A Machine-Checked Formalization of Σ-Protocols

Santiago Zanella-Béguelin[1]

Gilles Barthe[1] Daniel Hedin[1]

Benjamin Grégoire[2] Sylvain Heraud[2]

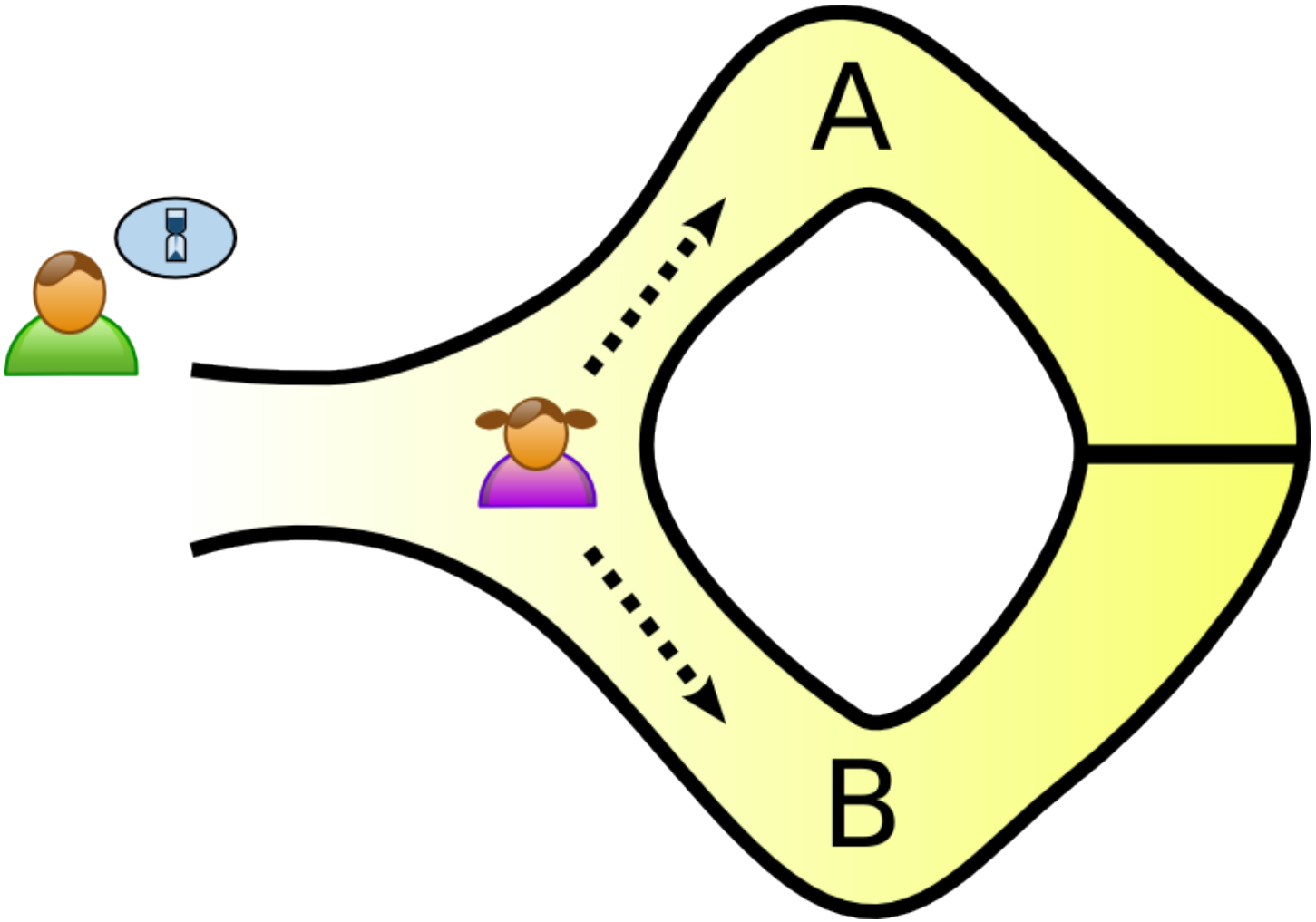
[1]IMDEA Software, Madrid, Spain

[2]INRIA Sophia Antipolis - Méditerranée, France

2010.07.18

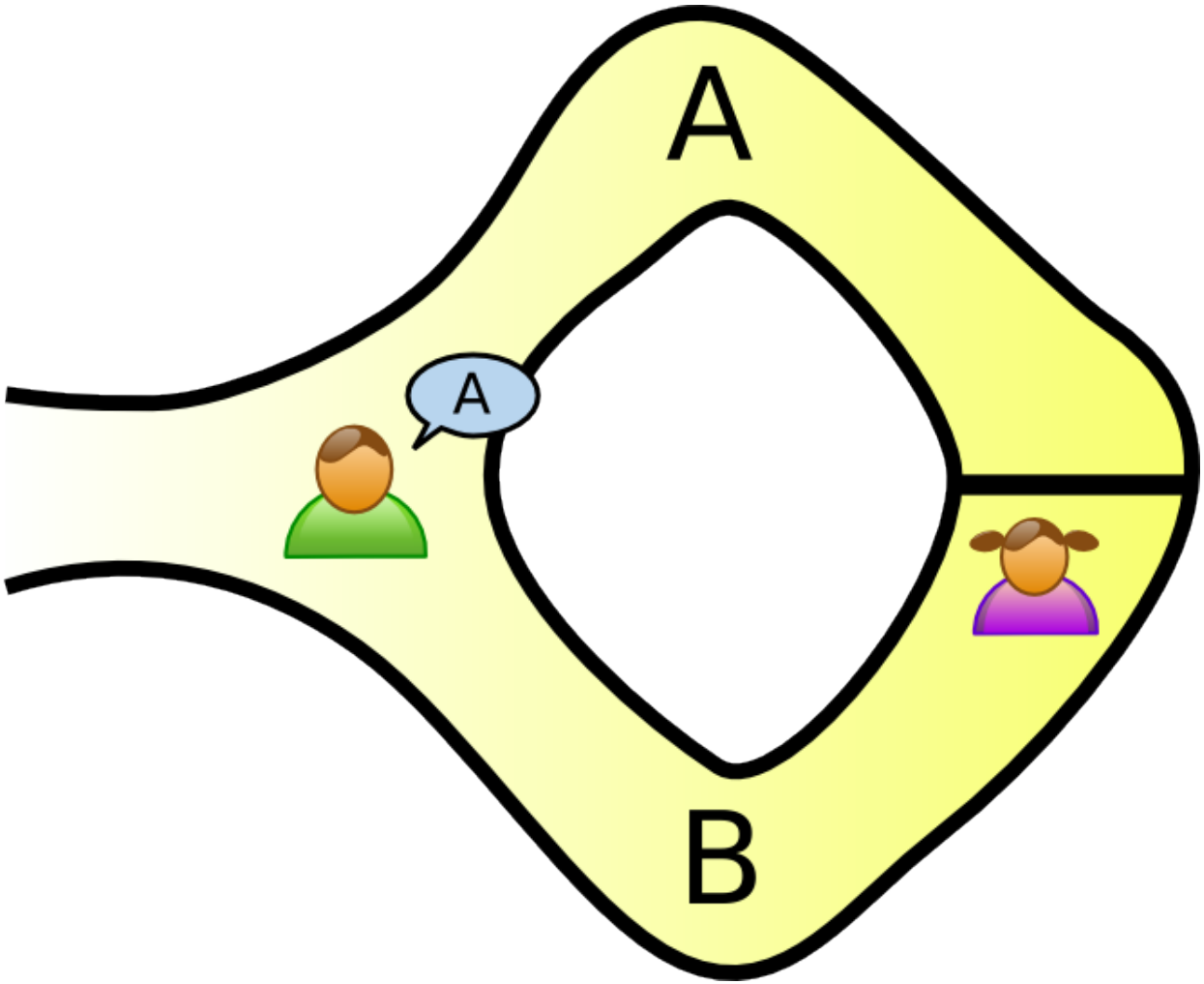
CSF 2010

Zero-Knowledge Proofs



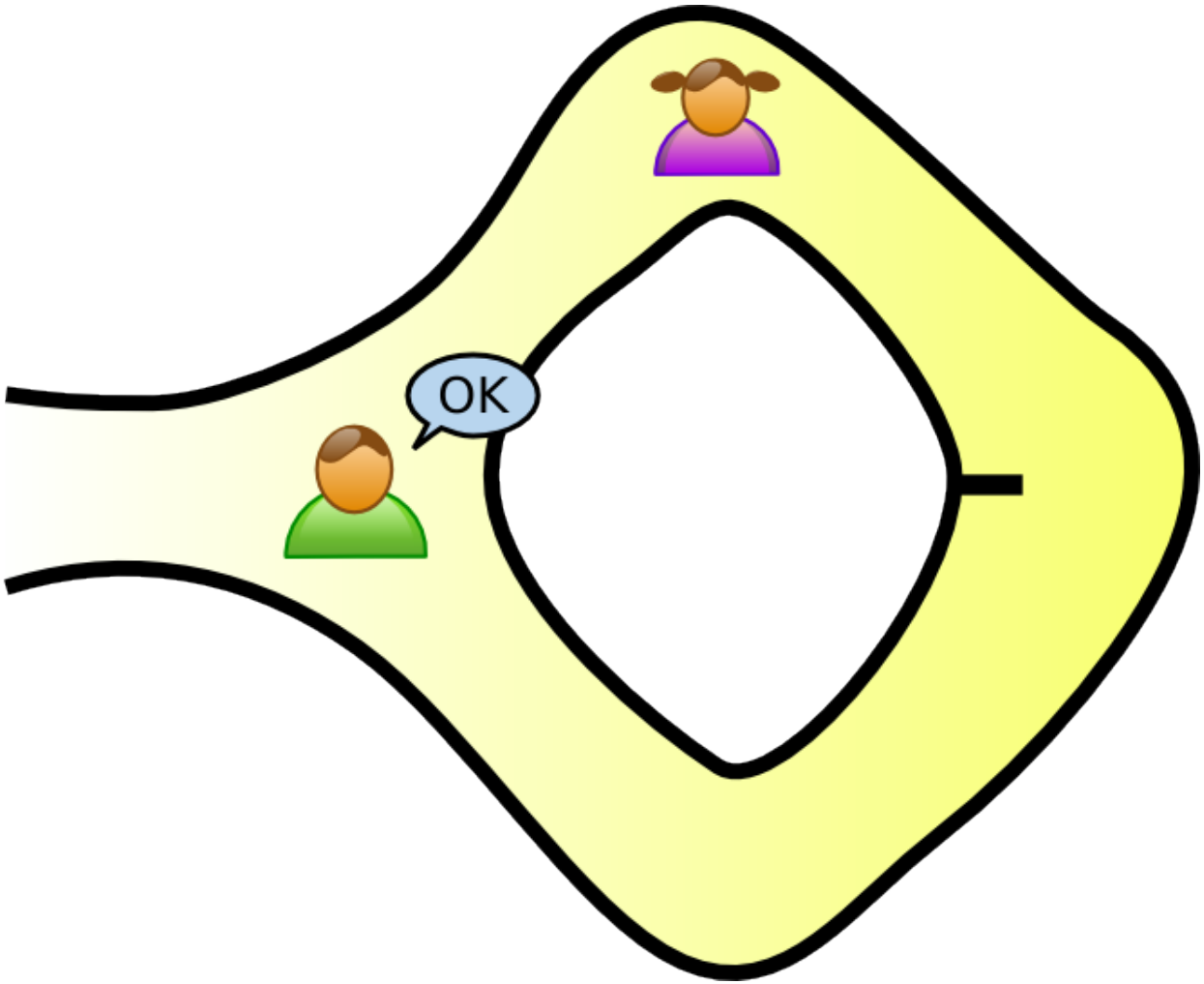
 Victor  Peggy

Zero-Knowledge Proofs



 Victor  Peggy

Zero-Knowledge Proofs



 Victor  Peggy

If you ever need to explain this to your kids

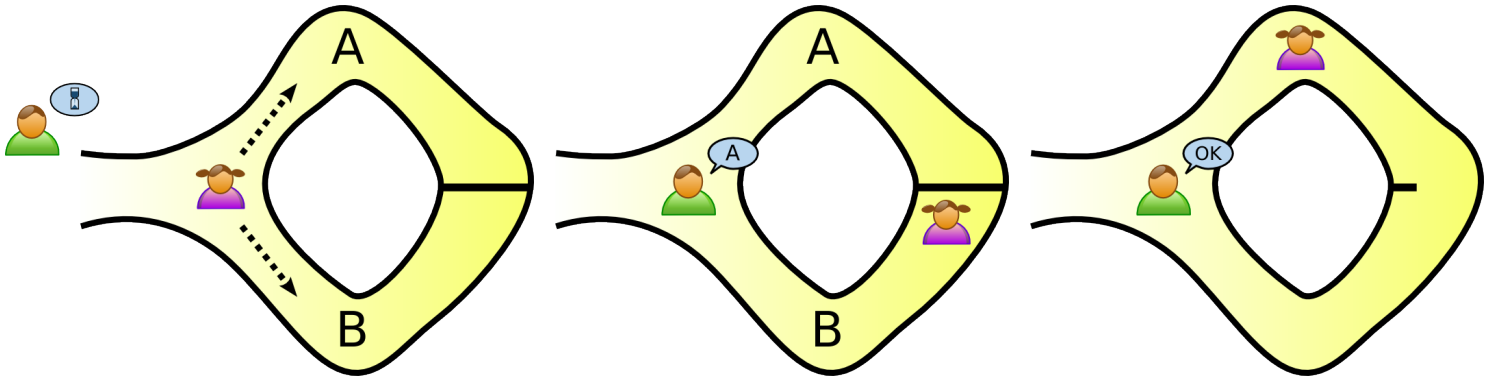
How to Explain Zero-Knowledge Protocols to your Children
Jean-Jacques Quisquater, Louis C. Guillou. CRYPTO'89

The Strange Cave of Ali Baba

winding passages: one to the left and the other to the right (The Entry of the Cave).

Ali Baba did not see which passage the thief ran into. Ali Baba had to choose which way to go, and he decided to go to the left. The left-hand passage ended in a dead end. Ali Baba searched all the way from the fork to the dead end, but he did not find the thief. Ali Baba said to himself that the thief was perhaps in the other passage. So he searched the right-hand passage, which also came to a dead end. But again he did not find the thief. "This cave is pretty strange," said Ali Baba to himself. "Where has my thief gone?"

Properties of Zero-Knowledge Proofs



- Completeness
  A honest prover always convinces a honest verifier
- Soundness
  A dishonest prover (almost) never convinces a verifier
- Zero-Knowledge
  A verifier doesn't learn anything from playing the protocol

Σ-Protocols

- Knowledge Relation R
