Towards Modular, Mechanized Proofs of Cryptographic Protocol Security

Alley Stoughton

Boston University

Joint Work with Ran Canetti, Assaf Kfoury and Mayank Varia

MACS Project Meeting

December 7, 2018

Project Background

- Securing computing systems is hard, especially those built using cryptographic schemes and protocols
- Proving the security of complex systems all in one “go” is unrealistic
- Instead a modular approach is needed
- Even with a modular approach, paper and pencil mathematics imperfectly copes with the complexity of security proofs
- Consequently, the mechanization of security proofs is desirable

Project Genesis and Goals

- Our project is an outgrowth of the MACS
- We are seeking to combine Universally Composable (UC) Security — as pioneered Canetti and others — with proof mechanization
  - UC Security is a refinement of the familiar real/ideal paradigm
- We are also interested in connecting such mechanized, modular high-level proofs with verified implementations

Real/Ideal Paradigm

- For a given protocol, two games:
  - a *real* game, based on the protocol
  - an *ideal* game, that's secure by construction
- The protocol is said to be *secure* iff a realistic adversary can't distinguish the real and ideal games, or, more precisely, iff the probability of it distinguishing the real and ideal games is negligible
- In a typical real/ideal formulation, the games drive the computation, calling the adversary at various points, and eventually returning the adversary's overall boolean judgment
- But this architecture doesn't support modularity

Universally Composable Security

- Universally Composable (UC) Security is a refinement of the real/ideal paradigm that *does* support modular proof development
- In UC, a real or ideal *functionality* interacts with
  - an *environment*, which supplies functionality inputs and consumes functionality outputs
  - an *adversary*, which is given certain powers to observe or corrupt the functionalities
- The environment and adversary also communicate
- In UC, the environment is in charge
- UC uses a coroutine style of message passing in which control is transferred along with data

Universal Composability

- We say that a real functionality `RF` *UC-realizes* an ideal functionality `IF` iff, for all adversaries `Adv`, there is a simulator `Sim` (which will be built out of `Adv` in a black box way), such that, for all environments `Env`, `Env` can't tell if it is interacting with
  - `RF`/`Adv` or
  - `IF`/`Sim`
- The UC Composition Theorem says that if `RF` UC-realizes `IF`, and ϕ is a functionality using `IF`, then $\phi^{\texttt{IF->RF}}$ UC-realizes ϕ
- UC was originally formalized in a computation theory style with systems of interactive Turing machines

Proof Mechanization

- Several frameworks have been developed for mechanizing cryptographic security proofs:
  - FCF (Petcher & Morrisett) is shallowly embedded in Coq
  - CryptHOL (Basin, Lochbihler & Sefidgar) is embedded in Isabelle/HOL
  - EasyCrypt (Barthe, Grégoire, Strub, …, Stoughton, …) is a standalone proof assistant, with a fairly small and well-studied TCB
    - Use of external SMT solvers is optional

EasyCrypt's Modules

- In EasyCrypt, cryptographic games are modeled as modules, which consists of global variables and procedures
- Modules may be parameterized, e.g., by adversaries
- Procedures are written in a simple imperative language, with while loops and random assignments (choosing values from probability sub-distributions)

EasyCrypt's Logics

- EasyCrypt has four logics:
  - a Probabilistic Relational Hoare Logic (pRHL) for proving relations between pairs of games
  - a Probabilisitic Hoare logic (pHL) for proving probabilistic facts about single games
  - an ordinary Hoare logic (HL)
  - an ambient higher-order logic for proving mathematical facts and connecting judgements from the other logics

EasyCrypt's Proofs and Theories

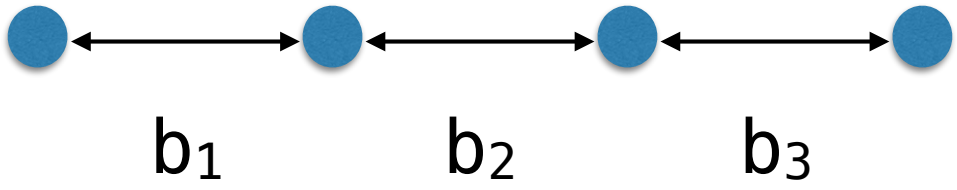
- Proofs are carried out using tactics, as in Coq
- Proofs are developed interactively, but may be replayed step-by-step or checked in batch mode
- Proofs may be structured as sequences of lemmas
- EasyCrypt theories may be used to group definitions, modules and lemmas together
- Theories may be specialized via cloning

