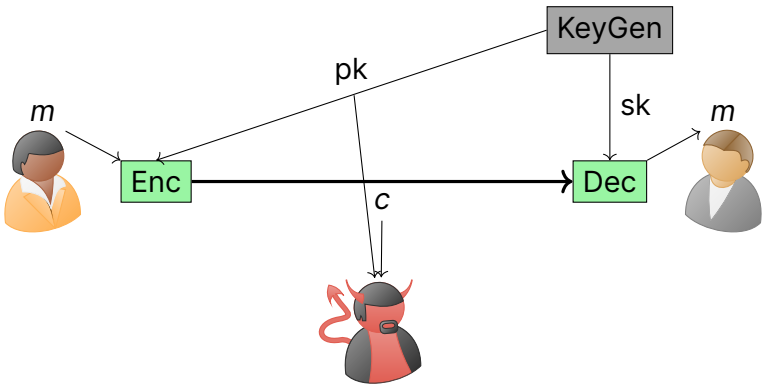
Resources in crypto proofs

Peeter Laud

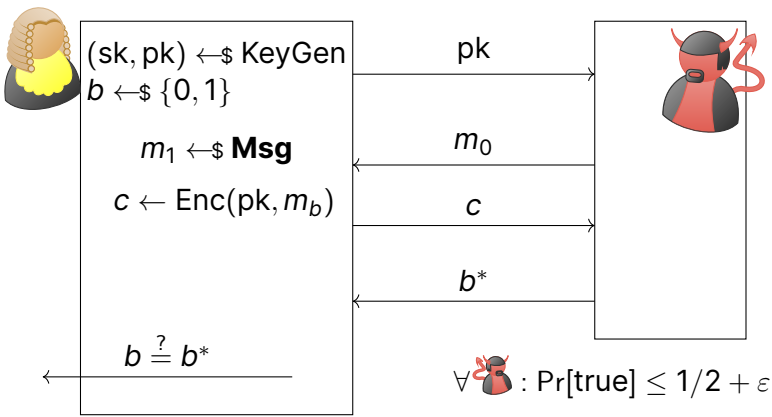
Public-key encryption



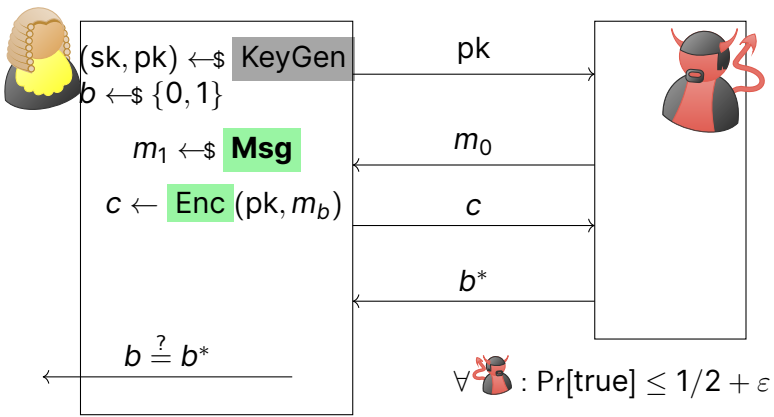
ElGamal encryption

- $\mathbb{G}$ — a large cyclic group. $\mathbb{G} = \langle g \rangle$. $|\mathbb{G}| = p$
- KeyGen works by: $\mathsf{sk} \leftarrow\!\!\$\ \mathbb{Z}_p$, $\mathsf{pk} \leftarrow g^{\mathsf{sk}}$
- Message space: $\mathbf{Msg} = \mathbb{G}$
- $\mathsf{Enc}(\mathsf{pk}, m) = (g^r, m \cdot \mathsf{pk}^r)$, where $r \leftarrow\!\!\$\ \mathbb{Z}_p$
- $\mathsf{Dec}(\mathsf{sk}, (c_1, c_2)) = c_2 / c_1^{\mathsf{sk}}$

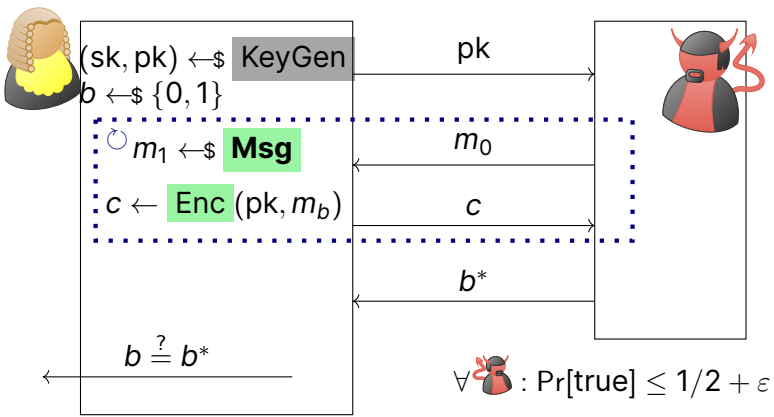
IND-CPA security



IND-CPA security

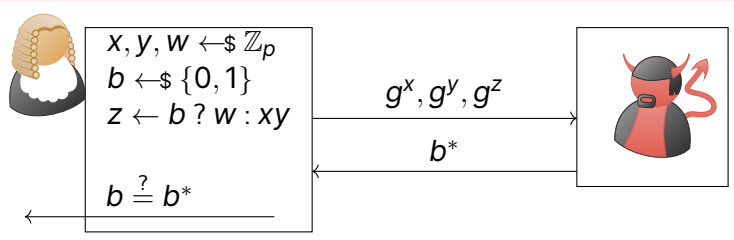


IND-CPA security



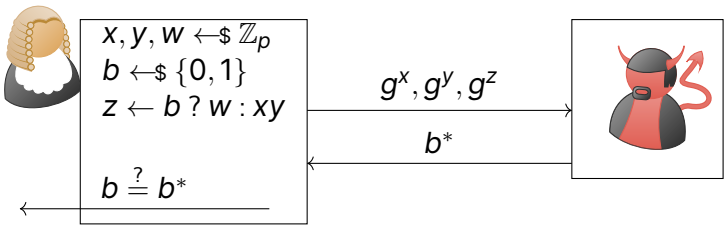
Hardness assumptions

Decisional Diffie-Hellman problem in $\mathbb{G}$



Hardness assumptions

Decisional Diffie-Hellman problem in $\mathbb{G}$

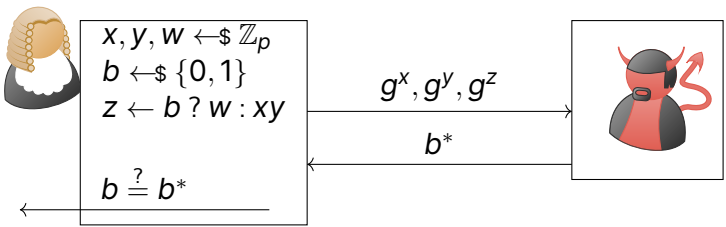


Hardness assumption

Assume that $\forall$  : $\Pr[\text{true}] \leq 1/2 + \varepsilon$

Hardness assumptions

Decisional Diffie-Hellman problem in $\mathbb{G}$



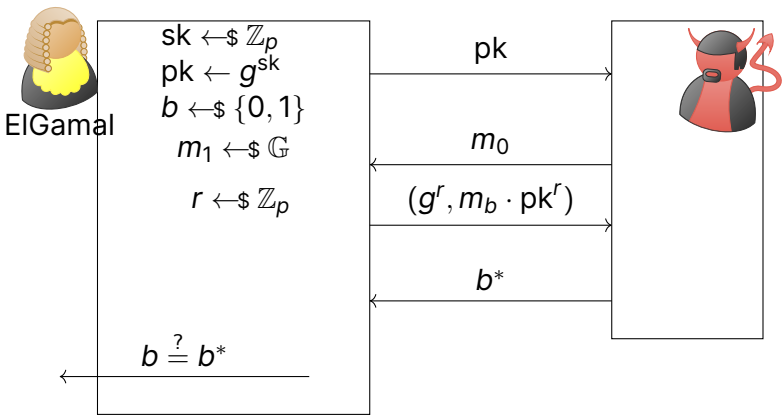
Hardness assumption

Assume that $\forall$  : $\Pr[\text{true}] \leq 1/2 + \varepsilon$

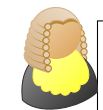
Polynomial running time?

- Yes (for )
- This presentation is a bit fast and loose with that

ElGamal is IND-CPA secure



ElGamal is IND-CPA secure



ElGamal

$\mathsf{sk} \leftarrow\$ \mathbb{Z}_p$

$\mathsf{pk} \leftarrow g^{\mathsf{sk}}$

$b \leftarrow\$ \{0,1\}$

$m_1 \leftarrow\$ \mathbb{G}$

$r \leftarrow\$ \mathbb{Z}_p$

pk

m_0

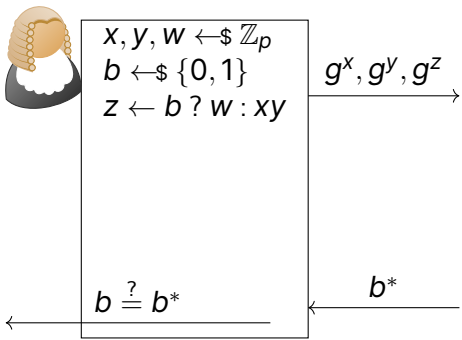
$(g^r, m_b \cdot \mathsf{pk}^r)$

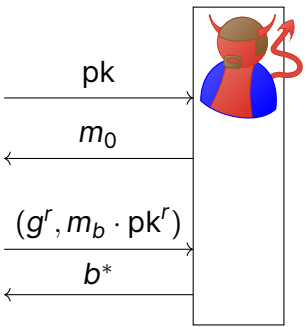
b^*

$b \stackrel{?}{=} b^*$

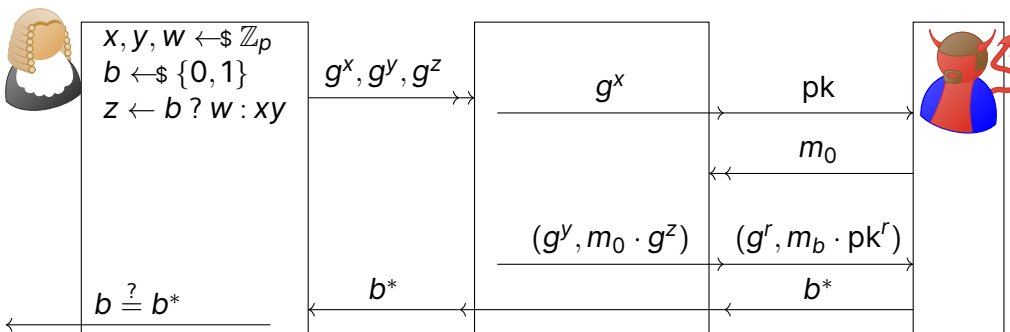
Suppose $\Pr[\text{true}] > 1/2 + \varepsilon$