- Prover knows x, w s.t. $R(x, w)$ / Verifier knows only x

Prover

Verifier

Computes commitment r → r

← c Samples challenge c

Computes response s → s Accepts/rejects response

Schnorr Protocol

Primes p, q such that $q|(p-1)$, g generator of $\mathbb{Z}_p$

$$R = \{(x, w) \mid x = g^w\} \subseteq \mathbb{Z}_p \times \mathbb{Z}_q$$

Prover

Verifier

$k \xleftarrow{\$} \mathbb{Z}_q;\ r \leftarrow g^k$

r

c

$c \xleftarrow{\$} [0..q-1]$

$s \leftarrow k + cw$

s

$g^s \stackrel{?}{=} r\ x^c$

Okamoto Protocol

Primes p, q such that $q|(p-1)$, $g_{1,2}$ generators of $\mathbb{Z}_q$

$$R = \{(x, (w_1, w_2)) \mid x = g_1^{w_1} g_2^{w_2}\} \subseteq \mathbb{Z}_p \times (\mathbb{Z}_q \times \mathbb{Z}_q)$$

Prover

Verifier

$k_{1,2} \xleftarrow{\$} \mathbb{Z}_q; r \leftarrow g_1^{k_1} g_2^{k_2}$

r

c

$c \xleftarrow{\$} [0..q-1]$

$s \leftarrow (k_1 + cw_1, k_2 + cw_2)$

s

$g_1^{s_1} g_2^{s_2} \stackrel{?}{=} (r_1\ x^c, r_2\ x^c)$

Fiat-Shamir Protocol

RSA modulus $N = pq$

$$R = \{(x, w) \mid x = w^2\} \subseteq \mathbb{Z}_N^* \times \mathbb{Z}_N^*$$

Prover

Verifier

$k \xleftarrow{\$} \mathbb{Z}_N^*;\ r \leftarrow k^2$

r

c

$c \xleftarrow{\$} [0..1]$

$s \leftarrow k \cdot w^c$

s

$s^2 \stackrel{?}{=} r\ x^c$

Guillou-Quisquater Protocol

RSA modulus $N = pq$, public exponent e

$$R = \{(x, w) \mid x = w^e\} \subseteq \mathbb{Z}_N^* \times \mathbb{Z}_N^*$$

Prover

Verifier

$k \xleftarrow{\$} \mathbb{Z}_N^*;\ r \leftarrow k^e$

r

c

$c \xleftarrow{\$} [0..e-1]$

$s \leftarrow k \cdot w^c$

s

$s^e \stackrel{?}{=} r\ x^c$

CertiCrypt: machine-checked crypto proofs

Certified framework for building and verifying crypto proofs in the Coq proof assistant

