Foundations for Cryptographic Reductions in CCSA Logics

David Baelde[1] Adrien Koutsos[2] Justine Sauvage[2]

[1]Univ Rennes, CNRS, IRISA, France

[2]Inria, France

Context: Cryptographic reductions

- Cryptographic protocols are critical parts of communication systems.
  $\Rightarrow$ **verification** provides strong security guarantees.

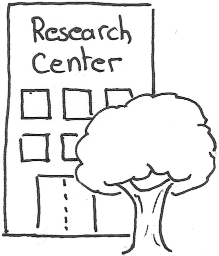
- Security proofs of protocols rely on **cryptographic reductions**.

$\mathcal{A}$ breaches security of protocol

$\Rightarrow_{\text{reduction}}$

$\mathcal{B}$ breaches security of primitive.

RFID protocol



m →

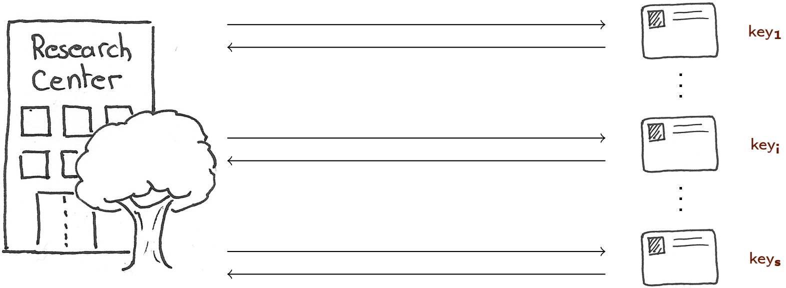
← $\langle \mathsf{r}, h(\langle \mathsf{r}, m\rangle, \mathsf{key})\rangle$

secret key

$\mathsf{r} \xleftarrow{\$}$

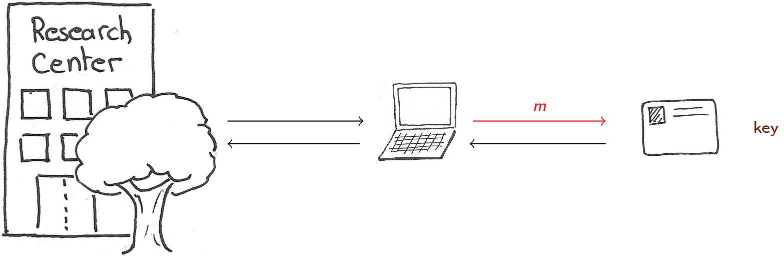
Cryptographic reduction: example

RFID protocol



Cryptographic reduction: example

RFID protocol



Key secrecy: Attackers cannot learn anything about the keys

Attacker's point of view:

$$\cdots, \langle \mathsf{r}, h(\langle \mathsf{r}, m \rangle, \mathsf{key}) \rangle, \cdots$$
$$\sim$$
$$\cdots, \langle \mathsf{r}, \mathsf{r}' \rangle, \qquad \cdots$$

Context: cryptographic games

A pseudo-random function is a function that "seems" random.

Game $G_{\#(\textbf{Left},\textbf{Right})}$

$$\underline{\texttt{oracle Init}} := \{k \xleftarrow{\$}; \mathsf{log} := []\}$$

$$\underline{\texttt{oracle Hash}}(x) := \{$$
$$\quad r \xleftarrow{\$}$$
$$\quad \texttt{if } (x \notin \mathsf{log}) \ \{$$
$$\qquad \mathsf{log} := x :: \mathsf{log}$$
$$\qquad \texttt{return } \boxed{\#(\mathsf{h}(\mathsf{x},\mathsf{k}),\ \mathsf{r})}$$
$$\quad \}$$
$$\}$$

Assumption: PRF

No polynomial-time adversary $\mathcal{B}$ can distinguish G_{Left} from G_{Right}.

