Universal Composition
with Responsive Environments

Jan Camenisch\textsuperscript{1}, Robert R. Enderlein\textsuperscript{1}, Stephan Krenn\textsuperscript{2},
Ralf Küsters\textsuperscript{3}, Daniel Rausch\textsuperscript{3}

\textsuperscript{1} IBM Research Zurich - Switzerland
\textsuperscript{2} AIT - Austria
\textsuperscript{3} University of Trier - Germany
Definition of **simulatability** (basic idea):

\[ P \preceq F \]
Simulation-Based Security

Definition of **simulatability** (basic idea):

\[ P \preceq F \]

- **Ideal** protocol/functionality:
  - e.g. ideal key exchange

- **Real** protocol:
  - e.g. IKE
Definition of **simulatability** (basic idea):

\[ P \leq F \]

**real** protocol
- e.g. IKE

**ideal** protocol/functionality
- e.g. ideal key exchange
Definition of **simulatability** (basic idea): 

\[ P \leq F \text{ iff } \]

- **real** protocol, e.g., IKE
- **ideal** protocol/functionality, e.g., ideal key exchange
Definition of simulatability (basic idea):

\[
\forall P \leq F \iff \exists P' = \text{ideal protocol/functionality} \\
\text{e.g. ideal key exchange}
\]

real protocol e.g. IKE
Definition of **simulatability** (basic idea):

\[ P \leq \mathcal{F} \text{ iff } \forall \exists \quad \text{ideal protocol/functionality} \quad \text{e.g. ideal key exchange} \]

\[ \text{real protocol} \quad \text{e.g. IKE} \]
Definition of *simulatability* (basic idea):

\[
\mathcal{P} \trianglelefteq \mathcal{F} \iff \forall \mathcal{E} \equiv \exists \mathcal{E}
\]

**ideal** protocol/functionality  
\text{e.g. ideal key exchange}

**real** protocol  
\text{e.g. IKE}
Compositional Protocol Analysis
Compositional Protocol Analysis

Assume:

\[ P \leq F \]
Compositional Protocol Analysis

Assume:

\[ P \leq F \]

e.g. ideal key exchange
Compositional Protocol Analysis

Assume:

\[ P \leq F \]

\[ \text{e.g. ideal key exchange} \]

Prove:
Compositional Protocol Analysis

Assume:

\[ P \leq F \]

e.g. ideal key exchange

e.g., some real-world protocol SSL/TLS, SSH, ...

Prove:

Q
Compositional Protocol Analysis

Assume:

\[ P \leq F \]

\[ \text{e.g., ideal key exchange} \]

e.g., some real-world protocol
SSL/TLS, SSH, ...

Prove:

\[ Q \leq F' \]
Compositional Protocol Analysis

Assume:

\[ P \leq F \]

- e.g. ideal key exchange
- e.g., some real-world protocol SSL/TLS, SSH, ...

Prove:

\[ Q \leq F' \]

- e.g. ideal secure channel
Compositional Protocol Analysis

Assume:

\[ P \leq F \]

- e.g. ideal key exchange

- e.g., some real-world protocol SSL/TLS, SSH, ...

Prove:

\[ Q \leq F' \]

- e.g. ideal secure channel
Compositional Protocol Analysis

Assume:

\[ P \leq F \]

- e.g., ideal key exchange
- e.g., some real-world protocol SSL/TLS, SSH, ...

Prove:

\[ Q \downarrow \]

\[ F \leq F' \]

- e.g. ideal secure channel
Compositional Protocol Analysis

Assume:

\[ \mathcal{P} \leq \mathcal{F} \]

- e.g. ideal key exchange
- e.g., some real-world protocol SSL/TLS, SSH, ...

Prove:

\[ \mathcal{Q} \downarrow \mathcal{F} \leq \mathcal{F}' \]

\[ \mathcal{Q} \downarrow \mathcal{F} \leq \mathcal{F}' \]
Compositional Protocol Analysis

Assume:

\[ P \leq F \]

- e.g. ideal key exchange
- e.g., some real-world protocol SSL/TLS, SSH, ...

Prove:

\[ Q \downarrow \leq F' \]

Composition Theorem

\[ Q \downarrow \leq F \downarrow \leq F' \]
Compositional Protocol Analysis

Assume:

\[ P \leq F \]

- e.g. ideal key exchange
- e.g., some real-world protocol SSL/TLS, SSH, ...

Prove:

\[ Q \downarrow \leq F \leq F' \]

- e.g. ideal secure channel

Composition Theorem

\[ Q \downarrow \leq P \leq F \leq F' \]

can now be used in more complex protocols
Models for Simulation-Based Security

- UC model [Canetti 2001]
- IITM model [Küsters 2006]
- GNUC model [Hofheinz, Shoup 2011]
- ...
What is the problem?
What is the problem?

* Urgent Requests
What is the problem?

* Urgent Requests
* Non-Responsiveness Problem
What is the problem?

