SaUCy

ANONYMOUS AUTHOR(S)

Text of abstract ....

Additional Key Words and Phrases: keyword1, keyword2, keyword3

1 INTRODUCTION

UC paper [Canetti 2001]. *TODO: Lots!*

2 OVERVIEW

3 ILC

Definition 3.1 (Protocol Emulation). Let π and ϕ be probabilistic polynomial time (p.p.t) protocols. We say that π UC-emulates ϕ if for any p.p.t. adversary $\mathcal{A}$ there exists a p.p.t. ideal-process adversary $\mathcal{S}$ such that for any balanced PPT environment $\mathcal{Z}$ we have:

$$\textsc{Exec}_{\phi,\mathcal{S},\mathcal{Z}} \approx \textsc{Exec}_{\pi,\mathcal{A},\mathcal{Z}}.$$

Definition 3.2 (Code-Based Protocol Emulation). Let $\mathcal{L}_\pi$ and $\mathcal{L}_\phi$ be p.p.t. program terms. We say that $\mathcal{L}_\pi$ UC-emulates $\mathcal{L}_\phi$ if for any p.p.t. adversary term $\mathcal{L}_\mathcal{A}$ there exists a p.p.t. ideal-process term $\mathcal{L}_\mathcal{S}$ such that for any balanced p.p.t. environment term $\mathcal{L}_\mathcal{Z}$ we have:

$$\textsc{Trace}_{\mathcal{L}_\phi,\mathcal{L}_\mathcal{S},\mathcal{L}_\mathcal{Z}} \approx_\epsilon \textsc{Trace}_{\mathcal{L}_\pi,\mathcal{L}_\mathcal{A},\mathcal{L}_\mathcal{Z}}.$$

How to define $\approx_\epsilon$ relation? Observational equivalence is a stronger notion than computational equivalance [Comon-Lundh and Cortier 2008]. However, they also show that indistinguishability (based on games) is soundly abstracted by trace equivalence.

4 METATHEORY

(1) Type soundness
(2) Confluence

5 IMPLEMENTATION

(1) Bidirectional type checker
(2) Replication

6 EXPERIMENTS

(1) Impossibility of UC commitments using standard assumptions [Canetti and Fischlin 2001].
(2) UC commitments construction using CRS

2018. 2475-1421/2018/1-ART1 $15.00
https://doi.org/

```
execUC(ℰ, π, 𝒜, ℱ)
ν z2p z2f z2a p2f p2a a2f.
// The environment chooses SID, conf, and corrupted parties
let (Corrupted, SID, conf) = ℰ{z2p, z2a, z2f}
// The protocol determines conf′
let conf′ = π.cmap(SID, conf)
| 𝒜{SID, conf, Corrupted, a2z, a2p, a2f}
| ℱ{SID, conf′, Corrupted, f2z, f2p, f2a}
// Create instances of parties on demand
let partyMap = ref empty
let newPartyPID = do
  ν f2pp z2pp.
  @partyMap[PID].f2p := f2pp
  @partyMap[PID].z2p := z2pp
  | forever do {m ← pp2f; (PID, m) → f2p}
  | forever do {m ← pp2z; (PID, m) → z2p}
  | π{SID, conf, p2f/pp2z, p2z/pp2z}
let getParty PID =
  if PID ∉ partyMap then newParty PID
  return @partyMap[PID]
| forever do
  (PID, m) ← z2p
  if PID ∈ Corrupted then Z2P(PID, m) → p2a
  else m → (getParty PID).z2p
| forever do
  (PID, m) ← f2p
  if PID ∈ Corrupted then F2P(PID, m) → p2a
  else m → (getParty PID).f2p
| forever do
  | A2P2F(PID, m) ← a2p
   if PID ∈ Corrupted then (PID, m) → p2f
  | A2P2Z(PID, m) ← a2p
   if PID ∈ Corrupted then (PID, m) → p2z
```

Fig. 1. Definition of the SaUCy execution model. The environment, are run as concurrent processes. A new instance of the protocol π is created, on demand, for each party PID. Messages sent to honest parties are routed according to their PID; messages sent to corrupted parties are instead diverted to the adversary.

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

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

(1) Upon receiving a value (Commit, sid, P_i, P_j, b) from P_i, where $b \in \{0, 1\}$, record the value b and send the message (Receipt, sid, P_i, P_j) to P_j and S. Ignore any subsequent Commit messages.

