SAUCY:

A CONCRETE FRAMEWORK FOR UNIVERSAL COMPOSABILITY

Kevin Liao[1], Abhiram Kothapalli[1], Matthew Hammer[2], Andrew Miller[1]
[1] UIUC, [2] CU Boulder

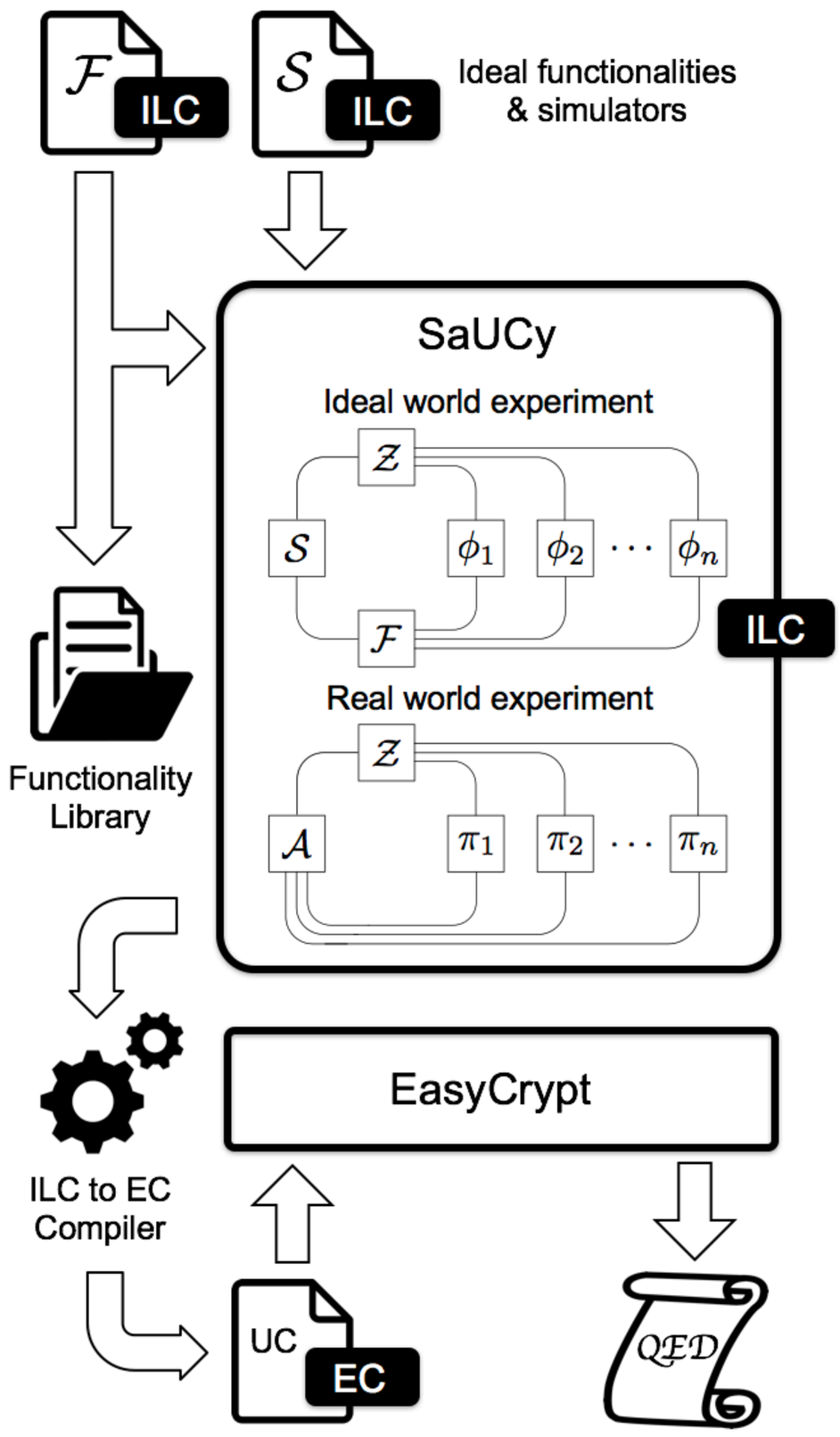
Motivation

- The Universal Composability (UC) framework is a general framework for analyzing cryptographic protocols.
- UC-secure protocols retain their guarantees even when concurrently composed with arbitrary other protocols.
- UC proofs exist as "pen-and-paper" proofs, making many of them error-prone and difficult to verify.
- Our goal is to place UC on a proper analytic foundation upon which we can construct mechanized UC proofs.

Objectives

- Interactive Lambda Calculus (ILC):
  - Programming language for constructing algorithmic entities in UC framework
  - Type system: Well-typed programs have valid executions in UC execution model
- Super Awesome Universal ComposabilitY (SaUCy)
  - UC execution model implemented in ILC
  - Library of ideal functionalities written in ILC
  - Compiler for translating ILC programs to proof languages (e.g., EasyCrypt modules)

SaUCy Framework



ILC Type System

Modes m, n, p ::= W | R | V (Write, Read and Value)

$\boxed{m \parallel n \Rightarrow p}$ The parallel composition of modes m and n is mode p.

$$\frac{m \parallel n \Rightarrow p}{n \parallel m \Rightarrow p}\ \text{sym} \qquad \frac{}{\mathsf{W} \parallel \mathsf{V} \Rightarrow \mathsf{W}}\ \text{wv} \qquad \frac{}{\mathsf{W} \parallel \mathsf{R} \Rightarrow \mathsf{W}}\ \text{wr} \qquad \frac{}{\mathsf{R} \parallel \mathsf{R} \Rightarrow \mathsf{R}}\ \text{rr}$$

