		$\mathcal{M}^{\textbf{t-usb}}_{\textbf{in}:\mathcal{RW}}$		✓✓✗	✓✓✗	✓✓✗	✓✓✗
		$\mathcal{M}^{\textbf{t-usb}}_{\textbf{in}:\mathcal{RW}}$	$\mathcal{M}^{\textbf{t-tls}}_{\textbf{io}:\mathcal{RO}}$	✓✓✗	✓✓✗	✓✓✗	✓✓✗
	FS	$\mathcal{M}^{\textbf{t-tls}}_{\textbf{io}:\mathcal{RO}}$		✖	✓✗✗	✖	✓✓✗
	FS	$\mathcal{M}^{\textbf{t-usb}}_{\textbf{in}:\mathcal{RO}}$		✖	✓✗✗	✖	✔
	FS	$\mathcal{M}^{\textbf{t-dis}}_{\textbf{io}:\mathcal{RW}}$		✔	✔	✖	✓✗✗
	FS	$\mathcal{M}^{\textbf{t-tls}}_{\textbf{io}:\mathcal{RW}}$		✖	✓✗✗	✖	✓✗✗
	FS	$\mathcal{M}^{\textbf{t-usb}}_{\textbf{in}:\mathcal{RW}}$		✖	✓✗✗	✖	✓✗✗
	FS	$\mathcal{M}^{\textbf{t-dis}}_{\textbf{io}:\mathcal{RW}}$	$\mathcal{M}^{\textbf{t-tls}}_{\textbf{io}:\mathcal{RO}}$	✖	✓✗✗	✖	✓✗✗
	FS	$\mathcal{M}^{\textbf{t-usb}}_{\textbf{in}:\mathcal{RO}}$	$\mathcal{M}^{\textbf{t-dis}}_{\textbf{io}:\mathcal{RW}}$	✖	✓✗✗	✖	✓✗✗
	FS	$\mathcal{M}^{\textbf{t-usb}}_{\textbf{in}:\mathcal{RW}}$	$\mathcal{M}^{\textbf{t-tls}}_{\textbf{io}:\mathcal{RO}}$	✖	✓✗✗	✖	✓✗✗
		$\mathcal{M}^{\textbf{u-tls}}_{\textbf{io}:\mathcal{RO}}$		✔	✔	✓✓✓	✔
		$\mathcal{M}^{\textbf{u-usb}}_{\textbf{in}:\mathcal{RO}}$		✔	✔	✓✓✓	✔
		$\mathcal{M}^{\textbf{u-tls}}_{\textbf{io}:\mathcal{RW}}$		✓✗✗	✔	✓✓✓	✓✓✓

Pros of U2F

- A possibility of privacy.
- Strong protection against phishing.

Cons of U2F

- No feedback to the user, cannot verify what is validated.
- Not independent from the computer, risk of malwares.

An introduction to the BC logic

Protocol and properties

A protocol

$$A \xrightarrow{\langle r,\, \mathtt{sign}(r, key)\rangle} B$$

A protocol

$$A \xrightarrow{\langle r,\, \mathtt{sign}(r, key)\rangle} B$$

| Checks the signature

Protocol and properties

A protocol

$$A \xrightarrow{\langle r,\, \mathtt{sign}(r, key)\rangle} B$$

| Checks the signature

$$\xleftarrow{<\text{"}ok\text{"}, r>}$$

Protocol and properties

A protocol

$$A \xrightarrow{\langle r, \texttt{sign}(r, key)\rangle} B$$

| Checks the signature

$$\xleftarrow{<\text{"}ok\text{"}, r>}$$

Security property

- Authentication - whenever B accepts, the message that B received was sent by A.

A protocol

$$A \xrightarrow{\langle r,\, \mathtt{sign}(r, key)\rangle} B$$

| Checks the signature

$$\xleftarrow{<\text{"}ok\text{"}, r>}$$

A protocol

$$A \xrightarrow{\langle r,\ \mathtt{sign}(r, key)\rangle} B$$

| Checks the signature

$$\xleftarrow{<\text{“}ok\text{”}, r>}$$

In BC

Protocols are modelled with sequences of terms:

$$\phi_0 := \langle r,\ \mathtt{sign}(r, key)\rangle$$

A quick introduction to the BC logic

A protocol

$$A \xrightarrow{\langle r,\, \mathtt{sign}(r, key)\rangle} B$$

| Checks the signature

$$\xleftarrow{<\text{“}ok\text{”}, r>}$$

In BC

Protocols are modelled with sequences of terms:

$$\phi_0 := \langle r,\ \mathtt{sign}(r, key)\rangle$$

$$\phi_1 := \phi_0, \begin{array}{l} \textbf{if } (\ \mathtt{checksign}(\mathtt{snd}(g_0(\phi_0)), pk(key))) = \mathtt{fst}(g_0(\phi_0)) \textbf{ then} \\ \quad < \text{“}ok\text{”}, \mathtt{fst}(g_0(\phi_0)) > \end{array}$$

A quick introduction to the BC logic

A protocol

$$A \xrightarrow{\langle r,\, \mathtt{sign}(r, key)\rangle} B$$

| Checks the signature

$$\xleftarrow{<\text{“}ok\text{”}, r>}$$

In BC

Protocols are modelled with sequences of terms:

$$\phi_0 := \langle r,\ \mathtt{sign}(r, key)\rangle$$

$$\phi_1 := \phi_0, \ \textbf{if } (\ \mathtt{checksign}(\mathtt{snd}(g_0(\phi_0)), pk(key))) = \mathtt{fst}(g_0(\phi_0))\ \textbf{then}$$
$$< \text{“}ok\text{”}, \mathtt{fst}(g_0(\phi_0)) >$$

A quick introduction to the BC logic

A protocol

$$A \xrightarrow{\langle r,\ \mathtt{sign}(r, key)\rangle} B$$

| Checks the signature

$$\xleftarrow{<\text{“}ok\text{”}, r>}$$

Attacker inputs represented with non instantiated function symbol

In BC

Protocols are modelled with sequences of terms:

$$\phi_0 := \langle r,\ \mathtt{sign}(r, key)\rangle$$

$$\phi_1 := \phi_0, \begin{array}{l} \textbf{if } (\ \mathtt{checksign}(\mathtt{snd}(g_0(\phi_0)), pk(key))) = \mathtt{fst}(g_0(\phi_0)) \textbf{ then} \\ \quad < \text{“}ok\text{”}, \mathtt{fst}(g_0(\phi_0)) > \end{array}$$

A quick introduction to the BC logic

How to reason on terms?
A first order logic built over a predicate that captures indistinguishability:

$$t_1 \sim t_2$$

A quick introduction to the BC logic

How to reason on terms?
A first order logic built over a predicate that captures indistinguishability:

$$t_1 \sim t_2$$

Indistinguishability

- A generic way to express all security properties.
- Any attacker can only distinguish between t_1 and t_2 with negligible probability.