(2) Upon receiving a value (Open, sid, P_i, P_j) from P_i, proceed as follows: If some value b was previously recorded, then send the message (Open, sid, P_i, P_j, b) to P_j and S and halt. Otherwise halt.

```
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)
```

7 RELATED WORK

EasyCrypt [Barthe et al. 2011], CertiCrypt [Barthe et al. 2009], CryptoVerif [Blanchet 2007], ProVerif [Blanchet 2005], RF* [Barthe et al. 2014], Cryptol [Lewis and Martin 2003], code-based game-playing proofs [Bellare and Rogaway 2006], symbolic UC [Böhl and Unruh 2016]

8 CONCLUSION

9 FUTURE WORK

REFERENCES

Gilles Barthe, Cédric Fournet, Benjamin Grégoire, Pierre-Yves Strub, Nikhil Swamy, and Santiago Zanella-Béguelin. 2014. Probabilistic relational verification for cryptographic implementations. In *ACM SIGPLAN Notices*, Vol. 49. ACM, 193–205.

Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago Zanella Béguelin. 2011. Computer-aided security proofs for the working cryptographer. In *Annual Cryptology Conference*. Springer, 71–90.

Gilles Barthe, Benjamin Grégoire, and Santiago Zanella Béguelin. 2009. Formal certification of code-based cryptographic proofs. *ACM SIGPLAN Notices* 44, 1 (2009), 90–101.

Mihir Bellare and Phillip Rogaway. 2006. The security of triple encryption and a framework for code-based game-playing proofs. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 409–426.

Bruno Blanchet. 2005. ProVerif automatic cryptographic protocol verifier user manual. *CNRS, Departement dInformatique, Ecole Normale Superieure, Paris* (2005).

Bruno Blanchet. 2007. CryptoVerif: Computationally sound mechanized prover for cryptographic protocols. In *Dagstuhl seminar âĂIJFormal Protocol Verification Applied*. 117.

Florian Böhl and Dominique Unruh. 2016. Symbolic universal composability. *Journal of Computer Security* 24, 1 (2016), 1–38.

Ran Canetti. 2001. Universally composable security: A new paradigm for cryptographic protocols. In *Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on*. IEEE, 136–145.

Ran Canetti and Marc Fischlin. 2001. Universally composable commitments. In *Annual International Cryptology Conference*. Springer, 19–40.

Hubert Comon-Lundh and Véronique Cortier. 2008. Computational soundness of observational equivalence. In *Proceedings of the 15th ACM conference on Computer and communications security*. ACM, 109–118.

Jeffrey R Lewis and Brad Martin. 2003. Cryptol: High assurance, retargetable crypto development and validation. In *Military Communications Conference, 2003. MILCOM'03. 2003 IEEE*, Vol. 2. IEEE, 820–825.

A APPENDIX

Value Types	$A, B ::= x$	Value variable
	$\mid \mathsf{unit}$	Unit value
	$\mid \mathsf{nat}$	Natural number
	$\mid A \times B$	Product
	$\mid A + B$	Sum type
	$\mid !A$	Intuitionistic type
	$\mid \mathbf{Rd}\, A$	Read channel
	$\mid \mathbf{Wr}\, A$	Write channel
	$\mid \mathbf{U}\, C$	Thunk type
Computation Types	$C, D ::= A \rightarrow C$	Value-consuming computation
	$\mid \mathbf{F}\, A$	Value-producing computation
Linear Typing Contexts	$\Delta ::= \cdot \mid \Delta, x : A$	
Intuitionisitic Typing Contexts	$\Gamma ::= \cdot \mid \Gamma, x : A$	

Fig. 2. Syntax of types and typing contexts

Values	$v ::= x$	
	$\mid \texttt{()}$	Unit value
	$\mid n$	Natural number
	$\mid \texttt{(}v_1\texttt{,}v_2\texttt{)}$	Pair of values
	$\mid \texttt{inj}_i\texttt{(}v\texttt{)}$	Injected value
	$\mid \texttt{chan(}c\texttt{)}$	Channel (either read or write end)
	$\mid \texttt{thunk(}e\texttt{)}$	Thunk (suspended, closed expression)
Expressions	$e ::= \texttt{split(}v, x_1.x_2.e\texttt{)}$	Pair elimination
	$\mid \texttt{case(}v, x_1.e_1, x_2.e_2\texttt{)}$	Injection elimination
	$\mid \texttt{ret(}v\texttt{)}$	Value-producing computation
	$\mid \texttt{let(}e_1, x.e_2\texttt{)}$	Let-binding/sequencing
	$\mid e\ v$	Function application
	$\mid \lambda x.\, e$	Function abstraction
	$\mid \texttt{force(}v\texttt{)}$	Unsuspend (force) a thunk
	$\mid \texttt{wr(}v_1 \leftarrow v_2\texttt{)}$	Write channel v_1 with value v_2
	$\mid \texttt{rd(}v\texttt{)}$	Read channel v
	$\mid \nu x.\, e$	Allocate channel as x in e
	$\mid e_1 \mid\!\triangleright\ e_2$	Fork e_1, continue as e_2
	$\mid e_1 \oplus e_2$	External choice between e_1 and e_2