Indistinguishability vs Real/Ideal Formulations

- Initially, most EasyCrypt proofs were about indistinguishability-style security formulations
- But Real/Ideal developments are also possible
  - E.g., my and Mayank's CSF 2017 paper on security of a simple secure database
  - Almeida etal.'s CCS 2017 paper on on secure function evaluation

Sequence of Games Approach

- In general, it takes some number of steps to connect the real and ideal games
  - Each step establishes an upper bound on the ability of the adversary to discriminate between the two games
  - The sum of these upper bounds is an upper bound on the ability of the adversary to discriminate the real and ideal games
  - Steps may be proved by reductions (maybe with their own game sequences), or upto-bad reasoning, …

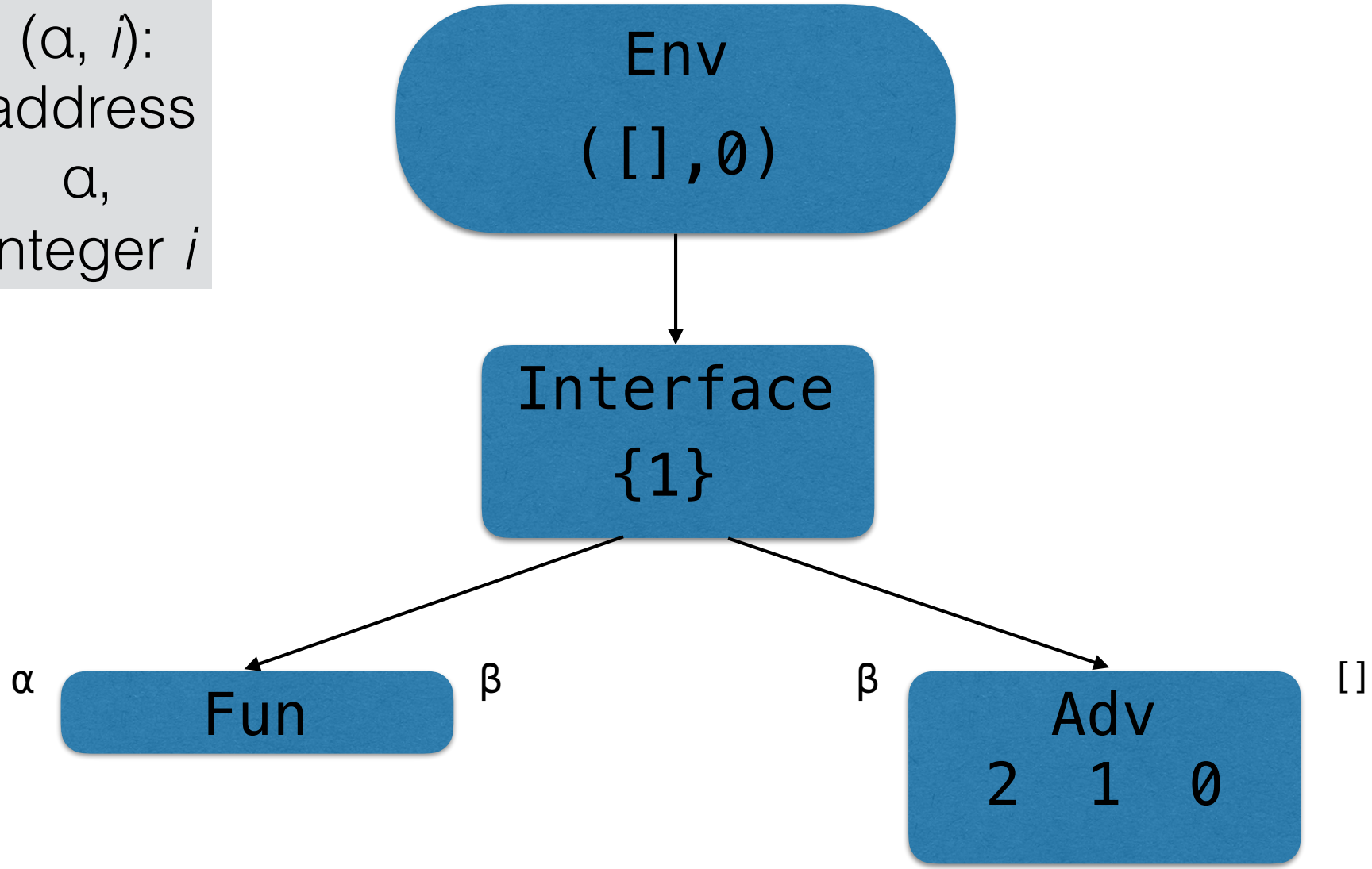


UC in EasyCrypt

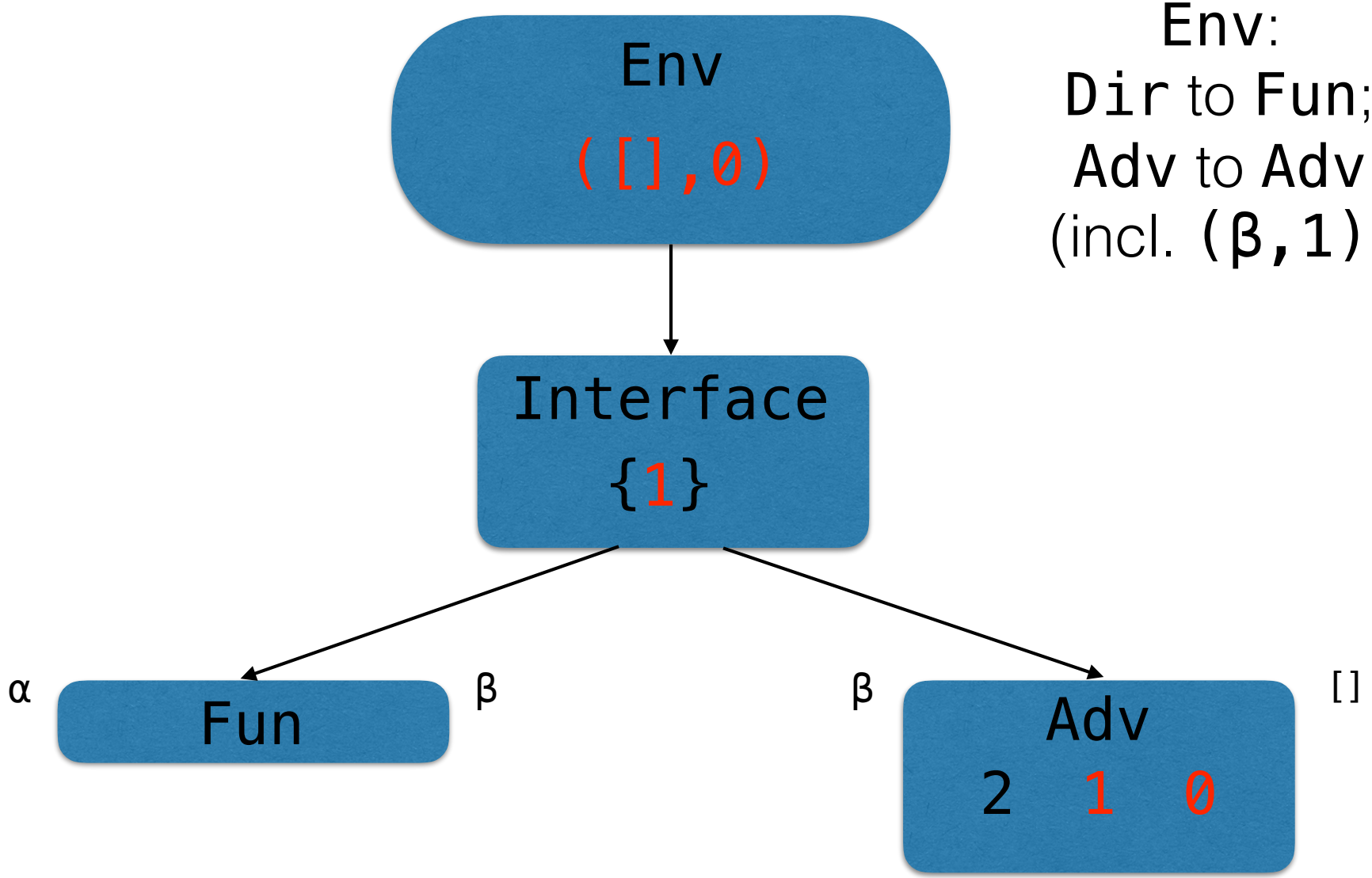
- We are in the early stages of researching how UC may be mechanized in EasyCrypt
- A major challenge is how to deal with UC's coroutine style of communication in EasyCrypt's procedural programming language
- Our approach is to give functionalities *addresses*, and to build abstractions that route messages to their destinations
- Two kinds of messages:
  - *direct* — supplying functionality inputs, returning their results
  - *adversarial* — other communications, including environment to adversary, functionality to adversary/ simulators

