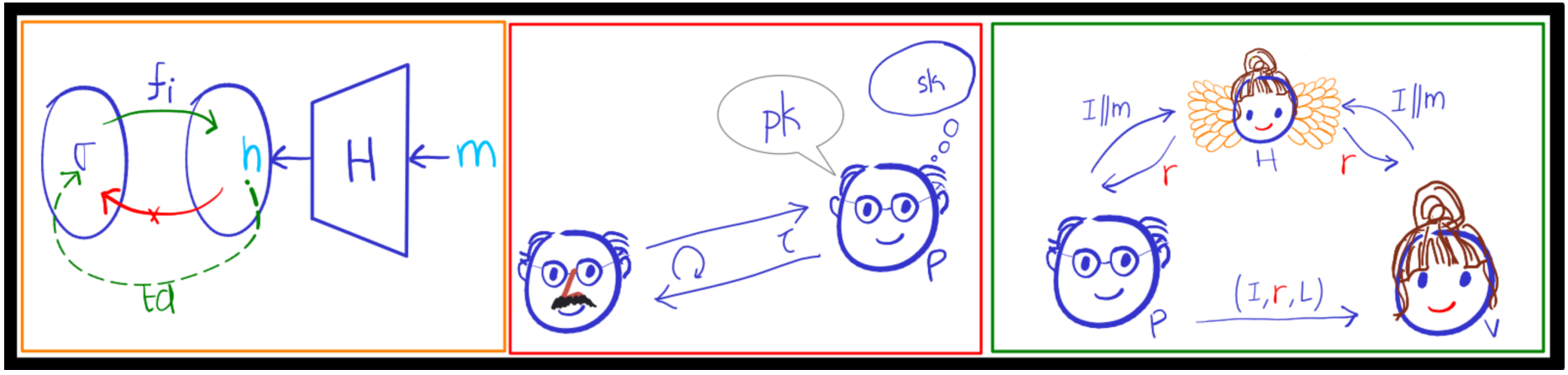



# CS409m: Introduction to Cryptography

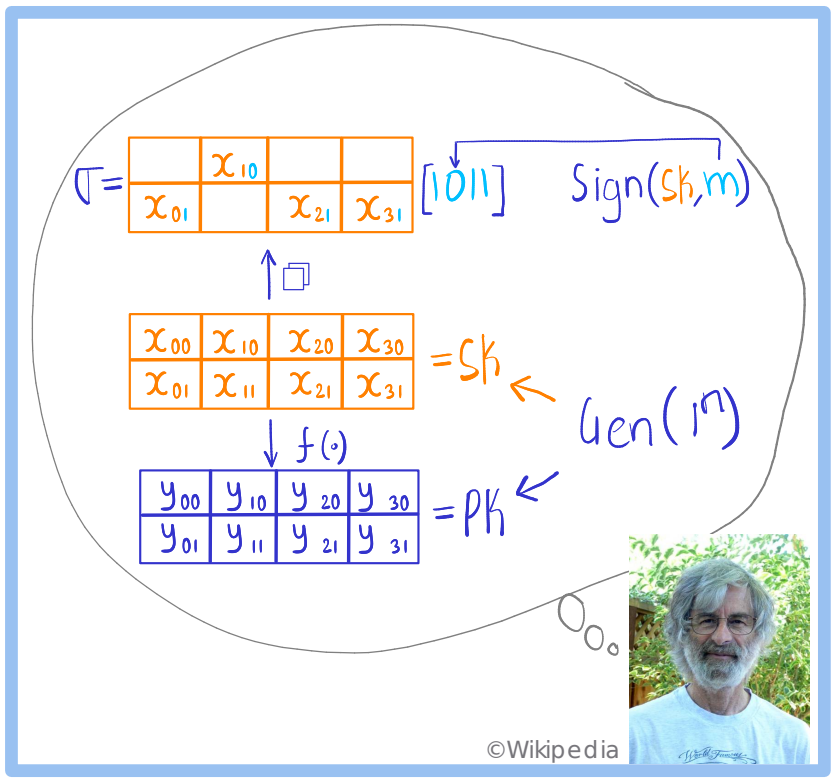
Lecture 17 (15/Oct/25)

Instructor: Chethan Kamath

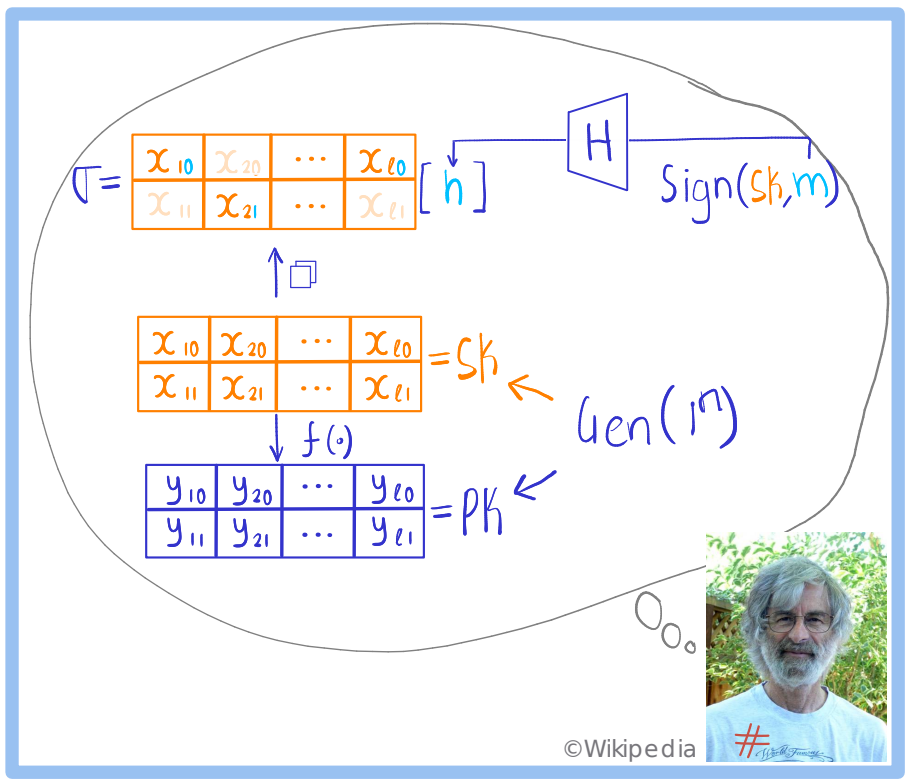
# Recall from Last Two Lectures...

- Primitive: digital signature (DS)
- Threat model: EU-CMA

## Lamport's One-Time DS




## Hash-then-Sign




Tools: OWF, CRHF and CRCF

# How to Construct CRCF in Theory?

- Based on DLP in $\mathbb{Z}_p^\times$: $\{H : (\mathbb{Z}_p^\times)^2 \times \mathbb{Z}_p^2 \to \mathbb{Z}_p^\times\}$, where

$$H_{g,h}(a, b) := g^a h^b \bmod p$$

# How to Construct CRCF in Theory?

- Based on DLP in $\mathbb{Z}_p^\times$: $\{H : (\mathbb{Z}_p^\times)^2 \times \mathbb{Z}_p^2 \to \mathbb{Z}_p^\times\}$, where

$$H_{g,h}(a, b) := g^a h^b \bmod p$$

How to compute discrete-log of $h$ given a collision $((a, b), (a', b'))$?

# How to Construct CRCF in Theory?

- Based on DLP in $\mathbb{Z}_p^\times$: $\{H : (\mathbb{Z}_p^\times)^2 \times \mathbb{Z}_p^2 \to \mathbb{Z}_p^\times\}$, where

$$H_{g,h}(a, b) := g^a h^b \bmod p$$

  - How to compute discrete-log of $h$ given a collision $((a, b), (a', b'))$?
- Based on subset-sum problem?

$$H_{a_1,\ldots,a_n}(x_1 \| \ldots \| x_n) := \sum_{i \in [1,n]} x_i a_i \bmod p$$

  - When is $H$ compressing?

# How to Construct CRCF in Theory?

- Based on DLP in $\mathbb{Z}_p^\times$: $\{H : (\mathbb{Z}_p^\times)^2 \times \mathbb{Z}_p^2 \to \mathbb{Z}_p^\times\}$, where

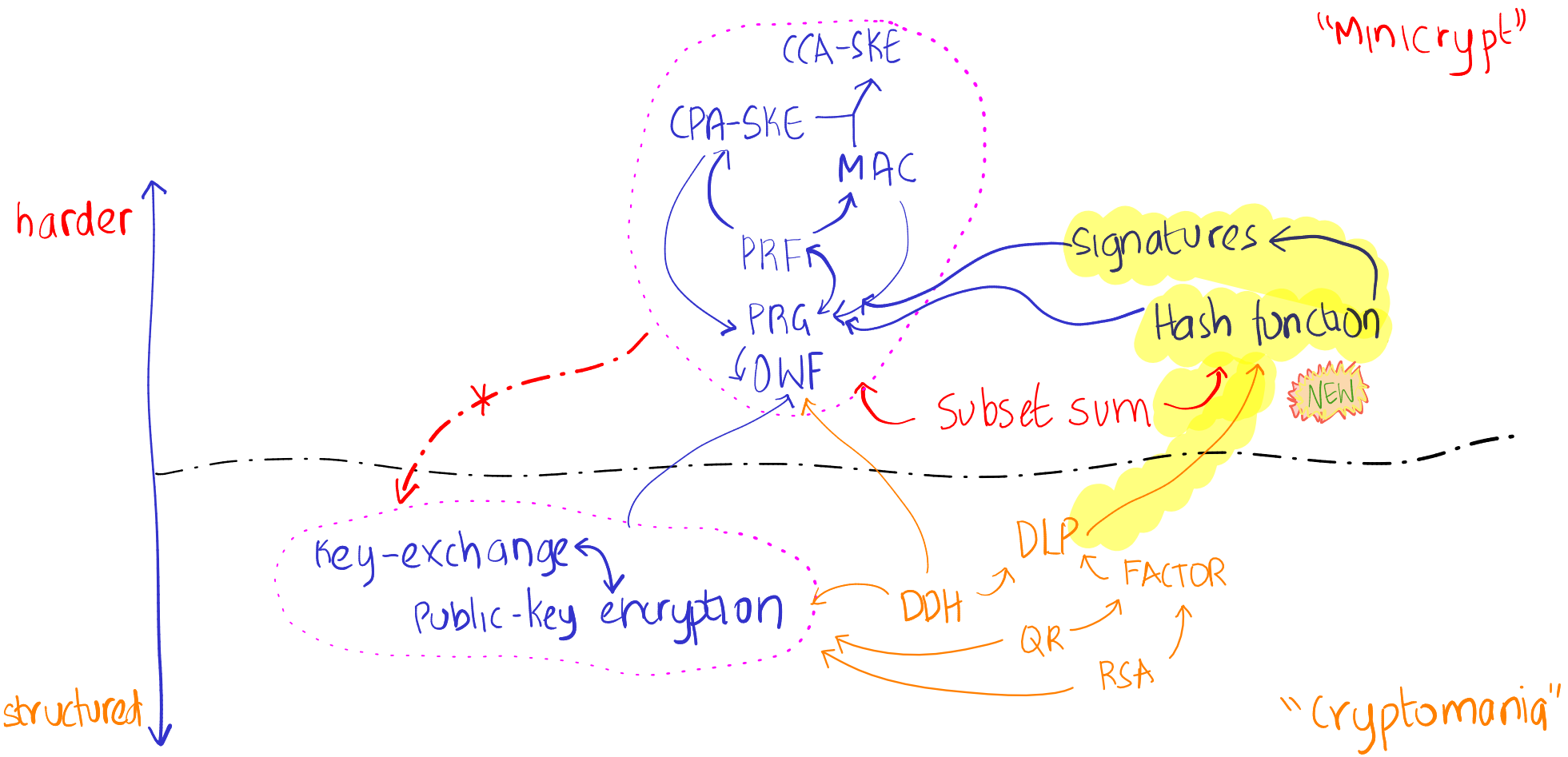
$$H_{g,h}(a, b) := g^a h^b \bmod p$$

  - How to compute discrete-log of $h$ given a collision $((a, b), (a', b'))$?
- Based on subset-sum problem?

$$H_{a_1,\ldots,a_n}(x_1 \| \ldots \| x_n) := \sum_{i \in [1,n]} x_i a_i \bmod p$$

  - When is $H$ compressing? When we set $p < 2^n$
  - How to solve subset-sum given a collision?