Fig. 3. Syntax of values and expressions

Modes $m, n, p ::= \mathsf{W} \mid \mathsf{R} \mid \mathsf{V}$ (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.

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

Note that in particular, the following mode compositions are *not derivable*:

- $\mathsf{W} \parallel \mathsf{W} \Rightarrow p$ is *not* derivable for any mode p
- $\mathsf{W}\ ;\ \mathsf{W} \Rightarrow p$ is *not* derivable for any mode p

Fig. 4. Syntax of modes; sequential and parallel mode composition.

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

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

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

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

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

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

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

$$
\begin{array}{ll}
\text{Channels} & \Sigma ::= \varepsilon \mid \Sigma, c \\
\text{Process pool} & \pi ::= \varepsilon \mid \pi, e \\
\text{Configurations} & C ::= \langle \Sigma; \pi \rangle \\
\text{Evaluation contexts} & E ::= \mathtt{let}(E, x.e) \\
& \quad \mid E\, v \\
& \quad \mid \bullet \\
\text{Read contexts} & R ::= \mathtt{rd}(\mathsf{chan}(c)) \oplus R \\
& \quad \mid R \oplus \mathtt{rd}(\mathsf{chan}(c)) \\
& \quad \mid \bullet
\end{array}
$$

$\boxed{e \longrightarrow e'}$ Expression e_1 reduces to e_2.

$$
\frac{}{\mathtt{let}(\mathtt{ret}(v), x.e) \longrightarrow [v/x]e}\ \text{let}
\qquad
\frac{}{(\lambda x.\, e)\, v \longrightarrow [v/x]e}\ \text{app}
\qquad
\frac{}{\mathtt{force}(\mathtt{thunk}(e)) \longrightarrow e}\ \text{force}
$$

$$
\frac{}{\mathtt{split}((v_1, v_2), x.y.e) \longrightarrow [v_1/x][v_2/y]e}\ \text{split}
\qquad
\frac{}{\mathtt{case}(\mathtt{inj}_i(v), x_1.e_1, x_2.e_2) \longrightarrow e_i[v/x_i]}\ \text{case}
$$

$\boxed{C_1 \equiv C_2}$ Configurations C_1 and C_2 are equivalent.

$$
\frac{\pi_1 \equiv_{\text{perm}} \pi_2}{\langle \Sigma; \pi_1 \rangle \equiv \langle \Sigma; \pi_2 \rangle}\ \text{permProcs}
$$

$\boxed{C_1 \longrightarrow C_2}$ Configuration C_1 reduces to C_2.

$$
\frac{e \longrightarrow e'}{\langle \Sigma; \pi, E[e] \rangle \longrightarrow \langle \Sigma; \pi, E[e]' \rangle}\ \text{local}
\qquad
\frac{}{\langle \Sigma; \pi, E[e_1 \mathrel{|\!\rhd} e_2] \rangle \longrightarrow \langle \Sigma; \pi, e_1, E[e_2] \rangle}\ \text{fork}
$$

$$
\frac{C_1 \equiv C_1' \qquad C_1' \longrightarrow C_2 \qquad C_2 \equiv C_2'}{C_1 \longrightarrow C_2'}\ \text{congr}
$$

$$
\frac{c \notin \Sigma}{\langle \Sigma; \pi, E[\nu x.\, e] \rangle \longrightarrow \langle \Sigma, c; \pi, E[[(\mathsf{chan}(c), \mathsf{chan}(c))/x]e] \rangle}\ \text{nu}
$$

$$
\frac{}{\langle \Sigma; \pi, E_1[R[\mathtt{rd}(\mathsf{chan}(c))]], E_2[\mathtt{wr}(\mathsf{chan}(c) \leftarrow v)] \rangle \longrightarrow \langle \Sigma; \pi, E_1[v], E_2[()] \rangle}\ \text{rw}
$$