- Combination of programming language techniques and cryptographic-specific tools
- Game-based methodology, natural to cryptographers
- Several case studies:
  - Encryption schemes: ElGamal, Hashed ElGamal, OAEP, IBE
  - Signature schemes: FDH, BLS
  - In this talk: Zero-Knowledge proofs

Inside CertiCrypt

- Semantics and cost model of probabilistic programs
- Standard tools to reason about probabilistic programs
  - Semantics-preserving program transformations
  - Observational equivalence
  - Relational Hoare Logic
  - Characterization of PPT programs

pWhile: a Probabilistic Programming Language

$$
\begin{array}{rcll}
\mathcal{I} & ::= & \mathcal{V} \leftarrow \mathcal{E} & \text{assignment} \\
 & | & \mathcal{V} \xleftarrow{\$} \mathcal{D}\mathcal{E} & \text{random sampling} \\
 & | & \text{if } \mathcal{E} \text{ then } \mathcal{C} \text{ else } \mathcal{C} & \text{conditional} \\
 & | & \text{while } \mathcal{E} \text{ do } \mathcal{C} & \text{while loop} \\
 & | & \mathcal{V} \leftarrow \mathcal{P}(\mathcal{E}, \ldots, \mathcal{E}) & \text{procedure call} \\
\mathcal{C} & ::= & \text{skip} & \text{nop} \\
 & | & \mathcal{I};\ \mathcal{C} & \text{sequence}
\end{array}
$$

$x \xleftarrow{\$} d$: sample x according to distribution d
Typically the uniform distribution on a finite set (e.g. $[0..n]$)

Deep Embedding

Syntax of programs formalized as an inductive type
Allows automation through reflection

Semantics

Measure Monad —courtesy of Christine Paulin

Distributions represented as functions of type

$$\mathcal{D}(A) \stackrel{\text{def}}{=} (A \to [0,1]) \to [0,1] \quad \text{s.t.}$$