A quick introduction to the BC logic

Axioms to model the assumptions about the cryptographic library.

A quick introduction to the BC logic

Axioms to model the assumptions about the cryptographic library.

EUF-CMA

$$\begin{aligned} &\mathtt{checksign}(t, pk(key)) = m \Rightarrow \\ &\qquad \bigvee_{\mathsf{sign}(x, key) \in \mathtt{St}(t)} (t = \mathsf{sign}(x, key) \wedge x = m) \end{aligned}$$

A quick introduction to the BC logic

Axioms to model the assumptions about the cryptographic library.

EUF-CMA

$$\texttt{checksign}(t, pk(key)) = m \Rightarrow$$
$$\bigvee_{\mathsf{sign}(x, key) \in \, \texttt{St}(t)} (t = \mathsf{sign}(x, key) \wedge x = m)$$

A quick introduction to the BC logic

Axioms to model the assumptions about the cryptographic library.

EUF-CMA

$$\begin{aligned} &\mathtt{checksign}(t, pk(key)) = m \Rightarrow \\ &\qquad \bigvee_{\mathsf{sign}(x, key) \in \mathtt{St}(t)} (t = \mathsf{sign}(x, key) \wedge x = m) \end{aligned}$$

Axioms to model the assumptions about the cryptographic library.

EUF-CMA

$$\mathtt{checksign}(t, pk(key)) = m \Rightarrow \bigvee_{\mathsf{sign}(x, key) \in \mathtt{St}(t)} (t = \mathsf{sign}(x, key) \wedge x = m)$$

A quick introduction to the BC logic

Axioms to model the assumptions about the cryptographic library.

EUF-CMA

$$\mathtt{checksign}(t, pk(key))) = m \Rightarrow$$
$$\bigvee_{\mathsf{sign}(x, key) \in \mathtt{St}(t)} (t = \mathsf{sign}(x, key) \wedge x = m)$$

A mechanized prover for the BC logic

Downsides of the BC logic

- To perform proofs, we have to look at all executions of the protocol, i.e., all possible sequences of terms.

Issues of the BC logic

Downsides of the BC logic

- To perform proofs, we have to look at all executions of the protocol, i.e., all possible sequences of terms.
- Proofs are tedious to perform by hand.

Issues of the BC logic

Downsides of the BC logic

- To perform proofs, we have to look at all executions of the protocol, i.e., all possible sequences of terms.
- Proofs are tedious to perform by hand.

Issues of the BC logic

Downsides of the BC logic

- To perform proofs, we have to look at all executions of the protocol, i.e., all possible sequences of terms.
- Proofs are tedious to perform by hand.
- Proofs only for a bounded number of sessions.

Our contributions[2]

- A meta-logic over BC, that allows to talk abstractly about executions of the protocol.

[2] D. Baelde, S. Delaune, C. Jacomme, A. Koutsos, S. Moreau. S&P'21

Our contributions[2]

- A meta-logic over BC, that allows to talk abstractly about executions of the protocol.
- An interactive prover for this meta-logic.

[2] D. Baelde, S. Delaune, C. Jacomme, A. Koutsos, S. Moreau. S&P'21

Our contributions[2]

- A meta-logic over BC, that allows to talk abstractly about executions of the protocol.
- An interactive prover for this meta-logic.

The Squirrel Prover

[2] D. Baelde, S. Delaune, C. Jacomme, A. Koutsos, S. Moreau. S&P'21

```
signature sign,checksign,pk

abstract ok : message
abstract error : message

name key : message
name r : index -> message

channel c

process A(i:index) =
  out(c, <r(i),sign(r(i),key)>)

process B =
  in(c,x);
  if checksign(snd(x),pk(key)) = fst(x) then
    out(c,<fst(x),ok>)
  else out(c,error)

system (!_i A(i) | !_i B).
```

-:**- auth.sp   Top (22,0)   (squirrel script +2 [1])

Applications Emplacements Emacs lun. 15:44

```
signature sign,checksign,pk

abstract ok : message
abstract error : message

name key : message
name r : index -> message

channel c

process A(i:index) =
  out(c, <r(i),sign(r(i),key)>)

process B =
  in(c,x);
  if checksign(snd(x),pk(key)) = fst(x) then
    out(c,<fst(x),ok>)
  else out(c,error)

system (!_i A(i) | !_i B).
```

-:**- auth.sp Top (22,0) (squirrel script +2 [1])

Applications Emplacements Emacs lun. 15:44 en 100%

```
signature sign,checksign,pk

abstract ok : message
abstract error : message

name key : message
name r : index -> message

channel c

process A(i:index) =
  out(c, <r(i),sign(r(i),key)>)

process B =
  in(c,x);
  if checksign(snd(x),pk(key)) = fst(x) then
    out(c,<fst(x),ok>)
  else out(c,error)

system (!_i A(i) | !_i B).

goal auth :
  forall (i:index),
    cond@B(i) =>
      exists (j:index),
        A(j) < B(i)
        && fst(input@B(i)) = fst(output@A(j)).
Proof.
```

-:**- auth.sp Top (33,0) (squirrel script +2 [1] Scripting)

```
[goal> Focused goal (1/1):
System: default/both
----------------------------------------
forall (i:index),
(cond@B(i) =>
 exists (j:index), (A(j) < B(i) && fst(input@B(i)) = fst(output@A(j))))
```

U:%*- *goals* All (8,0) (squirrel goals +2 [2])

U:%%- *response* All (1,0) (squirrel response [3])

Applications Emplacements Emacs lun. 15:45 en 100%

```
signature sign,checksign,pk

abstract ok : message
abstract error : message

name key : message
name r : index -> message

channel c

process A(i:index) =
  out(c, <r(i),sign(r(i),key)>)

process B =
  in(c,x);
  if checksign(snd(x),pk(key)) = fst(x) then
    out(c,<fst(x),ok>)
  else out(c,error)

system (!_i A(i) | !_i B).

goal auth :
  forall (i:index),
    cond@B(i) =>
      exists (j:index),
         A(j) < B(i)
         && fst(input@B(i)) = fst(output@A(j)).
Proof.
 simpl.
```

-:**- auth.sp Top (30,0) (squirrel script +2 [1] Scripting)

```
[goal> Focused goal (1/1):
System: default/both
Variables: i:index
H0: cond@B(i)
----------------------------------------
exists (j:index), (A(j) < B(i) && fst(input@B(i)) = fst(output@A(j)))
```

U:%*- *goals* All (1,0) (squirrel goals +2 [2])