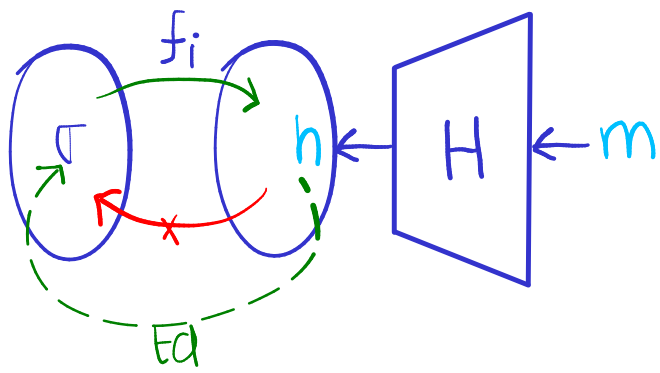
# The Cryptographic Landscape



# Plan for Today's Lecture...

- Task: *efficient* (many-time) digital signatures
- Threat model: EU-CMA

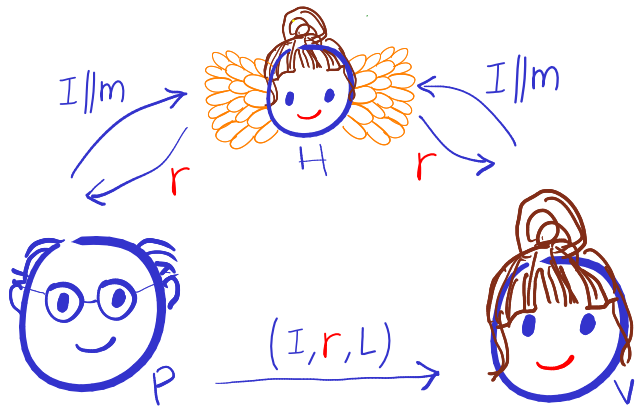
NEW Via Hash-then-Invert

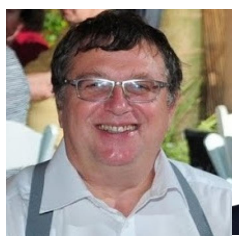




© rwc.iacr.org

Via Identification Protocols NEW



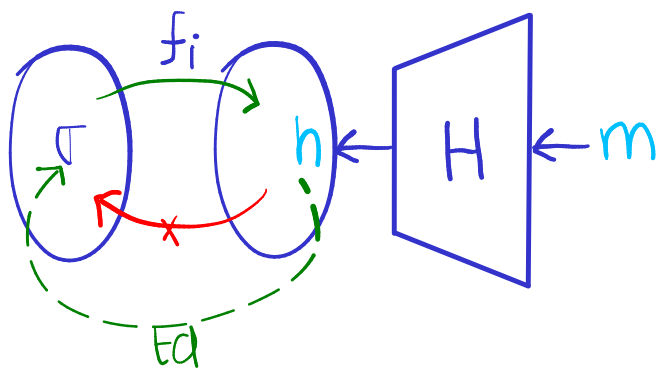



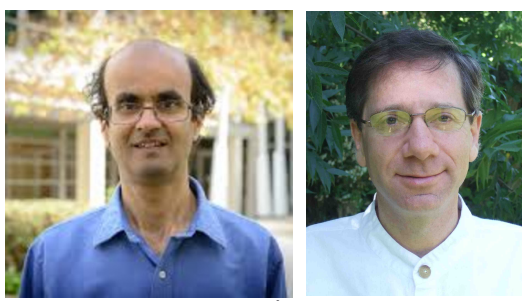

© Simons Institute © rwc.iacr.org

# Plan for Today's Lecture...

- Task: *efficient* (many-time) digital signatures
- Threat model: EU-CMA

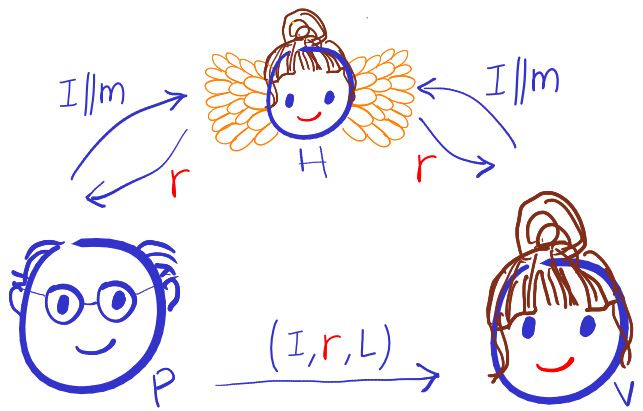

Via Hash-then-Invert

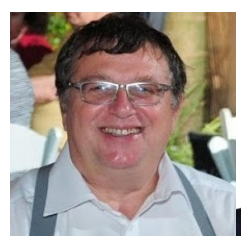


© rwc.iacr.org

Via Identification Protocols

© Simons Institute

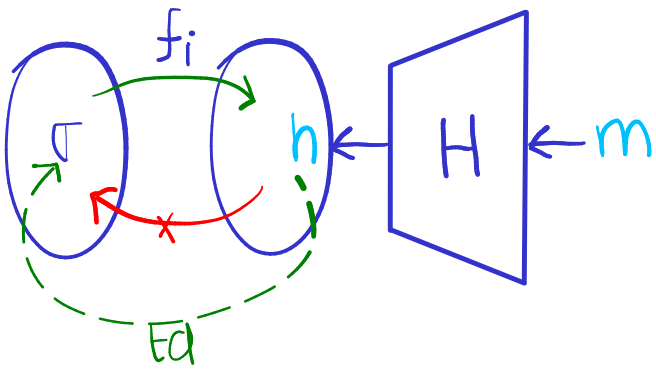
© rwc.iacr.org

Tools: Trapdoor perm., plug-and-pray, Fiat-Shamir Transform

# Plan for Today's Lecture...

- Task: *efficient* (many-time) digital signatures
- Threat model: EU-CMA


Via Hash-then-Invert




© rwc.iacr.org

Via Identification Protocols







© Simons Institute © rwc.iacr.org

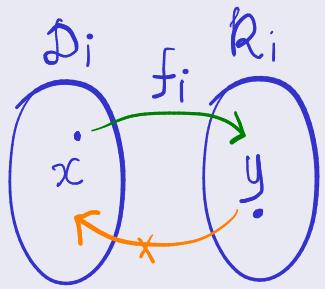
Tools: Trapdoor perm., plug-and-pray, Fiat-Shamir Transform

# *Collection* of OWFs

## Definition 1 (One-way function (OWF) collection)

A collection of functions $f := \{f_i : \mathcal{D}_i \to \mathcal{R}_i\}_{i \in \mathcal{I} \subseteq \{0,1\}^*}$ is one-way if

1. There is an efficient index-sampling algorithm Index
2. Each $f_i$ in collection is efficiently computable
3. For all PPT *inverters* Inv, the following is negligible:



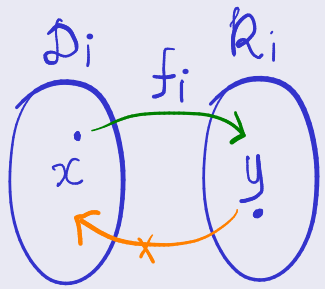

$$p(n) := \Pr_{\substack{i \leftarrow \mathsf{Index}(1^n) \\ x \leftarrow \mathcal{D}_i}} [\mathsf{Inv}(f_i(x)) \in f_i^{-1}(f_i(x))]$$

# *Collection* of OWFs

**Definition 1 (One-way function (OWF) collection)**

A collection of functions $f := \{f_i : \mathcal{D}_i \to \mathcal{R}_i\}_{i \in \mathcal{I} \subseteq \{0,1\}^*}$ is one-way if

1. There is an efficient index-sampling algorithm Index
2. Each $f_i$ in collection is efficiently computable
3. For all PPT *inverters* Inv, the following is negligible:




$$p(n) := \Pr_{\substack{i \leftarrow \mathsf{Index}(1^n) \\ x \leftarrow \mathcal{D}_i}} [\mathsf{Inv}(f_i(x)) \in f_i^{-1}(f_i(x))]$$

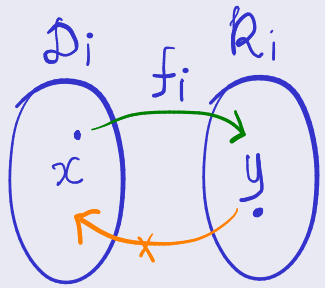
- One-way permutation (OWP): $\mathcal{D}_i = \mathcal{R}_i$ and $f_i$ injective

# *Collection* of OWFs

## Definition 1 (One-way function (OWF) collection)

A collection of functions $f := \{f_i : \mathcal{D}_i \to \mathcal{R}_i\}_{i \in \mathcal{I} \subseteq \{0,1\}^*}$ is one-way if

1. There is an efficient index-sampling algorithm Index
2. Each $f_i$ in collection is efficiently computable
3. For all PPT *inverters* Inv, the following is negligible:




$$p(n) := \Pr_{\substack{i \leftarrow \mathsf{Index}(1^n) \\ x \leftarrow \mathcal{D}_i}} [\mathsf{Inv}(f_i(x)) \in f_i^{-1}(f_i(x))]$$

- One-way permutation (OWP): $\mathcal{D}_i = \mathcal{R}_i$ and $f_i$ injective

Recall examples:

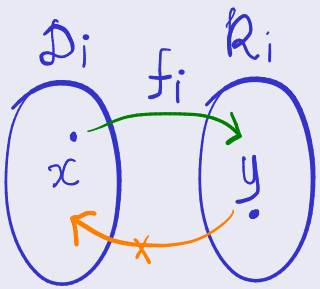
1. RSA function (power mod semiprime $N = pq$): $f_{N,e}(x) := x^e \bmod N$
2. Exponentiation function (modulo prime $p$): $f_{p,g}(x) := g^x \bmod p$

# Collection of *Trapdoor* OWP...

## Definition 2 (Trapdoor (one-way) permutation (TDP) collection)

A collection of permutations $f = \{f_i : \mathcal{D}_i \to \mathcal{D}_i\}_{i \in \mathcal{I} \subseteq \{0,1\}^*}$ is *trapdoor one-way* if

1. There is an efficient index+trapdoor sampling algorithm Index
2. Each $f_i$, $i \in \mathcal{I}$, is efficiently computable
3. For all PPT *inverters* Inv, the following is negligible:



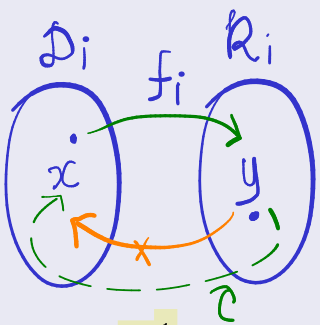

$$p(n) := \Pr_{\substack{(i,\tau) \leftarrow \mathsf{Index}(1^n) \\ x \leftarrow \mathcal{D}_i}} [\mathsf{Inv}(f_i(x)) \in f_i^{-1}(f_i(x))]$$

# Collection of *Trapdoor* OWP...

## Definition 2 (Trapdoor (one-way) permutation (TDP) collection)

A collection of permutations $f = \{f_i : \mathcal{D}_i \to \mathcal{D}_i\}_{i \in \mathcal{I} \subseteq \{0,1\}^*}$ is *trapdoor one-way* if

1. There is an efficient index+trapdoor sampling algorithm Index
2. Each $f_i$, $i \in \mathcal{I}$, is efficiently computable
3. For all PPT *inverters* Inv, the following is negligible:




$$p(n) := \Pr_{\substack{(i,\tau) \leftarrow \mathsf{Index}(1^n) \\ x \leftarrow \mathcal{D}_i}} [\mathsf{Inv}(f_i(x)) \in f_i^{-1}(f_i(x))]$$

4. $f_i^{-1}$ can be efficiently computed given trapdoor $\tau$ for $i$

# Collection of *Trapdoor* OWP...

- RSA function $\left\{f_{N,e} : \mathbb{Z}_N^\times \to \mathbb{Z}_N^\times\right\}_{N,e}$, defined as

$$f_{N,e}(x) := x^e \bmod N$$

  - $f_{N,e}$ is permutation when $GCD(e, (p-1)(q-1)) = 1$
  - One-way by RSA assumption

# Collection of *Trapdoor* OWP...

- RSA function $\{f_{N,e} : \mathbb{Z}_N^\times \to \mathbb{Z}_N^\times\}_{N,e}$, defined as

$$f_{N,e}(x) := x^e \bmod N$$

  - $f_{N,e}$ is permutation when $GCD(e, (p-1)(q-1)) = 1$
  - One-way by RSA assumption
  - ☆ The trapdoor is $d := e^{-1} \bmod (p-1)(q-1)$

# Collection of *Trapdoor* OWP...

- RSA function $\{f_{N,e} : \mathbb{Z}_N^\times \to \mathbb{Z}_N^\times\}_{N,e}$, defined as

$$f_{N,e}(x) := x^e \bmod N$$

  - $f_{N,e}$ is permutation when $GCD(e, (p-1)(q-1)) = 1$
  - One-way by RSA assumption
  - ☆ The trapdoor is $d := e^{-1} \bmod (p-1)(q-1)$

- Exponentiation function $\{f_{p,g}(x) : \mathbb{Z}_p^\times \to \mathbb{Z}_p^\times\}_{p,g}$, defined as

$$f_{p,g}(x) := g^x \bmod p$$

  - $f_{p,g}$ is permutation
  - One-way by discrete-log assumption

# Collection of *Trapdoor* OWP...

- RSA function $\{f_{N,e} : \mathbb{Z}_N^\times \to \mathbb{Z}_N^\times\}_{N,e}$, defined as

$$f_{N,e}(x) := x^e \bmod N$$

  - $f_{N,e}$ is permutation when $GCD(e, (p-1)(q-1)) = 1$
  - One-way by RSA assumption
  - ☆ The trapdoor is $d := e^{-1} \bmod (p-1)(q-1)$

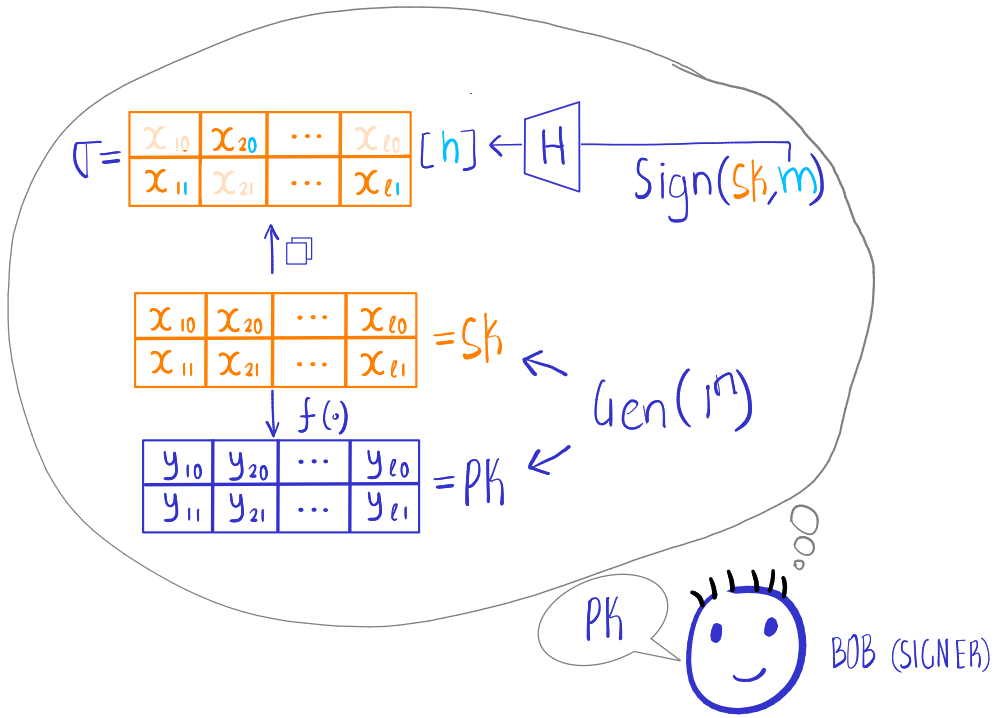
- Exponentiation function $\{f_{p,g}(x) : \mathbb{Z}_p^\times \to \mathbb{Z}_p^\times\}_{p,g}$, defined as

$$f_{p,g}(x) := g^x \bmod p$$

  - $f_{p,g}$ is permutation
  - One-way by discrete-log assumption
  - ⚠ We don't know a trapdoor!

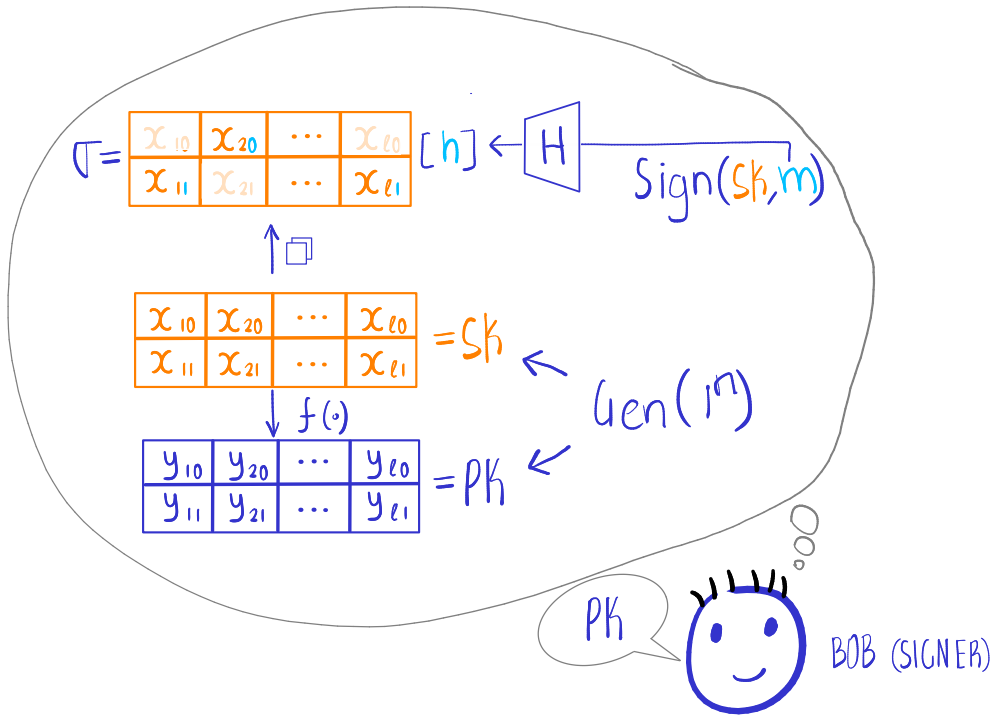
# Recall: "Hash-Then-Sign" Paradigm

- 1) Compute "hash" $h = H(k, m)$ 2) sign $h$ using Lamport's OTDS



# Recall: "Hash-Then-Sign" Paradigm

- 1) Compute "hash" $h = H(k, m)$ 2) sign $h$ using Lamport's OTDS

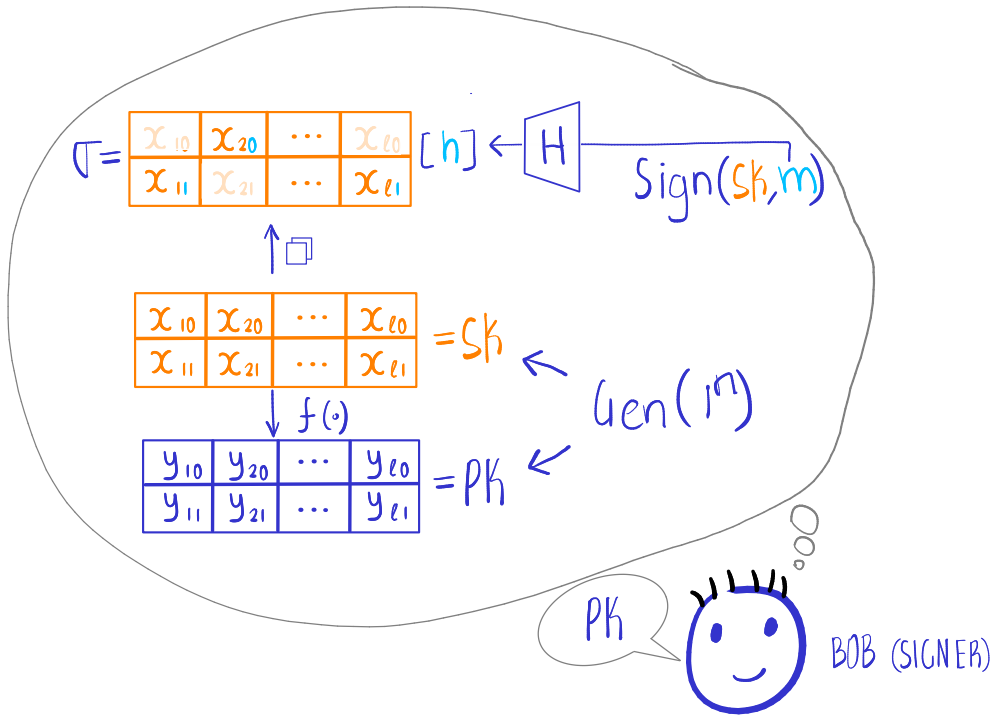



**Theorem 4 (from Lecture 16, rephrased)**

*If Lamport's scheme is OTDS and $H$ is CRHF then "hash-then-sign" scheme is a one-time EU-CMA for arbitrarily-long messages.*

