Categories for Cryptographic Composability

Riley Shahar
Advised by Angélica and Adam

Cryptography

The Simulation Paradigm

The Simulation Paradigm

In the *simulation paradigm*, the idea is to ask whether adversaries against an ideal functionality can "simulate" adversaries against a real protocol.

The Simulation Paradigm

In the *simulation paradigm*, the idea is to ask whether adversaries against an ideal functionality can "simulate" adversaries against a real protocol.

We need roughly four things:

The Simulation Paradigm

In the *simulation paradigm*, the idea is to ask whether adversaries against an ideal functionality can "simulate" adversaries against a real protocol.

We need roughly four things:

1. An ideal functionality we want to implement

The Simulation Paradigm

In the *simulation paradigm*, the idea is to ask whether adversaries against an ideal functionality can "simulate" adversaries against a real protocol.

We need roughly four things:

1. An ideal functionality we want to implement
2. A real protocol to analyze

The Simulation Paradigm

In the *simulation paradigm*, the idea is to ask whether adversaries against an ideal functionality can "simulate" adversaries against a real protocol.

We need roughly four things:

1. An ideal functionality we want to implement
2. A real protocol to analyze
3. A bound on adversarial strength

The Simulation Paradigm

In the *simulation paradigm*, the idea is to ask whether adversaries against an ideal functionality can "simulate" adversaries against a real protocol.

We need roughly four things:

1. An ideal functionality we want to implement
2. A real protocol to analyze
3. A bound on adversarial strength
4. A tool for comparing outcomes

The Simulation Paradigm

In the *simulation paradigm*, the idea is to ask whether adversaries against an ideal functionality can "simulate" adversaries against a real protocol.

We need roughly four things:

1. An ideal functionality we want to implement
2. A real protocol to analyze
3. A bound on adversarial strength
4. A tool for comparing outcomes

We will treat the more limited setting of N-party computation.

Ideal Functionalities

An *ideal functionality* is a function

$$f : \mathcal{X}_1, \ldots, \mathcal{X}_N \to \mathcal{Y}_1, \ldots, \mathcal{Y}_N.$$

Ideal Functionalities

An *ideal functionality* is a function

$$f : \mathcal{X}_1, \ldots, \mathcal{X}_N \to \mathcal{Y}_1, \ldots, \mathcal{Y}_N.$$

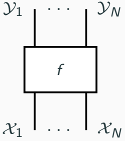
This is a "trusted third party."

Ideal Functionalities

An *ideal functionality* is a function

$$f : \mathcal{X}_1, \ldots, \mathcal{X}_N \to \mathcal{Y}_1, \ldots, \mathcal{Y}_N.$$

This is a "trusted third party." We can draw it as a gate:



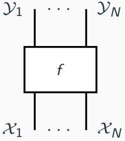
.

Ideal Functionalities

An *ideal functionality* is a function

$$f : \mathcal{X}_1, \ldots, \mathcal{X}_N \to \mathcal{Y}_1, \ldots, \mathcal{Y}_N.$$

This is a "trusted third party." We can draw it as a gate:



.

Running example: $f(x, y) = (*, xy)$.

Real Protocols

A *real protocol* is a list of N (interactive) algorithms.

Real Protocols

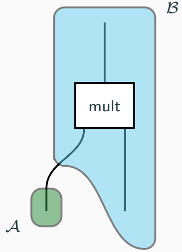
A *real protocol* is a list of N (interactive) algorithms.

$\mathcal{A}$		$\mathcal{B}$
Gets x		Gets y
	$\xrightarrow{x}$	
Output $*$		Output xy

Real Protocols

A *real protocol* is a list of N (interactive) algorithms.

$\mathcal{A}$		$\mathcal{B}$
Gets x		Gets y
	$\xrightarrow{x}$	
Output $*$		Output xy



Adversarial Behavior

We will treat *honest* adversaries. Usually you want to quantify over adversarial machines $\mathcal{A}'$.