ElGamal is IND-CPA secure

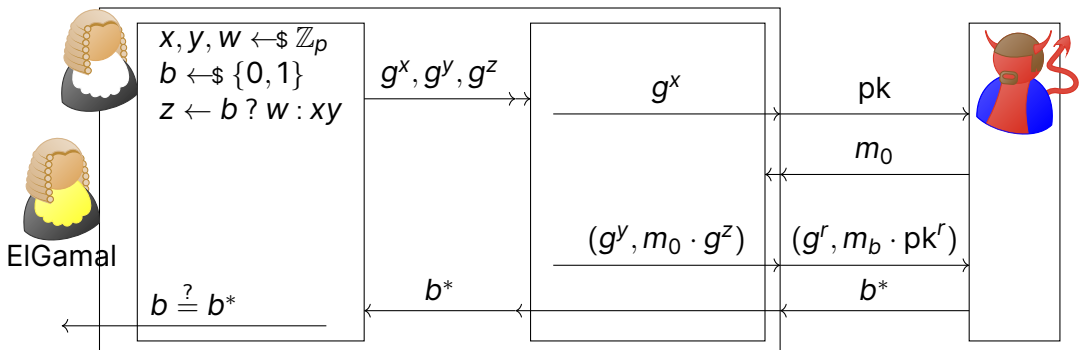




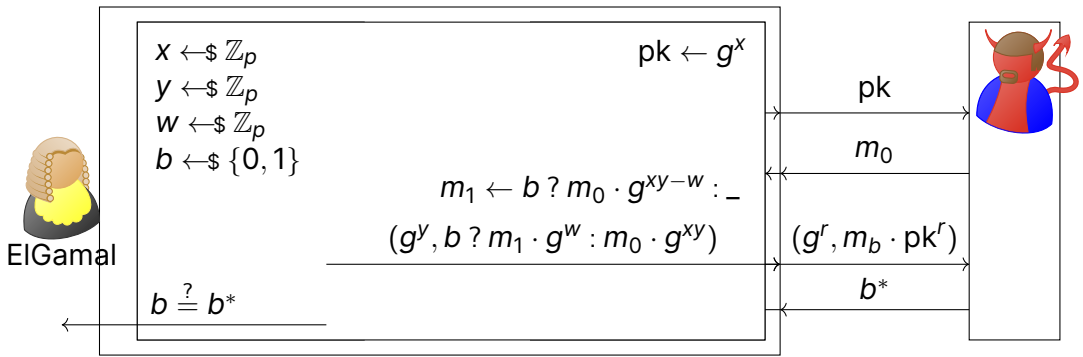
ElGamal is IND-CPA secure



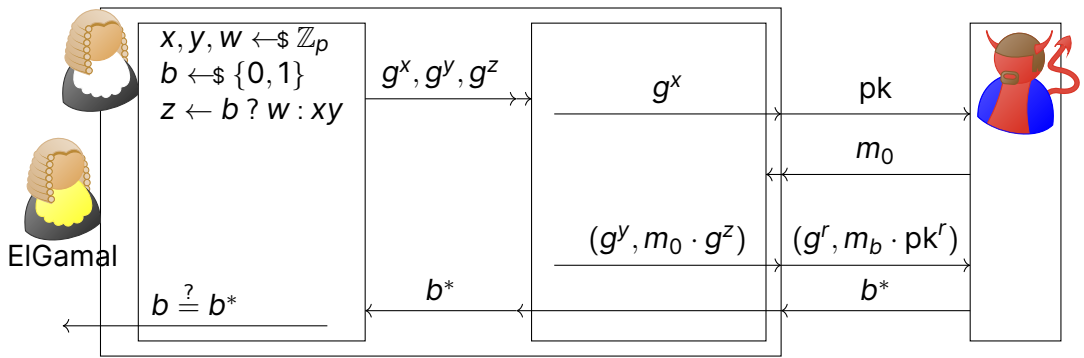
ElGamal is IND-CPA secure



ElGamal is IND-CPA secure

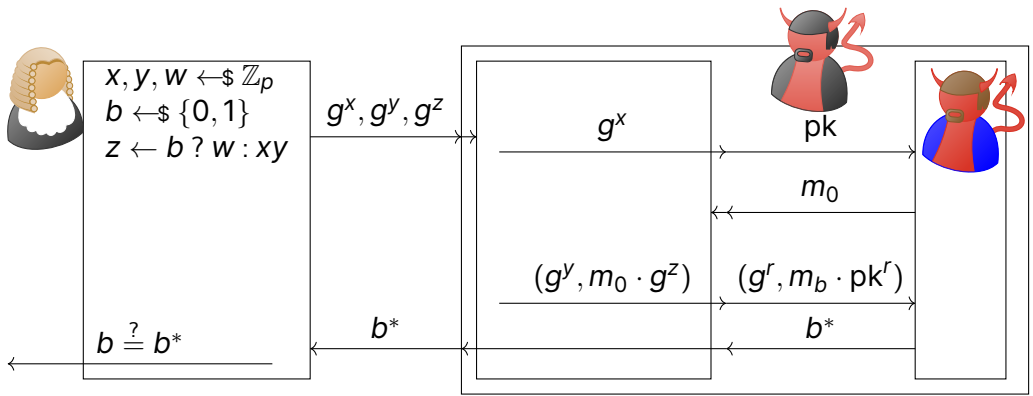


ElGamal is IND-CPA secure



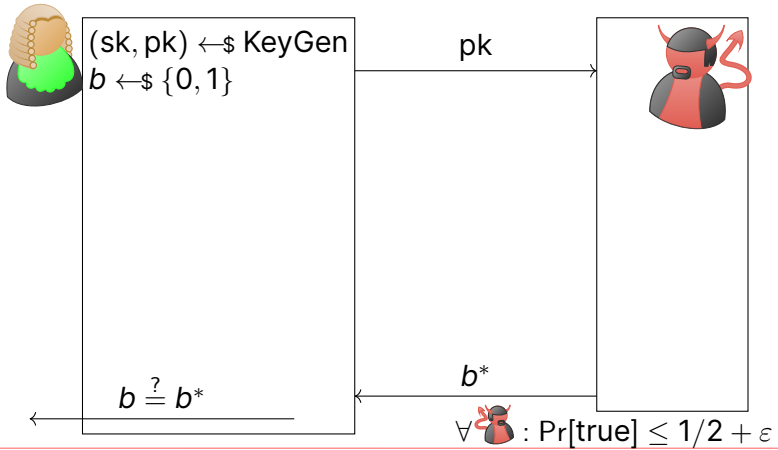
$$\Pr[\text{true}] > 1/2 + \varepsilon$$