# Recall: "Hash-Then-Sign" Paradigm

- 1) Compute "hash" $h = H(k, m)$ 2) sign $h$ using Lamport's OTDS



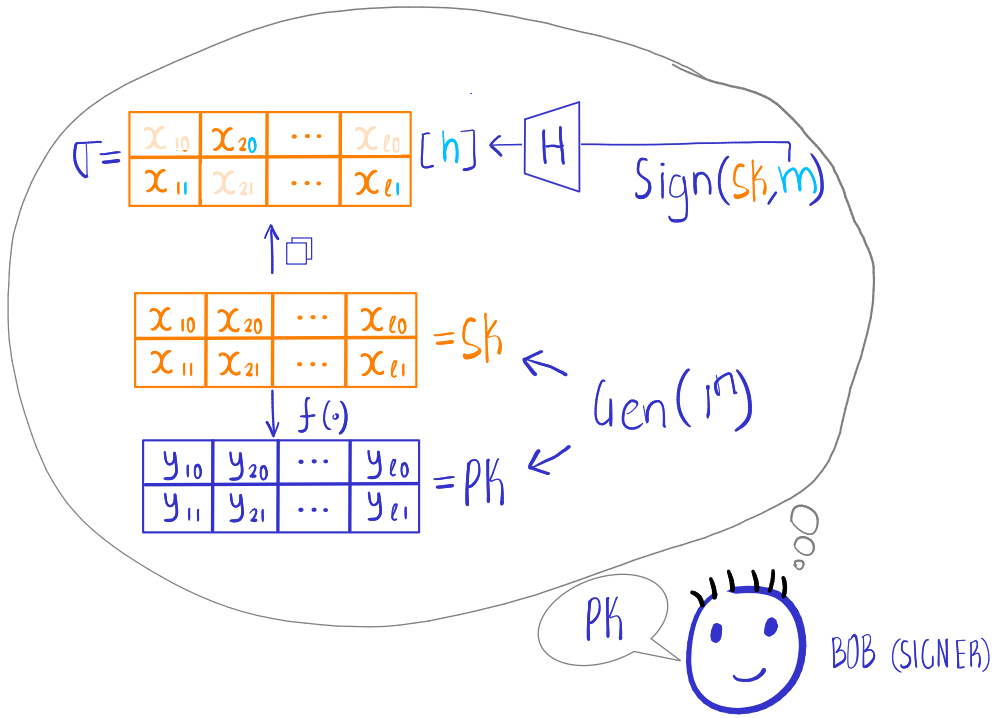

**Theorem 4 (from Lecture 16, rephrased)**

*If Lamport's scheme is OTDS and $H$ is CRHF then "hash-then-sign" scheme is a one-time EU-CMA for arbitrarily-long messages.*

How can a TDP be useful here?

# Recall: "Hash-Then-Sign" Paradigm

- 1) Compute "hash" $h = H(k, m)$ 2) sign $h$ using Lamport's OTDS




**Theorem 4 (from Lecture 16, rephrased)**

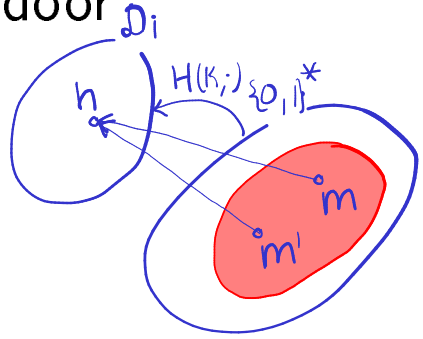
*If Lamport's scheme is OTDS and H is CRHF then "hash-then-sign" scheme is a one-time EU-CMA for arbitrarily-long messages.*

How can a TDP be useful here? To *replace* Lamport's OTS!

# TDP-based Signature: "Hash-then-Invert"

1) Compute "hash" $h = H(k, m)$ 2) *invert* $h$ using trapdoor

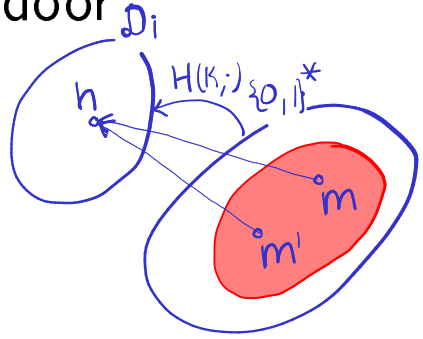
- "Full domain" hash function $H : \mathcal{K} \times \{0,1\}^* \to \mathcal{D}_i$



# TDP-based Signature: "Hash-then-Invert"

1) Compute "hash" $h = H(k, m)$ 2) *invert* $h$ using trapdoor

- "Full domain" hash function $H : \mathcal{K} \times \{0,1\}^* \to \mathcal{D}_i$

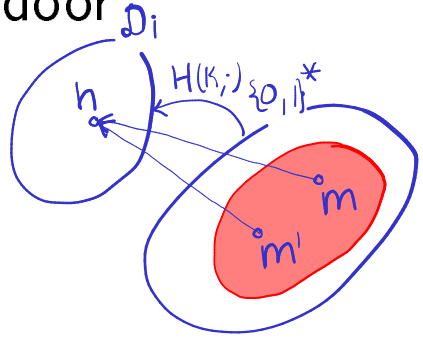


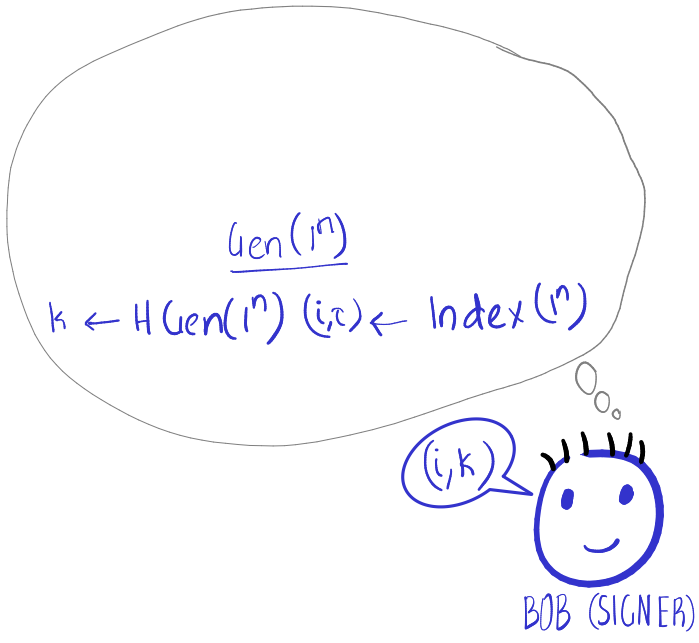


# TDP-based Signature: "Hash-then-Invert"

1) Compute "hash" $h = H(k, m)$ 2) *invert* $h$ using trapdoor

- "Full domain" hash function $H : \mathcal{K} \times \{0,1\}^* \to \mathcal{D}_i$

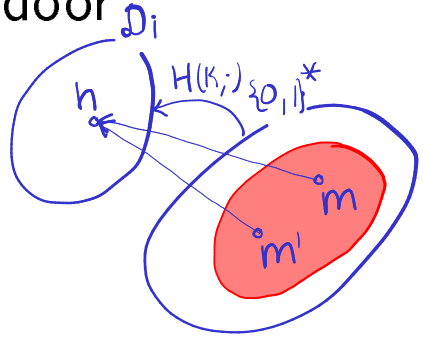


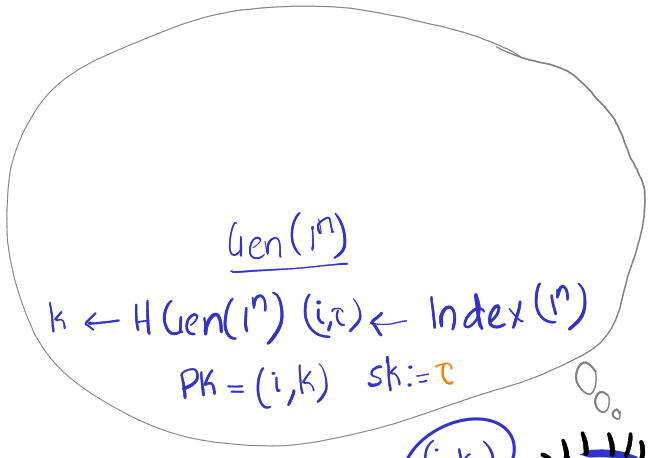



# TDP-based Signature: "Hash-then-Invert"

1) Compute "hash" $h = H(k, m)$ 2) *invert* $h$ using trapdoor

- "Full domain" hash function $H : \mathcal{K} \times \{0,1\}^* \to \mathcal{D}_i$

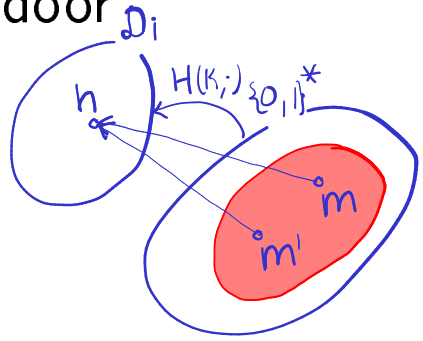


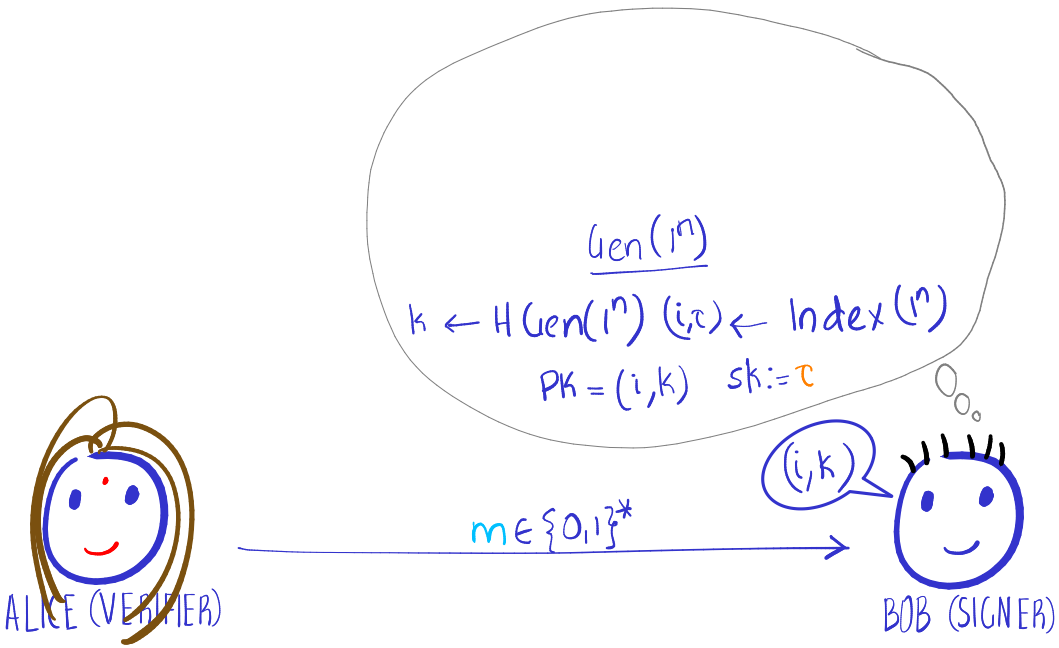



# TDP-based Signature: "Hash-then-Invert"

1) Compute "hash" $h = H(k, m)$ 2) *invert* $h$ using trapdoor

- "Full domain" hash function $H : \mathcal{K} \times \{0,1\}^* \to \mathcal{D}_i$

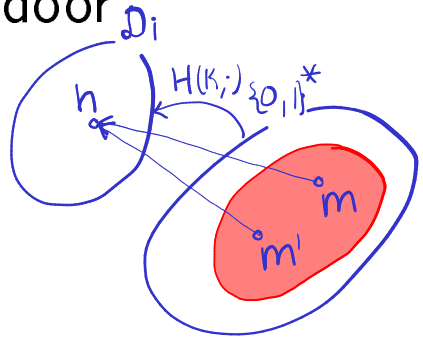


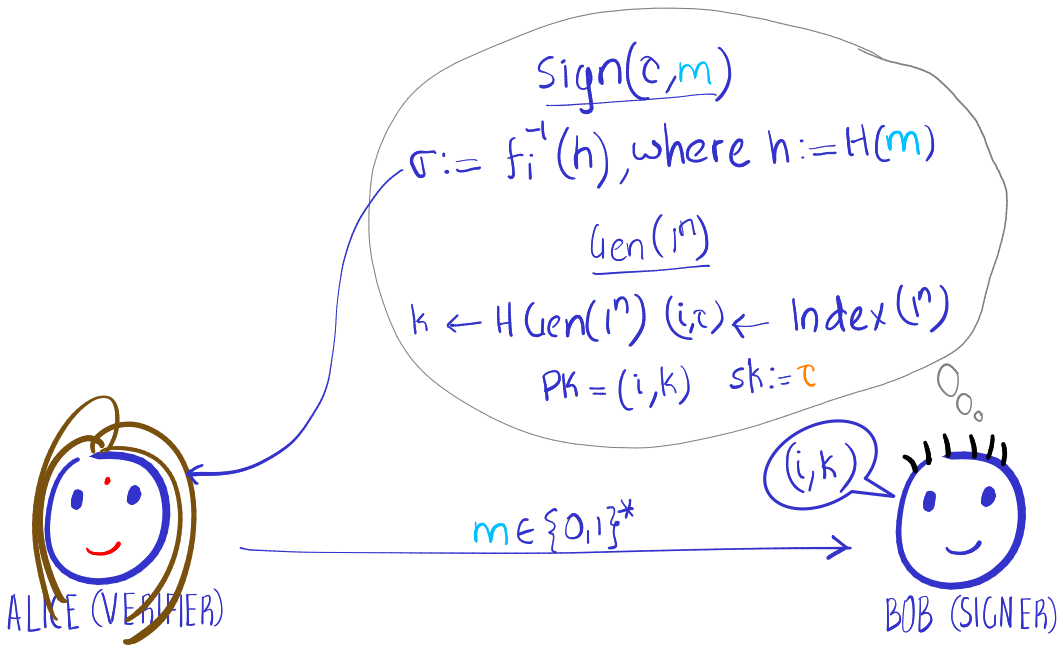



# TDP-based Signature: "Hash-then-Invert"

1) Compute "hash" $h = H(k, m)$ 2) *invert* $h$ using trapdoor

- "Full domain" hash function $H : \mathcal{K} \times \{0,1\}^* \to \mathcal{D}_i$