Computational Indistinguishability

Computational Indistinguishability

Definition
Two probability ensembles $\{X_n\}$ and $\{Y_n\}$ (over the set A) are *computationally indistinguishable* if for any distinguisher $\mathcal{D}$,

$$\Big| \Pr[\mathcal{D}(X_n) = 1] - \Pr[\mathcal{D}(Y_n) = 1] \Big| = \mathsf{negl}(n).$$

We write $\{X_n\} \stackrel{c}{\equiv} \{Y_n\}$.

Simulation Security

Simulation Security

Definition
A protocol $\langle \mathcal{A}_1, \ldots, \mathcal{A}_N \rangle$ is *secure* if for each choice of inputs $x_1, \ldots, x_N$ and each party i, there exists a simulator $\mathcal{S}$ such that

$$\mathcal{S}(x_i, f_i(x_1, \ldots, x_N)) \stackrel{c}{\equiv} \mathsf{view}_i^{\langle \mathcal{A}_1, \ldots, \mathcal{A}_N \rangle}(x_1, \ldots, x_N).$$

Simulation Security

Definition
A protocol $\langle \mathcal{A}_1, \ldots, \mathcal{A}_N \rangle$ is *secure* if for each choice of inputs $x_1, \ldots, x_N$ and each party i, there exists a simulator $\mathcal{S}$ such that

$$\mathcal{S}(x_i, f_i(x_1, \ldots, x_N)) \stackrel{c}{\equiv} \mathsf{view}_i^{\langle \mathcal{A}_1, \ldots, \mathcal{A}_N \rangle}(x_1, \ldots, x_N).$$

We need a grading on inputs: this is the *security parameter*.

Simulation Security

Definition
A protocol $\langle \mathcal{A}_1, \ldots, \mathcal{A}_N \rangle$ is *secure* if for each choice of inputs $x_1, \ldots, x_N$ and each party i, there exists a simulator $\mathcal{S}$ such that

$$\mathcal{S}(x_i, f_i(x_1, \ldots, x_N)) \overset{c}{\equiv} \mathsf{view}_i^{\langle \mathcal{A}_1, \ldots, \mathcal{A}_N \rangle}(x_1, \ldots, x_N).$$

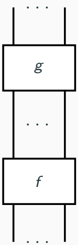
We need a grading on inputs: this is the *security parameter*.

Our running example is secure: the simulator can compute xy/y.

Composition

Composition

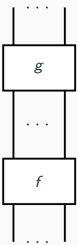
At least two ways to compose ideal functionalities:



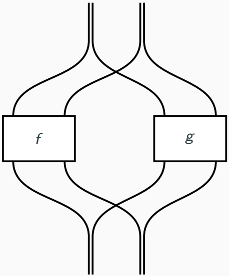
sequential composition

Composition

At least two ways to compose ideal functionalities:



and



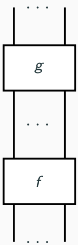
.

sequential composition

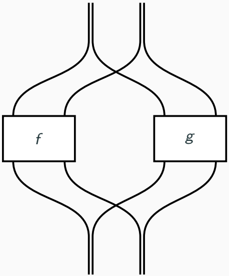
parallel composition

Composition

At least two ways to compose ideal functionalities:



and



.

sequential composition

parallel composition

Many more things to consider.

Universal Composability

Universal Composability

The protocol is talking to an *environment*.

Universal Composability

The protocol is talking to an *environment*.

Complexity costs:

- Low-level machine details

Universal Composability

The protocol is talking to an *environment*.

Complexity costs:

- Low-level machine details
- Many technical conditions on the composition theorem

Universal Composability

The protocol is talking to an *environment*.

Complexity costs:

- Low-level machine details
- Many technical conditions on the composition theorem
- High proof burdens

Why is this Hard?

Why is this Hard?

- Arbitrary adversarial behavior

Why is this Hard?

- Arbitrary adversarial behavior
- Asymptotic and polynomial bounds

Why is this Hard?