U:%%- *response* All (1,0) (squirrel response [3])

```
signature sign,checksign,pk

abstract ok : message
abstract error : message

name key : message
name r : index -> message

channel c

process A(i:index) =
  out(c, <r(i),sign(r(i),key)>)

process B =
  in(c,x);
  if checksign(snd(x),pk(key)) = fst(x) then
    out(c,<fst(x),ok>)
  else out(c,error)

system (!_i A(i) | !_i B).

goal auth :
  forall (i:index),
    cond@B(i) =>
      exists (j:index),
         A(j) < B(i)
         && fst(input@B(i)) = fst(output@A(j)).
Proof.
 simpl.
 expand cond@B(i).
```

```
[goal> Focused goal (1/1):
System: default/both
Variables: i:index
M0: checksign(snd(input@B(i)),pk(key)) = fst(input@B(i))
----------------------------------------
exists (j:index), (A(j) < B(i) && fst(input@B(i)) = fst(output@A(j)))
```

U:%*- *goals* All (1,0) (squirrel goals +2 [2])

-:**- auth.sp Top (31,0) (squirrel script +2 [1] Scripting)

U:%%- *response* All (1,0) (squirrel response [3])

Applications Emacs lun. 15:45

```
signature sign,checksign,pk

abstract ok : message
abstract error : message

name key : message
name r : index -> message

channel c

process A(i:index) =
  out(c, <r(i),sign(r(i),key)>)

process B =
  in(c,x);
  if checksign(snd(x),pk(key)) = fst(x) then
    out(c,<fst(x),ok>)
  else out(c,error)

system (!_i A(i) | !_i B).

goal auth :
  forall (i:index),
    cond@B(i) =>
      exists (j:index),
        A(j) < B(i)
        && fst(input@B(i)) = fst(output@A(j)).
Proof.
 simpl.
 expand cond@B(i).
 euf M0.
```

```
-:**-  auth.sp      Top (32,0)     (squirrel script +2 [1]  Scripting )
```

```
[goal> Focused goal (1/1):
System: default/both
Variables: i,i1:index
M0: checksign(snd(input@B(i)),pk(key)) = fst(input@B(i))
M1: r(i1) = fst(input@B(i))
T0: A(i1) < B(i)
----------------------------------------
exists (j:index), (A(j) < B(i) && fst(input@B(i)) = fst(output@A(j)))
```

```
U:%*-  *goals*      All (1,0)      (squirrel goals +2 [2] )
```

```
U:%%-  *response*   All (1,0)      (squirrel response [3] )
```

Applications Emplacements Emacs lun. 15:45 9 en 100%

```
signature sign,checksign,pk

abstract ok : message
abstract error : message

name key : message
name r : index -> message

channel c

process A(i:index) =
  out(c, <r(i),sign(r(i),key)>)

process B =
  in(c,x);
  if checksign(snd(x),pk(key)) = fst(x) then
    out(c,<fst(x),ok>)
  else out(c,error)

system (!_i A(i) | !_i B).

goal auth :
  forall (i:index),
    cond@B(i) =>
      exists (j:index),
         A(j) < B(i)
         && fst(input@B(i)) = fst(output@A(j)).
Proof.
 simpl.
 expand cond@B(i).
 euf M0.
 exists i1.
Qed.
```

```
[goal> Goal auth is proved
```

U:%%- *goals* All (1,0) (squirrel goals +2 [2])

-:**- auth.sp All (33,0) (squirrel script +2 [1] Scripting)

U:%%- *response* All (1,0) (squirrel response [3])

More details?

https://squirrel-prover.github.io/

A compositional framework inside the computational model

Composition?

The goal
To be able to make the proof of a composed protocol as a composition of proofs:

- smaller,
- reusable,
- and modular proofs.

Composition?

The goal
To be able to make the proof of a composed protocol as a composition of proofs:

- smaller,
- reusable,
- and modular proofs.

Top-down vs bottom-up

- Prove components universally secure (UC), and combine them together.
- Split a protocol into multiple components, and prove them secure in the context.

Composition?

The goal
To be able to make the proof of a composed protocol as a composition of proofs:

- smaller,
- reusable,
- and modular proofs.

Top-down vs bottom-up

- Prove components universally secure (UC), and combine them together.
- Split a protocol into multiple components, and prove them secure in the context.

Composition?

The goal
To be able to make the proof of a composed protocol as a composition of proofs:

- smaller,
- reusable,
- and modular proofs.

Top-down vs bottom-up

- Prove components universally secure (UC), and combine them together.
- Split a protocol into multiple components, and prove them secure in the context.

Limitations of the state of the art
Shared secrets and state passing and usability.

Our contributions

The composition framework[3]

- Handles parallel and sequential composition;

  unlike Blanchet, CSF'18, or Brzuska et al., CCS'11

[3] H. Comon, C. Jacomme and G. Scerri. CCS'20

Our contributions

The composition framework[3]

- Handles parallel and sequential composition;

  unlike Blanchet, CSF'18, or Brzuska et al., CCS'11

- allows to consider protocols with state passing and long term shared secrets;

  unlike Brzuska et al., ASIACRYPT'18

[3] H. Comon, C. Jacomme and G. Scerri. CCS'20

Our contributions

The composition framework[3]

- Handles parallel and sequential composition;

  unlike Blanchet, CSF'18, or Brzuska et al., CCS'11

- allows to consider protocols with state passing and long term shared secrets;

  unlike Brzuska et al., ASIACRYPT'18

- allows to reduce the security of multiple sessions to the security of a single one;

[3] H. Comon, C. Jacomme and G. Scerri. CCS'20

Our contributions

The composition framework[3]

- Handles parallel and sequential composition;

  unlike Blanchet, CSF'18, or Brzuska et al., CCS'11

- allows to consider protocols with state passing and long term shared secrets;

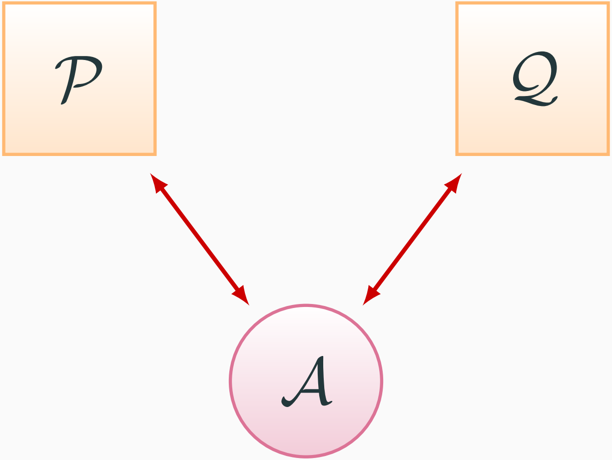
  unlike Brzuska et al., ASIACRYPT'18

- allows to reduce the security of multiple sessions to the security of a single one;
- naturally translates to the BC logic, and allows for the first time to perform proofs for an unbounded number of sessions with this logic.