Context: cryptographic reduction

$\langle \mathsf{r}_1, h(\langle \mathsf{r1}, m_1 \rangle, \mathsf{key}_1) \rangle, \langle \mathsf{r}_2, h(\langle \mathsf{r}_2, m_2 \rangle, \mathsf{key}_2) \rangle, \ldots$

Context: cryptographic reduction

$\langle \mathsf{r}_1, h(\langle \mathsf{r1}, m_1 \rangle, \mathsf{key}_1) \rangle, \langle \mathsf{r}_2, h(\langle \mathsf{r}_2, m_2 \rangle, \mathsf{key}_2) \rangle, \ldots$

$\langle \mathsf{r}_1, \mathsf{r}_1' \rangle, \langle \mathsf{r}_2, h(\langle \mathsf{r}_2, m_2 \rangle, \mathsf{key}_2) \rangle, \ldots$

Context: cryptographic reduction

$\langle r_1, h(\langle r1, m_1\rangle, key_1)\rangle, \langle r_2, h(\langle r_2, m_2\rangle, key_2)\rangle, \ldots$

$\langle r_1, r_1'\rangle, \langle r_2, h(\langle r_2, m_2\rangle, key_2)\rangle, \ldots$

Context: cryptographic reduction

$$\langle r_1, h(\langle r1, m_1\rangle, key_1)\rangle, \langle r_2, h(\langle r_2, m_2\rangle, key_2)\rangle, \ldots$$

$$\langle r_1, r_1'\rangle, \langle r_2, h(\langle r_2, m_2\rangle, key_2)\rangle, \ldots$$

$$\langle r_1, r_1'\rangle, \langle r_2, r_2'\rangle, \ldots$$

Context: cryptographic reduction

$$\langle \mathsf{r}_1, h(\langle \mathsf{r1}, m_1\rangle, \mathsf{key}_1)\rangle, \langle \mathsf{r}_2, h(\langle \mathsf{r}_2, m_2\rangle, \mathsf{key}_2)\rangle, \ldots$$

$$\langle \mathsf{r}_1, \mathsf{r}'_1\rangle, \langle \mathsf{r}_2, h(\langle \mathsf{r}_2, m_2\rangle, \mathsf{key}_2)\rangle, \ldots$$

$$\langle \mathsf{r}_1, \mathsf{r}'_1\rangle, \langle \mathsf{r}_2, \mathsf{r}'_2\rangle, \ldots$$

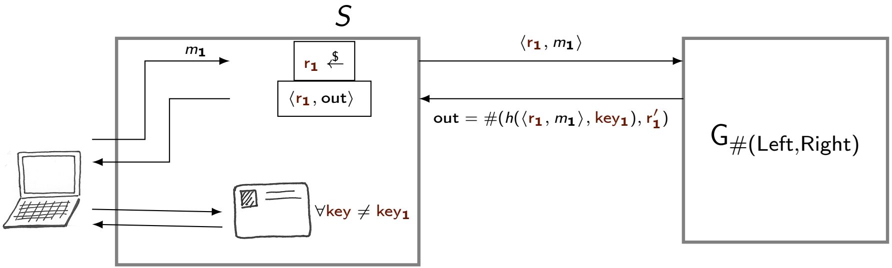
$$\vdots$$

$$\langle \mathsf{r}_1, \mathsf{r}'_1\rangle, \langle \mathsf{r}_2, \mathsf{r}'_2\rangle, \ldots$$

Context: cryptographic reduction

$$\langle r_1, h(\langle r1, m_1\rangle, key_1)\rangle, \cdots \sim \langle r_1, r_1'\rangle, \ldots$$

Build a simulator S such that:



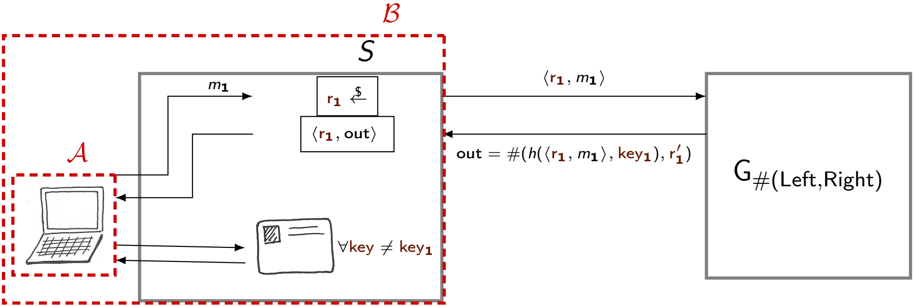
Context: cryptographic reduction

$\mathcal{A}$ against $\langle \mathsf{r}_1, h(\langle \mathsf{r}1, m_1\rangle, \mathsf{key}_1)\rangle, \cdots \sim \langle \mathsf{r}_1, \mathsf{r}'_1\rangle, \ldots$

$\Rightarrow_{\text{reduction}}$

$\mathcal{B}$ against PRF assumption

Build a simulator S such that:



Problem

Squirrel Prover:

- interactive proof assistant;
- relies on the CCSA logic;
- allows for proof mechanization.