- Arbitrary adversarial behavior
- Asymptotic and polynomial bounds
- Very general composition operations

Category Theory

Categories

Categories

A category has:

- objects $x, y, z, \ldots$;
- morphisms $x \xrightarrow{f} y, \ldots$;
- identities $x \xrightarrow{1_x} x$;
- composition $x \xrightarrow{f} y \xrightarrow{g} z$.

Categories

A category has:

- objects $x, y, z, \ldots;$
- morphisms $x \xrightarrow{f} y, \ldots;$
- identities $x \xrightarrow{1_x} x;$
- composition $x \xrightarrow{f} y \xrightarrow{g} z.$

$$
\begin{tikzcd}
x \arrow[r, "f"] \arrow[rr, bend left, "gf"] & y \arrow[r, "g"] \arrow[rr, bend right, "hg"'] & z \arrow[r, "h"] & w
\end{tikzcd}
$$

Categories

A category has:

- objects $x, y, z, \ldots$;
- morphisms $x \xrightarrow{f} y, \ldots$;
- identities $x \xrightarrow{1_x} x$;
- composition $x \xrightarrow{f} y \xrightarrow{g} z$.

$$\begin{tikzcd}
x \arrow[r, "f"] \arrow[rr, bend left, "gf"] & y \arrow[r, "g"] \arrow[rr, bend right, "hg"'] & z \arrow[r, "h"] & w
\end{tikzcd}$$

$$\begin{tikzcd}
x \arrow[r, "1_x"] \arrow[dr, "f"'] & x \arrow[d, "f"] \\
& y
\end{tikzcd}
\qquad
\begin{tikzcd}
x \arrow[r, "f"] \arrow[dr, "f"'] & y \arrow[d, "1_y"] \\
& y.
\end{tikzcd}$$

Categories

Examples:

A category has:

- objects $x, y, z, \dots;$
- morphisms $x \xrightarrow{f} y, \dots;$
- identities $x \xrightarrow{1_x} x;$
- composition $x \xrightarrow{f} y \xrightarrow{g} z.$

Categories

A category has:

- objects $x, y, z, \ldots;$
- morphisms $x \xrightarrow{f} y, \ldots;$
- identities $x \xrightarrow{1_x} x;$
- composition $x \xrightarrow{f} y \xrightarrow{g} z.$

Examples:

- $\textsc{Set}$: sets and functions;

Categories

A category has:

- objects $x, y, z, \ldots$;
- morphisms $x \xrightarrow{f} y, \ldots$;
- identities $x \xrightarrow{1_x} x$;
- composition $x \xrightarrow{f} y \xrightarrow{g} z$.

Examples:

- $\textsc{Set}$: sets and functions;
- $\textsc{Grp}$, $\textsc{Top}$, $\textsc{Vect}_{\Bbbk}$, $\ldots$;

Categories

A category has:

- objects $x, y, z, \ldots$;
- morphisms $x \xrightarrow{f} y, \ldots$;
- identities $x \xrightarrow{1_x} x$;
- composition $x \xrightarrow{f} y \xrightarrow{g} z$.

Examples:

- $\textsc{Set}$: sets and functions;
- $\textsc{Grp}$, $\textsc{Top}$, $\textsc{Vect}_{\Bbbk}$, $\ldots$;
- every poset;

Categories

A category has:

- objects $x, y, z, \ldots;$
- morphisms $x \xrightarrow{f} y, \ldots;$
- identities $x \xrightarrow{1_x} x;$
- composition $x \xrightarrow{f} y \xrightarrow{g} z.$

Examples:

- $\textsc{Set}$: sets and functions;
- $\textsc{Grp}$, $\textsc{Top}$, $\textsc{Vect}_{\mathbb{k}}$, $\ldots;$
- every poset;
- $\textsc{Chem}$: multisets of chemicals and reactions (Baez-Pollard 2017);

Categories

A category has:

- objects $x, y, z, \ldots$;
- morphisms $x \xrightarrow{f} y, \ldots$;
- identities $x \xrightarrow{1_x} x$;
- composition $x \xrightarrow{f} y \xrightarrow{g} z$.