[3] H. Comon, C. Jacomme and G. Scerri. CCS'20

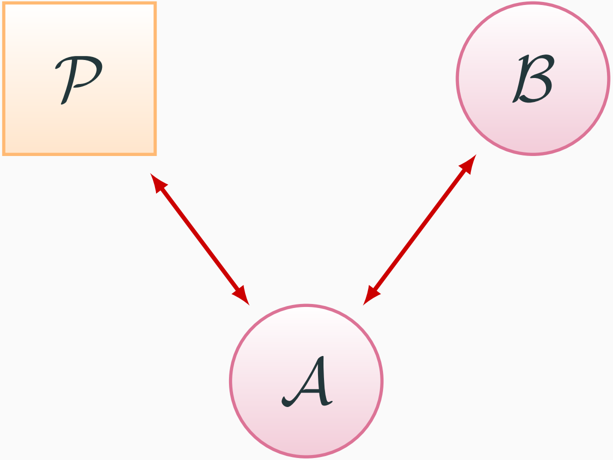
A classical proof technique

$\mathcal{A}$ is trying to break protocol $\mathcal{P}$, while also having access to $\mathcal{Q}$.



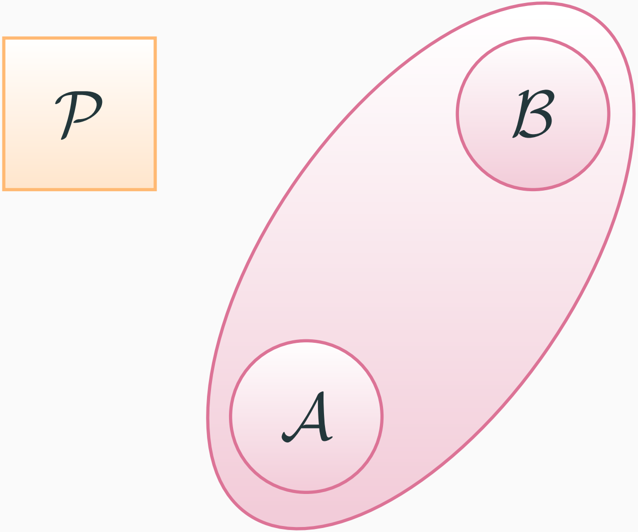
A classical proof technique

$\mathcal{A}$ is trying to break protocol $\mathcal{P}$, while also simulating $\mathcal{Q}$.



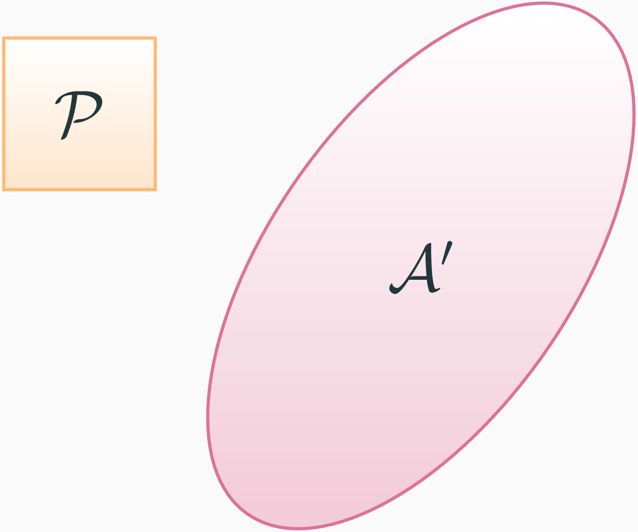
A classical proof technique

$\mathcal{A}$ is trying to break protocol $\mathcal{P}$, while also simulating $\mathcal{Q}$.



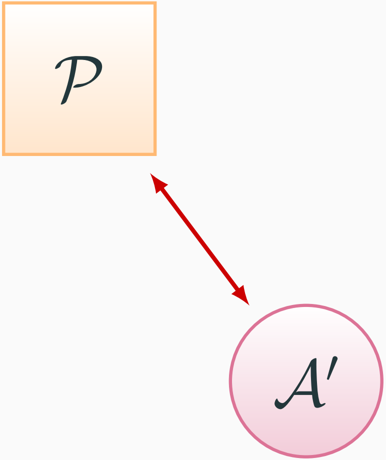
A classical proof technique

$\mathcal{A}$ is trying to break protocol $\mathcal{P}$, while also simulating $\mathcal{Q}$.



A classical proof technique

$\mathcal{A}$ is trying to break protocol $\mathcal{P}$, while also simulating $\mathcal{Q}$.



Simulation

The main idea

If $\mathcal{A}$ can simulate it, i.e., produce exactly all the same messages:

we remove Q from the picture!

Simulation

The main idea
If $\mathcal{A}$ can simulate it, i.e., produce exactly all the same messages:

we remove Q from the picture!

The difficulty
If P and Q share some secret key, $\mathcal{A}$ cannot simulate messages which require key.

The main idea - example on downgrade attacks

P = version 1 of the previous protocol

$$A \xrightarrow{\texttt{sign}(\langle r, \text{``v}_1\text{''}\rangle, key)} B$$

| Checks the signature

$$\xleftarrow{<r, \text{``}ok\text{''}>}$$

Q = version 2 of the previous protocol

$$A \xrightarrow{\texttt{sign}(\langle r, \text{``v}_2\text{''}\rangle, key)} B$$

| Checks the signature

$$\xleftarrow{<r, \text{``}ok\text{''}>}$$

The main idea

Example for signatures

- $\mathcal{Q}_{key}$ may produce $\texttt{sign}(< m, \text{“v}_1\text{”} >, key)$
- $\mathcal{P}_{key}$ may produce $\texttt{sign}(< m', \text{“v}_2\text{”} >, key)$

Example for signatures

- $\mathcal{Q}_{key}$ may produce $\texttt{sign}(< m, \text{"v}_1\text{"} >, key)$
- $\mathcal{P}_{key}$ may produce $\texttt{sign}(< m', \text{"v}_2\text{"} >, key)$

To prove $\mathcal{P}$ while abstracting $\mathcal{Q}$, the attacker must be able to produce $\texttt{sign}(< m', \text{"v}_1\text{"} >, key)$.

The main idea

Example for signatures

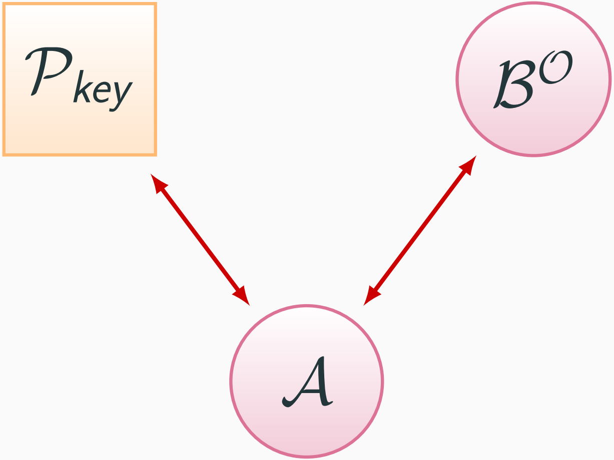
- $\mathcal{Q}_{key}$ may produce $\texttt{sign}(< m, \text{“v}_1\text{”} >, key)$
- $\mathcal{P}_{key}$ may produce $\texttt{sign}(< m', \text{“v}_2\text{”} >, key)$

To prove $\mathcal{P}$ while abstracting $\mathcal{Q}$, the attacker must be able to produce $\texttt{sign}(< m', \text{“v}_1\text{”} >, key)$.