Message Flow Architecture

ports
(α, *i*):
address
α,
integer *i*

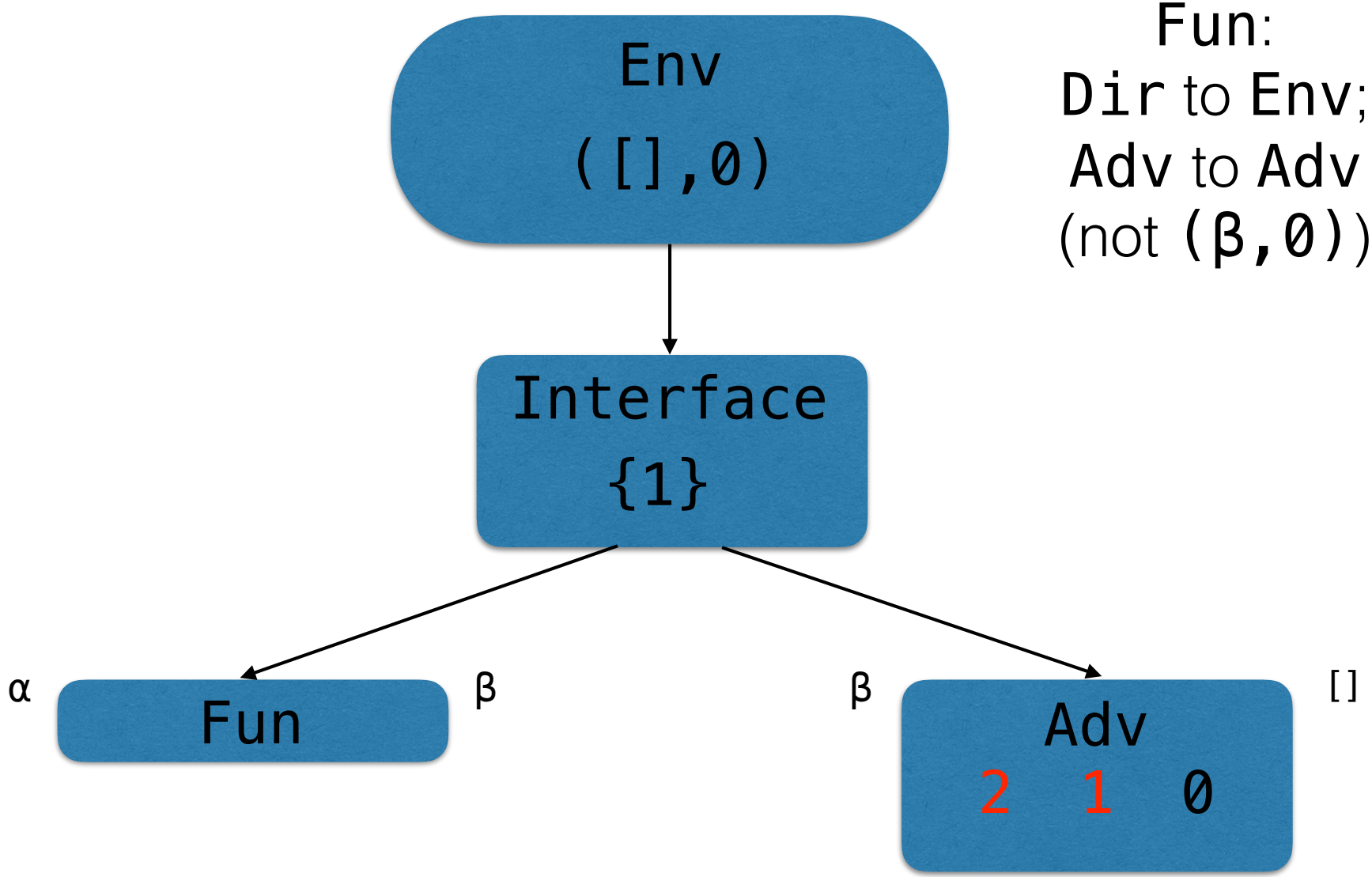