Examples:

- $\textsc{Set}$: sets and functions;
- $\textsc{Grp}$, $\textsc{Top}$, $\textsc{Vect}_{\Bbbk}$, $\ldots$;
- every poset;
- $\textsc{Chem}$: multisets of chemicals and reactions (Baez-Pollard 2017);
- functional programming languages & logics.

Functors

A *functor* $F : \mathcal{C} \to \mathcal{D}$ assigns:

- to each $x \in \mathcal{C}$, an $Fx \in \mathcal{D}$;
- to each $x \xrightarrow{f} y \in \mathcal{C}$, an $Fx \xrightarrow{Ff} Fy \in \mathcal{D}$;
- preserving identities and composition.

Functors

A *functor* $F : \mathcal{C} \to \mathcal{D}$ assigns:

- to each $x \in \mathcal{C}$, an $Fx \in \mathcal{D}$;
- to each $x \xrightarrow{f} y \in \mathcal{C}$, an $Fx \xrightarrow{Ff} Fy \in \mathcal{D}$;
- preserving identities and composition.

Examples:

Functors

A *functor* $F : \mathcal{C} \to \mathcal{D}$ assigns:

- to each $x \in \mathcal{C}$, an $Fx \in \mathcal{D}$;
- to each $x \xrightarrow{f} y \in \mathcal{C}$, an $Fx \xrightarrow{Ff} Fy \in \mathcal{D}$;
- preserving identities and composition.

Examples:

- the identity functor $1_{\mathcal{C}}$;

Functors

A *functor* $F : \mathcal{C} \to \mathcal{D}$ assigns:

- to each $x \in \mathcal{C}$, an $Fx \in \mathcal{D}$;
- to each $x \xrightarrow{f} y \in \mathcal{C}$, an $Fx \xrightarrow{Ff} Fy \in \mathcal{D}$;
- preserving identities and composition.

Examples:

- the identity functor $1_{\mathcal{C}}$;
- the power set (twice!);

Functors

A *functor* $F : \mathcal{C} \to \mathcal{D}$ assigns:

- to each $x \in \mathcal{C}$, an $Fx \in \mathcal{D}$;
- to each $x \xrightarrow{f} y \in \mathcal{C}$, an $Fx \xrightarrow{Ff} Fy \in \mathcal{D}$;
- preserving identities and composition.

Examples:

- the identity functor $1_{\mathcal{C}}$;
- the power set (twice!);
- forgetful functors;

Functors

A *functor* $F : \mathcal{C} \to \mathcal{D}$ assigns:

- to each $x \in \mathcal{C}$, an $Fx \in \mathcal{D}$;
- to each $x \xrightarrow{f} y \in \mathcal{C}$, an $Fx \xrightarrow{Ff} Fy \in \mathcal{D}$;
- preserving identities and composition.

Examples:

- the identity functor $1_{\mathcal{C}}$;
- the power set (twice!);
- forgetful functors;
- `List`, `Maybe`, etc.;

Functors

A *functor* $F : \mathcal{C} \to \mathcal{D}$ assigns:

- to each $x \in \mathcal{C}$, an $Fx \in \mathcal{D}$;
- to each $x \xrightarrow{f} y \in \mathcal{C}$, an $Fx \xrightarrow{Ff} Fy \in \mathcal{D}$;
- preserving identities and composition.

Examples:

- the identity functor $1_{\mathcal{C}}$;
- the power set (twice!);
- forgetful functors;
- `List`, `Maybe`, etc.;
- the free group, vector space, etc.

Symmetric Monoidal Categories

Symmetric Monoidal Categories

A *symmetric monoidal category* has:

- a product functor $\otimes$;
- a unit object I;
- "weak" associativity, unitality, and commutativity.