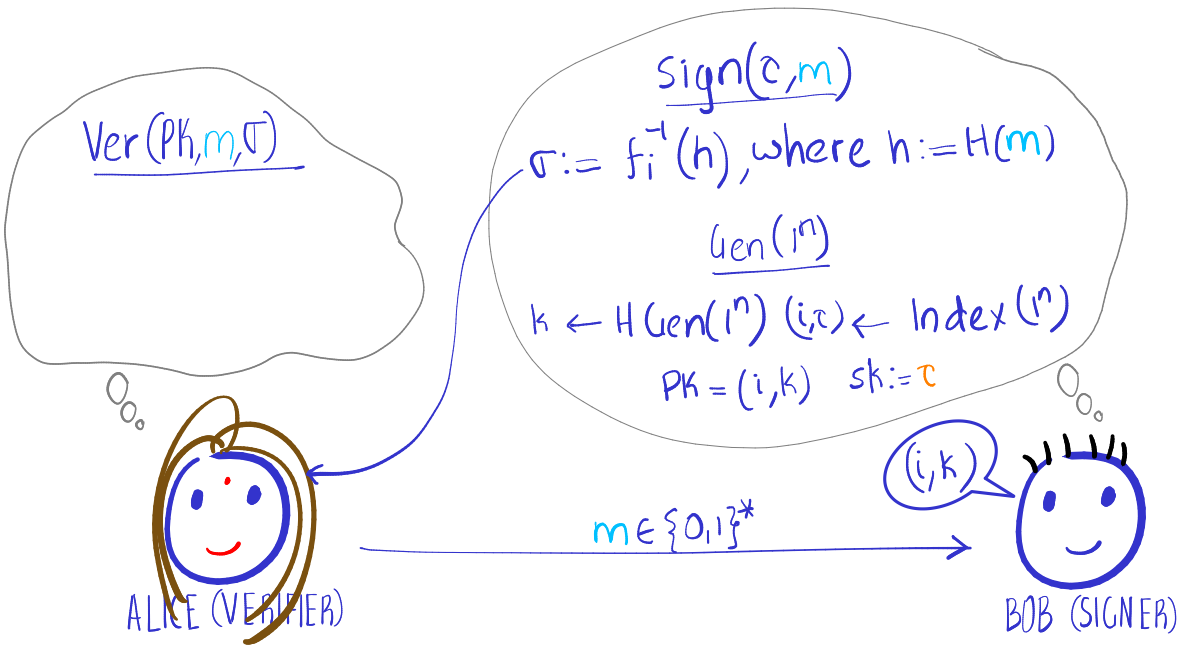


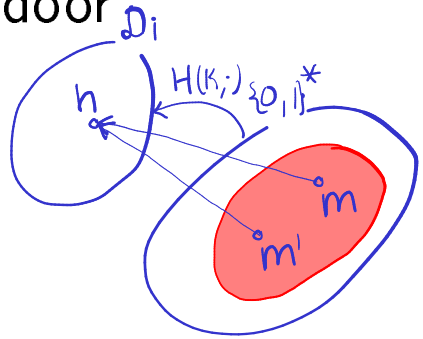



# TDP-based Signature: "Hash-then-Invert"

1) Compute "hash" $h = H(k, m)$ 2) *invert* $h$ using trapdoor

- "Full domain" hash function $H : \mathcal{K} \times \{0,1\}^* \to \mathcal{D}_i$

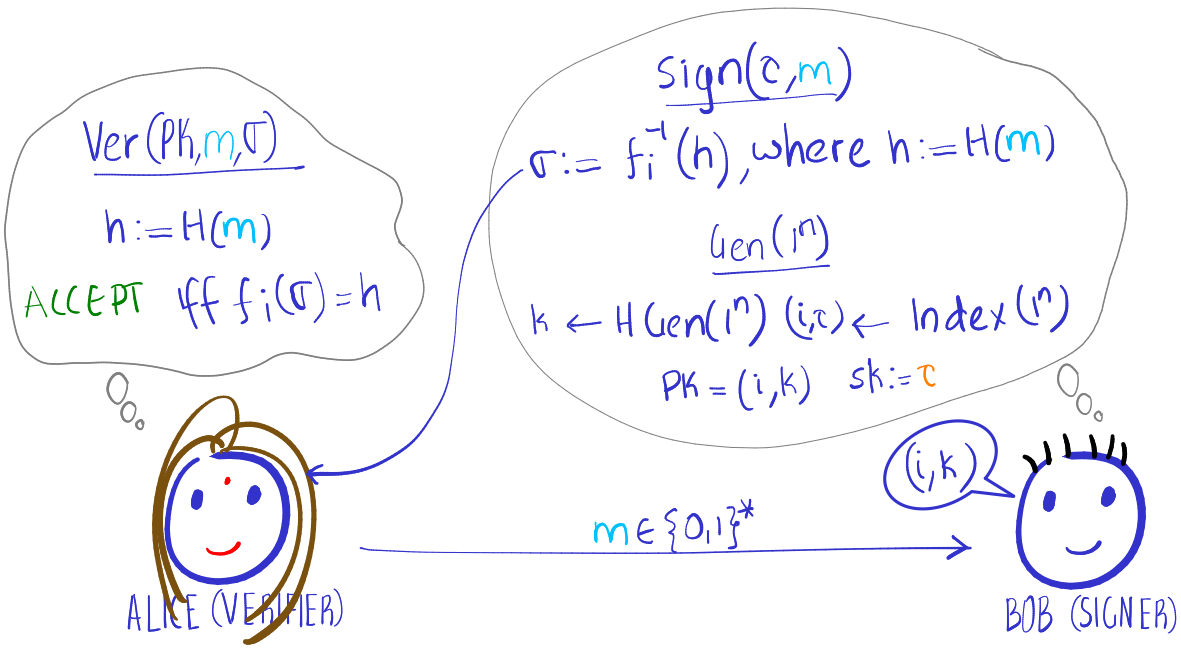


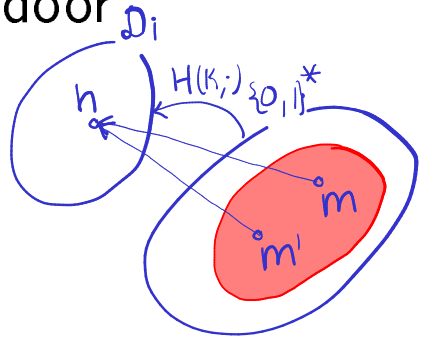



# TDP-based Signature: "Hash-then-Invert"

1) Compute "hash" $h = H(k, m)$ 2) *invert* $h$ using trapdoor

- "Full domain" hash function $H : \mathcal{K} \times \{0,1\}^* \to \mathcal{D}_i$

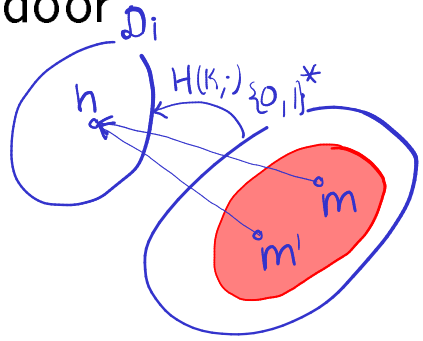


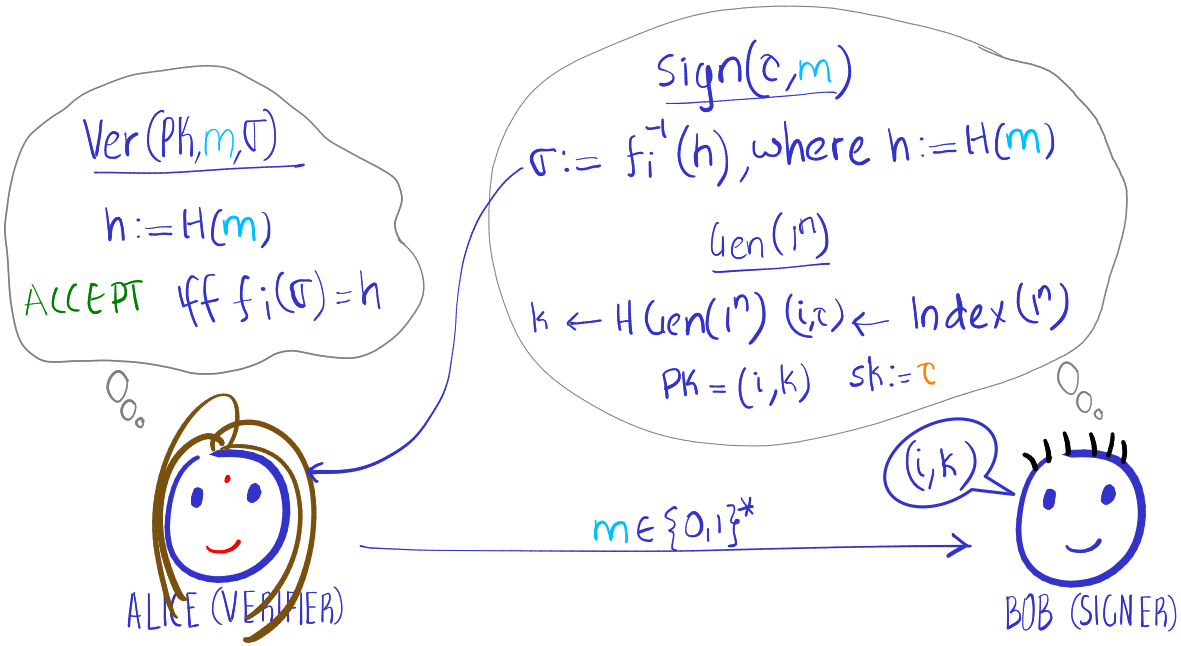



# TDP-based Signature: "Hash-then-Invert"

1) Compute "hash" $h = H(k, m)$ 2) *invert* $h$ using trapdoor

- "Full domain" hash function $H : \mathcal{K} \times \{0,1\}^* \to \mathcal{D}_i$




Efficiency, when using RSA function (i.e., RSA-FDH)

$$f_{N,e}(x) := x^e \bmod N$$

- Public key: $(N, e)$ and description of $H$
- Signature: one element of $\mathbb{Z}_N^\times$
- Signing/verification: one exponentiation + hash evaluation

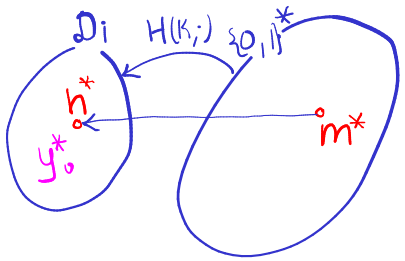
# Let's Prove Security of "Hash-then-Invert" ...

- Is $H$ being CRHF sufficient to prove security?

# Let's Prove Security of "Hash-then-Invert" ...

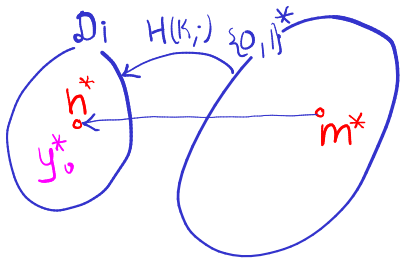
Is $H$ being CRHF sufficient to prove security? Seems not, problem:

1. Need to invert a *particular challenge* $y^*$
2. Forger could forger *any* message $m^*$



# Let's Prove Security of "Hash-then-Invert" ...

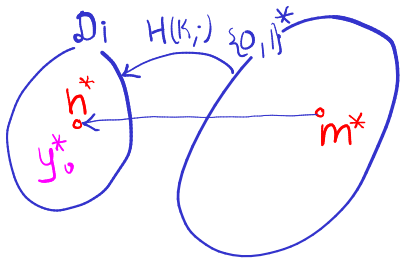
- Is $H$ being CRHF sufficient to prove security? Seems not, problem:
  1. Need to invert a *particular challenge* $y^*$
  2. Forger could forger *any* message $m^*$
- Exploit $H$ to "link" forgery $(\sigma^*, m^*)$ and challenge $y^*$



# Let's Prove Security of "Hash-then-Invert" ...

Is $H$ being CRHF sufficient to prove security? Seems not, problem:

1. Need to invert a *particular challenge* $y^*$
2. Forger could forger *any* message $m^*$



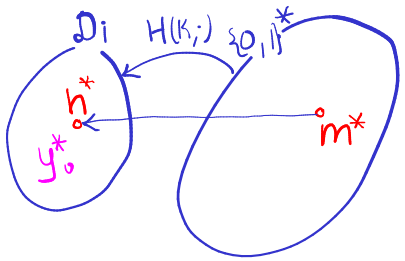

Exploit $H$ to "link" forgery $(\sigma^*, m^*)$ and challenge $y^*$

Solution: model $H$ as a *random-oracle*

# Let's Prove Security of "Hash-then-Invert" ...

Is $H$ being CRHF sufficient to prove security? Seems not, problem:

1. Need to invert a *particular challenge* $y^*$
2. Forger could forger *any* message $m^*$



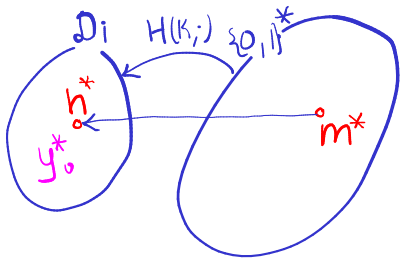

Exploit $H$ to "link" forgery $(\sigma^*, m^*)$ and challenge $y^*$

Solution: model $H$ as a *random-oracle*

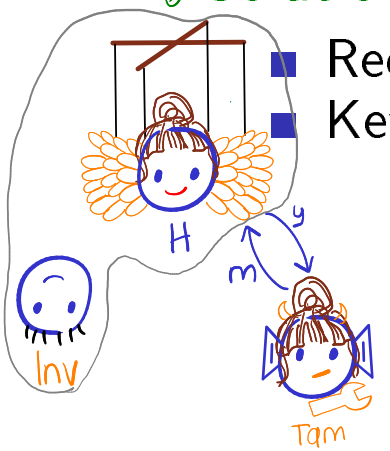
- Recall: $H$ is a *random function* that all parties have *oracle* access to

# Let's Prove Security of "Hash-then-Invert" ...

- Is $H$ being CRHF sufficient to prove security? Seems not, problem:
  1. Need to invert a *particular challenge* $y^*$
  2. Forger could forger *any* message $m^*$



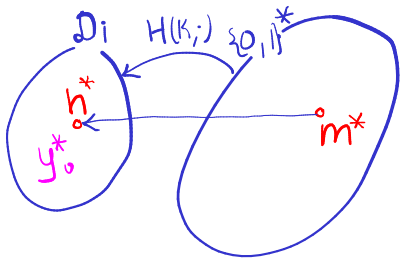

- Exploit $H$ to "link" forgery $(\sigma^*, m^*)$ and challenge $y^*$
- Solution: model $H$ as a *random-oracle*



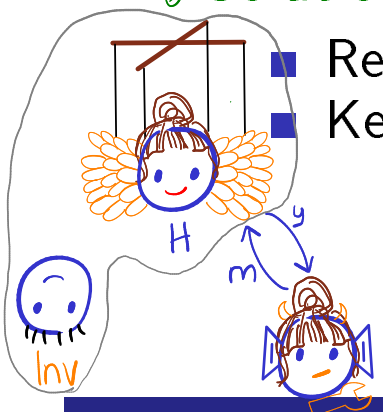

- Recall: $H$ is a *random function* that all parties have *oracle* access to
- Key idea: reduction "controls" $H$ (programming):
  - Constructs $H$ by on-the-fly/lazy sampling
  - A "fresh" query $m \in \{0,1\}^*$ replied with $y \leftarrow \mathcal{D}_i$
  - A "repeat" query $m$ responded *consistently* with $y$ (in table)

# Let's Prove Security of "Hash-then-Invert" ...

- Is $H$ being CRHF sufficient to prove security? Seems not, problem:
  1. Need to invert a *particular challenge* $y^*$
  2. Forger could forger *any* message $m^*$