ElGamal is IND-CPA secure

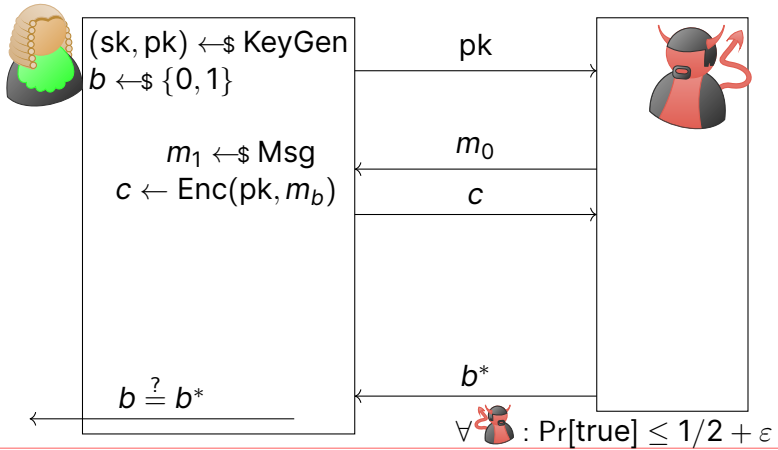


$$\Pr[\text{true}] > 1/2 + \varepsilon$$

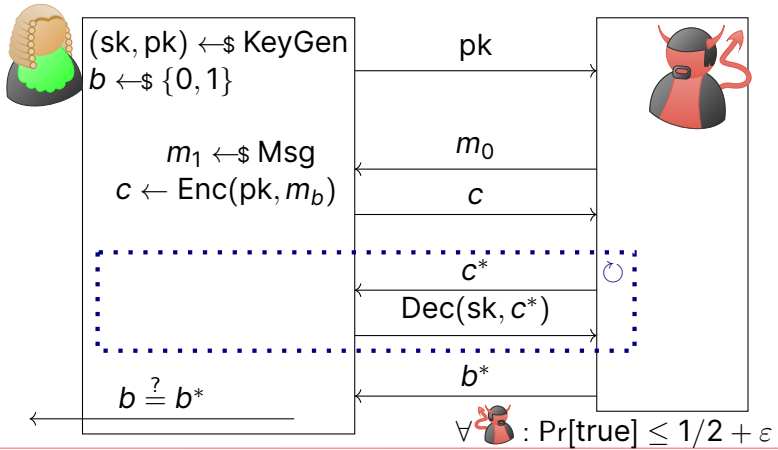
IND-CCA security



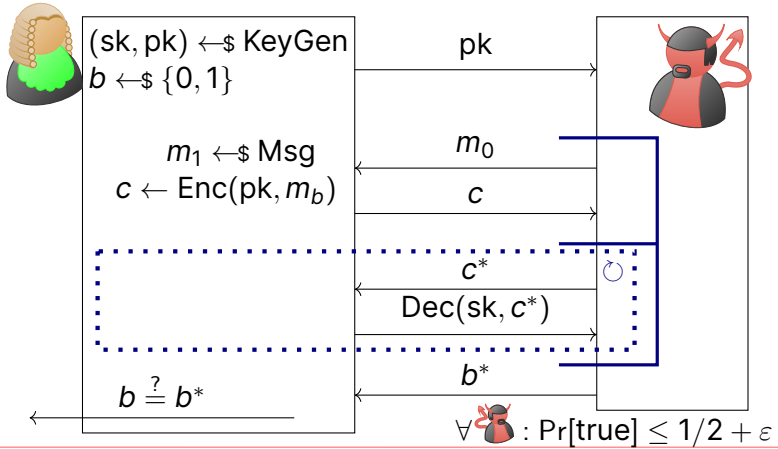
IND-CCA security



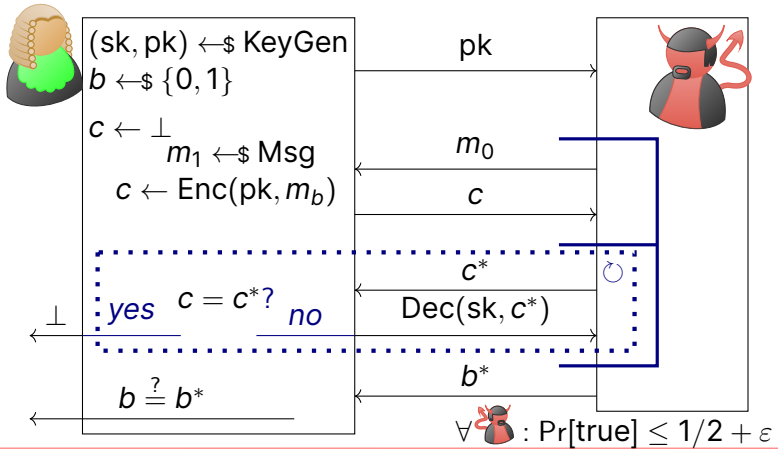
IND-CCA security



IND-CCA security

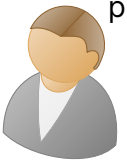


IND-CCA security



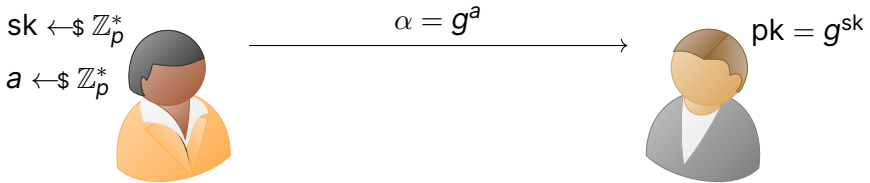
Proof of knowledge of exponent

$\mathsf{sk} \leftarrow\!\$\ \mathbb{Z}_p^*$

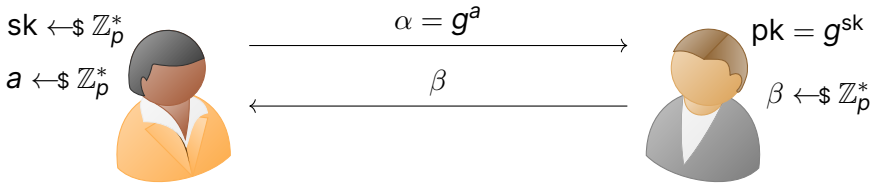


$\mathsf{pk} = g^{\mathsf{sk}}$

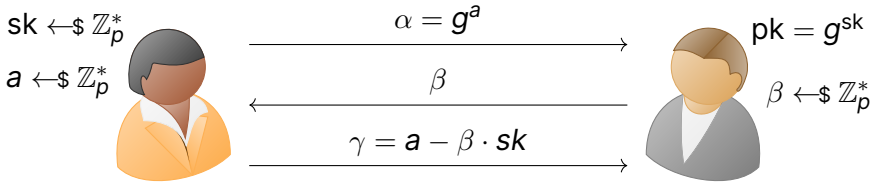
Proof of knowledge of exponent



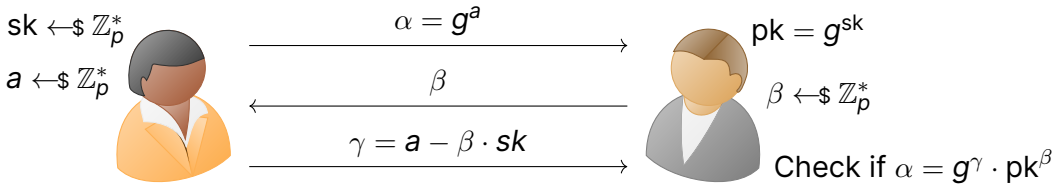
Proof of knowledge of exponent



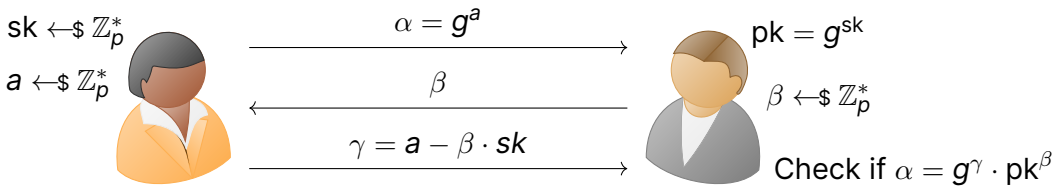
Proof of knowledge of exponent



Proof of knowledge of exponent

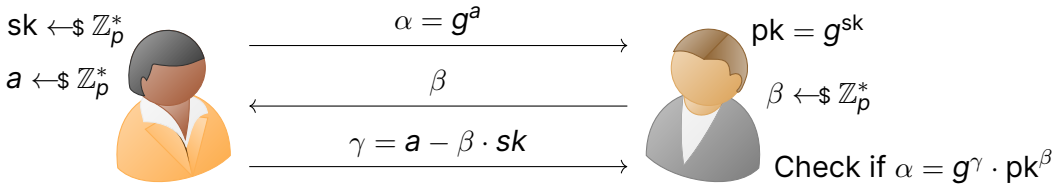


Proof of knowledge of exponent



- **Soundness:** is convinced: ability to respond to several different β-s (for the same α) implies knowledge of *sk*
  - $sk = (\gamma_1 - \gamma_2)/(\beta_2 - \beta_1)$

Proof of knowledge of exponent



- **Soundness:** is convinced: ability to respond to several different β-s (for the same α) implies knowledge of *sk*
  - sk $= (\gamma_1 - \gamma_2)/(\beta_2 - \beta_1)$
- **ZK:** learns nothing: can sample triples (α, β, γ) without
  - β, γ ←$ $\mathbb{Z}_p^*$. Compute $\alpha \leftarrow g^\gamma \cdot pk^\beta$

Fiat-Shamir heuristic

- β is random. A (one-way) hash function H has “random” outputs
- What if 🤔 herself computes $\beta = H(\alpha, \text{ctxt})$?
  - This is called the *Fiat-Shamir heuristic*
- The proof of knowledge is then a single message (α, γ) expressing that
  - 🤔 knows sk
  - 🤔 wanted to present this proof in the context ctxt
- 🧐 verifies by recomputing β, checking the equation

Fiat-Shamir heuristic

- β is random. A (one-way) hash function *H* has “random” outputs
- What if  herself computes $\beta = H(\alpha, \text{ctxt})$?
  - This is called the *Fiat-Shamir heuristic*
- The proof of knowledge is then a single message (α, γ) expressing that
  -  knows sk
  -  wanted to present this proof in the context ctxt
-  verifies by recomputing β, checking the equation
- If β is shorter than α, and α can be computed from $\beta, \gamma, \text{pk}, \text{ctxt}$, then the proof message may be (β, γ)

ElGamal with proof-of-knowledge of exponent

- $\mathbb{G}$ — a large cyclic group. $\mathbb{G} = \langle g \rangle$. $|\mathbb{G}| = p$
- KeyGen works by: $\mathsf{sk} \leftarrow\!\!\$\ \mathbb{Z}_p$, $\mathsf{pk} \leftarrow g^{\mathsf{sk}}$
- Message space: **Msg** $= \mathbb{G}$

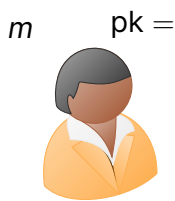
Enc(pk, m)

- $r, r' \leftarrow\!\!\$\ \mathbb{Z}_p$
- $c_1 \leftarrow g^r$, $c_2 \leftarrow m \cdot \mathsf{pk}^r$, $\alpha \leftarrow g^{r'}$
- $\beta \leftarrow H(c_1, c_2, \alpha)$
- $\gamma \leftarrow r' + \beta \cdot r$
- output $(c_1, c_2, \alpha, \gamma)$

Dec(sk, $(c_1, c_2, \alpha, \gamma)$)

- $\beta \leftarrow H(c_1, c_2, \alpha)$
- $g^\gamma \stackrel{?}{=} \alpha \cdot c_1^\beta$
- output c_2 / c_1^{sk}

Random oracle model



Random oracle model

m

$\text{pk} = g^{\text{sk}}$

sk

$r, r' \leftarrow\$ \mathbb{Z}_p$

$c_1 \leftarrow g^r$

$c_2 \leftarrow m \cdot \text{pk}^r$

$\alpha \leftarrow g^{r'}$

Random oracle model

m

$r, r' \leftarrow\$ \mathbb{Z}_p$

$c_1 \leftarrow g^r$

$c_2 \leftarrow m \cdot \mathrm{pk}^r$

$\alpha \leftarrow g^{r'}$

$\mathrm{pk} = g^{\mathrm{sk}}$

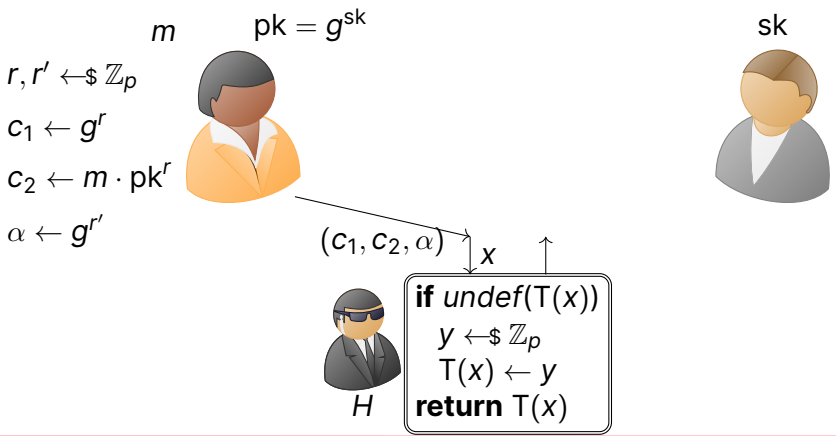
sk

x

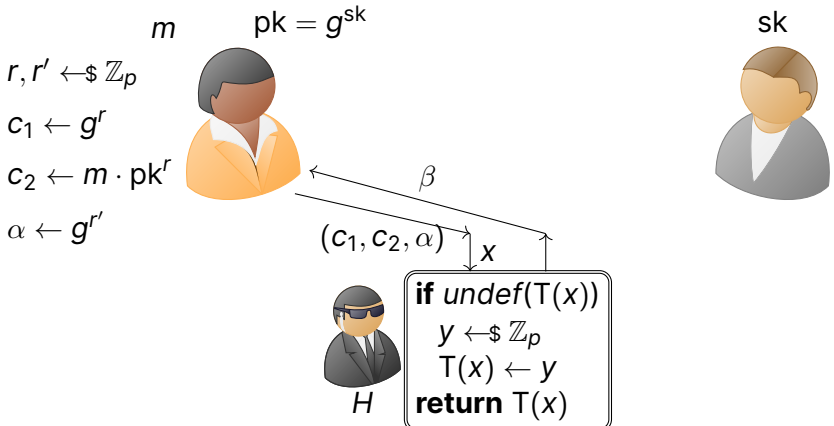
H

if *undef*$(\mathsf{T}(x))$
  $y \leftarrow\$ \mathbb{Z}_p$
  $\mathsf{T}(x) \leftarrow y$
return $\mathsf{T}(x)$