Symmetric Monoidal Categories

A *symmetric monoidal category* has:

- a product functor $\otimes$;
- a unit object I;
- "weak" associativity, unitality, and commutativity.

Examples:

Symmetric Monoidal Categories

A *symmetric monoidal category* has:

- a product functor $\otimes$;
- a unit object I;
- "weak" associativity, unitality, and commutativity.

Examples:

- $\textsc{Set}$ with $\times$ and $\{*\}$;

Symmetric Monoidal Categories

A *symmetric monoidal category* has:

- a product functor $\otimes$;
- a unit object I;
- "weak" associativity, unitality, and commutativity.

Examples:

- $\textsc{Set}$ with $\times$ and $\{*\}$;
- $\textsc{Vect}_{\mathbb{k}}$ with $\otimes$ and $\mathbb{k}$;

Symmetric Monoidal Categories

A *symmetric monoidal category* has:

- a product functor $\otimes$;
- a unit object I;
- "weak" associativity, unitality, and commutativity.

Examples:

- $\textsc{Set}$ with $\times$ and $\{*\}$;
- $\textsc{Vect}_{\mathbb{k}}$ with $\otimes$ and $\mathbb{k}$;
- $\textsc{Chem}$ with the union of multisets of molecules;

Symmetric Monoidal Categories

A *symmetric monoidal category* has:

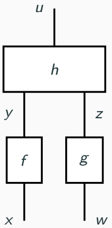
- a product functor $\otimes$;
- a unit object I;
- "weak" associativity, unitality, and commutativity.

Examples:

- $\textsc{Set}$ with $\times$ and $\{*\}$;
- $\textsc{Vect}_{\mathbb{k}}$ with $\otimes$ and $\mathbb{k}$;
- $\textsc{Chem}$ with the union of multisets of molecules;
- Concurrent languages with concurrent joining and the do-nothing program.

String Diagrams

Given $f : x \to y$, $g : w \to z$, and $h : y \otimes z \to u$, interpret



as $h(f \otimes g)$.

Categorical Cryptography

Three Layers of Abstraction

Three Layers of Abstraction

- An underlying symmetric monoidal category of computations;

Three Layers of Abstraction

- An underlying symmetric monoidal category of computations;
- A functorial construction of a SMC of protocols;

Three Layers of Abstraction

- An underlying symmetric monoidal category of computations;
- A functorial construction of a SMC of protocols;
- A security definition which works over any SMC.

Deterministic Computation

Deterministic Computation

First try: FinSet (finite sets and functions).

Deterministic Computation

First try: FinSet (finite sets and functions).

Second try: sets and computable functions.

Deterministic Computation

First try: FinSet (finite sets and functions).

Second try: sets and computable functions.

Third try: BinComp (sets of finite binary strings and computable functions).

Deterministic Computation

First try: FinSet (finite sets and functions).

Second try: sets and computable functions.

Third try: BinComp (sets of finite binary strings and computable functions).

Final try (Pavlovic 2014): Comp (binary-encoded sets and lifts of computable functions).

Efficient and Effectful Computation

New to us is a framework for categorically combining computational bounds and (monadic) effects.

Efficient and Effectful Computation

New to us is a framework for categorically combining computational bounds and (monadic) effects.

Name	Computational Bound	Effect
Enc	none	none
Comp	computability	none
CompStoch	computability	probability
Poly	poly-time computability	none
PPT	poly-time computability	probability

Non-interactive "Protocols"

The category $\mathcal{C} \times \mathcal{D}$ has:

- for objects, pairs (c, d) from $\mathcal{C}$ and $\mathcal{D}$;
- for morphisms, pairs (f, g) from $\mathcal{C}$ and $\mathcal{D}$;
- composition and identities componentwise.

States

States

The category $\mathrm{st}(\mathcal{C} \xrightarrow{F} \mathcal{D})$ has:

- for objects, maps $I \to Fx$ in $\mathcal{D}$;
- for morphisms $(I \xrightarrow{s} Fx) \to (I \xrightarrow{t} Fy)$, maps $x \xrightarrow{f} y$ in $\mathcal{C}$ such that $(Ff)s = t$;
- composition and identities as in $\mathcal{C}$.