$\hookrightarrow$ We may give an oracle to the attacker, allowing to obtain $\texttt{sign}(< m', \text{“v}_1\text{”} >, key)$ but not $\texttt{sign}(< m, \text{“v}_2\text{”} >, key)$

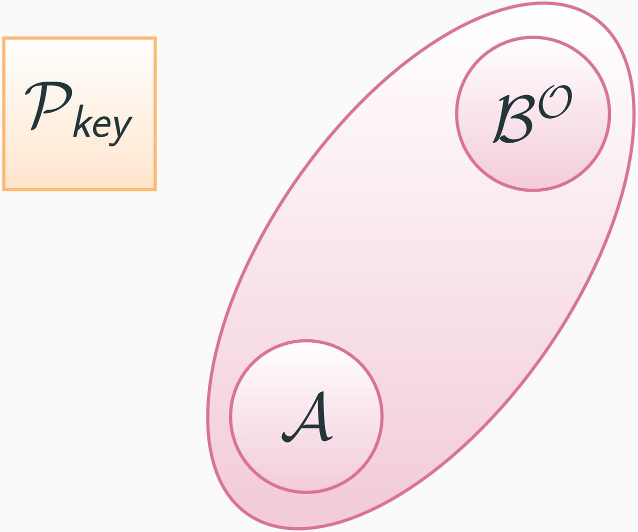
The main idea

$\mathcal{A}$ is trying to break protocol $\mathcal{P}$, while simulating $\mathcal{Q}$ thanks to oracle $\mathcal{O}$.



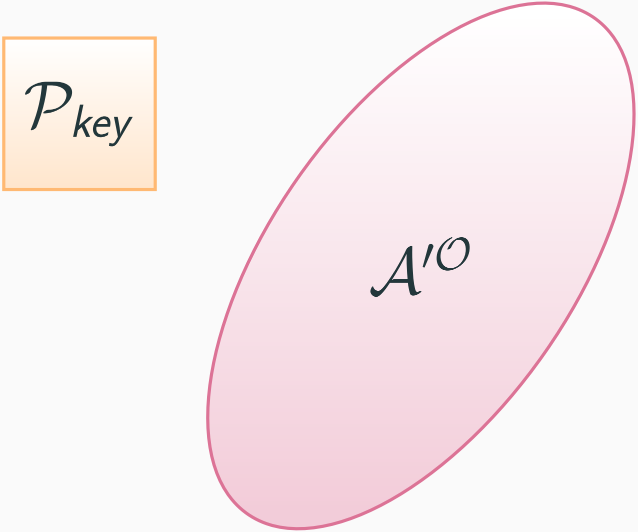
The main idea

$\mathcal{A}$ is trying to break protocol $\mathcal{P}$, while simulating $\mathcal{Q}$ thanks to oracle $\mathcal{O}$.



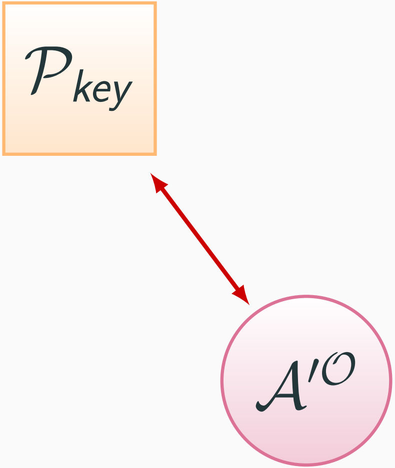
The main idea

$\mathcal{A}$ is trying to break protocol $\mathcal{P}$, while simulating $\mathcal{Q}$ thanks to oracle $\mathcal{O}$.



The main idea

$\mathcal{A}$ is trying to break protocol $\mathcal{P}$, while simulating $\mathcal{Q}$ thanks to oracle $\mathcal{O}$.



Simulatability

$\nu key.\mathcal{Q}_{key}$ is $\mathcal{O}$-simulatable

iff

there exists a PPT $\mathcal{A}^{\mathcal{O}}$ which, for any fixed value of *key*,

produces exactly the same distribution as $\mathcal{Q}_{key}$

Simulatability

$$\nu key.\mathcal{Q}_{key} \text{ is } \mathcal{O}\text{-simulatable}$$

iff

there exists a PPT $\mathcal{A}^{\mathcal{O}}$ which, for any fixed value of *key*,
produces exactly the same distribution as $\mathcal{Q}_{key}$

A protocol

$$\begin{aligned} Q := &\ \dots \\ &\ \textbf{out}(\text{sign}(\langle mess, \text{``v}_1\text{''}\rangle, key))) \end{aligned}$$

Simulatability

$$\nu key.\mathcal{Q}_{key} \text{ is } \mathcal{O}\text{-simulatable}$$

iff

there exists a PPT $\mathcal{A}^{\mathcal{O}}$ which, for any fixed value of *key*, produces exactly the same distribution as $\mathcal{Q}_{key}$

A protocol

$$Q := \dots$$
$$\quad \mathbf{out}(\mathrm{sign}(\langle mess, \text{“}v_1\text{”}\rangle, key)))$$

Signing oracle

$$\mathcal{O}^{\mathsf{sign}}_{key} : \mathrm{input}(m)$$
$$\quad \mathrm{output}(\mathrm{sign}(\langle m, \text{“}v_1\text{”}\rangle, key)))$$

Generic signing oracles

T signing oracle

$$\mathcal{O}^{\mathsf{sign}}_{T,sk} : \mathsf{input}(m)$$
$$\textbf{if } T(m) \textbf{ then}$$
$$\mathsf{output}(\mathsf{sign}(m, sk))$$

Generic signing oracles

T signing oracle

$$\mathcal{O}^{\text{sign}}_{T,sk} : \text{input}(m)$$
$$\textbf{if } T(m) \textbf{ then}$$
$$\text{output}(\text{sign}(m, sk))$$

T-EUF-CMA

$$\texttt{checksign}(t, pk(sk)) \doteq m \Rightarrow$$
$$T(m)$$
$$\bigvee_{\text{sign}(x,sk) \in \texttt{St}(t)} (t \doteq \text{sign}(x, sk) \wedge x \doteq m)$$

Conclusions about compositions

- Used as a proof of concept on SSH;
- proofs close to the classical ones;
- mechanizable.

$\hookrightarrow$ It was easy to extend Squirrel to support the generic axioms!

Conclusion

Summary of contributions

1. a methodology to analyze protocols in the symbolic model, but making the attacker as strong as possible, with a case study on multi-factor authentication;
    - C. Jacomme and S. Kremer. CSF'18 & ACM TOPS

Summary of contributions

1. a methodology to analyze protocols in the symbolic model, but making the attacker as strong as possible, with a case study on multi-factor authentication;
    - C. Jacomme and S. Kremer. CSF'18 & ACM TOPS
2. the Squirrel Prover, a mechanized prover in the BC logic.
    - D. Baelde, S. Delaune, C. Jacomme, A. Koutsos, S. Moreau. S&P 21.