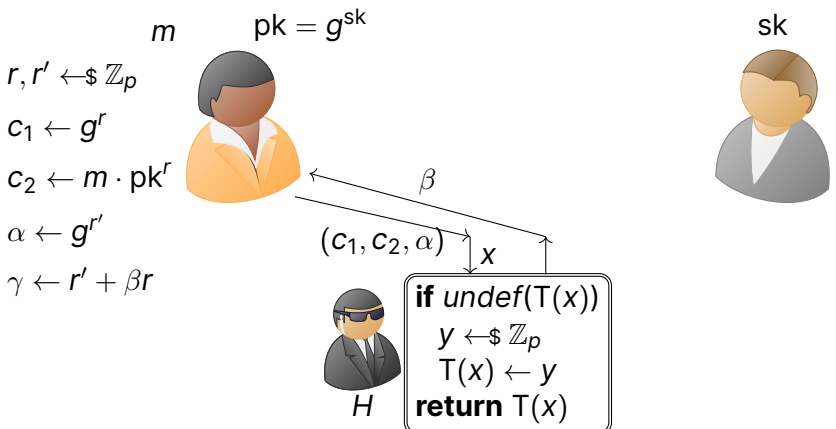
Random oracle model



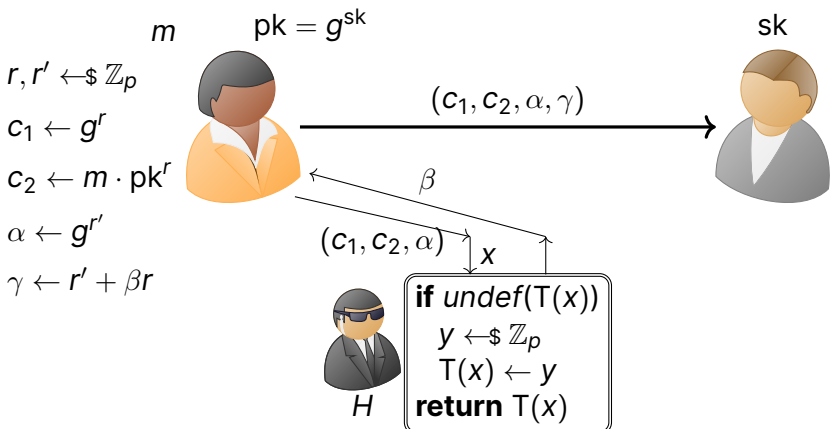
Random oracle model



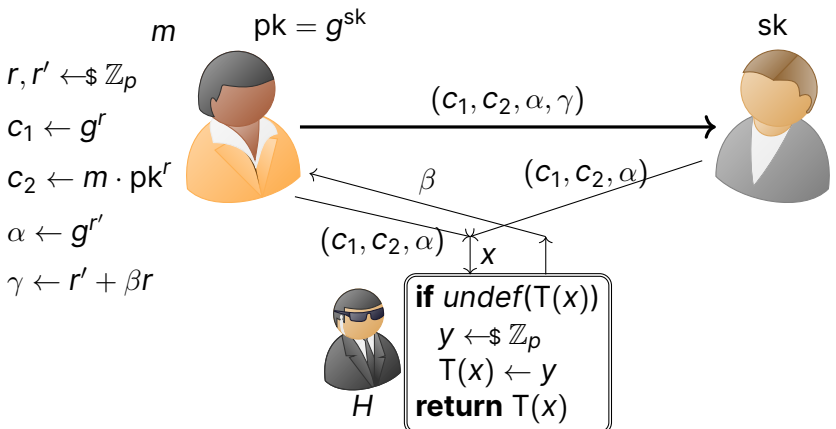
Random oracle model



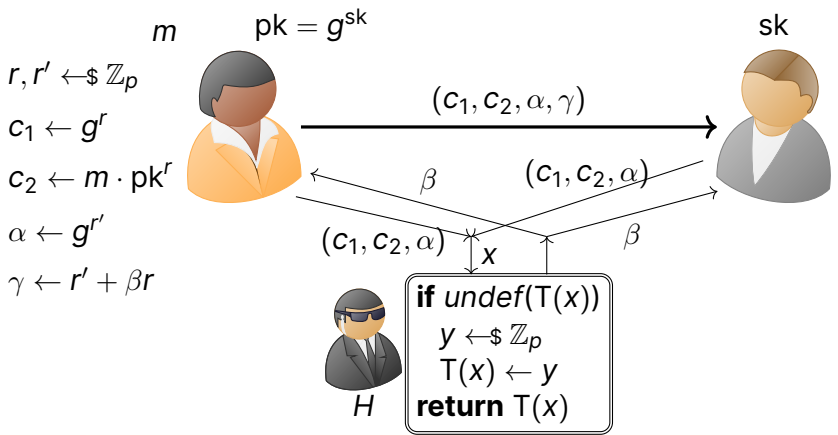
Random oracle model



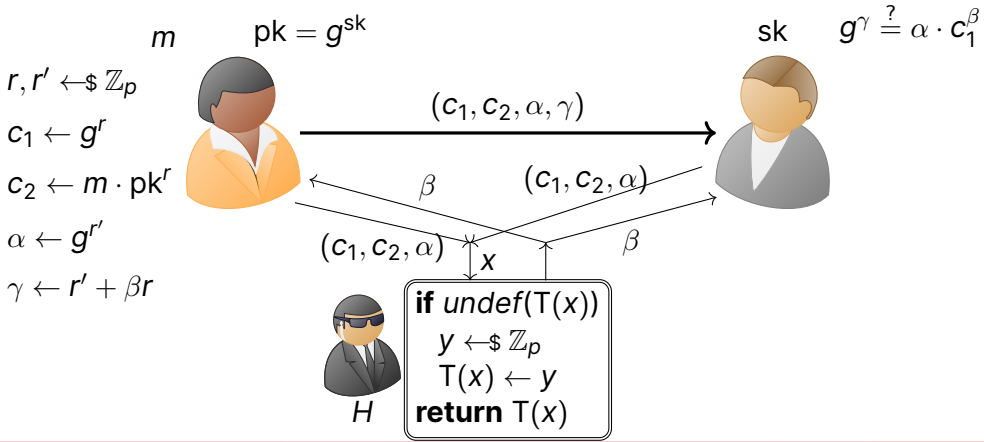
Random oracle model



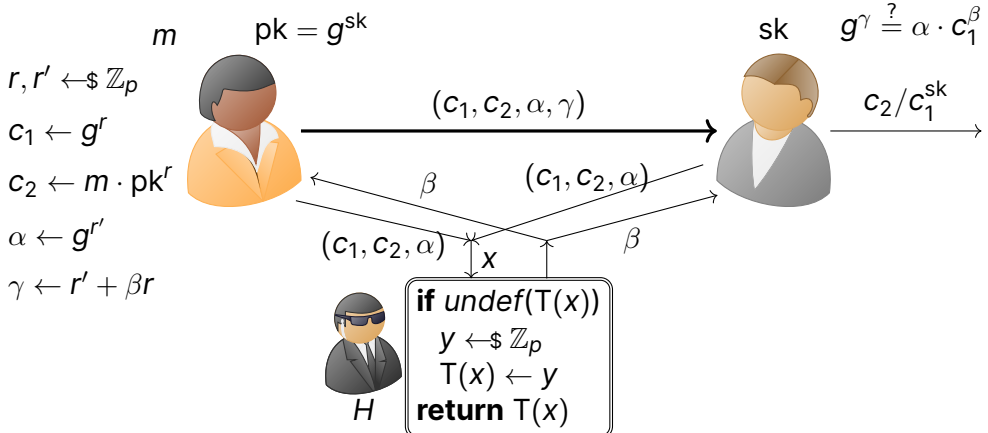
Random oracle model



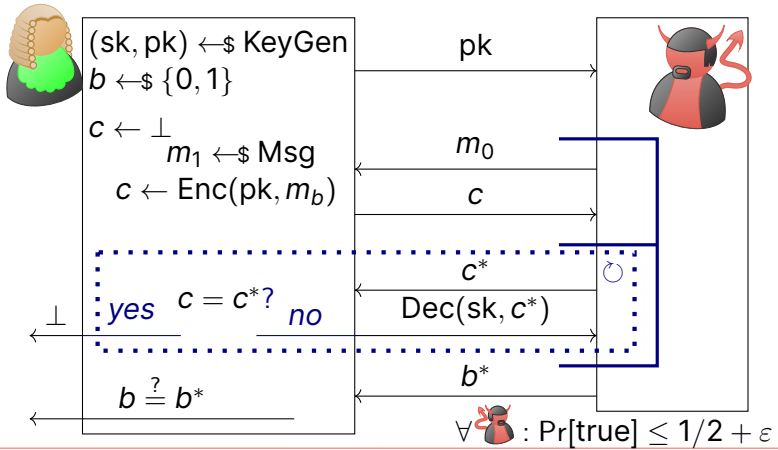
Random oracle model



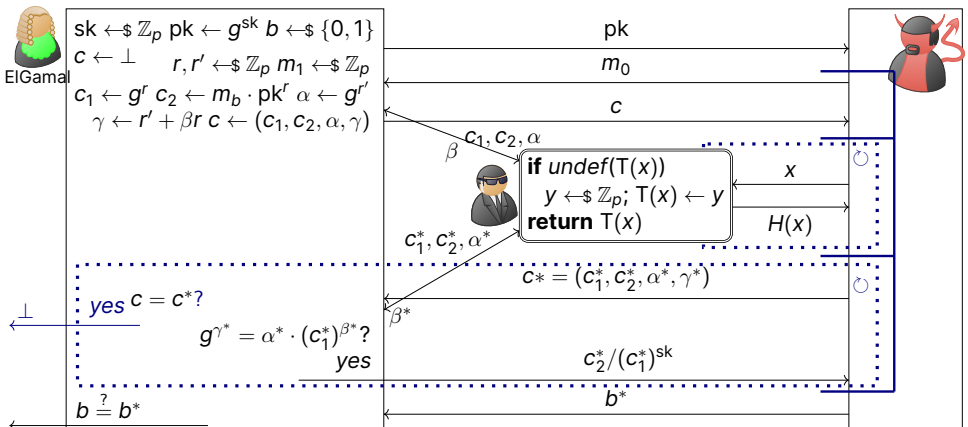
Random oracle model



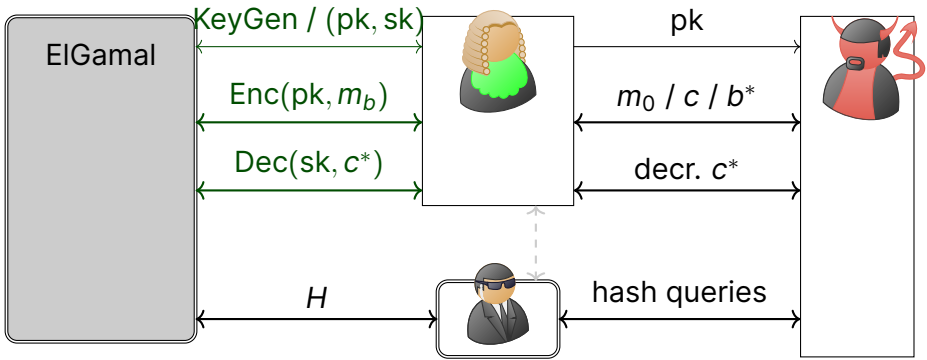
IND-CCA security



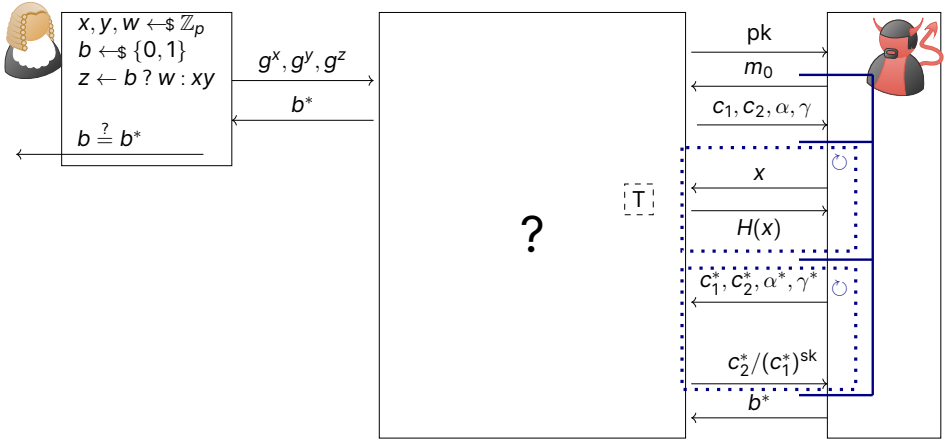
ElGamal with proof of kn. of. exp. & IND-CCA



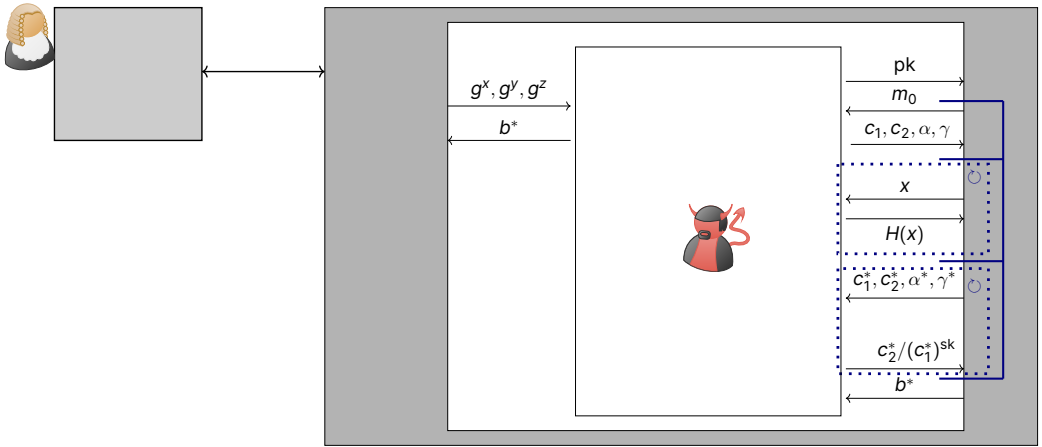
The same, with more boxes



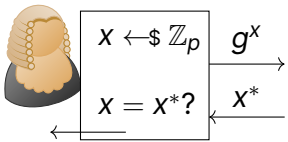
The reduction?