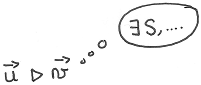
Cryptographic assumptions in Squirrel:
reasoning rules (tactics) for specific games (e.g. PRF, CCA1).

Problems:

- manually design and prove each new rule;
- implement each new rule in the tool.

$\Rightarrow$ out-of-reach for standard users and error prone.

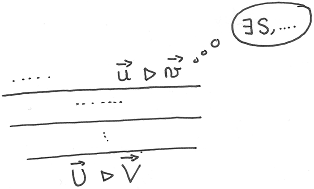
Contributions



- Framework for bi-deduction supporting cryptographic reductions.

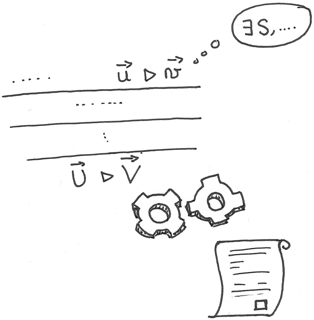
Contributions

- Framework for bi-deduction supporting cryptographic reductions.
- Proof system (implicitly build simulators through inference rules).



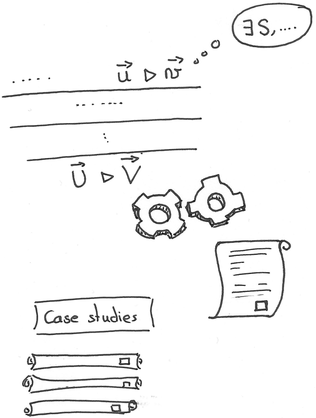
Contributions

- Framework for bi-deduction supporting cryptographic reductions.
- Proof system (implicitly build simulators through inference rules).
- Heuristic proof-search algorithm and its implementation in Squirrel.



Contributions

- Framework for bi-deduction supporting cryptographic reductions.
- Proof system (implicitly build simulators through inference rules).
- Heuristic proof-search algorithm and its implementation in Squirrel.
- Validation through case studies.



Bi-deduction predicate: starting point

Bideduction predicate

$$\vdash \#(\vec{u_0}, \vec{u_1}) \triangleright \#(\vec{v_0}, \vec{v_1})$$

There exists a simulator S such that

$$S_{G_{Left}}(\vec{u_0}) = \vec{v_0} \qquad S_{G_{Right}}(\vec{u_1}) = \vec{v_1}$$

Bi-deduction predicate: starting point

Bideduction predicate

$$\vdash \#(\vec{u_0}, \vec{u_1}) \triangleright \#(\vec{v_0}, \vec{v_1})$$

There exists a simulator S such that

$$S_{G_{Left}}(\vec{u_0}) = \vec{v_0} \qquad S_{G_{Right}}(\vec{u_1}) = \vec{v_1}$$

Compute $h(\langle \mathsf{r}, m \rangle, \mathsf{key})$ from $\langle \mathsf{r}, m \rangle$ and key?

$$\begin{aligned} \mathcal{S}() := \ & (\mathsf{x}_{pair}, \mathsf{x}_{key}) \leftarrow \mathcal{S}_1(); \\ & \mathsf{x}_{res} \leftarrow h(\mathsf{x}_{pair}, \mathsf{x}_{key}) \end{aligned}$$

$$\frac{\vdash \_ \triangleright \langle \mathsf{r}, m \rangle, \mathsf{key} \qquad \text{poly-time}(h)}{\vdash \_ \triangleright h(\langle \mathsf{r}, m \rangle, \mathsf{key})}$$

Bi-deduction predicate: handling randomness

Different sources of randomness:

- $\mathrm{key}_1 \rightarrow$ game
- $\mathrm{r}_1 \rightarrow$ simulator

$$\frac{}{(\mathrm{r}_1, \mathtt{T}_S) \vdash \_ \triangleright \mathrm{r}_1}$$

$\mathcal{S}() := \mathsf{x}_{r_1} \xleftarrow{\$}$

Constraint system:
associates samplings to their tag

Bi-deduction predicate: enabling oracle calls

$$\underline{\mathcal{S}()} := \begin{array}{l} \mathsf{x}_m \leftarrow \mathcal{S}_1(); \\ \mathsf{x}_\mathsf{h} \leftarrow G.\mathtt{Hash}(\mathsf{x}_m) \end{array}$$

$$\frac{\mathcal{C} \vdash \_ \triangleright \langle \mathsf{r}_1, m_1 \rangle}{\mathcal{C}' \vdash \_ \triangleright \#(h(\langle \mathsf{r}_1, m_1 \rangle, \mathsf{key}_1), \mathsf{r}'_1)}$$

with $\mathcal{C}' = \mathcal{C} \cdot (\mathsf{r}'_1, \mathtt{T}_G) \cdot (\mathsf{key}_1, \mathtt{T}_G)$

Bi-deduction predicate: enabling oracle calls

$$\underline{\mathcal{S}()} := \mathsf{x}_m \leftarrow \mathcal{S}_1(); \\ \mathsf{x}_\mathsf{h} \leftarrow G.\mathtt{Hash}(\mathsf{x}_m)$$

$$\frac{\mathcal{C}, \{\varphi\}\{\mathsf{log} = l\} \vdash \_ \rhd \langle \mathsf{r}_1, m_1 \rangle \quad \langle \mathsf{r}_1, m_1 \rangle \notin l}{\mathcal{C}', \{\varphi\}\{\mathsf{log} = \langle \mathsf{r}_1, m_1 \rangle :: l\} \vdash \_ \rhd \#(h(\langle \mathsf{r}_1, m_1 \rangle, \mathsf{key}_1), \mathsf{r}'_1)}$$

with $\mathcal{C}' = \mathcal{C} \cdot (\mathsf{r}'_1, \mathtt{T}_G) \cdot (\mathsf{key}_1, \mathtt{T}_G)$

Pre and post conditions
on game's memory

Bi-deduction predicate: wrapping up

Bideduction predicate

$$\mathcal{C}, \{\varphi\}\{\psi\} \vdash \vec{u} \rhd \vec{v}$$

There exists S such that whenever $\mathcal{C}$ is consistent:

- Randomness is used according to $\mathcal{C}$.
- From any memory in φ, S's execution yields a memory in ψ.
- $S(\vec{u}) = \vec{v}$.

Bi-deduction predicate: wrapping up

Bideduction predicate

$$\mathcal{C}, \{\varphi\}\{\psi\} \vdash \vec{u} \rhd \vec{v}$$

There exists S such that whenever $\mathcal{C}$ is consistent:

- Randomness is used according to $\mathcal{C}$.
- From any memory in φ, S's execution yields a memory in ψ.
- $S(\vec{u}) = \vec{v}$.

$$\frac{\mathcal{C}, \{\varphi_{\mathbf{init}}\}\{\psi\} \vdash \emptyset \rhd \#(\vec{v_0}, \vec{v_1})}{\vec{v_0} \sim \vec{v_1}}$$

whenever $\mathcal{C}$ is consistent.

Proof system overview

- Weakening rules
- Composing rules (loops, sequences)
- ...

Dup

$$\frac{\mathcal{E}, \Theta, C_\#, (\varphi_\#, \psi_\#) \vdash \vec{u}_\# \rhd \vec{v}_\#, \vec{t}_\#}{\mathcal{E}, \Theta, C_\#, (\varphi_\#, \psi_\#) \vdash \vec{u}_\# \rhd \vec{v}_\#, \vec{t}_\#, \vec{t}_\#}$$

Transitivity

$$\frac{\mathcal{E}, \Theta, C^1_\#, (\varphi_\#, \varphi'_\#) \vdash \vec{u}_\# \rhd \vec{t}_\# \qquad \mathcal{E}, \Theta, C^2_\#, (\varphi'_\#, \psi_\#) \vdash \vec{u}_\#, \vec{t}_\# \rhd \vec{v}_\#}{\mathcal{E}, \Theta, C^1_\# \cdot C^2_\#, (\varphi_\#, \psi_\#) \vdash \vec{u}_\# \rhd \vec{t}_\#, \vec{v}_\#}$$