* Urgent Requests
* Non-Responsiveness Problem

Our solution:
Responsive Environments
Urgent Requests

Protocols often have to exchange modeling related meta information with adversary:

\[ \mathcal{E} \]

\[ P \]
Urgent Requests

Protocols often have to exchange modeling related meta information with adversary:

- Ask for corruption status
Urgent Requests

Protocols often have to exchange modeling related meta information with adversary:

- Ask for corruption status
- Ask for cryptographic material (keys, algorithms,...)
Urgent Requests

Protocols often have to exchange modeling related meta information with adversary:

- Ask for corruption status
- Ask for cryptographic material (keys, algorithms, ...)
- Leak information
Urgent Requests

Protocols often have to exchange modeling related meta information with adversary:

- Ask for corruption status
- Ask for cryptographic material (keys, algorithms, ...)
- Leak information
- Signaling information ("new instance created")
Urgent Requests

Protocols often have to exchange modeling related meta information with adversary:

- Ask for corruption status
- Ask for cryptographic material (keys, algorithms,...)
- Leak information
- Signaling information ("new instance created")

⇒ Send a message $m$ (urgent request)
Non-Responsiveness Problem

Urgent requests do **not** model real network traffic
Non-Responsiveness Problem

- Urgent requests do **not model real network traffic**
- $\Rightarrow$ Real adversary cannot use them to mount attacks
Non-Responsiveness Problem

Urgent requests do not model real network traffic
⇒ Real adversary cannot use them to mount attacks
⇒ Natural to expect adversary in model to answer immediately
Non-Responsiveness Problem

Urgent requests do not model real network traffic
⇒ Real adversary cannot use them to mount attacks
⇒ Natural to expect adversary in model to answer immediately

Non-Responsiveness Problem
However, adversary can:

- Activate protocol in unexpected way
Non-Responsiveness Problem

Urgent requests do **not** model real network traffic

⇒ Real adversary cannot use them to mount attacks
⇒ Natural to expect adversary in model to answer immediately

Non-Responsiveness Problem

However, adversary can:

- Activate protocol in unexpected way
Non-Responsiveness Problem

Urgent requests do **not model real network traffic**

⇒ Real adversary cannot use them to mount attacks
⇒ Natural to expect adversary in model to answer immediately

Non-Responsiveness Problem

However, adversary can:

- Activate protocol in unexpected way
Urgent requests do not model real network traffic

⇒ Real adversary cannot use them to mount attacks
⇒ Natural to expect adversary in model to answer immediately

Non-Responsiveness Problem
However, adversary can:

- Activate protocol in unexpected way
Non-Responsiveness Problem

Urgent requests do **not model real network traffic**

⇒ Real adversary cannot use them to mount attacks
⇒ Natural to expect adversary in model to answer immediately

Non-Responsiveness Problem

However, adversary can:

- Activate protocol in unexpected way
- Activate and change state of other parts of the protocol
Non-Responsiveness Problem

Urgent requests do not model real network traffic

⇒ Real adversary cannot use them to mount attacks
⇒ Natural to expect adversary in model to answer immediately

Non-Responsiveness Problem

However, adversary can:

- Activate protocol in unexpected way
- Activate and change state of other parts of the protocol
Non-Responsiveness Problem

Urgent requests do not model real network traffic

⇒ Real adversary cannot use them to mount attacks
⇒ Natural to expect adversary in model to answer immediately

Non-Responsiveness Problem

However, adversary can:

- Activate protocol in unexpected way
- Activate and change state of other parts of the protocol
Non-Responsiveness Problem

Urgent requests do not model real network traffic
⇒ Real adversary cannot use them to mount attacks
⇒ Natural to expect adversary in model to answer immediately

Non-Responsiveness Problem
However, adversary can:

- Activate protocol in unexpected way
- Activate and change state of other parts of the protocol
Non-Responsiveness Problem

Urgent requests do **not model real network traffic**

⇒ Real adversary cannot use them to mount attacks
⇒ Natural to expect adversary in model to answer immediately

Non-Responsiveness Problem

However, adversary can:

- Activate protocol in unexpected way
- Activate and change state of other parts of the protocol
Non-Responsiveness Problem

Urgent requests do not model real network traffic
⇒ Real adversary cannot use them to mount attacks
⇒ Natural to expect adversary in model to answer immediately

Non-Responsiveness Problem
However, adversary can:

- Activate protocol in unexpected way
- Activate and change state of other parts of the protocol
- Block parts of the protocol
Non-Responsiveness Problem

Urgent requests do not model real network traffic

⇒ Real adversary cannot use them to mount attacks
⇒ Natural to expect adversary in model to answer immediately

Non-Responsiveness Problem
However, adversary can:

• Activate protocol in unexpected way
• Activate and change state of other parts of the protocol
• Block parts of the protocol
Non-Responsiveness Problem

Urgent requests do not model real network traffic
⇒ Real adversary cannot use them to mount attacks
⇒ Natural to expect adversary in model to answer immediately

Non-Responsiveness Problem
However, adversary can:

- Activate protocol in unexpected way
- Activate and change state of other parts of the protocol
- Block parts of the protocol
Non-Responsiveness Problem

Urgent requests do **not model real network traffic**

⇒ Real adversary cannot use them to mount attacks
⇒ Natural to expect adversary in model to answer immediately