- Exploit $H$ to "link" forgery $(\sigma^*, m^*)$ and challenge $y^*$
- Solution: model $H$ as a *random-oracle*




- Recall: $H$ is a *random function* that all parties have *oracle* access to
- Key idea: reduction "controls" $H$ (programming):
  - Constructs $H$ by on-the-fly/lazy sampling
  - A "fresh" query $m \in \{0,1\}^*$ replied with $y \leftarrow \mathcal{D}_i$
  - A "repeat" query $m$ responded *consistently* with $y$ (in table)


**Theorem 5**

*If $f$ is a TDP and $H$ is a random oracle then "hash-then-invert" is EU-CMA for arbitrarily-long messages.*

- Proof uses "plug and pray" (on the whiteboard)

# Plan for Today's Lecture...

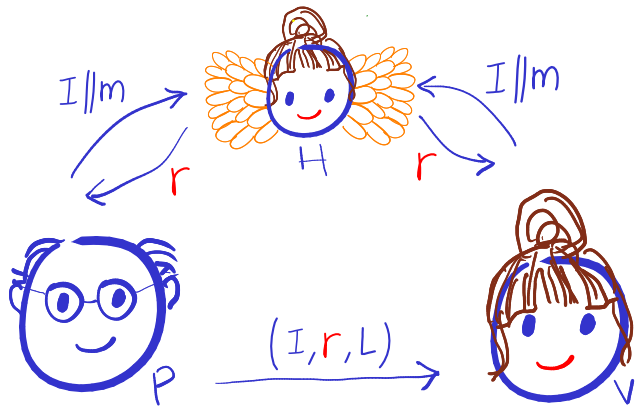
- Task: *efficient* (many-time) digital signatures
- Threat model: EU-CMA

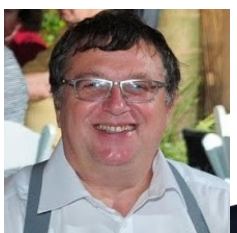
NEW Via Hash-then-Invert




© rwc.iacr.org

Via Identification Protocols NEW



© Simons Institute

© rwc.iacr.org

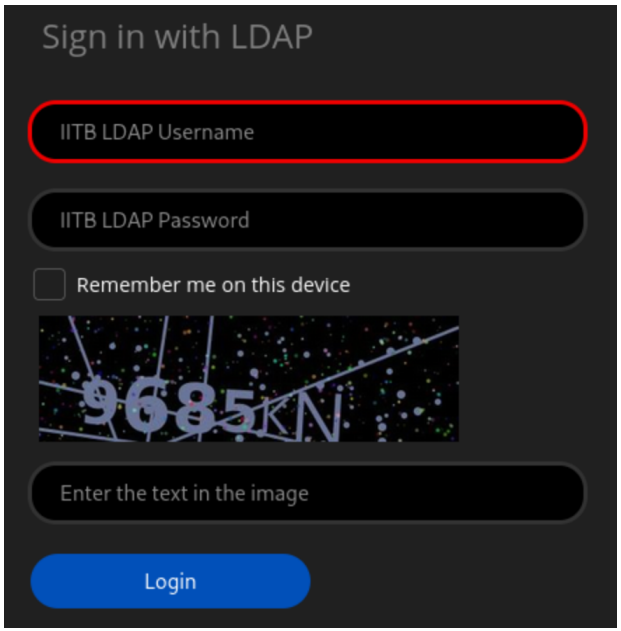
Tools: Trapdoor perm., plug-and-pray, Fiat-Shamir Transform

# Identification in Public-Key Setting

How do you (the prover) *identify* yourself to IITB webmail server (the verifier)?

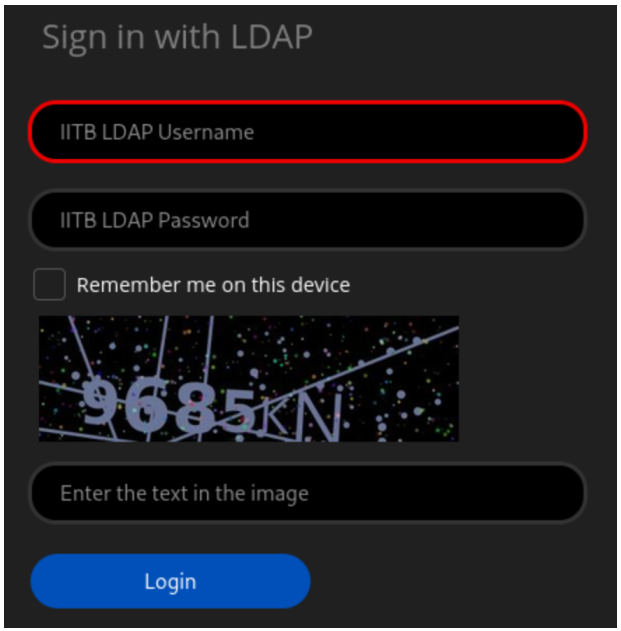
# Identification in Public-Key Setting

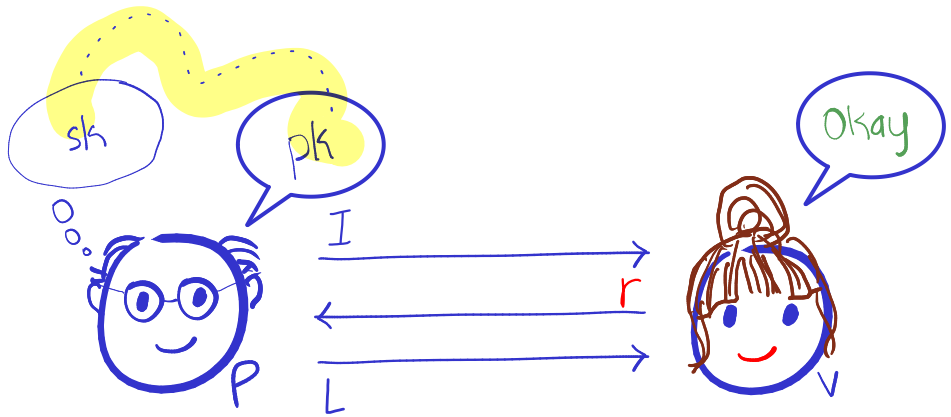
How do you (the prover) *identify* yourself to IITB webmail server (the verifier)? LDAP identity+password (via SSO)



# Identification in Public-Key Setting

How do you (the prover) *identify* yourself to IITB webmail server (the verifier)? LDAP identity+password (via SSO)

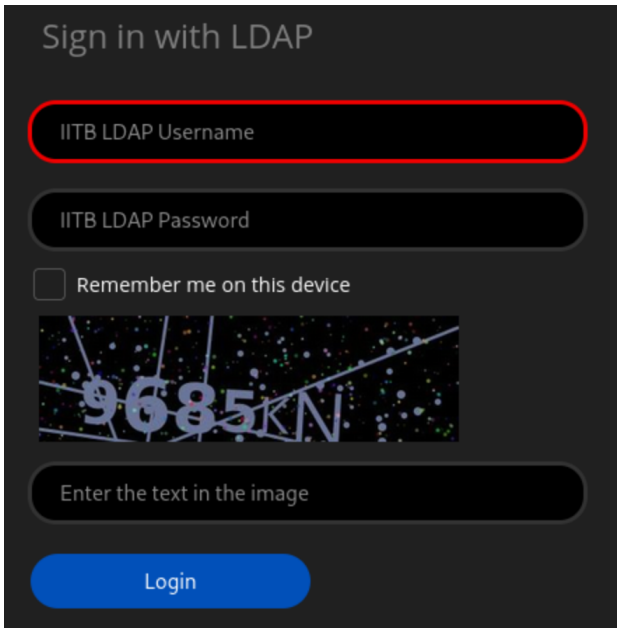


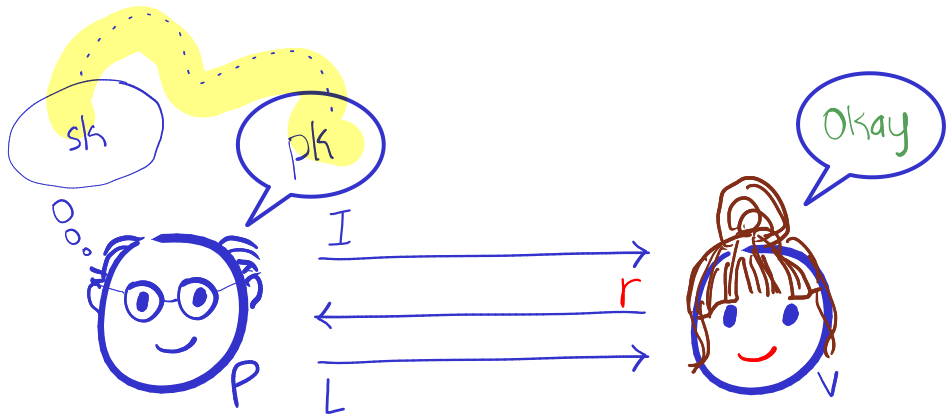




- Identification *in the public-key setting*:
  - Verifier (V) knows only the public key of the prover (P)
  - Verifier must be convinced it is communicating with P *rather than an imposter*

# Identification in Public-Key Setting

How do you (the prover) *identify* yourself to IITB webmail server (the verifier)? LDAP identity+password (via SSO)



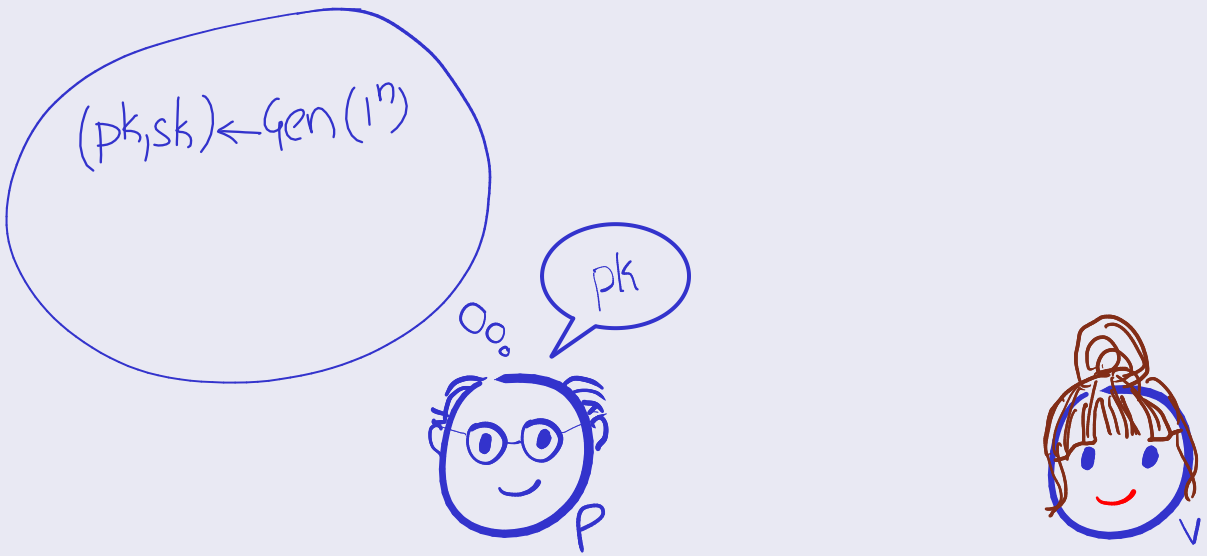




- Identification *in the public-key setting*:
  - Verifier (V) knows only the public key of the prover (P)
  - Verifier must be convinced it is communicating with P *rather than an imposter*
- E.g.: SSHing into a remote server

# Let's Look at the Syntax

## Definition 3 ((Three-Round) Identification (ID) Protocol...)

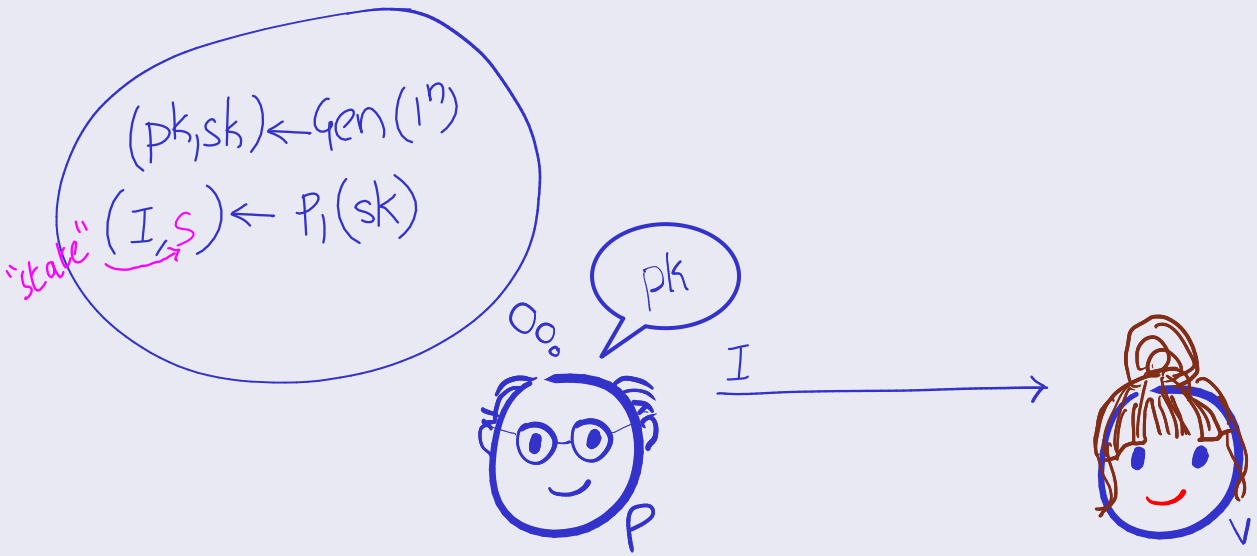
... $\Pi$ is a triple of PPT algorithms $(\mathsf{Gen}, \mathsf{P} = (\mathsf{P}_1, \mathsf{P}_2), \mathsf{V})$ with the following syntax:



# Let's Look at the Syntax

## Definition 3 ((Three-Round) Identification (ID) Protocol...)

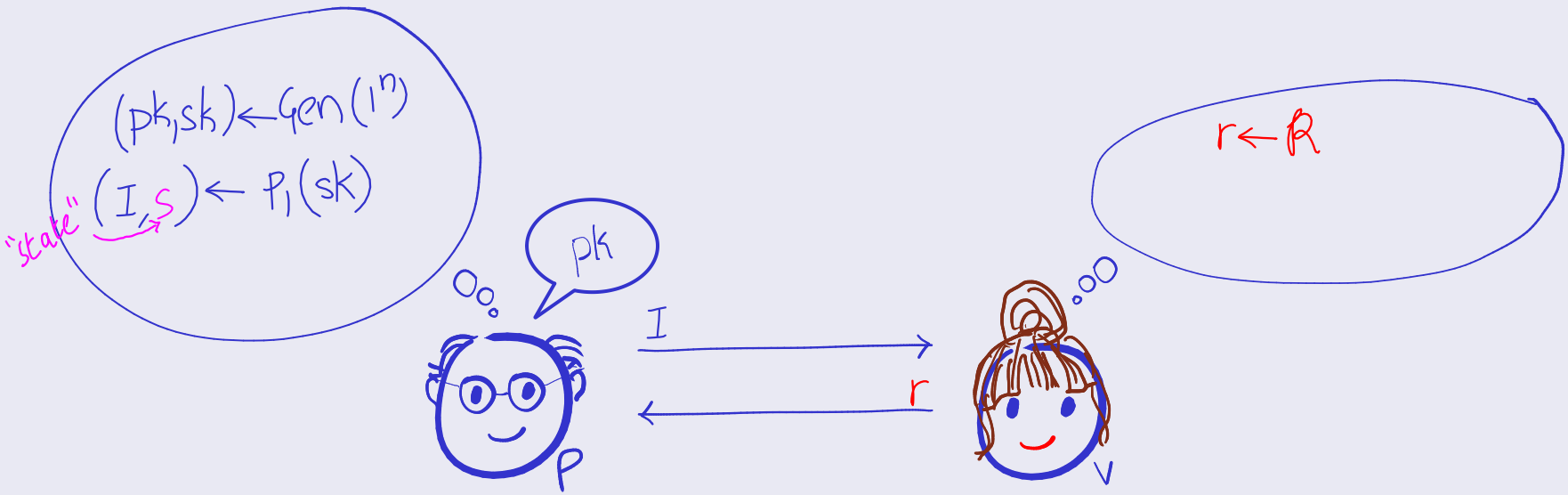
... $\Pi$ is a triple of PPT algorithms $(\mathsf{Gen}, \mathsf{P} = (\mathsf{P}_1, \mathsf{P}_2), \mathsf{V})$ with the following syntax:



# Let's Look at the Syntax

## Definition 3 ((Three-Round) Identification (ID) Protocol...)

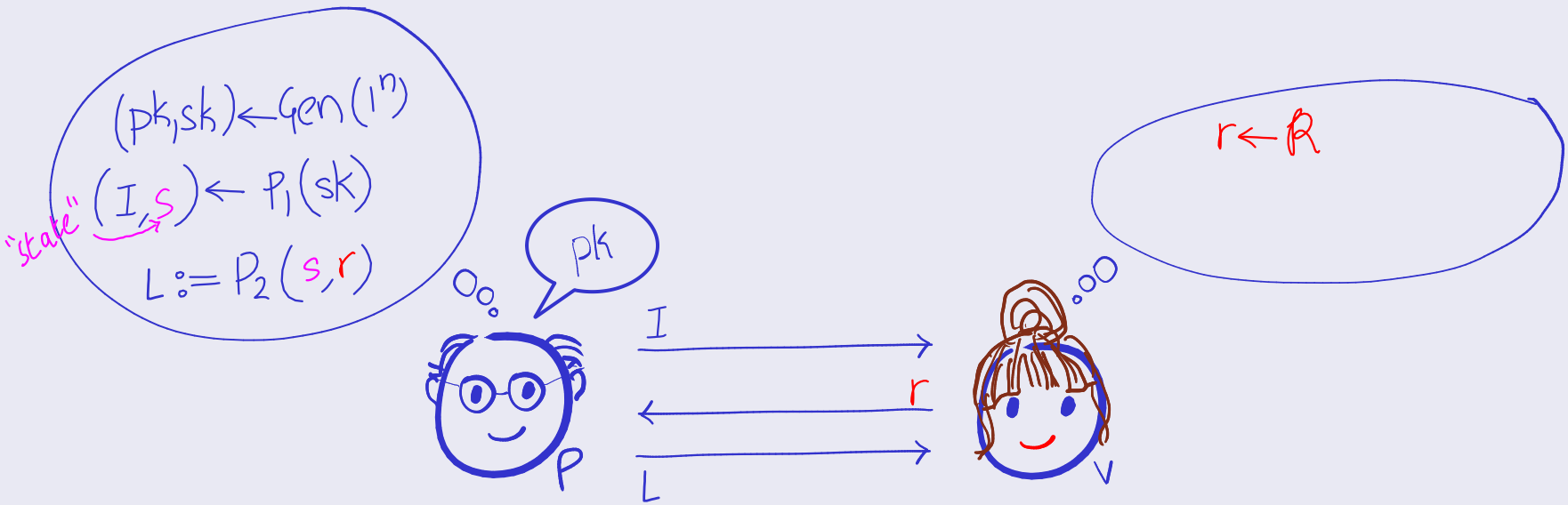
... $\Pi$ is a triple of PPT algorithms $(\mathsf{Gen}, \mathsf{P} = (\mathsf{P}_1, \mathsf{P}_2), \mathsf{V})$ with the following syntax:



# Let's Look at the Syntax

## Definition 3 ((Three-Round) Identification (ID) Protocol...)

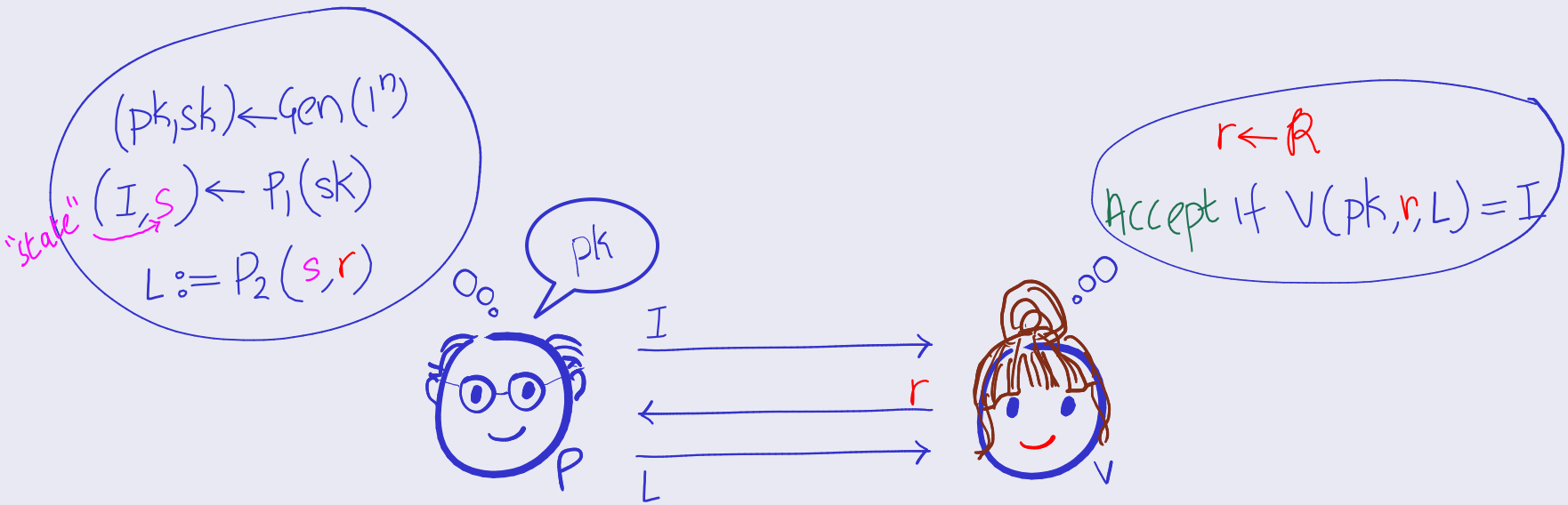
... $\Pi$ is a triple of PPT algorithms $(\mathsf{Gen}, \mathsf{P} = (\mathsf{P}_1, \mathsf{P}_2), \mathsf{V})$ with the following syntax:



# Let's Look at the Syntax

## Definition 3 ((Three-Round) Identification (ID) Protocol...)

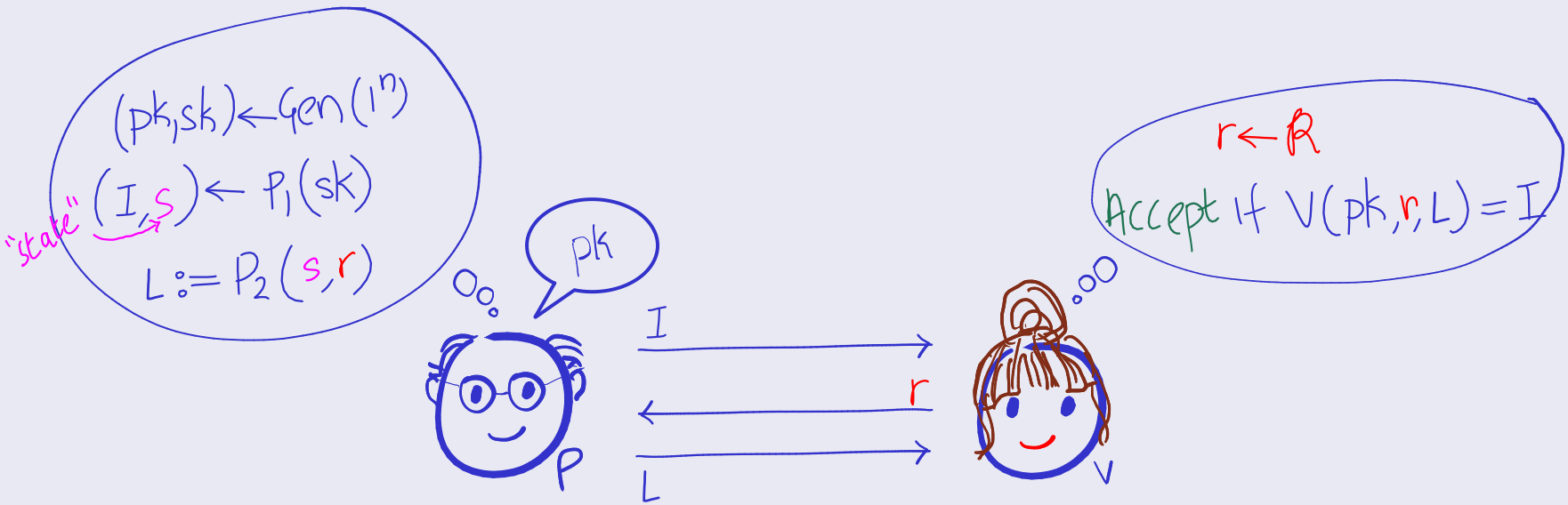
... $\Pi$ is a triple of PPT algorithms $(\mathsf{Gen}, \mathsf{P} = (\mathsf{P}_1, \mathsf{P}_2), \mathsf{V})$ with the following syntax:



# Let's Look at the Syntax

## Definition 3 ((Three-Round) Identification (ID) Protocol...)

... $\Pi$ is a triple of PPT algorithms $(\mathsf{Gen}, \mathsf{P} = (\mathsf{P}_1, \mathsf{P}_2), \mathsf{V})$ with the following syntax:




## Exercise 1

Define the correctness requirement

# How to Model Security?

- The impostor (who doesn't know the secret key):

  - May see several transcripts: via $\mathsf{Auth}_{sk}$ oracle
  - Should not be able to fool the verifier into accepting *in the protocol*

# How to Model Security?

- The impostor (who doesn't know the secret key):
  - May see several transcripts: via $\mathsf{Auth}_{sk}$ oracle
  - Should not be able to fool the verifier into accepting *in the protocol*

**Definition 4 (Security against passive attack)**

An ID protocol $\Pi := (\mathsf{Gen}, \mathsf{P} = (\mathsf{P}_1, \mathsf{P}_2), \mathsf{V})$ is secure against passive attacks if no PPT adversary Imp can break $\Pi$ in the following game with a non-negligible probability.


1. Give $pk$ to Imp, for $(pk, sk) \leftarrow \mathsf{Gen}(1^n)$

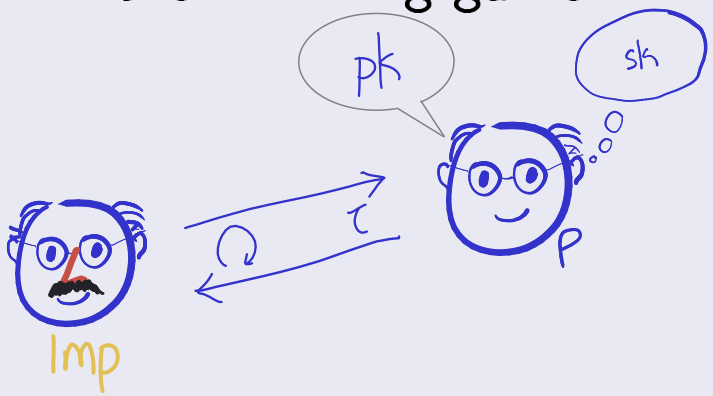
# How to Model Security?

- The impostor (who doesn't know the secret key):
  - May see several transcripts: via $\mathsf{Auth}_{sk}$ oracle
  - Should not be able to fool the verifier into accepting *in the protocol*

## Definition 4 (Security against passive attack)

An ID protocol $\Pi := (\mathsf{Gen}, \mathsf{P} = (\mathsf{P}_1, \mathsf{P}_2), \mathsf{V})$ is secure against passive attacks if no PPT adversary Imp can break $\Pi$ in the following game with a non-negligible probability.

1. Give $pk$ to Imp, for $(pk, sk) \leftarrow \mathsf{Gen}(1^n)$
2. Imp queries to $\mathsf{Auth}_{sk}()$ to get $\tau$s



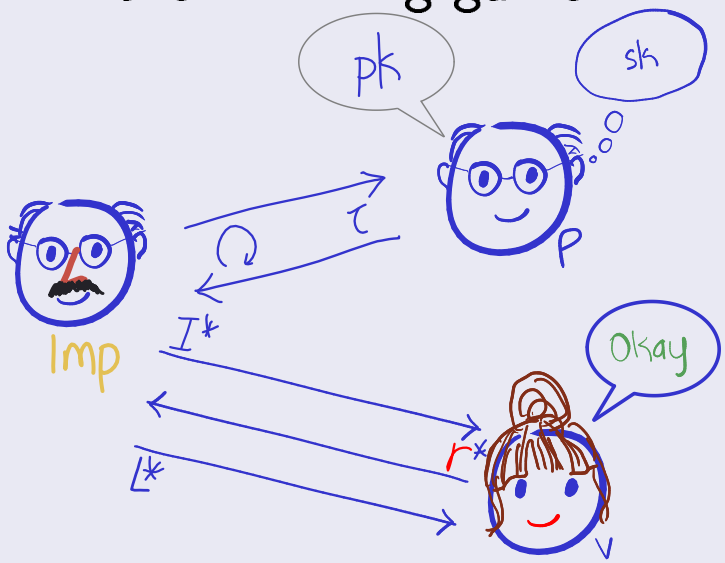
# How to Model Security?

- The impostor (who doesn't know the secret key):
  - May see several transcripts: via $\mathsf{Auth}_{sk}$ oracle
  - Should not be able to fool the verifier into accepting *in the protocol*

## Definition 4 (Security against passive attack)

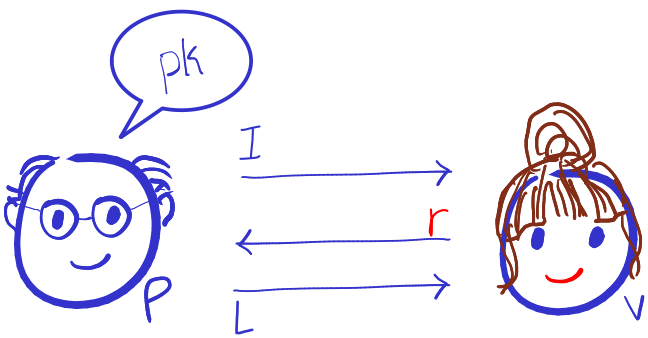
An ID protocol $\Pi := (\mathsf{Gen}, \mathsf{P} = (\mathsf{P}_1, \mathsf{P}_2), \mathsf{V})$ is secure against passive attacks if no PPT adversary Imp can break $\Pi$ in the following game with a non-negligible probability.

1. Give $pk$ to Imp, for $(pk, sk) \leftarrow \mathsf{Gen}(1^n)$
2. Imp queries to $\mathsf{Auth}_{sk}()$ to get $\tau$s
3. When Imp sends $I^*$, respond with $r^* \leftarrow \mathcal{R}$
4. Imp responds with $L^*$ and breaks $\Pi$ if $\mathsf{V}(pk, r^*, L^*) = I^*$



# Identification to Signatures: Fiat-Shamir Transform

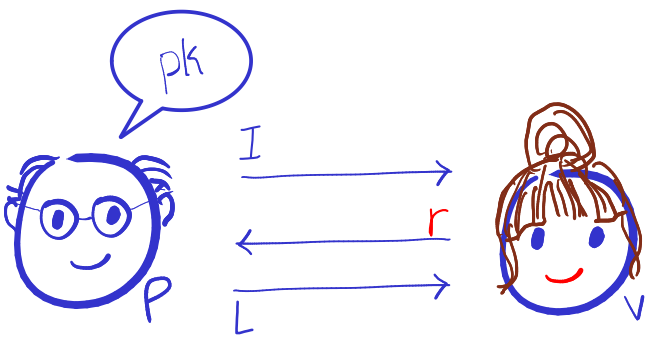
Intuition: ID protocol "proves" P's knowledge of $sk$



# Identification to Signatures: Fiat-Shamir Transform

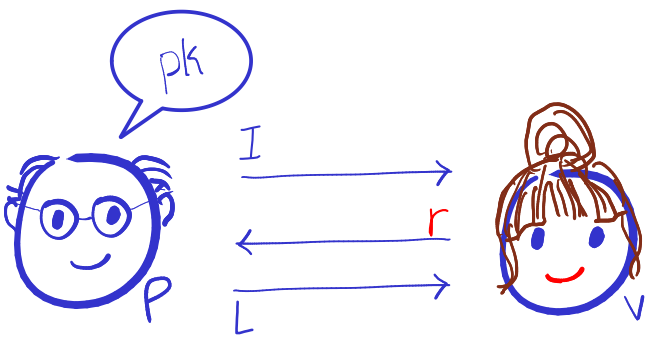
Intuition: ID protocol "proves" P's knowledge of *sk*

Problem: how to *non-interactively* generate V's message *r*?