1. $f \leq g \implies \mu(f) \leq \mu(g)$;
2. $\mu(\mathbb{1} - f) \leq 1 - \mu(f)$;
3. $f \leq \mathbb{1} - g \implies \mu(f + g) = \mu(f) + \mu(g)$;
4. $\mu(k \times f) = k \times \mu(f)$;
5. $\forall f : \mathbb{N} \xrightarrow{m} (A \xrightarrow{m} [0,1]).\ \mu(\sup\ f) \leq \sup\ (\lambda n.\ \mu(f(n))$

All arithmetic is in the unit interval $[0,1]$

$$\begin{array}{lll} \text{unit} : A \to \mathcal{D}(A) & \stackrel{\text{def}}{=} & \lambda x.\ \lambda f.\ f\ x \\ \text{bind} : \mathcal{D}(A) \to (A \to \mathcal{D}(B)) \to \mathcal{D}(B) & \stackrel{\text{def}}{=} & \lambda \mu.\ \lambda F.\ \lambda f.\ \mu(\lambda x.\ F\ x\ f) \end{array}$$

Semantics

Programs map an initial memory to a distribution on final memories

$$[\![c \in \mathcal{C}]\!] : \mathcal{M} \to \mathcal{D}(\mathcal{M})$$

To compute probabilities, just measure the characteristic function of the event:

$$\Pr[c, m : A] \stackrel{\text{def}}{=} [\![c]\!]\ m\ \mathbb{1}_A$$

Let $c = x \xleftarrow{\$} \{0,1\};\ y \xleftarrow{\$} \{0,1\}$

$$[\![c]\!]\ m\ f = \frac{1}{4} \begin{pmatrix} f(m\{0,0/x,y\}) + f(m\{0,1/x,y\}) + \\ f(m\{1,0/x,y\}) + f(m\{1,1/x,y\}) \end{pmatrix}$$

$$\Pr[c, m : x \leq y] = [\![c]\!]\ m\ \mathbb{1}_{(x \leq y)} = 3/4$$

Instrumented semantics to characterize efficient (PPT) programs:

$$[\![c \in \mathcal{C}]\!] : \mathcal{M} \to \mathcal{D}(\mathcal{M} \times \mathbb{N})$$

Semantics

Programs map an initial memory to a distribution on final memories

$$[\![c \in \mathcal{C}]\!] : \mathcal{M} \to \mathcal{D}(\mathcal{M})$$

To compute probabilities, just measure the characteristic function of the event:

$$\Pr[c, m : A] \stackrel{\text{def}}{=} [\![c]\!]\ m\ \mathbb{1}_A$$

Let $c = x \xleftarrow{\$} \{0,1\};\ y \xleftarrow{\$} \{0,1\}$

$$[\![c]\!]\ m\ f = \frac{1}{4} \begin{pmatrix} f(m\{0,0/x,y\}) + f(m\{0,1/x,y\}) + \\ f(m\{1,0/x,y\}) + f(m\{1,1/x,y\}) \end{pmatrix}$$

$$\Pr[c, m : x \leq y] = [\![c]\!]\ m\ \mathbb{1}_{(x \leq y)} = 3/4$$

Instrumented semantics to characterize efficient (PPT) programs:

$$[\![c \in \mathcal{C}]\!] : \mathcal{M} \to \mathcal{D}(\mathcal{M} \times \mathbb{N})$$

Observational Equivalence

Formal definition

$$f =_X g \quad \stackrel{\text{def}}{=} \quad \forall m_1\ m_2,\ m_1 =_X m_2 \implies f\ m_1 = g\ m_2$$

$$\vDash c_1 \simeq^I_O c_2 \quad \stackrel{\text{def}}{=} \quad \forall m_1\ m_2\ f\ g,\ m_1 =_I m_2 \wedge f =_O g \implies [\![c_1]\!]\ m_1\ f = [\![c_2]\!]\ m_2\ g$$

Example

$$\vDash x \xleftarrow{\$} \{0,1\}^k; y \leftarrow x \oplus z \simeq^{\{z\}}_{\{x,y,z\}} y \xleftarrow{\$} \{0,1\}^k; x \leftarrow y \oplus z$$

- Useful to relate probabilities

$$\frac{\text{fv}(E) \subseteq O \quad \vDash c_1 \simeq^I_O c_2 \quad m_1 =_I m_2}{\Pr[c_1, m_1 : A] = \Pr[c_2, m_2 : A]}$$

- Only a Partial Equivalence Relation

$$\vDash c \simeq^I_O c \qquad \text{not true in general}$$

- Generalizes information flow security (take $I = O = \mathcal{V}_{\text{low}}$)

Observational Equivalence

Formal definition

$$f =_X g \quad \stackrel{\text{def}}{=} \quad \forall m_1\ m_2,\ m_1 =_X m_2 \implies f\ m_1 = g\ m_2$$

$$\vDash c_1 \simeq_O^I c_2 \quad \stackrel{\text{def}}{=} \quad \forall m_1\ m_2\ f\ g,\ m_1 =_I m_2 \wedge f =_O g \implies [\![c_1]\!]\ m_1\ f = [\![c_2]\!]\ m_2\ g$$

Example

$$\vDash x \xleftarrow{\$} \{0,1\}^k; y \leftarrow x \oplus z \simeq_{\{x,y,z\}}^{\{z\}} y \xleftarrow{\$} \{0,1\}^k; x \leftarrow y \oplus z$$

- Useful to relate probabilities

$$\frac{\text{fv}(E) \subseteq O \quad \vDash c_1 \simeq_O^I c_2 \quad m_1 =_I m_2}{\Pr[c_1, m_1 : A] = \Pr[c_2, m_2 : A]}$$

- Only a Partial Equivalence Relation

$$\vDash c \simeq_O^I c \qquad \text{not true in general}$$

- Generalizes information flow security (take $I = O = \mathcal{V}_{\text{low}}$)

Proving program equivalence

Goal

$$\models c_1 \simeq^{I}_{O} c_2$$

A Relational Hoare Logic generalized to arbitrary relations

$$\frac{\models c_1 \sim c_2 : \Phi \Rightarrow \Phi' \quad \models c_1' \sim c_2' : \Phi' \Rightarrow \Phi''}{\models c_1; c_1' \sim c_2; c_2' : \Phi \Rightarrow \Phi''}[\text{Seq}]$$

$$\frac{\models c_1 \sim c_2 : \Psi \Rightarrow \Phi \quad \models c_2 \sim c_3 : \Psi' \Rightarrow \Phi'}{\models c_1 \sim c_3 : \Psi \circ \Psi' \Rightarrow \Phi \circ \Phi'}[\text{Comp}]$$

...