Non-Responsiveness Problem

However, adversary can:

- Activate protocol in unexpected way
- Activate and change state of other parts of the protocol
- Block parts of the protocol
Non-Responsiveness Problem

Urgent requests do not model real network traffic
⇒ Real adversary cannot use them to mount attacks
⇒ Natural to expect adversary in model to answer immediately

Protocol designers have to deal with unintended adversarial behavior:
Non-Responsiveness Problem

Urgent requests do **not model real network traffic**

⇒ Real adversary cannot use them to mount attacks

⇒ Natural to expect adversary in model to answer immediately

Protocol designers have to deal with unintended adversarial behavior:

- Difficult
Non-Responsiveness Problem

Urgent requests do not model real network traffic
⇒ Real adversary cannot use them to mount attacks
⇒ Natural to expect adversary in model to answer immediately

Protocol designers have to deal with unintended adversarial behavior:

• Difficult
• Not always possible
Non-Responsiveness Problem

Urgent requests do not model real network traffic

⇒ Real adversary cannot use them to mount attacks
⇒ Natural to expect adversary in model to answer immediately

Protocol designers have to deal with unintended adversarial behavior:

- Difficult
- Not always possible
- Complex specifications and proofs
Non-Responsiveness Problem

Urgent requests do not model real network traffic
⇒ Real adversary cannot use them to mount attacks
⇒ Natural to expect adversary in model to answer immediately

Protocol designers have to deal with unintended adversarial behavior:
• Difficult
• Not always possible
• Complex specifications and proofs
• Often ignored in the literature
Non-Responsiveness Problem

Urgent requests do **not model real network traffic**

⇒ Real adversary cannot use them to mount attacks
⇒ Natural to expect adversary in model to answer immediately

Protocol designers have to deal with unintended adversarial behavior:

- Difficult
- Not always possible
- Complex specifications and proofs
- Often ignored in the literature
  * Underspecified protocols
Non-Responsiveness Problem

Urgent requests do not model real network traffic
\[ \Rightarrow \] Real adversary cannot use them to mount attacks
\[ \Rightarrow \] Natural to expect adversary in model to answer immediately

Protocol designers have to deal with unintended adversarial behavior:

- Difficult
- Not always possible
- Complex specifications and proofs
- Often ignored in the literature
  - Underspecified protocols
  - Flawed proofs
Non-Responsiveness Problem

Urgent requests do **not model real network traffic**

⇒ Real adversary cannot use them to mount attacks
⇒ Natural to expect adversary in model to answer immediately

Protocol designers have to deal with unintended adversarial behavior:

- Difficult
- Not always possible
- Complex specifications and proofs
- Often ignored in the literature
  - Underspecified protocols
  - Flawed proofs
  - Hard to reuse functionalities
$F_{NIKE}$ from [Freire, Hesse, Hofheinz, 2014]

Upon input $(\text{init}, P_i, P_j)$ from $P_i$ [...] consider two cases:

- Corrupted session mode: if there exists $(\{P_i, P_j\}, K_{i,j})$ in $\Lambda_{\text{keys}}$, set $key = K_{i,j}$. Else, send $(\text{init}, P_i, P_j)$ to the adversary. After receiving $(\{P_i, P_j\}, K_{i,j})$ from the adversary, set $key = K_{i,j}$ and add $(\{P_i, P_j\}, K_{i,j})$ to $\Lambda_{\text{keys}}$.

- Honest session mode: [...]  

Return $(P_i, P_j, key)$ to $P_i$. 
Examples from the literature

$\mathcal{F}_{NIKE}$ from [Freire, Hesse, Hofheinz, 2014]

Upon input $(\text{init}, P_i, P_j)$ from $P_i$ [...] consider two cases:

- Corrupted session mode: if there exists $(\{P_i, P_j\}, K_{i,j})$ in $\Lambda_{\text{keys}}$, set $key = K_{i,j}$. Else, send $(\text{init}, P_i, P_j)$ to the adversary. After receiving $(\{P_i, P_j\}, K_{i,j})$ from the adversary, set $key = K_{i,j}$ and add $(\{P_i, P_j\}, K_{i,j})$ to $\Lambda_{\text{keys}}$.

- Honest session mode: [...]  

Return $(P_i, P_j, key)$ to $P_i$.

Lack of expressivity:

Functionality meant to model non-interactive key exchange, but is actually interactive
Examples from the literature

$\mathcal{F}_{\text{sok}}$ from [Chase, Lysyanskaya, 2006]

Upon receiving a value $(\text{Setup}, sid)$ from any party $P$, verify that $sid = (M_L, sid')$ for some $sid'$. If not, then ignore the request. Else, if this is the first time that $(\text{Setup}, sid)$ was received, hand $(\text{Setup}, sid)$ to the adversary; upon receiving $(\text{Algorithms}, sid, \text{Verify}, \text{Sign}, \text{Simsign}, \text{Extract})$ from the adversary, store these algorithms. Output the stored $(\text{Algorithms}, sid, \text{Sign}, \text{Verify})$ to $P$. 
Examples from the literature

$F_{\text{sok}}$ from [Chase, Lysyanskaya, 2006]

Upon receiving a value $(\text{Setup}, sid)$ from any party $P$, verify that $sid = (M_L, sid')$ for some $sid'$. If not, then ignore the request. Else, if this is the first time that $(\text{Setup}, sid)$ was received, **hand $(\text{Setup}, sid)$ to the adversary; upon receiving $(\text{Algorithms}, sid, \text{Verify}, \text{Sign}, \text{Simsign}, \text{Extract})$ from the adversary**, store these algorithms. Output the stored $(\text{Algorithms}, sid, \text{Sign}, \text{Verify})$ to $P$.