Message Flow Architecture



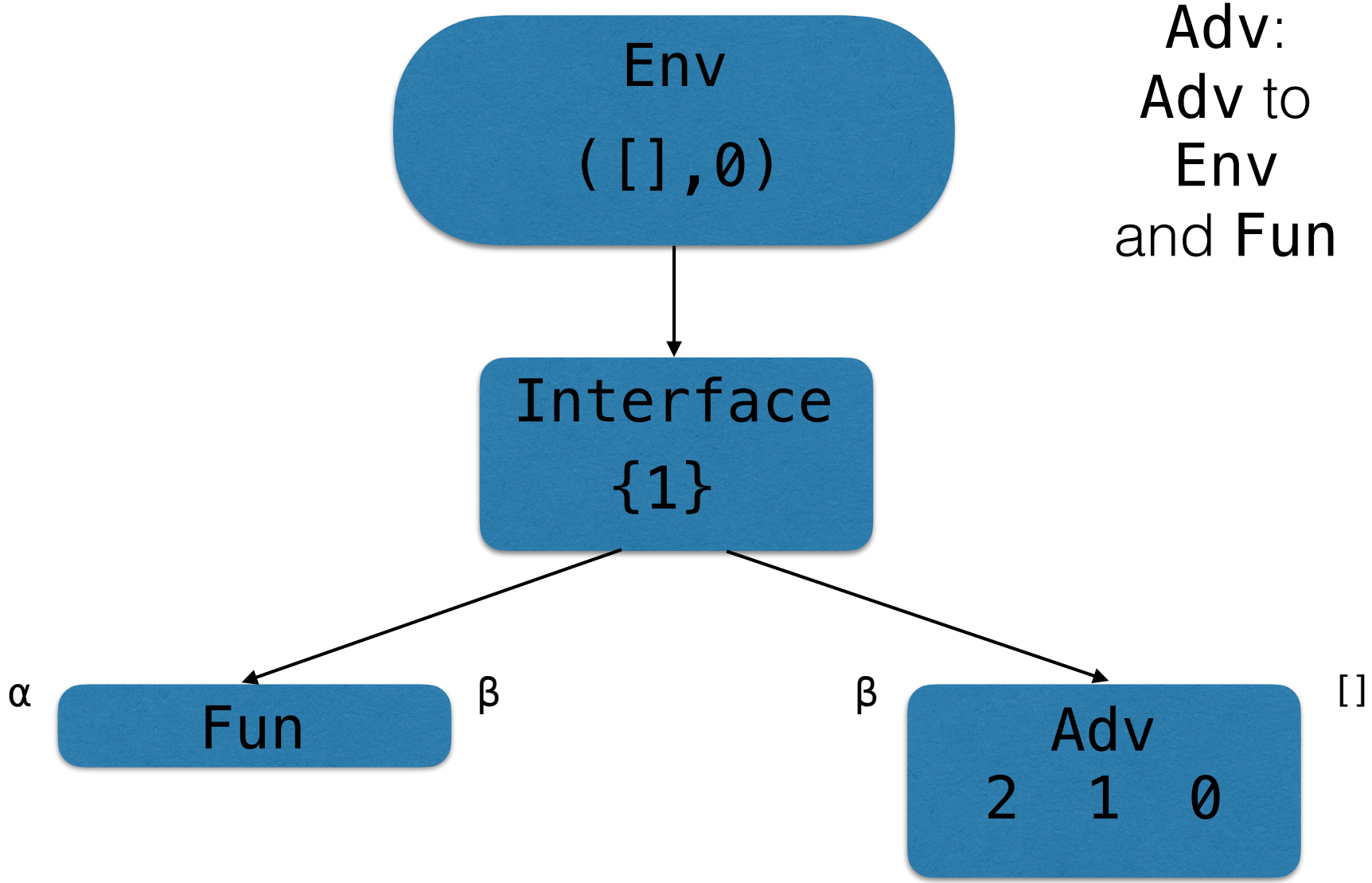
Env:
Dir to Fun;
Adv to Adv
(incl. (β,1))