Proving program equivalence

Goal

$$\vDash c_1 \simeq^{I}_{O} c_2$$

Mechanized program transformations

- Transformation: $T(c_1, c_2, I, O) = (c_1', c_2', I', O')$
- Soundness theorem

$$\frac{T(c_1, c_2, I, O) = (c_1', c_2', I', O') \qquad \vDash c_1' \simeq^{I'}_{O'} c_2'}{\vDash c_1 \simeq^{I}_{O} c_2}$$

- Reflection-based Coq tactic
  (replace reasoning by computation)

Proving program equivalence

Goal

$$\models c_1 \simeq^{I}_{O} c_2$$

Mechanized program transformations

- Dead code elimination (`deadcode`)
- Constant folding and propagation (`ep`)
- Procedure call inlining (`inline`)
- Code movement (`swap`)
- Common suffix/prefix elimination (`eqobs_hd`, `eqobs_tl`)

Proving program equivalence

Goal

$$\vDash c \simeq^{I}_{O} c$$

An –incomplete– tactic for self-equivalence
(`eqobs_in`)

- Does $\vDash c \simeq^{I}_{O} c$ hold?
- Analyze dependencies to compute I' s.t. $\vDash c \simeq^{I'}_{O} c$
- Check that $I' \subseteq I$
- Think about type systems for information flow security

Formalizing Σ-Protocols

Prover

Verifier

Computes commitment r

r

c

Samples challenge c

Computes response s

s

Accepts/rejects response

The protocols we consider are public-coin

Formalizing Σ-Protocols

Prover

Verifier

$(r, state) \leftarrow P_1(x, w)$

r

c

$c \leftarrow V_1(x, r)$

$s \leftarrow P_2(x, w, state, c)$

s

$b \leftarrow V_2(x, r, c, s)$

The protocols we consider are public-coin

Formalizing Σ-Protocols

Prover

Verifier

$(r, state) \leftarrow P_1(x, w)$

r

c

$c \xleftarrow{\$} C$

$s \leftarrow P_2(x, w, state, c)$

s

$b \leftarrow V_2(x, r, c, s)$

The protocols we consider are public-coin

Formalizing Σ-Protocols

A Σ-protocol is given by:

- Types for $x, w, r, s, state$
- A knowledge relation R
- A challenge set C
- Procedures P_1, P_2, V_2

The protocol can be seen as a program

protocol(x, w) :
$(r, state) \leftarrow \mathsf{P}_1(x, w)$;
$c \xleftarrow{\$} C$;
$s \leftarrow \mathsf{P}_2(x, w, state, c)$;
$b \leftarrow \mathsf{V}_2(x, r, c, s)$

Formalizing Σ-Protocols

Completeness

$$\forall x, w.\ R(x, w) \implies \Pr[\text{protocol}(x, w) : b = \text{true}] = 1$$

Soundness

$\exists \text{KE}.\ \forall x, r, c_1, c_2, s_1, s_2.$

$$\left.\begin{array}{l} x \in \text{dom}(R) \\ c_1 \neq c_2 \\ (x, r, c_1, s_1) \text{ accepting} \\ (x, r, c_2, s_2) \text{ accepting} \end{array}\right\} \implies$$

$$\Pr[w \leftarrow \text{KE}(x, r, c_1, c_2, s_1, s_2) : R(x, w)] = 1$$

Honest-Verifier ZK vs. Special Honest-Verifier ZK

protocol(x, w) :
$(r, state) \leftarrow \mathsf{P}_1(x, w)$;
$c \xleftarrow{\$} C$;
$s \leftarrow \mathsf{P}_2(x, w, state, c)$;
$b \leftarrow \mathsf{V}_2(x, r, c, s)$

protocol(x, w, c) :
$(r, state) \leftarrow \mathsf{P}_1(x, w)$;
$s \leftarrow \mathsf{P}_2(x, w, state, c)$;
$b \leftarrow \mathsf{V}_2(x, r, c, s)$

Special Honest-Verifier ZK

$$\exists \mathsf{S}.\ \forall x, w, c.\ R(x, w) \implies \vDash \text{protocol}(x, w, c) \simeq^{\{x,c\}}_{\{r,c,s\}} (r, s) \leftarrow \mathsf{S}(x, c)$$

Honest-Verifier ZK

$$\exists \mathsf{S}.\ \forall x, w.\ R(x, w) \implies \vDash \text{protocol}(x, w) \simeq^{\{x\}}_{\{r,c,s\}} (r, c, s) \leftarrow \mathsf{S}(x)$$

Honest-Verifier ZK vs. Special Honest-Verifier ZK

protocol(x, w) :
$(r, state) \leftarrow \mathsf{P}_1(x, w)$;
$c \xleftarrow{\$} C$;
$s \leftarrow \mathsf{P}_2(x, w, state, c)$;
$b \leftarrow \mathsf{V}_2(x, r, c, s)$

protocol(x, w, c) :
$(r, state) \leftarrow \mathsf{P}_1(x, w)$;
$s \leftarrow \mathsf{P}_2(x, w, state, c)$;
$b \leftarrow \mathsf{V}_2(x, r, c, s)$

Special Honest-Verifier ZK

$$\exists \mathsf{S}.\ \forall x, w, c.\ R(x, w) \implies \vDash \text{protocol}(x, w, c) \simeq^{\{x,c\}}_{\{r,c,s\}} (r, s) \leftarrow \mathsf{S}(x, c)$$

Honest-Verifier ZK

$$\exists \mathsf{S}.\ \forall x, w.\ R(x, w) \implies \vDash \text{protocol}(x, w) \simeq^{\{x\}}_{\{r,c,s\}} (r, c, s) \leftarrow \mathsf{S}(x)$$

Honest-Verifier ZK vs. Special Honest-Verifier ZK

Theorem

Every Σ-protocol satisfying special HVZK is HVZK.

Proof.

This simulator perfectly simulates protocol(x, w)*:*

$$\mathsf{S}'(x) : c \xleftarrow{\$} \{0,1\}^{\ell};\ (r, s) \leftarrow \mathsf{S}(x, c);\ \text{return } (r, c, s)$$

Theorem

Given a Σ-protocol satisfying just HVZK can be converted into a protocol satisfying special HVZK.