**Problems in proofs:**

Functionality might not receive algorithms, which is problematic for realizations based on $F_{\text{sok}}$
Examples from the literature

$\mathcal{F}_{D-Cert}$ from [Zhao, Zhang, Qin, Feng, 2014]

Upon receiving a value $(\text{Verify}, \text{sid}, m, \sigma)$ from some party $S'$, hand $(\text{Verify}, \text{sid}, m, \sigma)$ to the adversary. Upon receiving $(\text{Verified}, \text{sid}, m, \phi)$ from the adversary, do:

[...]

Output $(\text{Verified}, \text{sid}, m, f)$ to $S'$. 
$\mathcal{F}_{D\text{-Cert}}$ from [Zhao, Zhang, Qin, Feng, 2014]

Upon receiving a value $(\text{Verify}, sid, m, \sigma)$ from some party $S'$, hand $(\text{Verify}, sid, m, \sigma)$ to the adversary. Upon receiving $(\text{Verified}, sid, m, \phi)$ from the adversary, do:

[...]

Output $(\text{Verified}, sid, m, f)$ to $S'$.

**Unintended state changes and behavior:**

Adversary can corrupt signer of a signature during verification
⇒ Possible to accept invalid signatures
Examples from the literature

Realization of $\mathcal{F}_{\text{D-Cert}}$ from [Zhao, Zhang, Qin, Feng, 2014]

**Signature Protocol**: When activated with input $(\text{Sign}, sid, m)$, Party $S$ does:

[...]

$S$ **sends** $(\text{Sign}, (U, s), m)$ to $\mathcal{F}_{\text{SIG}}$. Upon receiving $(\text{Signature}, (U, s), m, \sigma)$ **from** $\mathcal{F}_{\text{SIG}}$, $S$ outputs $(\text{Signature}, sid, m, \sigma)$. 


Examples from the literature

Realization of $F_{D\text{-Cert}}$ from [Zhao, Zhang, Qin, Feng, 2014]

**Signature Protocol:** When activated with input $(\text{Sign}, sid, m)$, Party $S$ does:

[...]

$S$ sends $(\text{Sign}, (U, s), m)$ to $F_{\text{SIG}}$. Upon receiving $(\text{Signature}, (U, s), m, \sigma)$ from $F_{\text{SIG}}$, $S$ outputs $(\text{Signature}, sid, m, \sigma)$. 

\[ P \]
\[ \text{Signature} \]
\[ F_{\text{SIG}} \]
Examples from the literature

Realization of $\mathcal{F}_{D\text{-Cert}}$ from [Zhao, Zhang, Qin, Feng, 2014]

**Signature Protocol:** When activated with input $(\text{Sign}, \text{sid}, m)$, Party $S$ does:

[...] subroutine using urgent requests

$S$ sends $(\text{Sign}, (U, s), m)$ to $\mathcal{F}_{\text{SIG}}$. Upon receiving $(\text{Signature}, (U, s), m, \sigma)$ from $\mathcal{F}_{\text{SIG}}$, $S$ outputs $(\text{Signature}, \text{sid}, m, \sigma)$. 
Examples from the literature

Realization of $\mathcal{F}_{\text{D-Cert}}$ from [Zhao, Zhang, Qin, Feng, 2014]

**Signature Protocol:** When activated with input $(\text{Sign}, \text{sid}, m)$, Party $S$ does:

[...] subroutine using urgent requests

$S$ sends $(\text{Sign}, (U, s), m)$ to $\mathcal{F}_{\text{SIG}}$. Upon receiving $(\text{Signature}, (U, s), m, \sigma)$ from $\mathcal{F}_{\text{SIG}}$, $S$ outputs $(\text{Signature}, \text{sid}, m, \sigma)$. 
Examples from the literature

Realization of $\mathcal{F}_{D\text{-Cert}}$ from [Zhao, Zhang, Qin, Feng, 2014]

**Signature Protocol:** When activated with input $(\text{Sign}, sid, m)$, Party $S$ does:

[...]

$S$ sends $(\text{Sign}, (U, s), m)$ to $\mathcal{F}_{\text{SIG}}$. Upon receiving $(\text{Signature}, (U, s), m, \sigma)$ from $\mathcal{F}_{\text{SIG}}$, $S$ outputs $(\text{Signature}, sid, m, \sigma)$.

**Problem propagates to higher level protocols:**

Adversary is activated when calling a subroutine which models a local task.