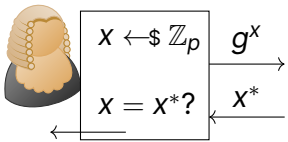
Meta-reductions

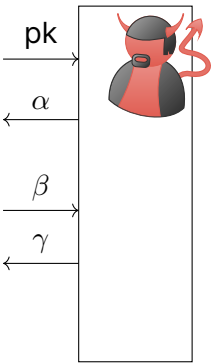


Rewinding / rewindability

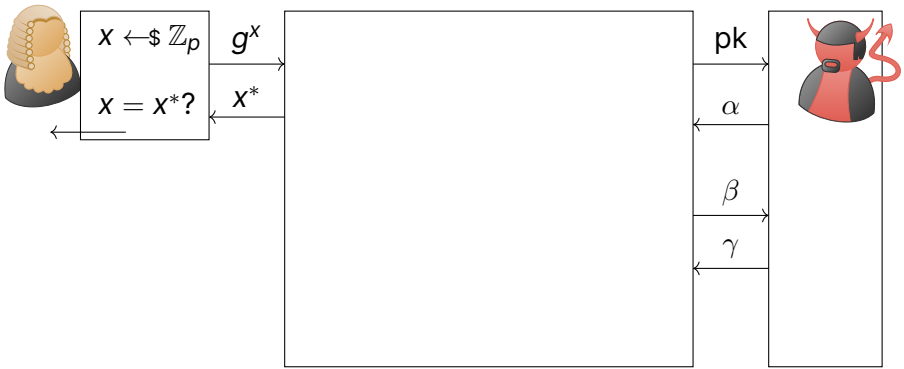


Rewinding / rewindability

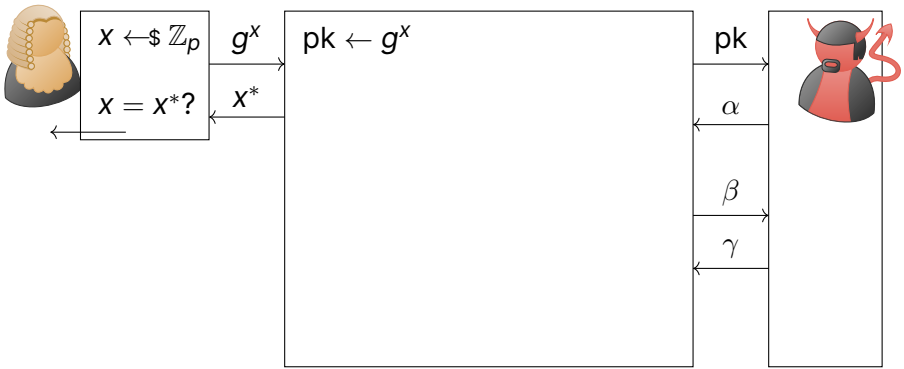




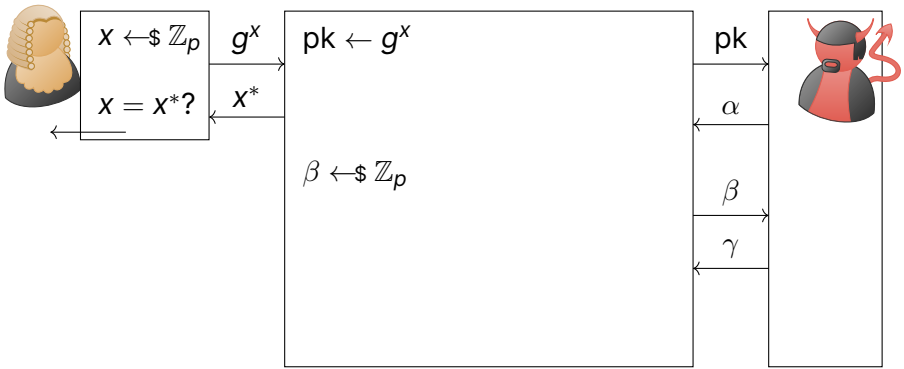
Rewinding / rewindability



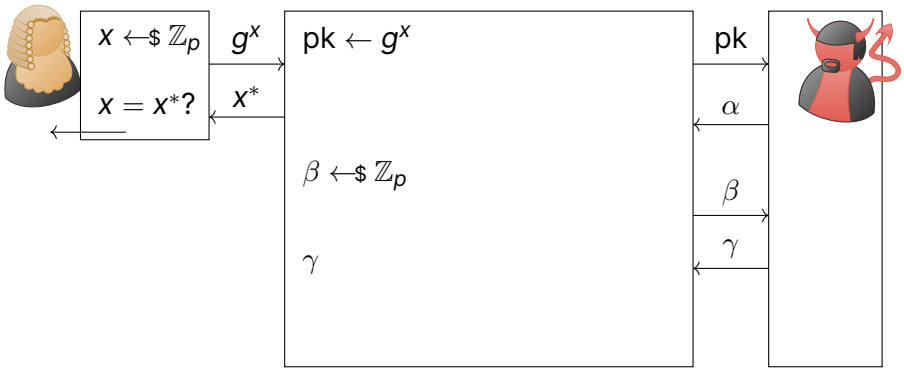
Rewinding / rewindability



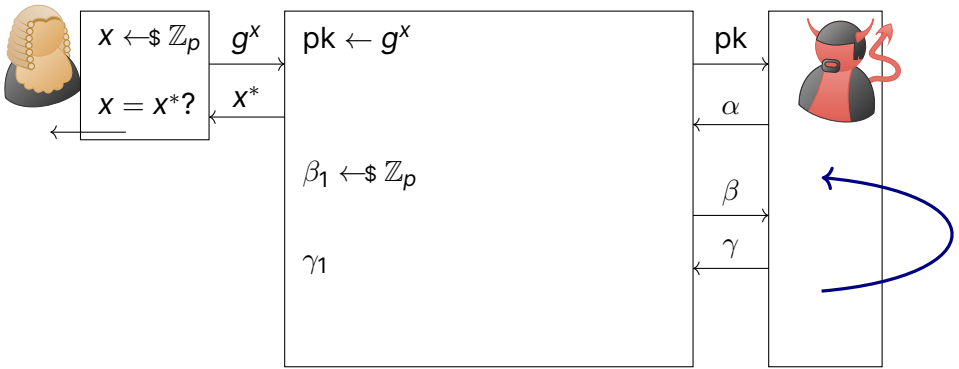
Rewinding / rewindability



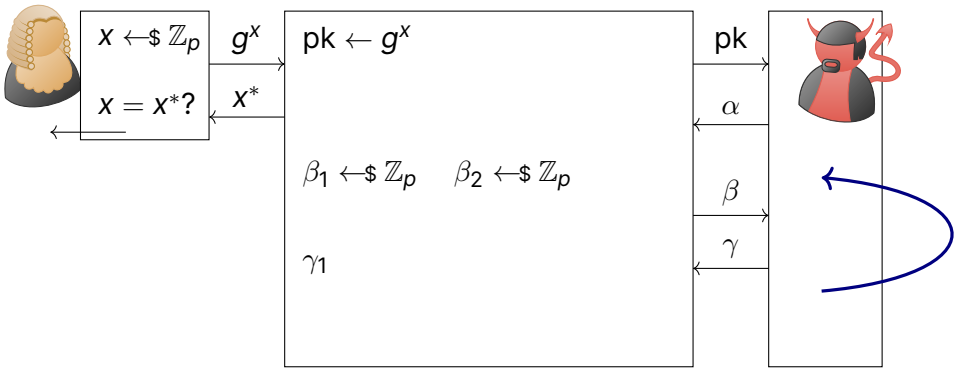
Rewinding / rewindability



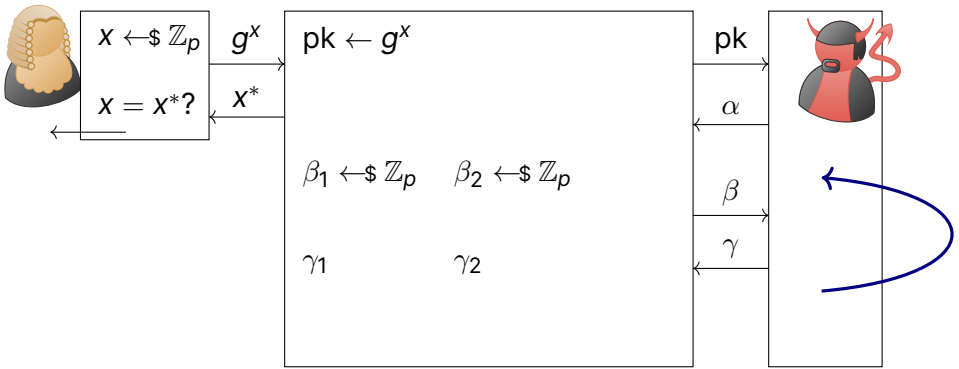
Rewinding / rewindability



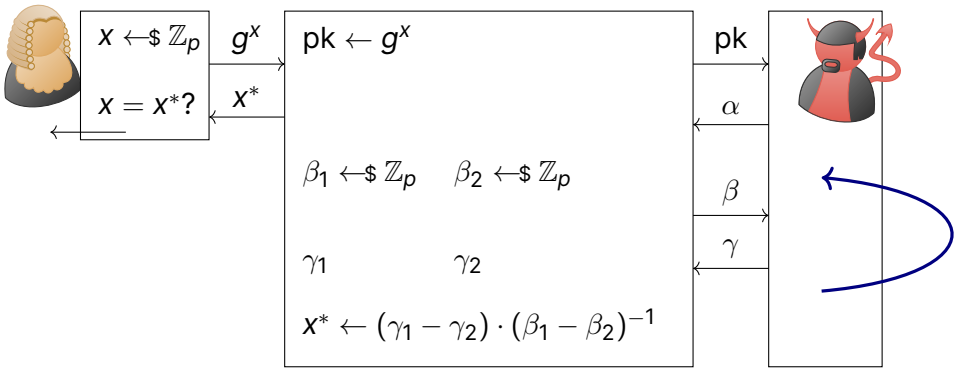
Rewinding / rewindability



Rewinding / rewindability



Rewinding / rewindability



Example: bulletproofs

- Public elements $g_1, \ldots, g_n, h_1, \ldots, h_n, \mathrm{P}, u \in \mathbb{G}$
- wants to convince  that she knows $a_1, \ldots, a_n, b_1, \ldots, b_n \in \mathbb{Z}_p$, such that

$$u^{\sum_{i=1}^{n} a_i b_i} \cdot \prod_{i=1}^{n} g_i^{a_i} h_i^{b_i} = \mathrm{P}$$

Example: bulletproofs

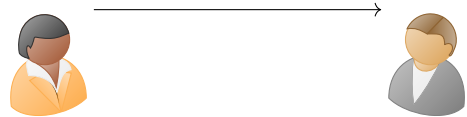
- Public elements $g_1, \ldots, g_n, h_1, \ldots, h_n, \mathrm{P}, u \in \mathbb{G}$
- wants to convince  that she knows $a_1, \ldots, a_n, b_1, \ldots, b_n \in \mathbb{Z}_p$, such that

$$u^{\sum_{i=1}^{n} a_i b_i} \cdot \prod_{i=1}^{n} g_i^{a_i} h_i^{b_i} = \mathrm{P}$$

- Privacy is not important
  - Could just send $a_1, \ldots, a_n, b_1, \ldots, b_n$
  - Can we be more efficient than this?