Message Flow Architecture



Fun:
Dir to **Env**;
Adv to **Adv**
(not **(β,0)**)

Message Flow Architecture



Adv:
Adv to
Env
and **Fun**

Initial Case Study

- As a first case study, we are formalizing and proving the security of secure message communication using a one-time pad, where the one-time pad is agreed by the parties using Diffie-Hellman key exchange
- We first proved the UC security of Diffie-Hellman key exchange
- We are now applying this theorem to the security of secure message communication using a one-time pad
- In both cases, we are assuming an adversary that can observe and delay, but not corrupt, communications between protocol parties

Initial Case Study

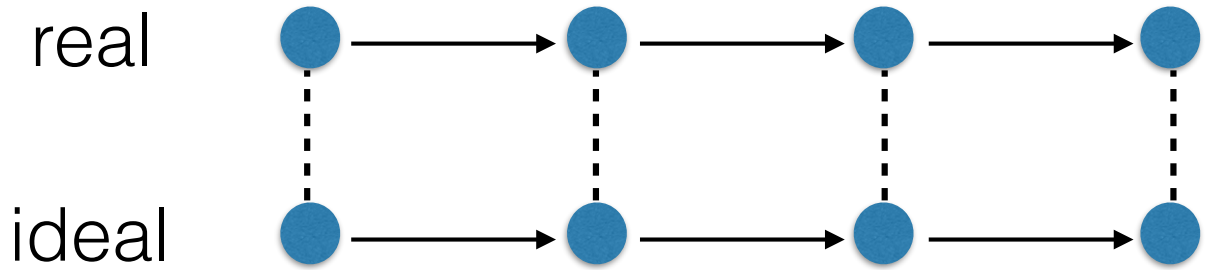
- Our case study is intended to test our architecture of embedding UC in EasyCrypt
- It involves:
  - an instance of the UC composition operator and theorem; and
  - the composition of simulators

Case Study Status

- Proof of Diffie-Hellman security is finished
- Proof of security of secure message communication has two gaps, which are rapidly being filled
- Case study is giving us confidence in the soundness of our architecture

Case Study Status

- Proofs use relational invariants allowing the related evolution of RF/Adv and IF/Sim(Adv) to be tracked
- Since the real and ideal worlds are structurally dissimilar, this means doing a lot of (manual) symbolic execution, guided by case analysis



Case Study Status

- Currently far too much effort needed to do proofs
  - Most pressing need: better support for guided symbolic execution needed
- Git repo:

`github.com/alleystoughton/EasyUC`

Research Plan

- Want to make it much easier to carry out UC security formalizations and proofs in EasyCrypt
- New EasyCrypt tactics for guided symbolic execution
- Domain specific language for UC functionalities involving coroutine communication
- Translating DSL into EasyCrypt
- Automate UC's Composition Theorem in EasyCrypt
- Adding DSL to EasyCrypt as alternative programming language
- Increasingly complex case studies…
- Connection with verified implementations

Our research group at Boston University,

led by Ran Canetti, Assaf Kfoury, Mayank Varia and myself,

is actively looking to recruit graduate students wanting to work at the

intersection of cryptography and formal methods/programming languages