The behavior of $\mathcal{P}$ in this case is undefined.
Examples from the literature

Realization of $\mathcal{F}_{D\text{-}Cert}$ from [Zhao, Zhang, Qin, Feng, 2014]

**Signature Protocol:** When activated with input $(\text{Sign}, \text{sid}, m)$, Party $S$ does:

[...]

$S$ sends $(\text{Sign}, (U, s), m)$ to $\mathcal{F}_{\text{SIG}}$. Upon receiving $(\text{Signature}, (U, s), m, \sigma)$ from $\mathcal{F}_{\text{SIG}}$, $S$ outputs $(\text{Signature}, \text{sid}, m, \sigma)$.

**Problem propagates to higher level protocols:**

Adversary is activated when calling a subroutine which models a local task.

The behavior of $\mathcal{P}$ in this case is undefined.
Examples from the literature

Realization of $F_{D-Cert}$ from [Zhao, Zhang, Qin, Feng, 2014]

**Signature Protocol:** When activated with input $(\text{Sign}, \text{sid}, m)$, Party $S$ does:

- [subroutine using urgent requests]

... 

$S$ sends $(\text{Sign}, (U, s), m)$ to $F_{SIG}$. Upon receiving $(\text{Signature}, (U, s), m, \sigma)$ from $F_{SIG}$, $S$ outputs $(\text{Signature}, \text{sid}, m, \sigma)$.

Problem propagates to higher level protocols:

Adversary is activated when calling a subroutine which models a local task.

The behavior of $P$ in this case is undefined.
Examples from the literature

Realization of $\mathcal{F}_{\text{D-Cert}}$ from [Zhao, Zhang, Qin, Feng, 2014]

**Signature Protocol:** When activated with input $(\text{Sign}, sid, m)$, Party $S$ does:

[...] subroutine using urgent requests

$S$ sends $(\text{Sign}, (U, s), m)$ to $\mathcal{F}_{\text{SIG}}$. Upon receiving $(\text{Signature}, (U, s), m, \sigma)$ from $\mathcal{F}_{\text{SIG}}$, $S$ outputs $(\text{Signature}, sid, m, \sigma)$.

**Problem propagates to higher level protocols:**

Adversary is activated when calling a subroutine which models a local task.

The behavior of $\mathcal{P}$ in this case is undefined.
Examples from the literature

Realization of $F_{D\text{-Cert}}$ from [Zhao, Zhang, Qin, Feng, 2014]

**Signature Protocol:** When activated with input $(\text{Sign, sid}, m)$, Party $S$ does:


[start of subroutine]

$S$ sends $(\text{Sign, (U, s), m})$ to $F_{\text{SIG}}$. Upon receiving $(\text{Signature, (U, s), m, } \sigma)$ from $F_{\text{SIG}}$, $S$ outputs $(\text{Signature, sid, m, } \sigma)$.

[end of subroutine]

**Problem propagates to higher level protocols:**

Adversary is activated when calling a subroutine which models a local task. The behavior of $P$ in this case is undefined.
Examples from the literature

Realization of $\mathcal{F}_{D\text{-Cert}}$ from [Zhao, Zhang, Qin, Feng, 2014]

**Signature Protocol**: When activated with input $(\text{Sign}, sid, m)$, Party $S$ does:

- subroutine using urgent requests

$S$ sends $(\text{Sign}, (U, s), m)$ to $\mathcal{F}_{\text{SIG}}$. Upon receiving $(\text{Signature}, (U, s), m, \sigma)$ from $\mathcal{F}_{\text{SIG}}$, $S$ outputs $(\text{Signature}, sid, m, \sigma)$.

Idealization cannot express properties of realization:

Unlike $\mathcal{F}_{\text{SIG}}$, realization $\mathcal{P}_{\text{SIG}}$ is indeed local.

Problems from previous slides do not exist when using $\mathcal{P}_{\text{SIG}}$. 
Dealing with the Non-Responsiveness Problem

Workarounds for full specifications:
Dealing with the Non-Responsiveness Problem

Workarounds for full specifications:

- Blocking requests while waiting for adversary
Dealing with the Non-Responsiveness Problem

Workarounds for full specifications:

- Blocking requests while waiting for adversary
- Sending error messages while waiting for adversary
Dealing with the Non-Responsiveness Problem

Workarounds for full specifications:

- Blocking requests while waiting for adversary
- Sending error messages while waiting for adversary
- Queuing new requests
Dealing with the Non-Responsiveness Problem

Workarounds for full specifications:

- Blocking requests while waiting for adversary
- Sending error messages while waiting for adversary
- Queuing new requests
- Code upload constructs
Dealing with the Non-Responsiveness Problem

Workarounds for full specifications:

- Blocking requests while waiting for adversary
- Sending error messages while waiting for adversary
- Queuing new requests
- Code upload constructs
- Resort to a default
Dealing with the Non-Responsiveness Problem

Workarounds for full specifications:

- Blocking requests while waiting for adversary
- Sending error messages while waiting for adversary
- Queuing new requests
- Code upload constructs
- Resort to a default

However:

- Not generally applicable
Dealing with the Non-Responsiveness Problem

Workarounds for full specifications:

- Blocking requests while waiting for adversary
- Sending error messages while waiting for adversary
- Queuing new requests
- Code upload constructs
- Resort to a default

However:

- Not generally applicable
- Usually need tailor-made solutions
Dealing with the Non-Responsiveness Problem

Workarounds for full specifications:

- Blocking requests while waiting for adversary
- Sending error messages while waiting for adversary
- Queuing new requests
- Code upload constructs
- Resort to a default

However:

- Not generally applicable
- Usually need tailor-made solutions
- Unnecessarily complicate specifications and proofs
Dealing with the Non-Responsiveness Problem

Workarounds for full specifications:

- Blocking requests while waiting for adversary
- Sending error messages while waiting for adversary
- Queuing new requests
- Code upload constructs
- Resort to a default

However:

- Not generally applicable
- Usually need tailor-made solutions
- Unnecessarily complicate specifications and proofs

Also:

Does not address unintended state changes or limited expressivity
We introduce responsive environments and responsive adversaries
We introduce **responsive environments** and **responsive adversaries**.
Our Solution

We introduce **responsive environments** and **responsive adversaries**

![Diagram](image.png)
Our Solution

We introduce responsive environments and responsive adversaries
Our Solution

We introduce **responsive environments** and **responsive adversaries**

![Diagram](image)
We introduce **responsive environments** and **responsive adversaries**
Non-Responsiveness Problem

Urgent requests do not model real network traffic
⇒ Real adversary cannot use them to mount attacks
⇒ Natural to expect adversary in model to answer immediately

Non-Responsiveness Problem

However, adversary can:

- Activate protocol in unexpected way
- Activate and change state of other parts of the protocol
- Block parts of the protocol
Non-Responsiveness Problem

Urgent requests do not model real network traffic
⇒ Real adversary cannot use them to mount attacks
⇒ Natural to expect adversary in model to answer immediately

Non-Responsiveness Problem
However, adversary can:

- Activate protocol in unexpected way
- Actuate and change state of other parts of the protocol
- Block parts of the protocol

\[ E \xrightarrow{\text{\textbullet}} P \xrightarrow{\text{\textbullet}} F \]
Non-Responsiveness Problem

Urgent requests do **not model real network traffic**
⇒ Real adversary cannot use them to mount attacks
⇒ Natural to expect adversary in model to answer immediately

Non-Responsiveness Problem

However, adversary can:

- Activate protocol in an expected way
- Activate and change state of other parts of the protocol
- Block parts of the protocol
Our Solution

We introduce responsive environments and responsive adversaries

- Natural solution, solves the problem entirely
Our Solution

We introduce responsive environments and responsive adversaries

- Natural solution, solves the problem entirely
- Simple, elegant, easy to use
Our Solution

We introduce *responsive environments* and *responsive adversaries*

- Natural solution, solves the problem entirely
- Simple, elegant, easy to use
- Solves problems from the literature
Our Solution

We introduce **responsive environments** and **responsive adversaries**

- Natural solution, solves the problem entirely
- Simple, elegant, easy to use
- Solves problems from the literature
- Applicable to all UC-style models (exemplified for UC, IITM, GNUC)
Our Solution

We introduce **responsive environments** and **responsive adversaries**

We provide detailed definitions and full proofs for the IITM model, including:
Our Solution

We introduce **responsive environments and responsive adversaries**

We provide detailed definitions and full proofs for the IITM model, including:

- Formal definitions of urgent requests, responsive environments, responsive adversaries
Our Solution

We introduce **responsive environments** and **responsive adversaries**

We provide detailed definitions and full proofs for the IITM model, including:

- Formal definitions of urgent requests, responsive environments, responsive adversaries
- Various security notions (dummy UC, strong simulatability, black-box simulatability, ...)

![Diagram showing the relationship between \( \mathcal{E} \), \( \mathcal{P} \), and \( \mathcal{F} \) with an interaction protocol and security notions.]
We introduce **responsive environments** and **responsive adversaries**

We provide detailed definitions and full proofs for the IITM model, including:

- Formal definitions of urgent requests, responsive environments, responsive adversaries
- Various security notions (dummy UC, strong simulatability, black-box simulatability, ...)
- Reflexivity and transitivity of security notions
Our Solution

We introduce **responsive environments** and **responsive adversaries**

We provide detailed definitions and full proofs for the IITM model, including:

- Formal definitions of urgent requests, responsive environments, responsive adversaries
- Various security notions (dummy UC, strong simulatability, black-box simulatability, ...)
- Reflexivity and transitivity of security notions
- Composition theorems
\( \mathcal{F}_{\text{NIKE}} \) from [Freire, Hesse, Hofheinz, 2014]

Upon input \((\text{init}, P_i, P_j)\) from \(P_i\) [...] consider two cases:

- Corrupted session mode: if there exists \((\{P_i, P_j\}, K_{i,j})\) in \(\Lambda_{\text{keys}}\), set \(key = K_{i,j}\). Else, send \((\text{init}, P_i, P_j)\) to the adversary. After receiving \((\{P_i, P_j\}, K_{i,j})\) from the adversary, set \(key = K_{i,j}\) and add \((\{P_i, P_j\}, K_{i,j})\) to \(\Lambda_{\text{keys}}\).