The protocol

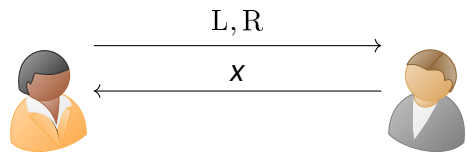
- Let $m = n/2$.



The protocol

- Let $m = n/2$. computes and sends to :

$$\mathrm{L} \leftarrow u^{\sum_{i=1}^{m} a_i b_{i+m}} \cdot \prod_{i=1}^{m} g_{i+m}^{a_i} h_i^{b_{i+m}} \qquad \mathrm{R} \leftarrow u^{\sum_{i=1}^{m} a_{i+m} b_i} \cdot \prod_{i=1}^{m} g_i^{a_{i+m}} h_{i+m}^{b_i}$$



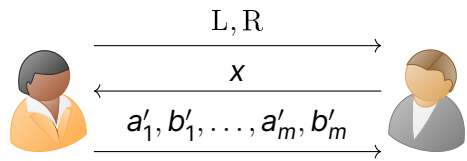
The protocol

- Let $m = n/2$. computes and sends to :

$$\mathrm{L} \leftarrow u^{\sum_{i=1}^{m} a_i b_{i+m}} \cdot \prod_{i=1}^{m} g_{i+m}^{a_i} h_i^{b_{i+m}} \qquad \mathrm{R} \leftarrow u^{\sum_{i=1}^{m} a_{i+m} b_i} \cdot \prod_{i=1}^{m} g_i^{a_{i+m}} h_{i+m}^{b_i}$$

- sends random challenge $x \xleftarrow{\$} \mathbb{Z}_p$

The protocol

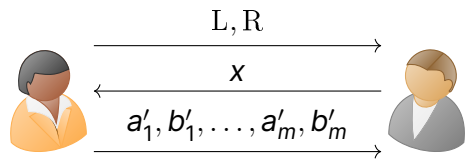


- Let $m = n/2$. computes and sends to :

$$\mathrm{L} \leftarrow u^{\sum_{i=1}^{m} a_i b_{i+m}} \cdot \prod_{i=1}^{m} g_{i+m}^{a_i} h_i^{b_{i+m}} \qquad \mathrm{R} \leftarrow u^{\sum_{i=1}^{m} a_{i+m} b_i} \cdot \prod_{i=1}^{m} g_i^{a_{i+m}} h_{i+m}^{b_i}$$

- sends random challenge $x \xleftarrow{\$} \mathbb{Z}_p$
- sends $a'_i = xa_i + x^{-1}a_{i+m}$ and $b'_i = x^{-1}b_i + xb_{i+m}$ to $(1 \le i \le m)$

The protocol



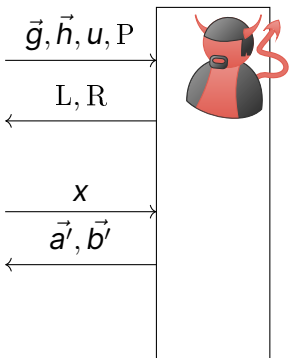
- Let $m = n/2$. computes and sends to :

$$\mathrm{L} \leftarrow u^{\sum_{i=1}^{m} a_i b_{i+m}} \cdot \prod_{i=1}^{m} g_{i+m}^{a_i} h_i^{b_{i+m}} \qquad \mathrm{R} \leftarrow u^{\sum_{i=1}^{m} a_{i+m} b_i} \cdot \prod_{i=1}^{m} g_i^{a_{i+m}} h_{i+m}^{b_i}$$

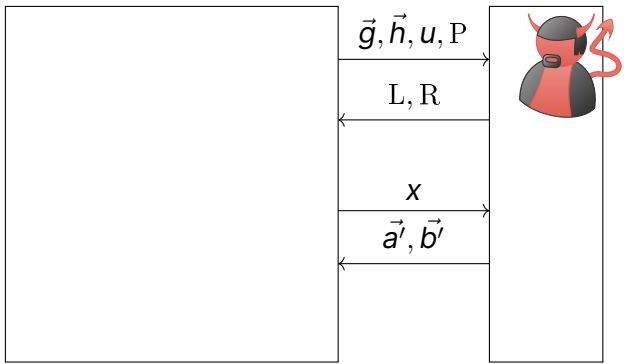
- sends random challenge $x \xleftarrow{\$} \mathbb{Z}_p$
- sends $a'_i = xa_i + x^{-1}a_{i+m}$ and $b'_i = x^{-1}b_i + xb_{i+m}$ to $(1 \le i \le m)$
- checks

$$\mathrm{L}^{x^2} \mathrm{P} \mathrm{R}^{x^{-2}} \stackrel{?}{=} u^{\sum_{i=1}^{m} a'_i b'_i} \cdot \prod_{i=1}^{m} \left(g_i^{x^{-1}} g_{i+m}^{x} \right)^{a'_i} \left(h_i^{x} h_{i+m}^{x^{-1}} \right)^{b'_i}$$

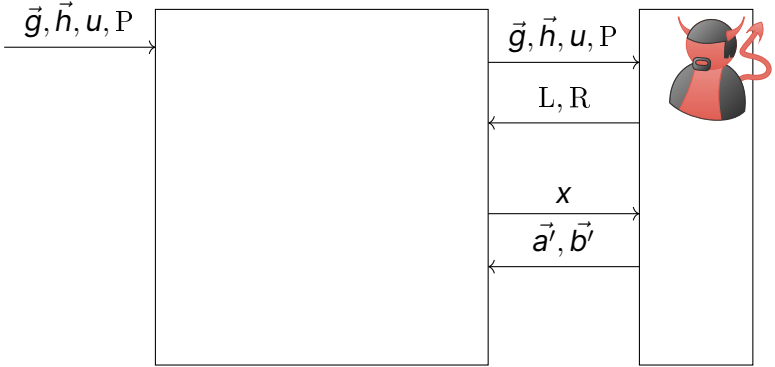
Reduction



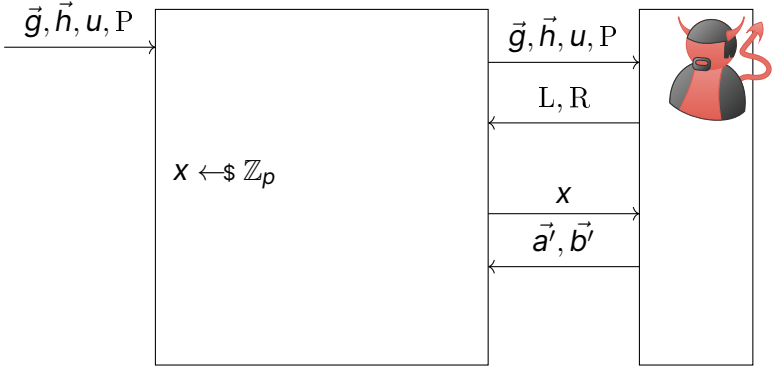
Reduction



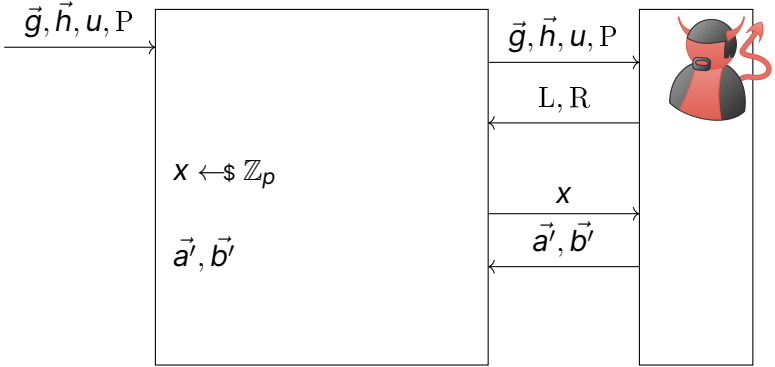
Reduction


$\vec{g}, \vec{h}, u, \mathrm{P}$
$\vec{g}, \vec{h}, u, \mathrm{P}$
L, R
x
$\vec{a}', \vec{b}'$

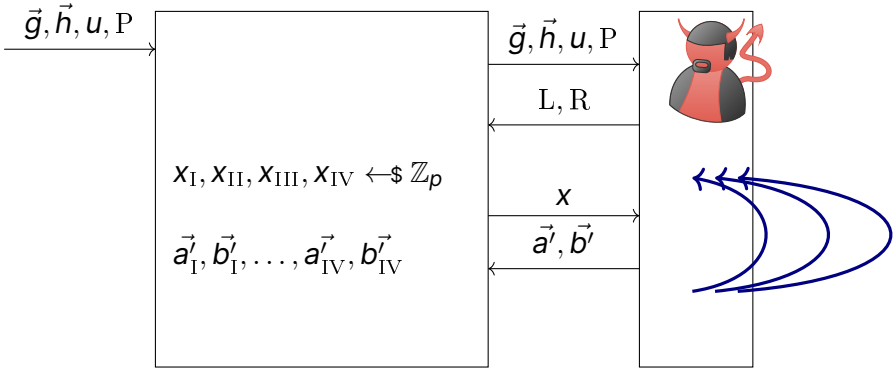
Reduction


$\vec{g}, \vec{h}, u, \mathrm{P}$
$x \leftarrow\$ \mathbb{Z}_p$
$\vec{g}, \vec{h}, u, \mathrm{P}$
L, R
x
$\vec{a'}, \vec{b'}$

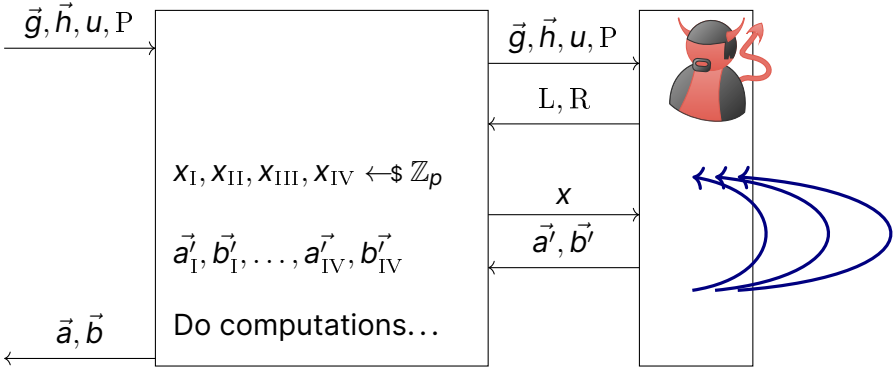
Reduction

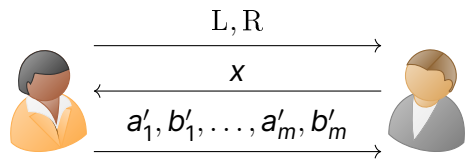

$\vec{g}, \vec{h}, u, \mathrm{P}$
$x \leftarrow\$ \mathbb{Z}_p$
$\vec{a}', \vec{b}'$
$\vec{g}, \vec{h}, u, \mathrm{P}$
L, R
x
$\vec{a}', \vec{b}'$