Proof.

$$\begin{aligned}
\mathsf{P}'_1(x, w) &\stackrel{def}{=} (r, state) \leftarrow \mathsf{P}_1(x, w);\ c' \xleftarrow{\$} \{0,1\}^k; \\
&\quad \text{return } ((r, c'), (state, c'))) \\
\mathsf{P}'_2(x, w, (state, c'), c) &\stackrel{def}{=} s \leftarrow \mathsf{P}_2(x, w, state, c \oplus c');\ \text{return } s \\
\mathsf{V}'_2(x, (r, c'), c, s) &\stackrel{def}{=} b \leftarrow \mathsf{V}_2(x, r, c \oplus c', s);\ \text{return } b
\end{aligned}$$

Σ^{ϕ}-Protocols

Let ϕ be a homomorphism from an additive group $(\mathcal{G}, \oplus)$ to a multiplicative group $(\mathcal{H}, \otimes)$

$$\phi(a \oplus b) = \phi(a) \otimes \phi(b)$$

Homomorphism ϕ is special if there exists

1. a constant $v \in \mathbb{Z}$
2. a PPT-computable function $u : \mathcal{H} \to \mathcal{G}$

such that $\forall x \in \phi[\mathcal{G}]$

$$\phi(u(x)) = x^v$$

Σ^{ϕ}-Protocols

- A special homomorphism ϕ
- $c^{+} \in \mathbb{N}$ smaller than any prime divisor of special exponent v

This protocol is a ZK proof of knowledge of preimages of ϕ:

$$R = \{(x, w) \mid x = \phi(w)\}$$

Prover

Verifier

$y \xleftarrow{\$} \mathcal{G};\ r \leftarrow \phi(y)$ — r →

← c — $c \xleftarrow{\$} [0..c^{+}]$

$s \leftarrow y \oplus cw$ — s →

$\phi(s) \stackrel{?}{=} r \otimes x^{c}$

Σ^{ϕ}-Protocols (Special HVZK)

The following simulator perfectly simulates the protocol:

$\mathsf{S}(x, c)$:
$s \xleftarrow{\$} \mathcal{G}$;
$r \leftarrow \phi(s) \oplus x^{-c}$;
return (r, c, s)

$$\models \mathit{protocol}(x, w, c) \simeq^{\{x,c\} \wedge R(x,w)}_{\{r,c,s\}} \mathsf{S}(x, c)$$

Σ^{ϕ}-Protocols (Special HVZK)

$\mathsf{protocol}(x, w, c)$:
$(r, state) \leftarrow \mathsf{P}_1(x, w)$;
$s \leftarrow \mathsf{P}_2(x, w, state, c)$;
$b \leftarrow \mathsf{V}_2(x, r, c, s)$

$\overset{\{x,c\} \wedge R(x,w)}{\simeq}_{\{r,c,s\}}$

$simulation(x, c)$:
$s \xleftarrow{\$} \mathcal{G}$;
$r \leftarrow \phi(s) \oplus x^{-c}$

Σ^{ϕ}-Protocols (Special HVZK)

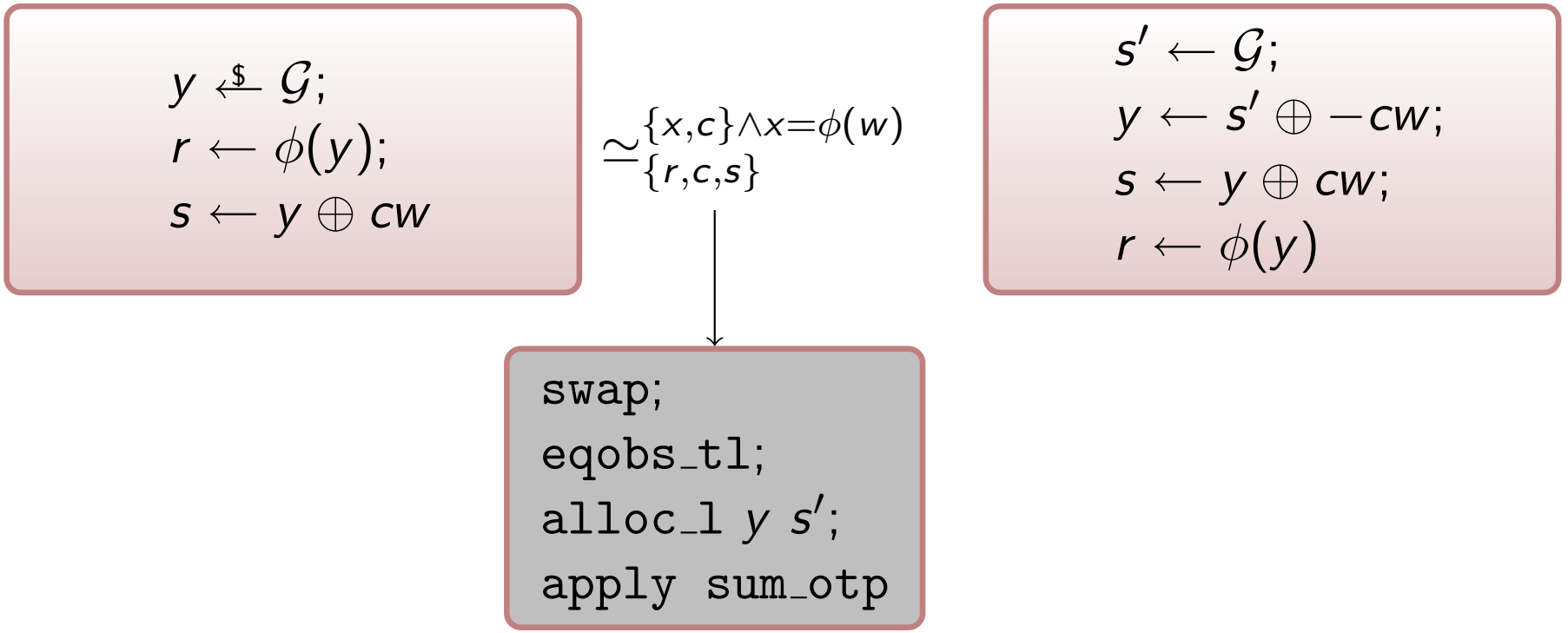
$\mathsf{protocol}(x, w, c)$:
$(r, state) \leftarrow \mathsf{P}_1(x, w)$;
$s \leftarrow \mathsf{P}_2(x, w, state, c)$;
$b \leftarrow \mathsf{V}_2(x, r, c, s)$