- Honest session mode: […]

Return \((P_i, P_j, key)\) to \(P_i\).
Solve problems from the literature

$\mathcal{F}_{\text{NIKE}}$ from [Freire, Hesse, Hofheinz, 2014]

Upon input $(\text{init}, P_i, P_j)$ from $P_i$ [...] consider two cases:

- Corrupted session mode: if there exists $(\{P_i, P_j\}, K_{i,j})$ in $\Lambda_{\text{keys}}$, set $\text{key} = K_{i,j}$. Else, send $(\text{Respond}, \text{init}, P_i, P_j)$ to the adversary. After receiving $(\{P_i, P_j\}, K_{i,j})$ from the adversary, set $\text{key} = K_{i,j}$ and add $(\{P_i, P_j\}, K_{i,j})$ to $\Lambda_{\text{keys}}$.

- Honest session mode: [...]  

Return $(P_i, P_j, \text{key})$ to $P_i$. 

\( \mathcal{F}_{NIKE} \) from [Freire, Hesse, Hofheinz, 2014]

Upon input \((\text{init}, P_i, P_j)\) from \(P_i\) [...] consider two cases:

- **Corrupted session mode:** if there exists \(\{P_i, P_j\}, K_{i,j}\) in \(\Lambda_{\text{keys}}\), set \(\text{key} = K_{i,j}\). Else, **send** \((\text{Respond}, \text{init}, P_i, P_j)\) to the adversary. **After receiving** \(\{P_i, P_j\}, K_{i,j}\) from the adversary, set \(\text{key} = K_{i,j}\) and **add** \(\{P_i, P_j\}, K_{i,j}\) to \(\Lambda_{\text{keys}}\).

- **Honest session mode:** [...] 

Return \((P_i, P_j, \text{key})\) to \(P_i\).
Solve problems from the literature

$F_{NIKE}$ from [Freire, Hesse, Hofheinz, 2014]

Upon input $(\text{init}, P_i, P_j)$ from $P_i$ [...] consider two cases:

- Corrupted session mode: if there exists $(\{P_i, P_j\}, K_{i,j})$ in $\Lambda_{\text{keys}}$, set $key = K_{i,j}$. Else, send $(\text{Respond}, \text{init}, P_i, P_j)$ to the adversary. After receiving $(\{P_i, P_j\}, K_{i,j})$ from the adversary, set $key = K_{i,j}$ and add $(\{P_i, P_j\}, K_{i,j})$ to $\Lambda_{\text{keys}}$.

- Honest session mode: [...]

Return $(P_i, P_j, key)$ to $P_i$.  

immediate response
$F_{sok}$ from [Chase, Lysyanskaya, 2006]

Upon receiving a value $(\text{Setup}, sid)$ from any party $P$, verify that $sid = (M_L, sid')$ for some $sid'$. If not, then ignore the request. Else, if this is the first time that $(\text{Setup}, sid)$ was received, hand $(\text{Respond}, \text{Setup}, sid)$ to the adversary; upon receiving $(\text{Algorithms}, sid, \text{Verify}, \text{Sign}, \text{Simsign}, \text{Extract})$ from the adversary, store these algorithms. Output the stored $(\text{Algorithms}, sid, \text{Sign}, \text{Verify})$ to $P$. 
\( \mathcal{F}_{\text{sok}} \) from [Chase, Lysyanskaya, 2006]

Upon receiving a value \((\text{Setup}, sid)\) from any party \(P\), verify that \(sid = (M_L, sid')\) for some \(sid'\). If not, then ignore the request. Else, if this is the first time that \((\text{Setup}, sid)\) was received, hand \((\text{Respond}, \text{Setup}, sid)\) to the adversary; upon receiving \((\text{Algorithms}, sid, \text{Verify}, \text{Sign}, \text{Simsign}, \text{Extract})\) from the adversary, store these algorithms. Output the stored \((\text{Algorithms}, sid, \text{Sign}, \text{Verify})\) to \(P\).
Solve problems from the literature

$F_{sok}$ from [Chase, Lysyanskaya, 2006]

Upon receiving a value $(\text{Setup}, sid)$ from any party $P$, verify that $sid = (M_L, sid')$ for some $sid'$. If not, then ignore the request. Else, if this is the first time that $(\text{Setup}, sid)$ was received, hand $(\text{Respond, Setup, } sid)$ to the adversary; upon receiving $(\text{Algorithms, } sid, \text{Verify, Sign, Simgen, Extract})$ from the adversary, store these algorithms. Output the stored $(\text{Algorithms, } sid, \text{Sign, Verify})$ to $P$. 
Solve problems from the literature

$\mathcal{F}_{D-Cert}$ from [Zhao, Zhang, Qin, Feng, 2014]

Upon receiving a value $(\text{Verify}, sid, m, \sigma)$ from some party $S'$, hand $(\text{Respond}, \text{Verify}, sid, m, \sigma)$ to the adversary. Upon receiving $(\text{Verified}, sid, m, \phi)$ from the adversary, do:

[...]