States

The category $\mathrm{st}(\mathcal{C} \xrightarrow{F} \mathcal{D})$ has:

- for objects, maps $I \to Fx$ in $\mathcal{D}$;
- for morphisms $(I \xrightarrow{s} Fx) \to (I \xrightarrow{t} Fy)$, maps $x \xrightarrow{f} y$ in $\mathcal{C}$ such that $(Ff)s = t$;
- composition and identities as in $\mathcal{C}$.

We think of $\mathcal{C}$ as “free” processes in $\mathcal{D}$.

Combs

Objects: finite multisets of pairs of objects in $\mathcal{C}$.

Combs

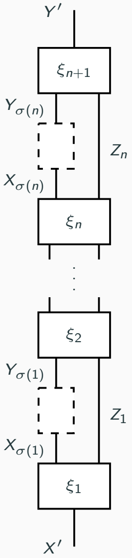
Objects: finite multisets of pairs of objects in $\mathcal{C}$.

Goal: states are multisets of morphisms.

Combs

Objects: finite multisets of pairs of objects in $\mathcal{C}$.

Goal: states are multisets of morphisms.



Assigning Resources

In n-comb($\mathcal{C}$), there are two ways we control the assignment of resources:

- a function α assigns each pair in the codomain some resources from the domain;
- a permutation σ orders the resources assigned to each comb.

Let's play with this!

The category of protocols

The category

$$\mathrm{prot}_N(\mathcal{C}) := \mathrm{st}(\text{n-comb}(\mathcal{C}^N) \xrightarrow{\text{n-comb}(\otimes^{N-1})} \text{n-comb}(\mathcal{C}))$$

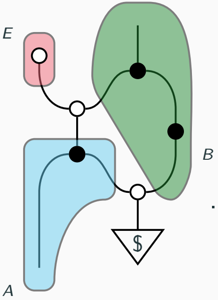
has:

- for objects, objects maps $(X_1 \otimes \cdots \otimes X_N) \to (Y_1 \otimes \cdots \otimes Y_N)$ in $\mathcal{C}$;
- for morphisms, combs drawn from $\mathcal{C}^N$ which preserve the chosen maps.

Take an object with maps , , , , and  forming a *Hopf object*, and a map 

forming an *integral*.

Take an object with maps , , , , and  forming a *Hopf object*, and a map  forming an *integral*.



Adversarial Behavior

Adversarial Behavior

Definition
An *attack model* $\mathbb{A}$ consists of, for each morphism f in $\mathcal{C}$, a collection of morphisms $\mathbb{A}f$ such that:

Adversarial Behavior

Definition
An *attack model* $\mathbb{A}$ consists of, for each morphism f in $\mathcal{C}$, a collection of morphisms $\mathbb{A}f$ such that:

1-4) $\cdots$

5) If $h \in \mathbb{A}(f \otimes g)$ such that $\operatorname{dom} h = x \otimes y$ for some objects x and y, then there is some $h' \in \mathbb{A}1_{\operatorname{cod} f \otimes \operatorname{cod} g}$, $f' \in \mathbb{A}f$, and $g' \in \mathbb{A}g$ such that $\operatorname{dom} f' = x$, $\operatorname{dom} g' = y$, and $h = h' \circ (f' \otimes g')$.

The Security Definition

Definition

Let $\mathcal{C} \xrightarrow{F} \mathcal{D} \xrightarrow{G} \mathcal{E}$ be a string of symmetric monoidal functors so that F is strong monoidal. Let $\mathbb{A}$ be an attack model on $\mathcal{D}$. Let $f : (x, s) \to (y, t)$ be a map in $\mathrm{st}(GF)$ and let a be a map in $\mathcal{D}$ with $\operatorname{dom} a = Fx$.