# Identification to Signatures: Fiat-Shamir Transform

- Intuition: ID protocol "proves" P's knowledge of $sk$
- Problem: how to *non-interactively* generate V's message $r$?
  - Replace V with a random oracle $H : \{0,1\}^* \to \mathcal{R}$
  - Signature on $m$ is the "flattened" transcript where $r := H(I \| m)$



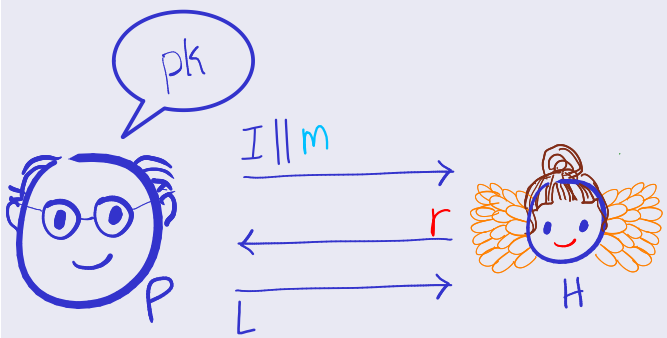
# Identification to Signatures: Fiat-Shamir Transform

Intuition: ID protocol "proves" P's knowledge of $sk$

Problem: how to *non-interactively* generate V's message $r$?

- Replace V with a random oracle $H : \{0,1\}^* \to \mathcal{R}$
- Signature on $m$ is the "flattened" transcript where $r := H(I \| m)$

**Construction 1** ($\Pi := (\mathsf{Gen}, (\mathsf{P}_1, \mathsf{P}_2), \mathsf{V}) \mapsto \Sigma := (\mathsf{Gen}, \mathsf{Sign}^H, \mathsf{Ver}^H)$)




- $\mathsf{Sign}^H(sk, m)$: output $\sigma := (I, r, L)$, where $(I, s) \leftarrow \mathsf{P}_1(sk)$, $r := H(I \| m)$ and $L := \mathsf{P}_2(s, r)$

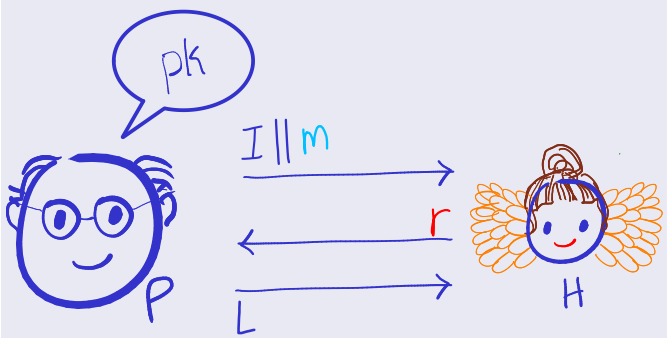
# Identification to Signatures: Fiat-Shamir Transform

Intuition: ID protocol "proves" P's knowledge of $sk$

Problem: how to *non-interactively* generate V's message $r$?

- Replace V with a random oracle $H : \{0,1\}^* \to \mathcal{R}$
- Signature on $m$ is the "flattened" transcript where $r := H(I \| m)$

## Construction 1 ($\Pi := (\mathsf{Gen}, (\mathsf{P}_1, \mathsf{P}_2), \mathsf{V}) \mapsto \Sigma := (\mathsf{Gen}, \mathsf{Sign}^H, \mathsf{Ver}^H)$)

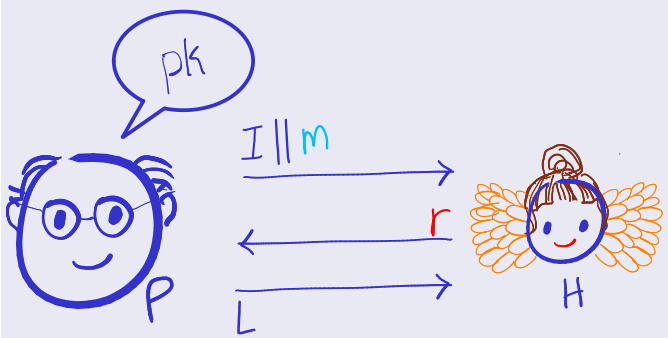



- $\mathsf{Sign}^H(sk, m)$: output $\sigma := (I, r, L)$, where $(I, s) \leftarrow \mathsf{P}_1(sk)$, $r := H(I \| m)$ and $L := \mathsf{P}_2(s, r)$
- $\mathsf{Ver}^H(pk, \sigma, m)$: accept iff $r = H(I \| m)$ and $\mathsf{V}(pk, r, L) = I$

# Identification to Signatures: Fiat-Shamir Transform

- Intuition: ID protocol "proves" P's knowledge of $sk$
- Problem: how to *non-interactively* generate V's message $r$?
  - Replace V with a random oracle $H : \{0,1\}^* \to \mathcal{R}$
  - Signature on $m$ is the "flattened" transcript where $r := H(I \| m)$

## Construction 1 ($\Pi := (\mathsf{Gen}, (\mathsf{P}_1, \mathsf{P}_2), \mathsf{V}) \mapsto \Sigma := (\mathsf{Gen}, \mathsf{Sign}^H, \mathsf{Ver}^H)$)




- $\mathsf{Sign}^H(sk, m)$: output $\sigma := (I, r, L)$, where $(I, s) \leftarrow \mathsf{P}_1(sk)$, $r := H(I \| m)$ and $L := \mathsf{P}_2(s, r)$
- $\mathsf{Ver}^H(pk, \sigma, m)$: accept iff $r = H(I \| m)$ and $\mathsf{V}(pk, r, L) = I$

## Theorem 6

*If $\Pi$ is secure against passive attacks, then $\Sigma$ is EU-CMA secure if $H$ is modelled as random oracle*

# Schnorr's ID Protocol...

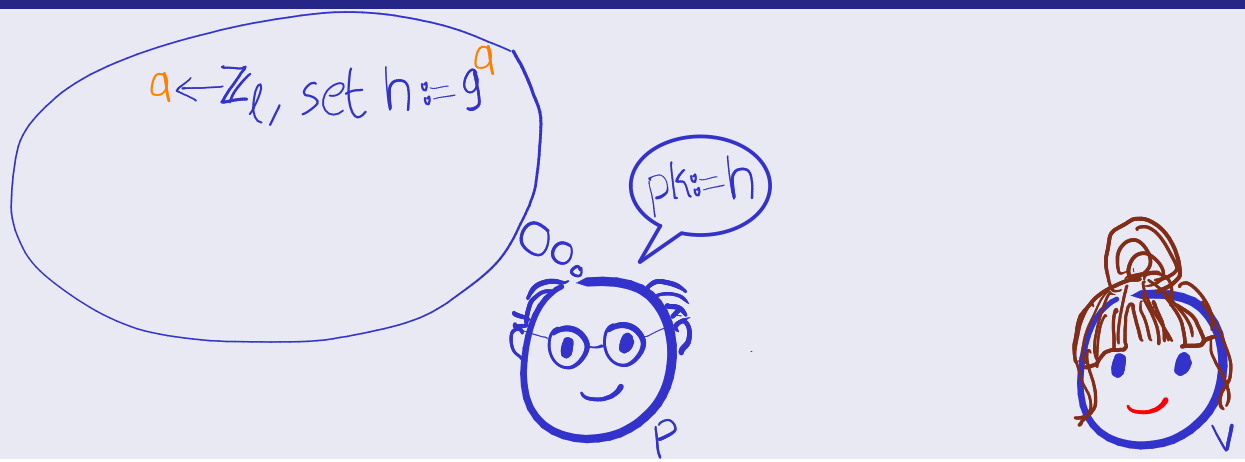
Recall Elgamal PKE over group $\mathbb{G}$ of prime order $\ell$

- Gen: public key is $h := g^x$ and secret key is $x \in \mathbb{Z}_\ell$

# Schnorr's ID Protocol...

- Recall Elgamal PKE over group $\mathbb{G}$ of prime order $\ell$
  - Gen: public key is $h := g^x$ and secret key is $x \in \mathbb{Z}_\ell$
  - Schnorr's ID protocol: authenticates Elgamal public key

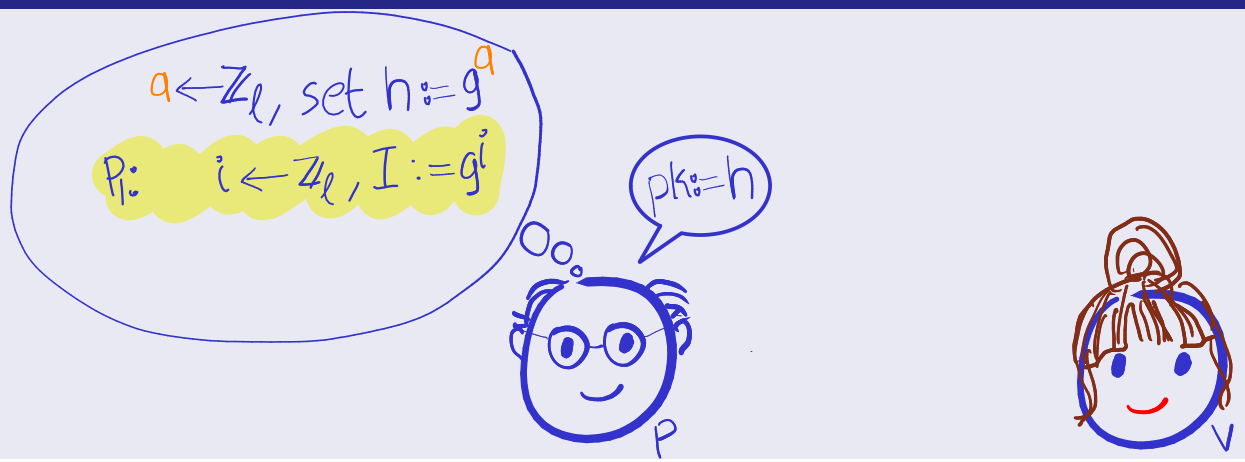
**Protocol 1 (Schnorr's ID protocol $\Pi := (\mathsf{Gen}, (\mathsf{P}_1, \mathsf{P}_2), \mathsf{V})$)**



# Schnorr's ID Protocol...

- Recall Elgamal PKE over group $\mathbb{G}$ of prime order $\ell$
  - Gen: public key is $h := g^x$ and secret key is $x \in \mathbb{Z}_\ell$
  - Schnorr's ID protocol: authenticates Elgamal public key

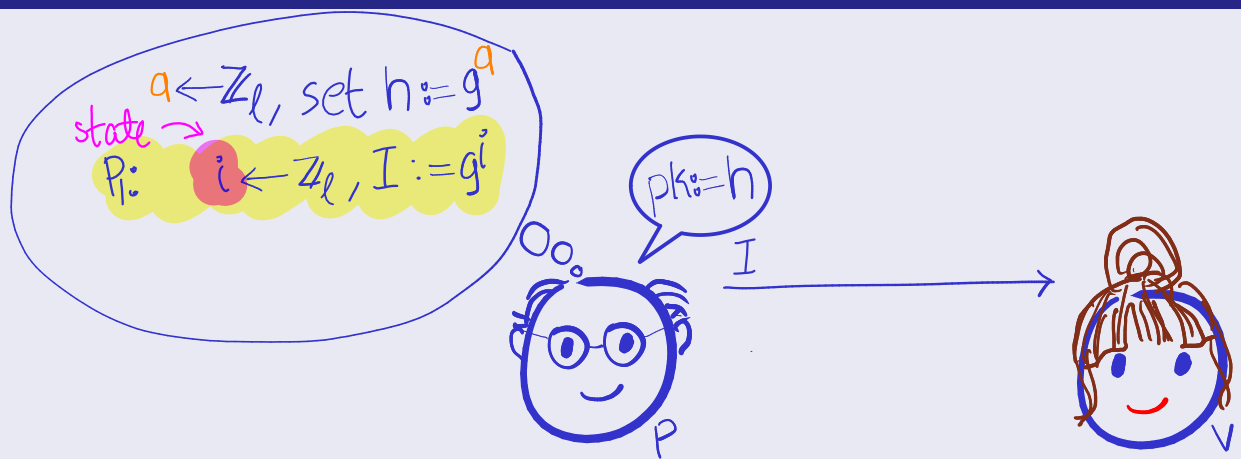
**Protocol 1 (Schnorr's ID protocol $\Pi := (\mathsf{Gen}, (\mathsf{P}_1, \mathsf{P}_2), \mathsf{V})$)**



# Schnorr's ID Protocol...

- Recall Elgamal PKE over group $\mathbb{G}$ of prime order $\ell$
  - Gen: public key is $h := g^x$ and secret key is $x \in \mathbb{Z}_\ell$
  - Schnorr's ID protocol: authenticates Elgamal public key

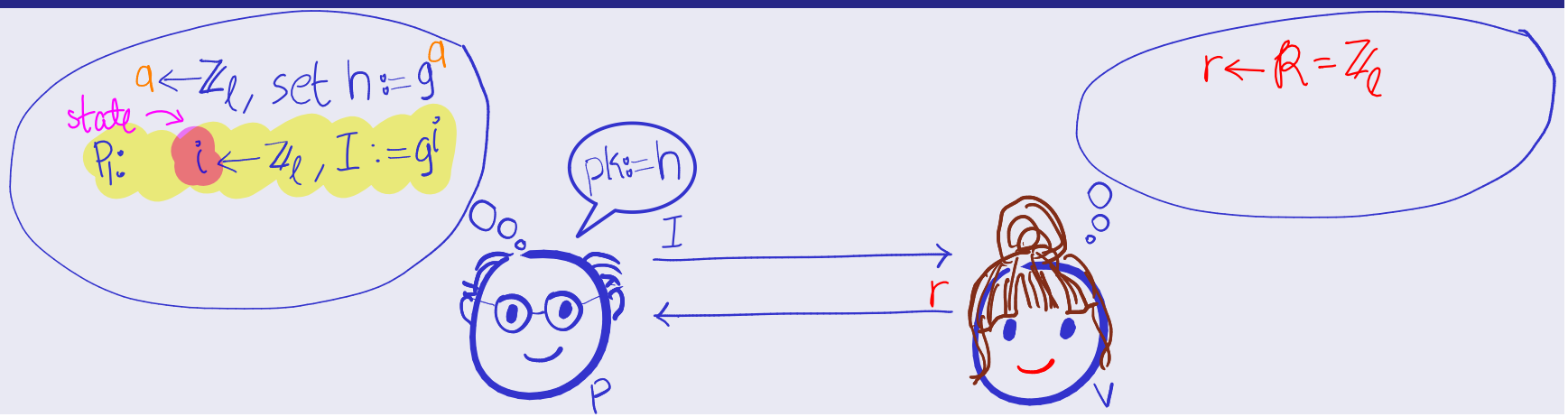
**Protocol 1 (Schnorr's ID protocol $\Pi := (\mathsf{Gen}, (\mathsf{P}_1, \mathsf{P}_2), \mathsf{V})$)**



# Schnorr's ID Protocol...

- Recall Elgamal PKE over group $\mathbb{G}$ of prime order $\ell$
  - Gen: public key is $h := g^x$ and secret key is $x \in \mathbb{Z}_\ell$
  - Schnorr's ID protocol: authenticates Elgamal public key