$\simeq^{\{x,c\} \wedge x = \phi(w)}_{\{r,c,s\}}$

$y \xleftarrow{\$} \mathcal{G}$;
$r \leftarrow \phi(y)$;
$s \leftarrow y \oplus cw$

$\texttt{inline\_l}\ \mathsf{P}_1$;
$\texttt{inline\_l}\ \mathsf{P}_2$;
$\texttt{ep}; \texttt{deadcode}$

Σ^{ϕ}-Protocols (Special HVZK)



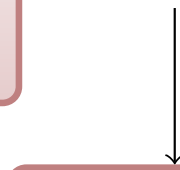
Lemma sum_otp : $\vDash x \xleftarrow{\$} \mathcal{G};\ y \leftarrow x \oplus z \sim x \xleftarrow{\$} \mathcal{G};\ y \leftarrow x : \Psi \Rightarrow =_{\{y\}}$

Σ^{ϕ}-Protocols (Special HVZK)

$s' \leftarrow \mathcal{G}$;
$y \leftarrow s' \oplus -cw$;
$s \leftarrow y \oplus cw$;
$r \leftarrow \phi(y)$

$\simeq^{\{x,c\} \wedge x = \phi(w)}_{\{r,c,s\}}$

$s' \leftarrow \mathcal{G}$;
$s \leftarrow s'$;
$r \leftarrow \phi(s') \otimes \phi(w)^{-c}$



ep;
deadcode

Σ^{ϕ}-Protocols (Special HVZK)

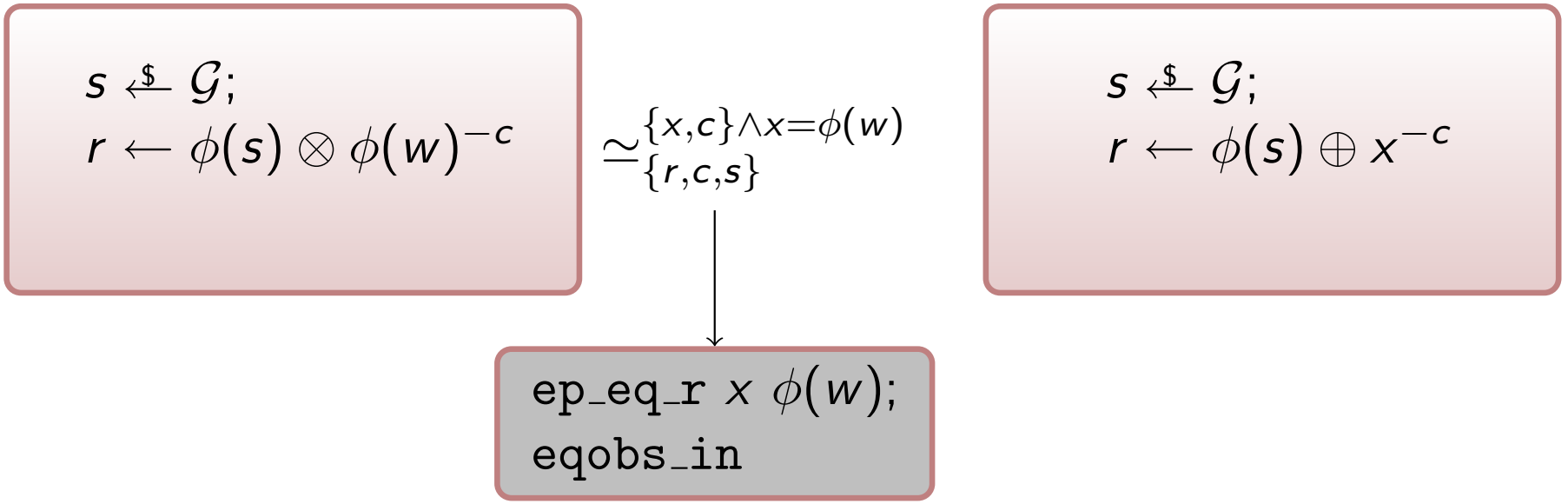
$s' \leftarrow \mathcal{G}$;
$s \leftarrow s'$;
$r \leftarrow \phi(s') \otimes \phi(w)^{-c}$

$\simeq^{\{x,c\} \wedge x = \phi(w)}_{\{r,c,s\}}$

`alloc_r` s s'

$s \xleftarrow{\$} \mathcal{G}$;
$r \leftarrow \phi(s) \otimes \phi(w)^{-c}$

Σ^{ϕ}-Protocols (Special HVZK)



Σ^{ϕ}-Protocols (Special HVZK)

$s \xleftarrow{\$} \mathcal{G}$;
$r \leftarrow \phi(s) \oplus x^{-c}$

$\simeq^{\{x,c\} \wedge x=\phi(w)}_{\{r,c,s\}}$

$simulation(x, c)$:
$(r, s) \leftarrow \mathsf{S}(x, c)$

```
inline_r S;
ep;
deadcode
```

Formalized Σ^ϕ-Protocols

Protocol	$\mathcal{G} \to \mathcal{H}$	$\phi(x)$	$u(x)$	v
Schnorr	$\mathbb{Z}_q^+ \to \mathbb{Z}_p^*$	g^x	0	q
Okamoto	$(\mathbb{Z}_q^+, \mathbb{Z}_q^+) \to \mathbb{Z}_p^*$	$g_1^{x_1} \otimes g_2^{x_2}$	$(0,0)$	q
Diffie-Hellman	$\mathbb{Z}_q^+ \to \mathbb{Z}_p^* \times \mathbb{Z}_p^*$	(g^x, g^{bx})	0	q
Fiat-Shamir	$\mathbb{Z}_N^* \to \mathbb{Z}_N^*$	x^2	x	2
Guillou-Quisquater	$\mathbb{Z}_N^* \to \mathbb{Z}_N^*$	x^e	x	e
Feige-Fiat-Shamir	$\{-1,1\} \times \mathbb{Z}_N^* \to \mathbb{Z}_N^*$	$s.x^2$	$\lvert x \rvert$	2

All these protocols are proved sound, complete and sHVZK in Coq

Combination of Σ^ϕ-protocols

Special homomorphism are closed under direct product

Proof.

Special homomorphisms $\phi_1 : \mathcal{G}_1 \to \mathcal{H}_1$, $\phi_2 : \mathcal{G}_2 \to \mathcal{H}_2$

$$\begin{array}{rcl} \phi & : & \mathcal{G}_1 \times \mathcal{G}_2 \to \mathcal{H}_1 \times \mathcal{H}_2 \\ \phi(x_1, x_2) & = & (\phi(x_1), \phi(x_2)) \end{array}$$

- $v \stackrel{\text{def}}{=} \text{lcm}(v_1, v_2)$
- $u(x_1, x_2) \stackrel{\text{def}}{=} (u_1(x_1)^{v/v_1}, u_2(x_2)^{v/v_2})$

A cheap and efficient way of combining Σ^ϕ-protocols to prove knowledge of several preimages!