Output $(\text{Verified}, sid, m, f)$ to $S'$. 

Solve problems from the literature

$F_{D\text{-Cert}}$ from [Zhao, Zhang, Qin, Feng, 2014]

Upon receiving a value $(\text{Verify}, sid, m, \sigma)$ from some party $S'$, hand $(\text{Respond}, \text{Verify}, sid, m, \sigma)$ to the adversary. Upon receiving $(\text{Verified}, sid, m, \phi)$ from the adversary, do:

[...]

Output $(\text{Verified}, sid, m, f)$ to $S'$.
Solve problems from the literature

$F_{D-Cert}$ from [Zhao, Zhang, Qin, Feng, 2014]

Upon receiving a value $\langle \text{Verify, } sid, m, \sigma \rangle$ from some party $S'$, hand $\langle \text{Respond, Verify, } sid, m, \sigma \rangle$ to the adversary. Upon receiving $\langle \text{Verified, } sid, m, \phi \rangle$ from the adversary, do:

[...]

Output $\langle \text{Verified, } sid, m, f \rangle$ to $S'$.
Realization of $\mathcal{F}_{\text{D-Cert}}$ from [Zhao, Zhang, Qin, Feng, 2014]

**Signature Protocol:** When activated with input $(\text{Sign}, \text{sid}, m)$, Party $S$ does:

[...]

$S$ sends $(\text{Sign}, (U, s), m)$ to $\mathcal{F}_{\text{SIG}}$. Upon receiving $(\text{Signature}, (U, s), m, \sigma)$ from $\mathcal{F}_{\text{SIG}}$, $S$ outputs $(\text{Signature}, \text{sid}, m, \sigma)$. 
Solve problems from the literature

Realization of $\mathcal{F}_{D\text{-Cert}}$ from [Zhao, Zhang, Qin, Feng, 2014]

**Signature Protocol:** When activated with input $(\text{Sign}, sid, m)$, Party $S$ does:

[...]

$S$ sends $(\text{Sign}, (U, s), m)$ to $\mathcal{F}_{\text{SIG}}$. Upon receiving $(\text{Signature}, (U, s), m, \sigma)$ from $\mathcal{F}_{\text{SIG}}$, $S$ outputs $(\text{Signature}, sid, m, \sigma)$.

signature returned immediately
Solve problems from the literature

Realization of $\mathcal{F}_{D\text{-Cert}}$ from [Zhao, Zhang, Qin, Feng, 2014]

**Signature Protocol:** When activated with input $(\text{Sign}, sid, m)$, Party $S$ does:

[...]

$S$ sends $(\text{Sign}, (U, s), m)$ to $\mathcal{F}_{\text{SIG}}$. Upon receiving $(\text{Signature}, (U, s), m, \sigma)$ from $\mathcal{F}_{\text{SIG}}$, $S$ outputs $(\text{Signature}, sid, m, \sigma)$.

signature returned immediately

[Diagram of protocol flow]
Conclusion

- Protocols often exchange meta information by what we call *urgent requests*
Conclusion

- Protocols often exchange meta information by what we call **urgent requests**

- **Non-Responsiveness Problem:**
  Adversary might not answer immediately
Conclusion

- Protocols often exchange meta information by what we call **urgent requests**

- **Non-Responsiveness Problem:**
  Adversary might not answer immediately
  
  * Complicates protocol specifications and security proofs
Conclusion

- Protocols often exchange meta information by what we call **urgent requests**

- **Non-Responsiveness Problem:**
  Adversary might not answer immediately
  * Complicates protocol specifications and security proofs
  * No simple, general solution for adjusting protocols
Conclusion

- Protocols often exchange meta information by what we call **urgent requests**

- **Non-Responsiveness Problem:**
  Adversary might not answer immediately
  * Complicates protocol specifications and security proofs
  * No simple, general solution for adjusting protocols
  * Limited expressiveness
Conclusion

- Protocols often exchange meta information by what we call **urgent requests**

- **Non-Responsiveness Problem:**
  Adversary might not answer immediately
  * Complicates protocol specifications and security proofs
  * No simple, general solution for adjusting protocols
  * Limited expressiveness
  * Often ignored in the literature
Conclusion

- Protocols often exchange meta information by what we call urgent requests

- Non-Responsiveness Problem:
  Adversary might not answer immediately
  * Complicates protocol specifications and security proofs
  * No simple, general solution for adjusting protocols
  * Limited expressiveness
  * Often ignored in the literature
    - Underspecified protocols
    - Flawed proofs
    - Hard to reuse functionalities
Conclusion

- Our solution: Responsive environments/adversaries
  Easy to use, gets rid of the problem entirely, fixes literature
Conclusion

- Our solution: Responsive environments/adversaries
  Easy to use, gets rid of the problem entirely, fixes literature

Use our framework!
It makes your life much easier!
Conclusion

- Our solution: Responsive environments/adversaries
  Easy to use, gets rid of the problem entirely, fixes literature

Use our framework!
It makes your life much easier!

Thanks for your attention!