Reduction


$\vec{g}, \vec{h}, u, \mathrm{P}$
$x_{\mathrm{I}}, x_{\mathrm{II}}, x_{\mathrm{III}}, x_{\mathrm{IV}} \leftarrow\$ \mathbb{Z}_p$
$\vec{a'_{\mathrm{I}}}, \vec{b'_{\mathrm{I}}}, \ldots, \vec{a'_{\mathrm{IV}}}, \vec{b'_{\mathrm{IV}}}$
$\vec{g}, \vec{h}, u, \mathrm{P}$
L, R
x
$\vec{a'}, \vec{b'}$

Reduction


$\vec{g}, \vec{h}, u, \mathrm{P}$
$x_{\mathrm{I}}, x_{\mathrm{II}}, x_{\mathrm{III}}, x_{\mathrm{IV}} \leftarrow\$ \mathbb{Z}_p$
$\vec{a'_{\mathrm{I}}}, \vec{b'_{\mathrm{I}}}, \ldots, \vec{a'_{\mathrm{IV}}}, \vec{b'_{\mathrm{IV}}}$
Do computations...
$\vec{a}, \vec{b}$
$\vec{g}, \vec{h}, u, \mathrm{P}$
L, R
x
$\vec{a'}, \vec{b'}$



Recursion

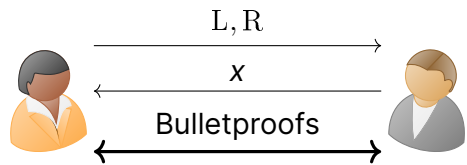
- Let $m = n/2$. computes and sends to :

$$\mathrm{L} \leftarrow u^{\sum_{i=1}^{m} a_i b_{i+m}} \cdot \prod_{i=1}^{m} g_{i+m}^{a_i} h_i^{b_{i+m}} \qquad \mathrm{R} \leftarrow u^{\sum_{i=1}^{m} a_{i+m} b_i} \cdot \prod_{i=1}^{m} g_i^{a_{i+m}} h_{i+m}^{b_i}$$

- sends random challenge $x \xleftarrow{\$} \mathbb{Z}_p$
- sends $a'_i = xa_i + x^{-1}a_{i+m}$ and $b'_i = x^{-1}b_i + xb_{i+m}$ to $(1 \le i \le m)$
- checks

$$\mathrm{L}^{x^2}\mathrm{P}\mathrm{R}^{x^{-2}} \stackrel{?}{=} u^{\sum_{i=1}^{m} a'_i b'_i} \cdot \prod_{i=1}^{m} \left(g_i^{x^{-1}} g_{i+m}^{x}\right)^{a'_i} \left(h_i^{x} h_{i+m}^{x^{-1}}\right)^{b'_i}$$

Recursion



- Let $m = n/2$.  computes and sends to  :

$$\mathrm{L} \leftarrow u^{\sum_{i=1}^{m} a_i b_{i+m}} \cdot \prod_{i=1}^{m} g_{i+m}^{a_i} h_i^{b_{i+m}} \qquad \mathrm{R} \leftarrow u^{\sum_{i=1}^{m} a_{i+m} b_i} \cdot \prod_{i=1}^{m} g_i^{a_{i+m}} h_{i+m}^{b_i}$$

-  sends random challenge $x \xleftarrow{\$} \mathbb{Z}_p$
-  convinces  that she knows a'_i, b'_i $(1 \le i \le m)$, such that

$$\mathrm{L}^{x^2} \mathrm{P} \mathrm{R}^{x^{-2}} \stackrel{?}{=} u^{\sum_{i=1}^{m} a'_i b'_i} \cdot \prod_{i=1}^{m} \left(g_i^{x^{-1}} g_{i+m}^{x} \right)^{a'_i} \left(h_i^{x} h_{i+m}^{x^{-1}} \right)^{b'_i}$$

Game-based vs. ideal functionality based definitions

Think of functionalities, talking to multiple parties

Game-based

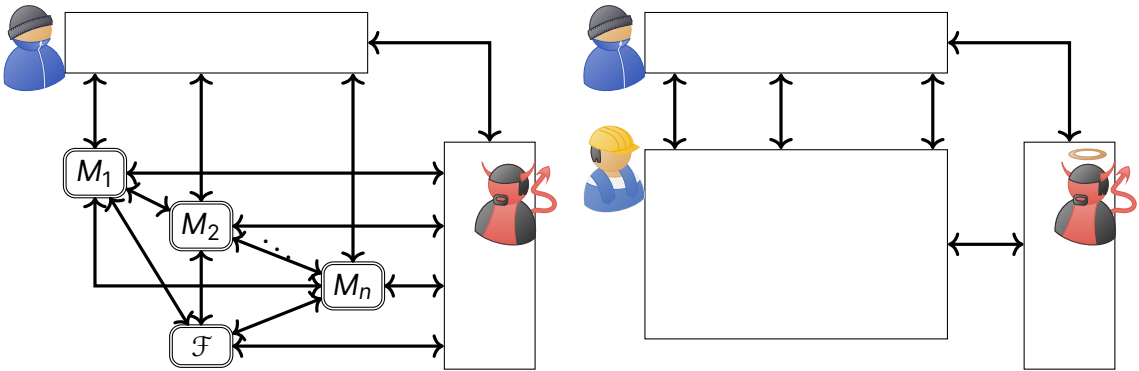
- Implementation must satisfy a number of properties
- Each property is given by a game b/w  and 
  -  runs some parts of the implementation, and does some checks
-  may not “win”

Id. fun. based

- There’s an ideal machine 
  - Talks to all parties
  - Also talks to 
- Implementation must be *at least as secure* as 

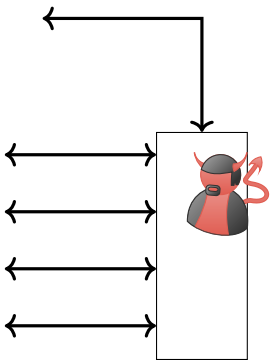
Benefit of this style of definition: we are less likely to mis-specify the security requirements

Universal composability

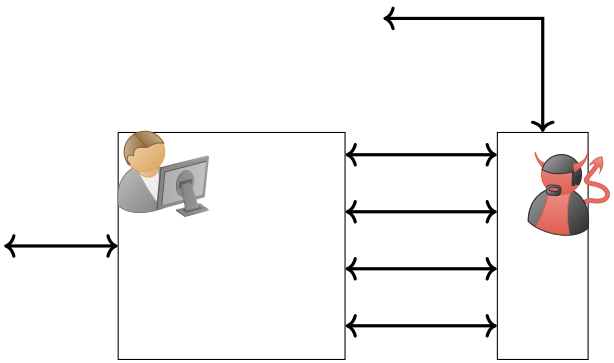


“at least as secure as”: For each , there must be a , such that the views of are indistinguishable in these two scenarios

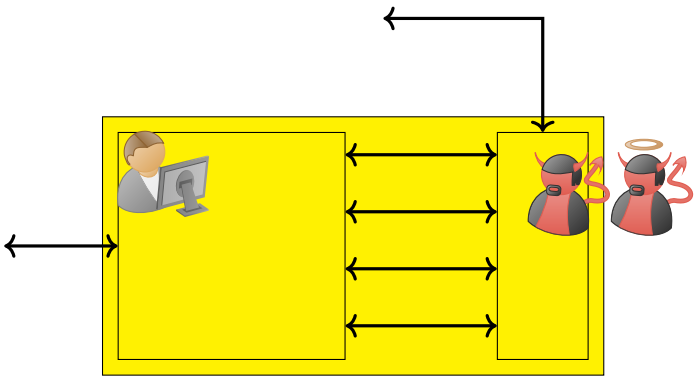
Simulator, or: where do we get 



Simulator, or: where do we get 



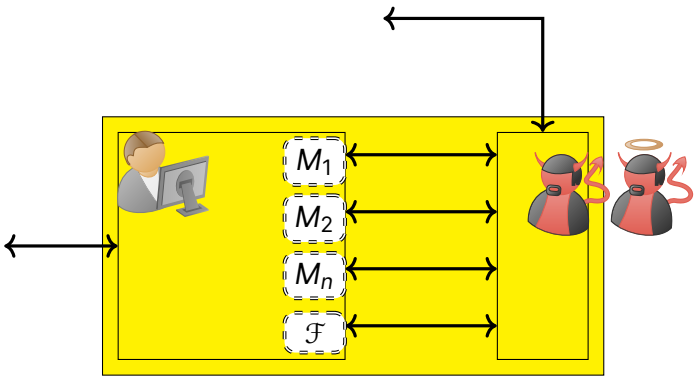
Simulator, or: where do we get 



To do (in proofs):

- Propose 
- Show that the real implementation is indistinguishable from the composition of  and 

Simulator, or: where do we get 



To do (in proofs):

- Propose 
- Show that the real implementation is indistinguishable from the composition of  and 

Example. 👷 for signatures. Key generation

- P_i (in 🥷) asks 👷 for pk_i

- 👷 asks 😇 for some bit-string pk_i
- 👷 receives pk_i from 😇
- 👷 stores (i, pk_i) in the table of keys
- 👷 sends pk_i to P_i

Example.  for signatures. Key generation

- P_i (in ) asks  for pk_i
-  ignores the request if $(i, \ldots)$ is in the table of keys. Otherwise:
-  asks  for some bit-string pk_i
-  receives pk_i from 
-  stores (i, pk_i) in the table of keys
-  sends pk_i to P_i

Example. for signatures. Signing

- P_i (in) asks to sign message M
- ignores the request if $(i, \ldots)$ is not in the table of keys. Otherwise:
- sends M to and asks for the value σ of a signature
- receives σ from
- stores $(\mathrm{pk}_i, M, \sigma, 1)$ in the table of signatures
- sends σ to P_i

Example. for signatures. Verification

- P_i (in) asks to verify $(\mathrm{pk}_j, M, \sigma)$
- If $(\mathrm{pk}_j, M, \sigma, b)$ is in the table of signatures, then answers "b" to P_i
- If $(\ldots, \mathrm{pk}_j)$ is in the table of keys, but there is no $(\mathrm{pk}_j, M, \ldots)$ in the table of signatures, then answers "0" to P_i
- Otherwise, sends $(\mathrm{pk}_j, M, \sigma)$ to , getting back b^*
- stores $(\mathrm{pk}_j, M, \sigma, b^*)$ in the table of signatures, and answers "b^*" to P_i

Example. Signatures. Implementation

- Let **Sig** $=$ (**KeyGen**, **Sign**, **Verify**) be an existentially unforgeable (under chosen-message attacks) signature scheme
- Machine M_i is totally intuitive:
  - uses **KeyGen** to generate a key; stores sk
  - uses **Sign**$(\mathrm{sk}, \cdot)$ to sign
  - uses **Verify** to verify signatures

Resources

- There is a well-defined interface b/w  and  , and b/w  and 
-  makes queries (sends messages) to  , and vice versa
- (In general:)  makes queries (sends messages) to  and receives replies
  - May correspond to  playing a *corrupted* party
  - May correspond to  simulating  controlling the network
- (As we saw:)  makes queries to  and receives replies
  - Assumption:  responds

Alternative to responsive 

- At the start,  gives **KeyGen**, **Sig**, **Verify** to 
  - These algorithms are given as executable code
- Instead of consulting  ,  runs these algorithms
  - But maintains and uses the tables, too

Alternative to responsive 

- At the start,  gives **KeyGen**, **Sig**, **Verify** to 
  - These algorithms are given as executable code
- Instead of consulting  ,  runs these algorithms
  - But maintains and uses the tables, too
- In general…this is less flexible
  - Recall $\mathcal{F}$ as part of 

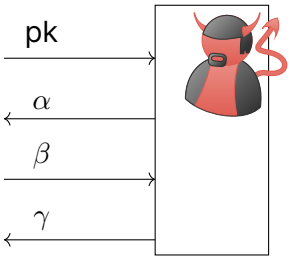
Information flows

- Making queries to  gives information to 
- Can we model that  do not learn anything while  responds to the query?