Summary of contributions

1. a methodology to analyze protocols in the symbolic model, but making the attacker as strong as possible, with a case study on multi-factor authentication;
    - C. Jacomme and S. Kremer. CSF'18 & ACM TOPS
2. the Squirrel Prover, a mechanized prover in the BC logic.
    - D. Baelde, S. Delaune, C. Jacomme, A. Koutsos, S. Moreau. S&P 21.
3. composition results to allow modular proofs of complex protocols in the computational model;
    - H. Comon, C. Jacomme, and G. Scerri. CCS'20

Summary of contributions

1. a methodology to analyze protocols in the symbolic model, but making the attacker as strong as possible, with a case study on multi-factor authentication;
    - C. Jacomme and S. Kremer. CSF'18 & ACM TOPS
2. the Squirrel Prover, a mechanized prover in the BC logic.
    - D. Baelde, S. Delaune, C. Jacomme, A. Koutsos, S. Moreau. S&P 21.
3. composition results to allow modular proofs of complex protocols in the computational model;
    - H. Comon, C. Jacomme, and G. Scerri. CCS'20
4. symbolic methods for deciding basic proof steps in computational proofs, formulated as problems on probabilistic programs;
    - G. Barthe, X. Fan, J. Gancher, B. Grégoire, C. Jacomme, and E. Shi. CCS'18
    - G. Barthe, B. Grégoire, C. Jacomme, S. Kremer, and P-Y. Strub. CSF'19
    - G. Barthe, C. Jacomme, and S. Kremer. LICS'20

What's next

Modularity
Apply/extend the composition framework to more complex protocols and properties.

(e-voting protocols, forward secrecy for key-exchanges)

What's next

Modularity
Apply/extend the composition framework to more complex protocols and properties.
(e-voting protocols, forward secrecy for key-exchanges)

Automation
Improve the automation,

- at the high-level, in the Squirrel Prover; (e.g. with SMT solvers)
- at the low-level, through SolvEq.

What's next

Modularity
Apply/extend the composition framework to more complex protocols and properties. (e-voting protocols, forward secrecy for key-exchanges)

Automation
Improve the automation,

- at the high-level, in the Squirrel Prover; (e.g. with SMT solvers)
- at the low-level, through SolvEq.

Collaboration
There will not be one tool to rule them all. Use each for what it does best and combine formally the guarantees.

Privacy

What's next - in real life

Privacy

- It **matters**.

Privacy

- It **matters**.
- Most people, corporation and states **don't care about it**.

Privacy

- It **matters**.
- Most people, corporation and states **don't care about it**.

I would like to try to do something about that...

Some appendixes

Composition on an example

Main intuition

Basic Theorem Example - Parallel Composition
Given two protocols $\mathcal{P}$ and $\mathcal{Q}$, with $\overline{n} = \mathcal{N}(P) \cap \mathcal{N}(Q)$, if:

- $\nu \overline{n}.\mathcal{Q}$ is $\mathcal{O}$-simulatable;
- $A \models \mathcal{P} \sim \mathcal{P}'$;
- the axioms A are sound for machines with access to $\mathcal{O}$.

Then $A \models \mathcal{P} \| \mathcal{Q} \sim \mathcal{P}' \| \mathcal{Q}$.

A small DDH example

Signed DDH

$A(a, skA)$ $\qquad\qquad$ $B(b, skB)$

$\xrightarrow{\text{sign}(g^a, skA)}$

$x_B = g^a$

$\xleftarrow{\text{sign}(<g^a, g^b>, skB)}$

$x_A = g^b$

$\xrightarrow{\text{sign}(<g^a, g^b>, skA)}$

$k_A = x_A^a$ $\qquad\qquad$ $k_B = x_B^b$

A small DDH example

The security property:

$$\|^{i \leq N}(A(a_i, skA); \textbf{out}(k_A) \| B(b_i, skB); \textbf{out}(k_B))$$

$$\sim$$

$$\begin{aligned}
&\|^{i \leq N-1}(A(a_i, skA); \textbf{out}(k_A) \| B(b_i, skB); \textbf{out}(k_B)) \\
&\| \;\; A(a_N, skA);\ \textbf{if}\ x_A = g^{b_N}\ \textbf{then out}(k_{N,N}) \\
&\qquad\qquad\qquad \textbf{else if}\ x_A \notin \{g^{b_i}\}_{1 \leq i \leq N}\ \textbf{then}\ \perp \\
&\| \;\; B(b_N, skB);\ \textbf{if}\ x_B = g^{a_N}\ \textbf{then out}(k_{N,N}) \\
&\qquad\qquad\qquad \textbf{else if}\ x_B \notin \{g^{a_i}\}_{1 \leq i \leq N}\ \textbf{then}\ \perp
\end{aligned}$$

A small DDH example

The final security property:
Let's assume the attacker can simulate

$$\|^{i \leq N-1}(A(a_i, skA); \mathbf{out}(k_A) \| B(b_i, skB); \mathbf{out}(k_B))$$

.

A small DDH example

The final security property:
Let's assume the attacker can simulate

$$\|^{i \leq N-1}(A(a_i, skA); \mathbf{out}(k_A) \| B(b_i, skB); \mathbf{out}(k_B))$$

. We can simply prove:

$$
\begin{array}{c}
A(a_N, skA); \mathbf{out}(k_A) \| B(b_N, skB); \mathbf{out}(k_B) \\
\sim \\
A(a_N, skA);\ \textbf{if } x_A = g^{b_N} \textbf{ then out}(k_{N,N}) \\
\qquad \textbf{else if } x_A \notin \{g^{b_i}\}_{1 \leq i \leq N} \textbf{ then } \bot \\
\|\ B(b_N, skB);\ \textbf{if } x_B = g^{a_N} \textbf{ then out}(k_{N,N}) \\
\qquad \textbf{else if } x_B \notin \{g^{a_i}\}_{1 \leq i \leq N} \textbf{ then } \bot
\end{array}
$$

A small DDH example

The final security property:
Let's assume the attacker can simulate

$$\|^{i\leq N-1}(A(a_i, skA); \mathbf{out}(k_A) \| B(b_i, skB); \mathbf{out}(k_B))$$

. We can simply prove:

$$\begin{array}{l} A(a_N, skA); \mathbf{out}(k_A) \| B(b_N, skB); \mathbf{out}(k_B) \\ \sim \\ A(a_N, skA);\ \mathbf{if}\ x_A = g^{b_N}\ \mathbf{then}\ \mathbf{out}(k_{N,N}) \\ \qquad \mathbf{else\ if}\ x_A \notin \{g^{b_i}\}_{1\leq i\leq N}\ \mathbf{then}\ \bot \\ \|\ B(b_N, skB);\ \mathbf{if}\ x_B = g^{a_N}\ \mathbf{then}\ \mathbf{out}(k_{N,N}) \\ \qquad \mathbf{else\ if}\ x_B \notin \{g^{a_i}\}_{1\leq i\leq N}\ \mathbf{then}\ \bot \end{array}$$

$\hookrightarrow$ How to simulate the $N-1$ sessions ?