Definition

Let $\mathcal{C} \xrightarrow{F} \mathcal{D} \xrightarrow{G} \mathcal{E}$ be a string of symmetric monoidal functors so that F is strong monoidal. Let $\mathbb{A}$ be an attack model on $\mathcal{D}$. Let $f : (x, s) \to (y, t)$ be a map in $\mathrm{st}(GF)$ and let a be a map in $\mathcal{D}$ with $\mathrm{dom}\, a = Fx$. Then f is *secure against the attack* a if there is an attack $a' \in \mathbb{A}(1_{Fy})$ such that $\mathrm{dom}\, a' = Fy$, $\mathrm{cod}\, a = \mathrm{cod}\, a'$, and the following diagram commutes in $\mathcal{E}$:

$$\begin{tikzcd} I \arrow[r, "s"] \arrow[d, "t"'] & GFx \arrow[d, "Ga"] \\ GFy \arrow[r, "Ga'"'] & G\,\mathrm{cod}\, a. \end{tikzcd}$$

The Security Definition

Definition

Let $\mathcal{C} \xrightarrow{F} \mathcal{D} \xrightarrow{G} \mathcal{E}$ be a string of symmetric monoidal functors so that F is strong monoidal. Let $\mathbb{A}$ be an attack model on $\mathcal{D}$. Let $f : (x, s) \to (y, t)$ be a map in $\mathrm{st}(GF)$ and let a be a map in $\mathcal{D}$ with $\operatorname{dom} a = Fx$. Then f is *secure against the attack* a if there is an attack $a' \in \mathbb{A}(1_{Fy})$ such that $\operatorname{dom} a' = Fy$, $\operatorname{cod} a = \operatorname{cod} a'$, and the following diagram commutes in $\mathcal{E}$:

$$\begin{tikzcd} I \arrow[r, "s"] \arrow[d, "t"'] & GFx \arrow[d, "Ga"] \\ GFy \arrow[r, "Ga'"'] & G \operatorname{cod} a. \end{tikzcd}$$

Further, f is $\mathbb{A}$-*secure* if it is secure against all attacks in $\mathbb{A}Ff$ with domain Fx.

The Composition Theorem

Let $\mathcal{C} \xrightarrow{F} \mathcal{D} \xrightarrow{G} \mathcal{E}$ be a string of symmetric monoidal functors so that F is strong monoidal. Let $\mathbb{A}$ be an attack model on $\mathcal{D}$. Then the class of $\mathbb{A}$-secure maps is closed under monoidal product and sequential composition.

A 2-Categorical Generalization

We usually want security up to some reduction relation, e.g. computational indistinguishability.

A 2-Categorical Generalization

We usually want security up to some reduction relation, e.g. computational indistinguishability.

$$\begin{tikzcd} I \arrow[r, "s"] \arrow[d, "t"'] & GFx \arrow[d, "Ga"] \\ GFy \arrow[r, "Ga'"'] \arrow[ur, phantom, "\nearrow"] & G\,\mathrm{cod}\,a. \end{tikzcd}$$

A 2-Categorical Generalization

We usually want security up to some reduction relation, e.g. computational indistinguishability.

$$\begin{tikzcd} I \arrow[r, "s"] \arrow[d, "t"'] & GFx \arrow[d, "Ga"] \\ GFy \arrow[r, "Ga'"'] \arrow[ur, phantom, "\nearrow"] & G\,\mathrm{cod}\,a. \end{tikzcd}$$

What about *bi*categories?

Evaluation

Evaluation

Some concerns:

- Limitations of attack models and the security definition

Evaluation

Some concerns:

- Limitations of attack models and the security definition
- Limited forms of composition

Evaluation

Some concerns:

- Limitations of attack models and the security definition
- Limited forms of composition
- Many-round composition, dynamic linking, etc

Evaluation

Some concerns:

- Limitations of attack models and the security definition
- Limited forms of composition
- Many-round composition, dynamic linking, etc

Thanks for your time!