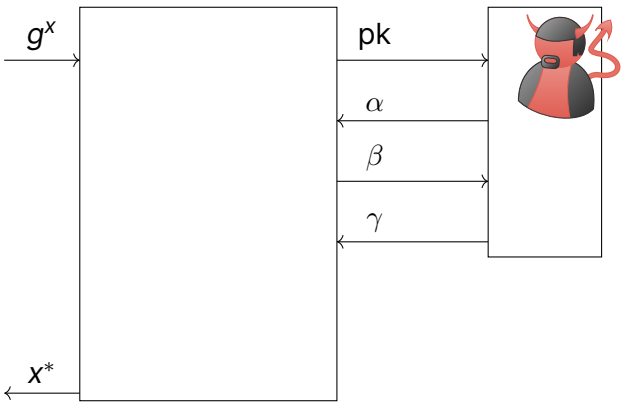
Information flows

- Making queries to  gives information to 
- Can we model that  do not learn anything while  responds to the query?
- Is the ability to update descriptions of algorithms sufficient?

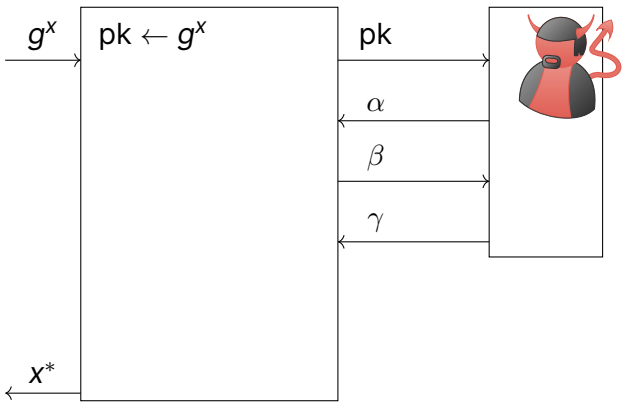
Knowledge extraction



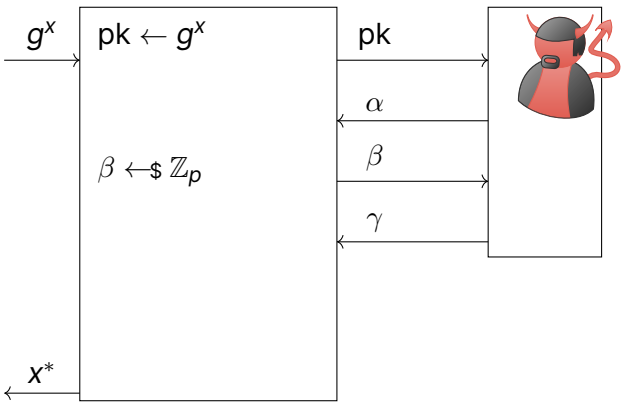
Knowledge extraction



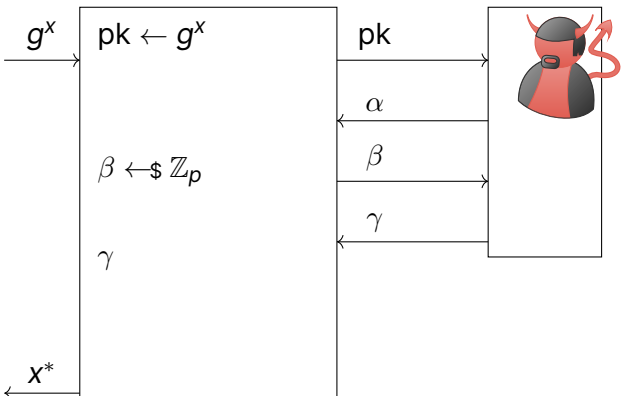
Knowledge extraction



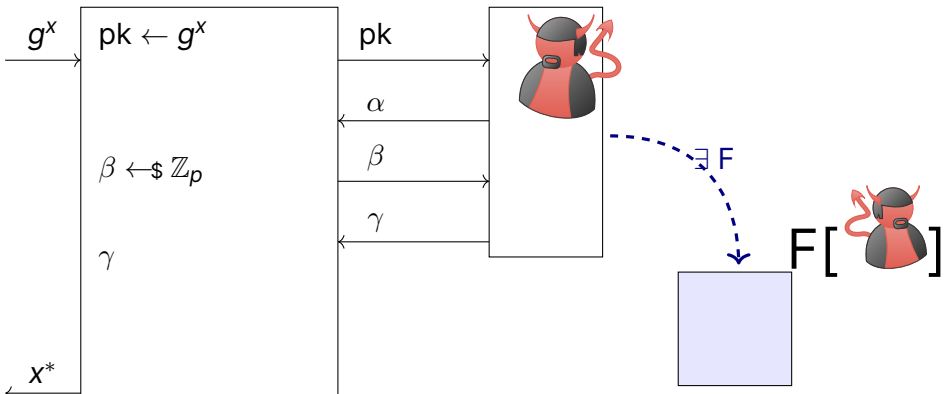
Knowledge extraction



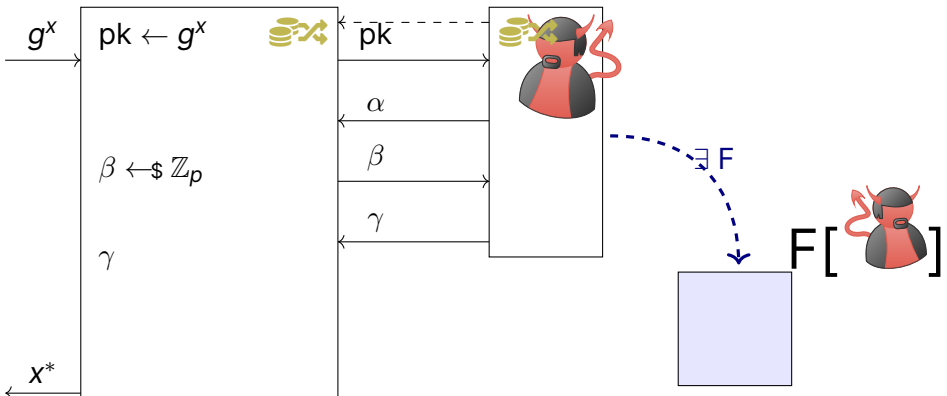
Knowledge extraction



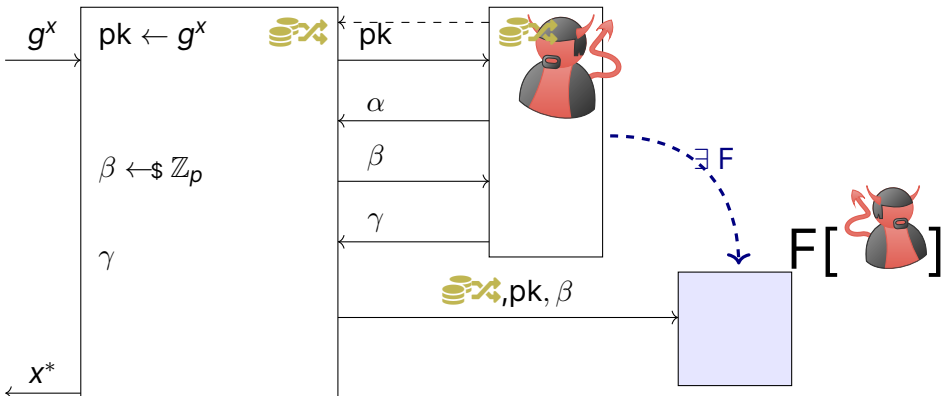
Knowledge extraction



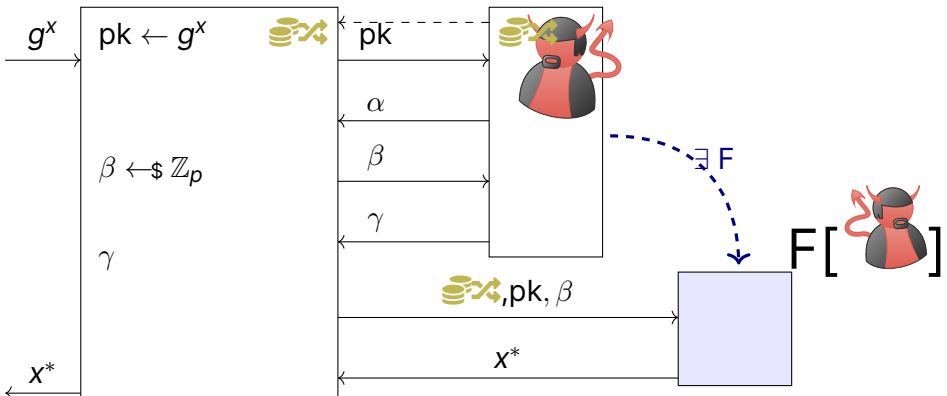
Knowledge extraction



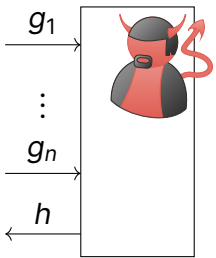
Knowledge extraction



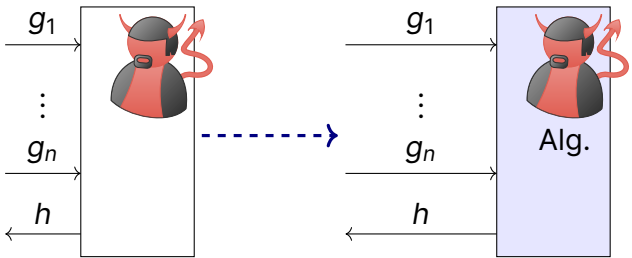
Knowledge extraction



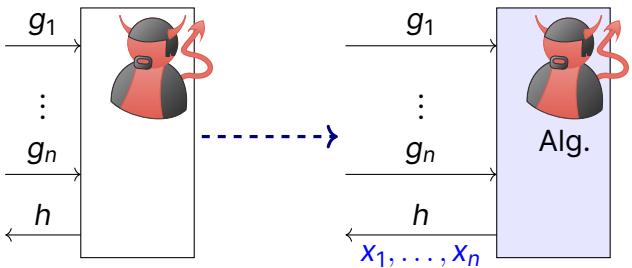
Algebraic Group Model



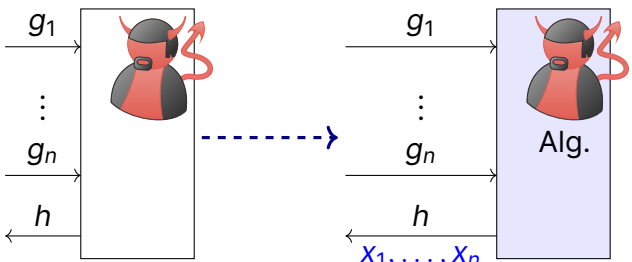
Algebraic Group Model



Algebraic Group Model



Algebraic Group Model



Such that $h = g_1^{x_1} \cdots g_n^{x_n}$