Refl

$$\frac{}{\mathcal{E}, \Theta, \emptyset, (\varphi_\#, \varphi_\#) \vdash \vec{u}_\#, t_\# \rhd t_\#}$$

FA

$$\frac{\mathcal{E}, \Theta, C_\#, (\varphi_\#, \psi_\#) \vdash \vec{u}_\# \rhd \vec{v}_\#, (t^1_\# \mid f_\#), \ldots, (t^n_\# \mid f_\#) \qquad \mathcal{E}, \Theta \vdash \mathsf{adv}(g)}{\mathcal{E}, \Theta, C_\#, (\varphi_\#, \psi_\#) \vdash \vec{u}_\# \rhd \vec{v}_\#, (g\ t^1_\# \ \ldots\ t^n_\# \mid f_\#)}$$

If-then-else

$$\frac{\mathcal{E}, \Theta, C_\#, (\varphi_\#, \psi_\#) \vdash \vec{u}_\# \rhd \vec{v}_\#, (b_\# \mid f_\#), (t_\# \mid f_\# \wedge b_\#), (t'_\# \mid f_\# \wedge \neg b_\#)}{\mathcal{E}, \Theta, C_\#, (\varphi_\#, \psi_\#) \vdash \vec{u}_\# \rhd \vec{v}_\#, (\text{if } b_\# \text{ then } t_\# \text{ else } t'_\# \mid f_\#)}$$

Lambda

$$\frac{(\mathcal{E}, x : \tau), \Theta, C_\#, (\varphi_\#, \varphi_\#) \vdash \vec{u}_\#, x \rhd (t_\# \mid f_\#) \qquad \mathcal{E}, x : \tau \vdash t_\# : \tau_b \qquad \tau_b \in \mathbb{B} \qquad \mathsf{enum}(\tau)}{\mathcal{E}, \Theta, \prod_{(x:\tau)} .C_\#, (\varphi_\#, \varphi_\#) \vdash \vec{u}_\# \rhd (\lambda(x : \tau).t_\# \mid f_\#)}$$

Induction

$$\frac{(\mathcal{E}, x : \tau), \Theta, C_\#, (\varphi_\#, \varphi_\#) \vdash \vec{u}_\#, (\lambda(y : \tau).\text{if } y < x \text{ then } t[x \mapsto y] \mid f_\#), x \rhd (t_\# \mid f_\#) \qquad \mathcal{E}, x : \tau \vdash t_\# : \tau_b \qquad \tau_b \in \mathbb{B} \qquad \mathsf{finite}(\tau) \qquad \mathsf{fixed}(\tau) \qquad \mathcal{E}, \Theta \vdash \mathsf{well\text{-}founded}_\tau(<) \mathbin{\tilde{\wedge}} \mathsf{adv}(<)}{\mathcal{E}, \Theta, \prod_{(x:\tau)} C_\#, (\varphi_\#, \varphi_\#) \vdash \vec{u}_\# \rhd (\lambda(x : \tau).t_\# \mid f_\#)}$$

Name

$$\frac{\mathcal{E}, \Theta, C_\#, (\varphi_\#, \psi_\#) \vdash \vec{u}_\# \rhd (t_\# \mid f_\#)}{\mathcal{E}, \Theta, C_\# \cdot \{(\emptyset, n, t_\#, \top_S, f_\#)\}, (\varphi_\#, \psi_\#) \vdash \vec{u}_\# \rhd (n\ t_\# \mid f_\#)}$$

Figure 6: Selected set of rules.

Automation

Heuristic proof search:

- goal-directed;
- constraint directed;
- greedily applies oracle calls.

Case studies:

Protocol	Hypotheses	Property
Basic Hash	EUF-MAC and PRF	Unlinkability
Hash Lock	PRF	Strong secrecy
Private Authentication	$CCA_{\$}$ **NEW**	Anonymity
NSL (partial)	$CCA2$ **NEW**	Strong secrecy

Conclusion

What we have done:

- Formal framework linking games, simulators, and the logic.
- Bi-deduction judgment to build simulators interacting with games.
- Proof system for bi-deduction.
- Implementation of proof-search algorithm.

On going and future work:

- Improve the proof-search heuristic (ongoing).
- "Stress test" on larger protocol (ongoing).
- Apply to other frameworks (e.g. EasyCrypt).

(contact me: justine.sauvage@inria.fr)