$\boxed{m \,;\, n \Rightarrow p}$ The sequential composition of modes m and n is mode p.

$$\frac{}{\mathsf{V} \,;\, n \Rightarrow n}\ \text{v*} \qquad \frac{}{\mathsf{W} \,;\, \mathsf{V} \Rightarrow \mathsf{W}}\ \text{wv} \qquad \frac{}{\mathsf{R} \,;\, n \Rightarrow \mathsf{R}}\ \text{r*}$$

$$\frac{}{\mathsf{W} \,;\, \mathsf{R} \Rightarrow \mathsf{W}}\ \text{wr}$$

$\boxed{\Delta; \Gamma \vdash e : C \rhd m}$ Under Δ and Γ, expression e has type C and mode m.

$$\frac{\Delta; \Gamma \vdash v : A}{\Delta; \Gamma \vdash \mathtt{ret}(v) : \mathbf{F}\, A \rhd \mathsf{V}}\ \text{ret} \qquad \frac{\begin{array}{c} m_1 \,;\, m_2 \Rightarrow m_3 \\ \Delta_1; \Gamma \vdash e_1 : \mathbf{F}\, A \rhd m_1 \\ \Delta_2, x : A; \Gamma \vdash e_2 : C \rhd m_2 \end{array}}{\Delta_1, \Delta_2; \Gamma, x : A \vdash \mathtt{let}(e_1, x.e_2) : C \rhd m_3}\ \text{let}$$

$$\frac{\cdot; \Gamma \vdash v : A}{\cdot; \Gamma \vdash \mathtt{ret}(v) : \mathbf{F}\,(!A) \rhd \mathsf{V}}\ \text{ret!} \qquad \frac{\Delta_1; \Gamma \vdash v : !A \qquad \Delta_2; \Gamma, x : A \vdash e : C \rhd m}{\Delta_1, \Delta_2; \Gamma, x : A \vdash \mathtt{let!}(v, x.e) : C \rhd m}\ \text{let!}$$

$$\frac{\Delta; \Gamma \vdash e : C \rhd m}{\Delta; \Gamma \vdash \lambda x.\, e : A \to C \rhd m}\ \text{lam} \qquad \frac{\Delta_1; \Gamma \vdash v : A \qquad \Delta_2; \Gamma \vdash e : A \to C \rhd m}{\Delta_1, \Delta_2; \Gamma \vdash e\, v : C \rhd m}\ \text{app}$$

$$\frac{\Delta, x : (\mathbf{Rd}\, A \times !(\mathbf{Wr}\, A)); \Gamma \vdash e : C \rhd m}{\Delta; \Gamma \vdash \nu x.\, e : C \rhd m}\ \text{nu}$$

$$\frac{\Delta; \Gamma \vdash v : \mathbf{Rd}\, A}{\Delta \vdash \mathtt{rd}(v) : \mathbf{F}\,(A \times (\mathbf{Rd}\, A)) \rhd \mathsf{R}}\ \text{rd} \qquad \frac{\Delta_1; \Gamma \vdash v_1 : \mathbf{Wr}\, A \qquad \Delta_2; \Gamma \vdash v_2 : A}{\Delta_1, \Delta_2 \vdash \mathtt{wr}(v_1 \leftarrow v_2) : \mathbf{F}\,\mathsf{unit} \rhd \mathsf{W}}\ \text{wr}$$

$$\frac{\begin{array}{c} m_1 \parallel m_2 \Rightarrow m_3 \\ \Delta_1; \Gamma \vdash e_1 : C \rhd m_1 \\ \Delta_2; \Gamma \vdash e_2 : D \rhd m_2 \end{array}}{\Delta_1, \Delta_2 \vdash e_1 \mid\rhd e_2 : D \rhd m_3}\ \text{fork} \qquad \frac{\begin{array}{c} \Delta_1; \Gamma \vdash e_1 : C \rhd \mathsf{R} \\ \Delta_2; \Gamma \vdash e_2 : C \rhd \mathsf{R} \end{array}}{\Delta_1, \Delta_2 \vdash e_1 \oplus e_2 : C \rhd \mathsf{R}}\ \text{choice}$$

Example Functionality in ILC

Functionality $\mathcal{F}_{\text{COM}}$

$\mathcal{F}_{\text{COM}}$ proceeds as follows, running with parties $P_1, \ldots, P_n$ and an adversary S.

1. Upon receiving a value $(\mathtt{Commit}, sid, P_i, P_j, b)$ from P_i, where $b \in \{0, 1\}$, record the value b and send the message $(\mathtt{Receipt}, sid, P_i, P_j)$ to P_j and S. Ignore any subsequent Commit messages.
2. Upon receiving a value $(\mathtt{Open}, sid, P_i, P_j)$ from P_i, proceed as follows: If some value b was previously recorded, then send the message $(\mathtt{Open}, sid, P_i, P_j, b)$ to P_j and S and halt. Otherwise halt.

ILC $\mathcal{F}_{\text{COM}}$

```
let F_com = lam S .
  let ('Commit, sid, P_i, P_j, b) = rd ?p2f in
    req mem b {0,1} in
    wr (('Receipt, sid, P_i, P_j), {P_j, S}) → ?f2p ;
    let ('Open, sid, P_i, P_j) = rd ?p2f in
    wr (('Open, sid, P_i, P_j, b), {P_j, S}) → ?f2p
in
  nu f2p, p2f .
    |▷ (F_com S)
```

Future Work

- Full ILC and SaUCy implementations
- Use ILC and SaUCy to analyze complex protocols, especially involving blockchain protocols and smart contracts

This material is based upon work supported by the National Science Foundation Graduate Research Fellowship.