Simulating the sessions

What must the attacker be able to produce ?
He must be able to start some A:

$$\forall 1 \leq i \leq N-1.\ \mathrm{sign}(g^{a_i}, skA)$$

Simulating the sessions

What must the attacker be able to produce ?
He must be able to start some A:

$$\forall 1 \leq i \leq N-1.\ \text{sign}(g^{a_i}, skA)$$

And for any DDH share r he receives, he should be able to produce:

- $\forall 1 \leq i \leq N-1.\ \text{sign}(< g^{a_i}, r >, skA)$

Simulating the sessions

What must the attacker be able to produce ?
He must be able to start some A:

$$\forall 1 \leq i \leq N-1.\ \text{sign}(g^{a_i}, skA)$$

And for any DDH share r he receives, he should be able to produce:

- $\forall 1 \leq i \leq N-1.\ \text{sign}(< g^{a_i}, r >, skA)$
- $\forall 1 \leq i \leq N-1.\ \text{sign}(< r, g^{b_i} >, skB)$

Generic signing oracles

T signing oracle

$$\mathcal{O}^{\text{sign}}_{T,sk} : \text{input}(m)$$
$$\textbf{if } T(m) \textbf{ then}$$
$$\quad \text{output}(\text{sign}(m, sk)))$$

Generic signing oracles

T signing oracle

$$\mathcal{O}^{\text{sign}}_{T,sk} : \text{input}(m)$$
$$\textbf{if } T(m) \textbf{ then}$$
$$\text{output}(\text{sign}(m, sk)))$$

Give the attacher access to $\mathcal{O}^{sign}_{T,skA}$ and $\mathcal{O}^{sign}_{T,skB}$ with:

$$T(m) = \text{true} \Leftrightarrow \exists 1 \leq i \leq N-1, r. \begin{cases} m = g^{a_i} \\ m =< g^{a_i}, r > \\ m =< r, g^{b_i} > \end{cases}$$

Generic signing oracles

T signing oracle

$$\mathcal{O}^{\text{sign}}_{T,sk} : \text{input}(m) \\ \textbf{if } T(m) \textbf{ then} \\ \text{output}(\text{sign}(m, sk)))$$

Give the attacher access to $\mathcal{O}^{sign}_{T,skA}$ and $\mathcal{O}^{sign}_{T,skB}$ with:

$$T(m) = \text{true} \Leftrightarrow \exists 1 \leq i \leq N-1, r. \begin{cases} m = g^{a_i} \\ m =< g^{a_i}, r > \\ m =< r, g^{b_i} > \end{cases}$$

$\hookrightarrow$ How to make the proof for such attackers ?

Generic axioms

T-EUF-CMA

For any computable function T, for all terms t such that sk only appears in key position:

$$\begin{aligned}&\texttt{checksign}(t, pk(sk))) \Rightarrow\\&\quad T(\texttt{getmess}(t))\\&\quad \bigvee_{\mathsf{sign}(x,sk)\in \texttt{St}(t)} (t \doteq \mathsf{sign}(x, sk)))\\&\sim \mathit{true}\end{aligned}$$

The final proof

Assumption

$$\mathtt{checksign}(t, pk(sk)) \Rightarrow$$
$$\exists 1 \leq i \leq N-1, r.\ \ \mathtt{getmess}(t) \in \{g^{a_i}, < g^{a_i}, r >, < r, g^{b_i} >\}$$
$$\bigvee_{\mathsf{sign}(x,sk) \in \mathtt{St}(t)} (t \doteq \mathsf{sign}(x, sk)))$$
$$\sim true$$

The final proof

Assumption

$$\begin{aligned}
&\texttt{checksign}(t, pk(sk))) \Rightarrow \\
&\quad \exists 1 \leq i \leq N-1, r.\ \ \texttt{getmess}(t) \in \{g^{a_i}, < g^{a_i}, r >, < r, g^{b_i} >\} \\
&\quad \bigvee_{\mathsf{sign}(x,sk) \in \texttt{St}(t)} (t \doteq \mathsf{sign}(x, sk))) \\
&\sim true
\end{aligned}$$

$$\wedge \quad DDH : g^{a_N}, g^{b_N}, g^{a_N b_N} \sim g^{a_N}, g^{b_N}, k_{N,N}$$

The final proof

Goal

$$A(a_N, skA); \mathbf{out}(k_A) \| B(b_N, skB); \mathbf{out}(k_B)$$
$$\sim$$
$$\begin{aligned} & A(a_N, skA);\ \mathbf{if}\ x_A = g^{b_N}\ \mathbf{then\ out}(k_{N,N}) \\ & \qquad \mathbf{else\ if}\ x_A \notin \{g^{b_i}\}_{1 \leq i \leq N}\ \mathbf{then}\ \bot \\ \| \ & B(b_N, skB);\ \mathbf{if}\ x_B = g^{a_N}\ \mathbf{then\ out}(k_{N,N}) \\ & \qquad \mathbf{else\ if}\ x_B \notin \{g^{a_i}\}_{1 \leq i \leq N}\ \mathbf{then}\ \bot \end{aligned}$$

The final proof

Synchronization

$$
\begin{aligned}
&A(a_N, skA); \textbf{ if } x_A = g^{b_N} \textbf{ then out}(g^{a_N b_N}) \\
&\qquad\qquad \textbf{else if } x_A \notin \{g^{b_i}\}_{1\le i\le N} \textbf{ then out}(x_A^{a_N}) \\
\parallel\ &B(b_N, skB); \textbf{ if } x_B = g^{a_N} \textbf{ then out}(g^{a_N b_N}) \\
&\qquad\qquad \textbf{else if } x_B \notin \{g^{a_i}\}_{1\le i\le N} \textbf{ then out}(x_B^{b_N})
\end{aligned}
$$

$$\sim$$

$$
\begin{aligned}
&A(a_N, skA); \textbf{ if } x_A = g^{b_N} \textbf{ then out}(k_{N,N}) \\
&\qquad\qquad \textbf{else if } x_A \notin \{g^{b_i}\}_{1\le i\le N} \textbf{ then } \bot \\
\parallel\ &B(b_N, skB); \textbf{ if } x_B = g^{a_N} \textbf{ then out}(k_{N,N}) \\
&\qquad\qquad \textbf{else if } x_B \notin \{g^{a_i}\}_{1\le i\le N} \textbf{ then } \bot
\end{aligned}
$$