Combination of Σ^ϕ-protocols

Special homomorphism are closed under direct product

Proof.

Special homomorphisms $\phi_1 : \mathcal{G}_1 \to \mathcal{H}_1$, $\phi_2 : \mathcal{G}_2 \to \mathcal{H}_2$

$$\begin{array}{rcl} \phi & : & \mathcal{G}_1 \times \mathcal{G}_2 \to \mathcal{H}_1 \times \mathcal{H}_2 \\ \phi(x_1, x_2) & = & (\phi(x_1), \phi(x_2)) \end{array}$$

- $v \stackrel{\text{def}}{=} \text{lcm}(v_1, v_2)$
- $u(x_1, x_2) \stackrel{\text{def}}{=} (u_1(x_1)^{v/v_1}, u_2(x_2)^{v/v_2})$

A cheap and efficient way of combining Σ^ϕ-protocols to prove knowledge of several preimages!

...which bring us to combining arbitrary Σ-protocols

Combination of Σ-Protocols

Given

- a Σ-protocol (P^1, V^1) for relation R_1
- a Σ-protocol (P^2, V^2) for relation R_2

Two basic ways of combining them. Given (x_1, x_2)

- AND-combination: prove knowledge of (w_1, w_2) such that

$$R_1(x_1, w_1) \ \textit{AND} \ R_2(x_2, w_2)$$

$$R \stackrel{\text{def}}{=} \{((x_1, x_2), (w_1, w_2)) \mid (x_1, w_1) \in R_1 \wedge (x_2, w_2) \in R_2\}$$

- OR-combination: prove knowledge of a w such that

$$R_1(x_1, w) \ \textit{OR} \ R_2(x_2, w)$$

without revealing which is the case

$$R \stackrel{\text{def}}{=} \{((x_1, x_2), w) \mid (x_1, w) \in R_1 \ \vee (x_2, w) \in R_2\}$$

Combination of Σ-Protocols

Given

- a Σ-protocol (P^1, V^1) for relation R_1
- a Σ-protocol (P^2, V^2) for relation R_2

Two basic ways of combining them. Given (x_1, x_2)

- AND-combination: prove knowledge of (w_1, w_2) such that

$$R_1(x_1, w_1) \ \textit{AND} \ R_2(x_2, w_2)$$

$$R \stackrel{\text{def}}{=} \{((x_1, x_2), (w_1, w_2)) \mid (x_1, w_1) \in R_1 \wedge (x_2, w_2) \in R_2\}$$

- OR-combination: prove knowledge of a w such that

$$R_1(x_1, w) \ \textit{OR} \ R_2(x_2, w)$$

without revealing which is the case

$$\sout{R \stackrel{\text{def}}{=} \{((x_1, x_2), w) \mid (x_1, w) \in R_1 \ \vee (x_2, w) \in R_2\}}$$

Combination of Σ-Protocols

Given

- a Σ-protocol (P^1, V^1) for relation R_1
- a Σ-protocol (P^2, V^2) for relation R_2

Two basic ways of combining them. Given (x_1, x_2)

- AND-combination: prove knowledge of (w_1, w_2) such that

$$R_1(x_1, w_1) \ \textit{AND} \ R_2(x_2, w_2)$$

$$R \stackrel{\text{def}}{=} \{((x_1, x_2), (w_1, w_2)) \mid (x_1, w_1) \in R_1 \wedge (x_2, w_2) \in R_2\}$$

- OR-combination: prove knowledge of a w such that

$$R_1(x_1, w) \ \textit{OR} \ R_2(x_2, w)$$

without revealing which is the case

$$R \stackrel{\text{def}}{=} \left\{ ((x_1, x_2), w) \;\middle|\; \begin{array}{l} (x_1, w) \in R_1 \wedge x_2 \in \text{dom}(R_2) \vee \\ (x_2, w) \in R_2 \wedge x_1 \in \text{dom}(R_1) \end{array} \right\}$$

AND-Combination

Prover

Verifier

$r_1 \leftarrow P_1^1(x_1, w_1)$
$r_2 \leftarrow P_1^2(x_2, w_2)$

r_1, r_2

c

$c \xleftarrow{\$} \{0,1\}^\ell$

$s_1 \leftarrow P_2^1(x_1, w_1, c)$
$s_2 \leftarrow P_2^2(x_2, w_2, c)$

s_1, s_2

$V_2^1(x_1, r_1, c, s_1) \wedge$
$V_2^2(x_2, r_2, c, s_2)$

- Using the same challenge for both proofs is the key
- Only possible if protocols are Special HVZK

OR-Combination

Suppose w is a witness for x_1, i.e. $R_1(x, w)$

Prover

Verifier

$r_1 \leftarrow P_1^1(x_1, w)$
$c_2 \xleftarrow{\$} \{0,1\}^\ell$
$(r_2, s_2) \leftarrow S_2(x_2, c_2)$

r_1, r_2

c

$c \xleftarrow{\$} \{0,1\}^\ell$

$c_1 \leftarrow c_2 \oplus c$
$s_1 \leftarrow P_2^1(x_1, w, c_1)$

s_1, s_2

$c = c_1 \oplus c_2 \wedge$
$V_2^1(x_1, r_1, c, s_1) \wedge$
$V_2^2(x_2, r_2, c, s_2)$

Need to know that $x_2 \in \mathrm{dom}(R_2)$

Summary

Theory of Σ-protocols formalized in Coq

- ZK proofs of preimages of homomorphisms
- Boolean (monotonic) combination of ZK proofs
- Clarification of a recurrent mistake in *OR*-proofs
- Short proofs of several practically relevant Σ-protocols

The Road Ahead

Related work

- Maurer @ AFRICACRYPT'09:
  Unifying ZK Proofs of Knowledge
- Backes, Grochulla, Hritcu, Maffei @ CSF'09:
  Achieving Security Despite Compromise Using ZK
- Almeida et al. @ ESORICS'10:
  A Certifying Compiler for ZK PoK Based on Σ-Protocols

Beyond perfect ZK

- Statistical ZK
- Computational ZK

EasyCrypt

Increasing abstraction and automation will make verifiable security a reasonable and profitable alternative for cryptographers
(talk at FCC on Tuesday)