The final proof

Synchronization

$$A(a_N, skA);\ \textbf{if } x_A = g^{b_N} \textbf{ then out}(g^{a_N b_N})$$
$$\textbf{else if } x_A \notin \{g^{b_i}\}_{1 \leq i \leq N} \textbf{ then out}(x_A^{a_N})$$
$$\parallel B(b_N, skB);\ \textbf{if } x_B = g^{a_N} \textbf{ then out}(g^{a_N b_N})$$
$$\textbf{else if } x_B \notin \{g^{a_i}\}_{1 \leq i \leq N} \textbf{ then out}(x_B^{b_N})$$
$$\sim$$
$$A(a_N, skA);\ \textbf{if } x_A = g^{b_N} \textbf{ then out}(k_{N,N})$$
$$\textbf{else if } x_A \notin \{g^{b_i}\}_{1 \leq i \leq N} \textbf{ then } \bot$$
$$\parallel B(b_N, skB);\ \textbf{if } x_B = g^{a_N} \textbf{ then out}(k_{N,N})$$
$$\textbf{else if } x_B \notin \{g^{a_i}\}_{1 \leq i \leq N} \textbf{ then } \bot$$

The final proof

Synchronization

$$
\begin{aligned}
& A(a_N, skA);\ \text{if } x_A = g^{b_N} \text{ then out}(g^{a_N b_N}) \\
& \qquad \text{else if } x_A \notin \{g^{b_i}\}_{1 \leq i \leq N} \text{ then } \text{out}(x_A^{a_N}) \\
\| \ & B(b_N, skB);\ \text{if } x_B = g^{a_N} \text{ then out}(g^{a_N b_N}) \\
& \qquad \text{else if } x_B \notin \{g^{a_i}\}_{1 \leq i \leq N} \text{ then out}(x_B^{b_N})
\end{aligned}
$$

$$\sim$$

$$
\begin{aligned}
& A(a_N, skA);\ \text{if } x_A = g^{b_N} \text{ then out}(k_{N,N}) \\
& \qquad \text{else if } x_A \notin \{g^{b_i}\}_{1 \leq i \leq N} \text{ then } \bot \\
\| \ & B(b_N, skB);\ \text{if } x_B = g^{a_N} \text{ then out}(k_{N,N}) \\
& \qquad \text{else if } x_B \notin \{g^{a_i}\}_{1 \leq i \leq N} \text{ then } \bot
\end{aligned}
$$

The final proof

Synchronization

Proof steps Split the conditonals into four cases and,

The final proof

Synchronization
Proof steps Split the conditonals into four cases and,

1. use DDH to show indistinguishability,

The final proof

Synchronization
Proof steps Split the conditonals into four cases and,

1. use DDH to show indistinguishability,
2. use T-EUF-CMA, to show that $x_A \notin \{g^{b_i}\}_{1 \leq i \leq N}$ is never true (e.g, $\perp$ unreachable),

The final proof

Synchronization
Proof steps Split the conditonals into four cases and,

1. use DDH to show indistinguishability,
2. use T-EUF-CMA, to show that $x_A \notin \{g^{b_i}\}_{1 \leq i \leq N}$ is never true (e.g, $\perp$ unreachable),
3. similar to (2);
4. similar to (2);

Formal composition theorems

A core theorem

Composition without replication

Let $C[\_{1}, \ldots, \_{n}]$ be a context such that the variable k_i is bound in each hole $\_i$ and $P_1(x), \ldots, P_n(x)$ be parametrized protocols, such that all channels are disjoint. Given an oracle $\mathcal{O}$, with $\overline{n} \supset \mathcal{N}(C) \cap \mathcal{N}(P_1, \ldots, P_n)$, if, with $k'_1, \ldots, k'_n$ fresh names,

1. $C[\mathbf{out}(1, k_1), \ldots, \mathbf{out}(n, k_n)] \cong_{\mathcal{O}} C[\mathbf{out}(1, k'_1), \ldots, \mathbf{out}(n, k'_n)]$
2. $\nu\overline{n}.\mathbf{in}(x).P_1(x) \| \ldots \| \mathbf{in}(x).P_n(x)$ is $\mathcal{O}$-simulatable

Then $C[P_1(k_1), \ldots, P_n(k_n)] \cong_{\mathcal{O}} C[P_1(k'_1), \ldots, P_n(k'_n)]$

A core theorem

Unbounded parallel Composition

Let $\mathcal{O}_r$ be an oracle and Ax a set of axioms both parametrized by a sequence of names $\overline{s}$. Let $\overline{p}$ be a sequence of shared secrets, $P(\overline{x})$, $R(\overline{x}, \overline{y}, \overline{z})$ and $Q(\overline{x}, \overline{y})$ be parametrized protocols. If we have, for a sequence of names $\overline{lsid}$ and any integers n, if with $\overline{s} = \overline{lsid}_1, \dots, \overline{lsid}_n$ n copies of $\overline{lsid}$:

1. $\forall\ 1 \leq i \leq n, \nu\overline{p}.t_{R(\overline{p},\overline{lsid}_i,\overline{s})}$ is $\mathcal{O}_r$ simulatable.
2. Ax is $\mathcal{O}_r$ sound.
3. $Ax \models t_{P(\overline{p})} \sim t_{Q(\overline{p},\overline{s})}$

Then, for any integer n:

$$\begin{aligned} &P(\overline{p}) \| \ !_n R(\overline{p}, \overline{lsid}, \overline{s}) \\ &\quad \cong Q(\overline{p}, \overline{s}) \| \ !_n R(\overline{p}, \overline{lsid}, \overline{s}) \end{aligned}$$

A core theorem

Unbounded parallel Composition
Let $\mathcal{O}_r$ be an oracle and Ax a set of axioms both parametrized by a sequence of names $\overline{s}$. Let $\overline{p}$ be a sequence of shared secrets, $P(\overline{x}, \overline{y})$ and $Q(\overline{x}, \overline{y}, \overline{z})$ be parametrized protocols. If we have, for sequences of names $\overline{lsid}_p, \overline{lsid}_q$ and any integers n, if with $\overline{s} = \overline{lsid}_{p,1}, \dots, \overline{lsid}_{p,n}, \dots, \overline{lsid}_{q,n}$ sequences of copies of $\overline{lsid}_p, \overline{lsid}_q$

1. $\forall\ 1 \leq i \leq n, \nu\overline{p}.t_{P(\overline{p}, \overline{lsid}_{p,i})}$ is $\mathcal{O}_r$ simulatable.
2. $\forall\ 1 \leq i \leq n, \nu\overline{p}.t_{Q(\overline{p}, \overline{lsid}_{q,i}, \overline{s})}$ is $\mathcal{O}_r$ simulatable.
3. Ax is $\mathcal{O}_r$ sound.
4. $Ax \models t_{P(\overline{p}, \overline{lsid}_p)} \sim t_{Q(\overline{p}, \overline{lsid}_q, \overline{s})}$

Then, for any integers n:

$$!_n P(\overline{p}, \overline{lsid}_p) \cong_{\mathcal{O}} !_n Q(\overline{p}, \overline{s}, \overline{lsid}_q)$$