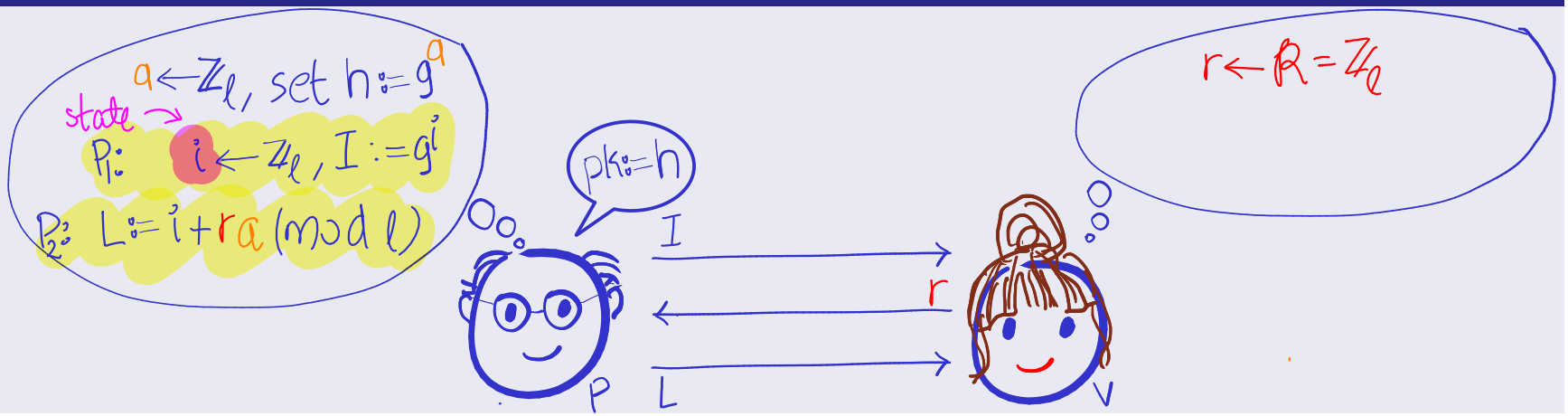
**Protocol 1 (Schnorr's ID protocol $\Pi := (\mathsf{Gen}, (\mathsf{P}_1, \mathsf{P}_2), \mathsf{V})$)**



# Schnorr's ID Protocol...

- Recall Elgamal PKE over group $\mathbb{G}$ of prime order $\ell$
  - Gen: public key is $h := g^x$ and secret key is $x \in \mathbb{Z}_\ell$
  - Schnorr's ID protocol: authenticates Elgamal public key

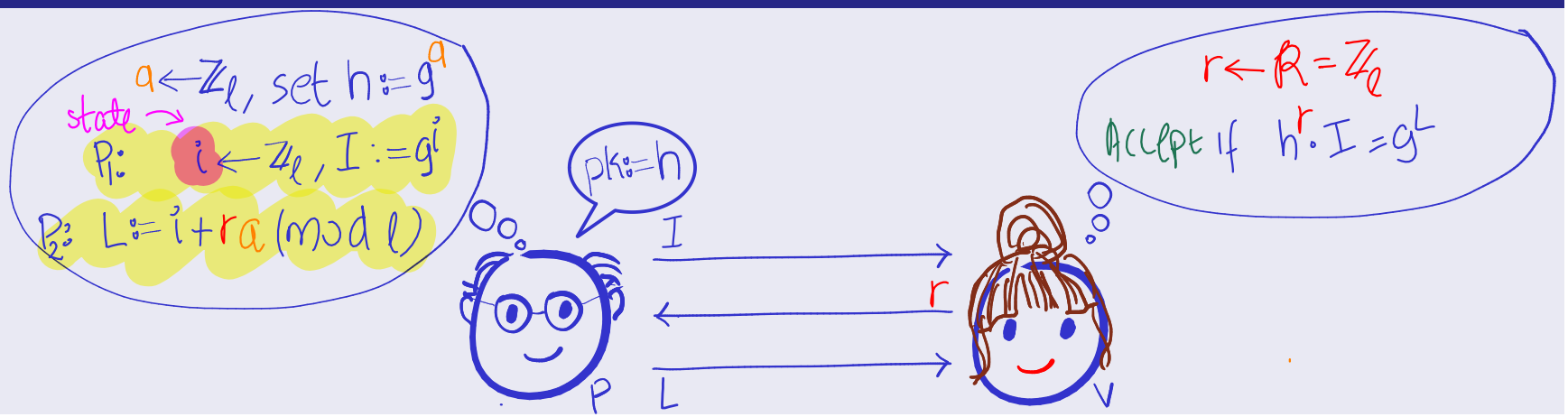
**Protocol 1 (Schnorr's ID protocol $\Pi := (\mathsf{Gen}, (\mathsf{P}_1, \mathsf{P}_2), \mathsf{V})$)**



# Schnorr's ID Protocol...

- Recall Elgamal PKE over group $\mathbb{G}$ of prime order $\ell$
  - Gen: public key is $h := g^x$ and secret key is $x \in \mathbb{Z}_\ell$
  - Schnorr's ID protocol: authenticates Elgamal public key

**Protocol 1 (Schnorr's ID protocol $\Pi := (\mathsf{Gen}, (\mathsf{P}_1, \mathsf{P}_2), \mathsf{V})$)**

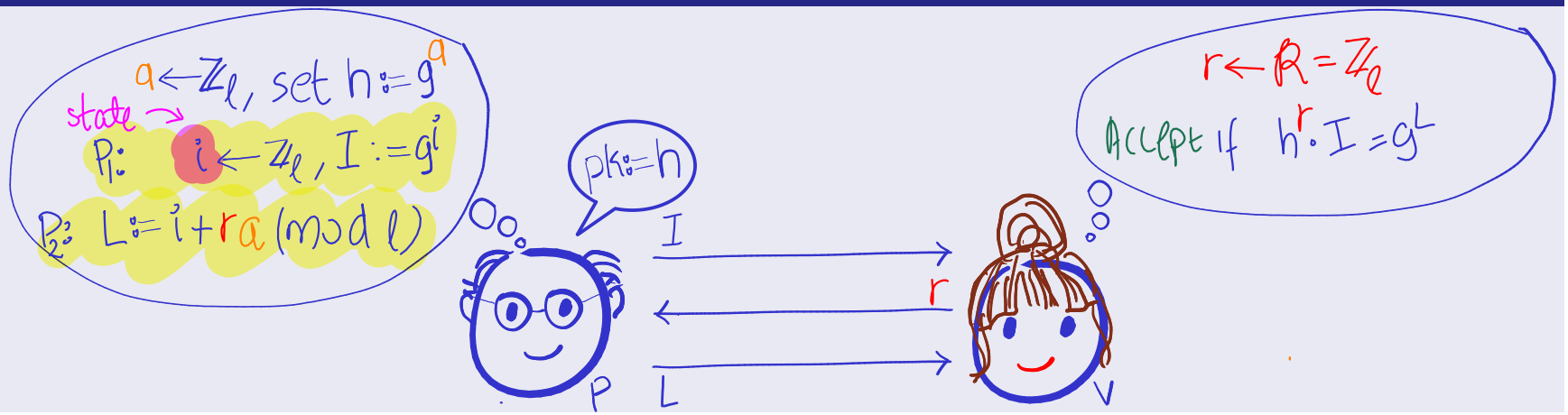


# Schnorr's ID Protocol...

Recall Elgamal PKE over group $\mathbb{G}$ of prime order $\ell$

- Gen: public key is $h := g^x$ and secret key is $x \in \mathbb{Z}_\ell$
- Schnorr's ID protocol: authenticates Elgamal public key

**Protocol 1 (Schnorr's ID protocol $\Pi := (\mathsf{Gen}, (\mathsf{P}_1, \mathsf{P}_2), \mathsf{V})$)**



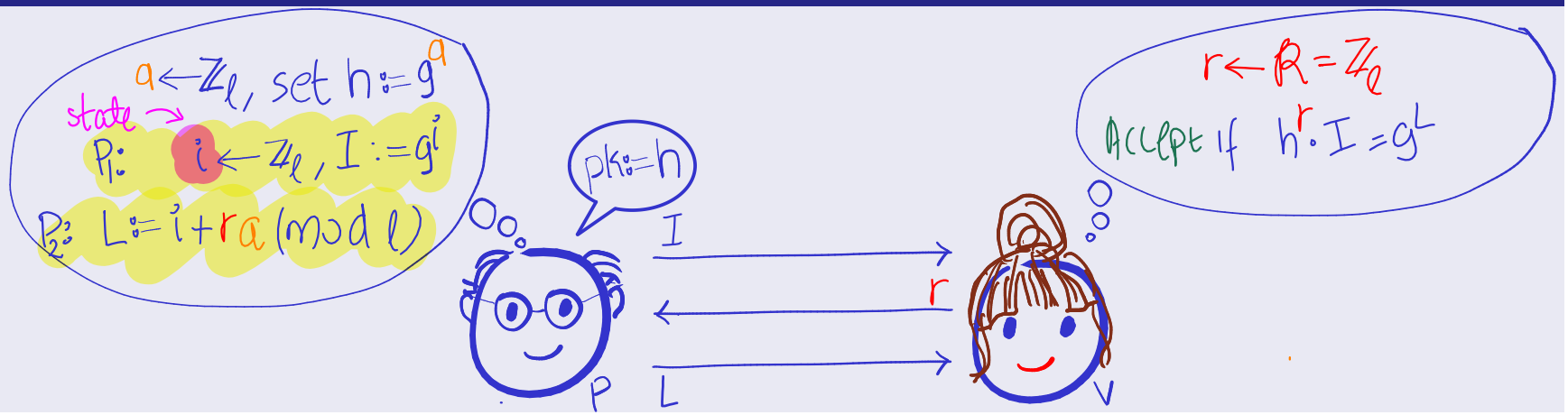

- Intuition for $\Pi$'s passive security:
  - Can Imp compute correct $L$ without knowing $x$?

# Schnorr's ID Protocol...

Recall Elgamal PKE over group $\mathbb{G}$ of prime order $\ell$

- Gen: public key is $h := g^x$ and secret key is $x \in \mathbb{Z}_\ell$
- Schnorr's ID protocol: authenticates Elgamal public key

**Protocol 1 (Schnorr's ID protocol $\Pi := (\mathsf{Gen}, (\mathsf{P}_1, \mathsf{P}_2), \mathsf{V})$)**




- Intuition for $\Pi$'s passive security:

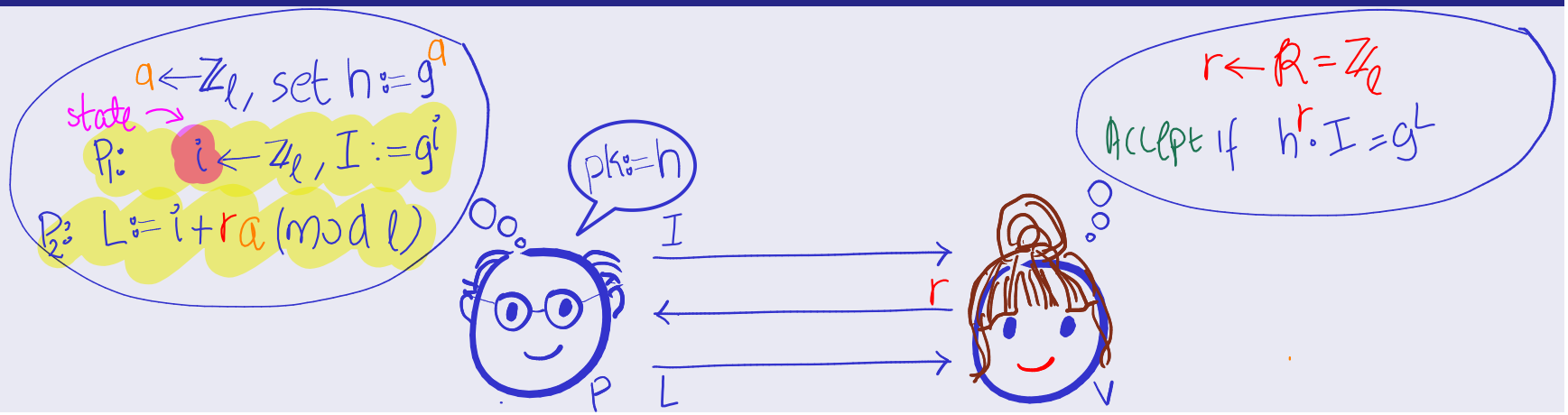
  - Can Imp compute correct $L$ without knowing $x$? Seems not
    - It is possible to "extract" $x$ from Imp if fools V a lot!
  - Does transcript reveal $x$?

# Schnorr's ID Protocol...

Recall Elgamal PKE over group $\mathbb{G}$ of prime order $\ell$

- Gen: public key is $h := g^x$ and secret key is $x \in \mathbb{Z}_\ell$
- Schnorr's ID protocol: authenticates Elgamal public key

**Protocol 1 (Schnorr's ID protocol $\Pi := (\mathsf{Gen}, (\mathsf{P}_1, \mathsf{P}_2), \mathsf{V})$)**




- Intuition for $\Pi$'s passive security:

  - Can Imp compute correct $L$ without knowing $x$? Seems not
    - It is possible to "extract" $x$ from Imp if fools V a lot!
  - Does transcript reveal $x$? No, it can be "simulated"!

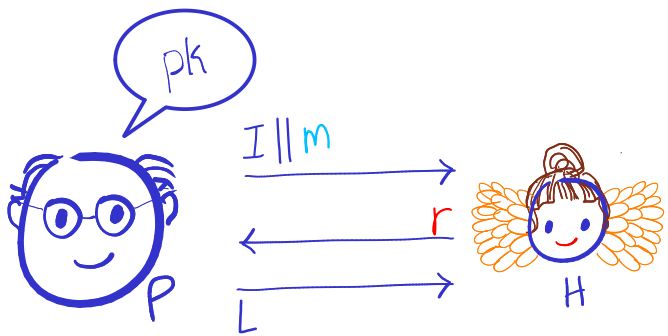
1. Sample $r, s \leftarrow \mathbb{Z}_\ell$
2. Output $(g^s \cdot h^{-r}, r, s)$

# Schnorr's ID Protocol...

## Theorem 7

*If discrete-log assumption holds in $\mathbb{G}$, then $\Pi$ is secure against passive attacks*

- Schnorr signature: apply Fiat-Shamir transform to $\Pi$




## Exercise 2

Formally describe Schnorr signature

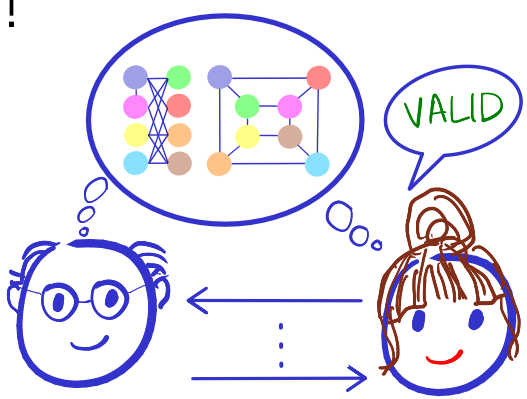
# Recap/Next Module

- Efficient digital signatures
  - RSA-FDH: via hash-then-invert
    - Tools: trapdoor permutation, random oracle programming
  - Schnorr signature: via identification protocol
    - Tool: Fiat-Shamir Transform
  - Other efficient signatures: ECDSA and EdDSA
    - Coming up in Lab Exercise 4!

# Recap/Next Module

- Efficient digital signatures
  - RSA-FDH: via hash-then-invert
    - Tools: trapdoor permutation, random oracle programming
  - Schnorr signature: via identification protocol
    - Tool: Fiat-Shamir Transform
  - Other efficient signatures: ECDSA and EdDSA
    - Coming up in Lab Exercise 4!

- Next Module: Applications!
  - Zero-knowledge proof
  - eVoting
  - TLS/SSL
  - Secure messaging
  - Zerocash




helios
Trust the vote.

©Wikipedia

# References

1. Details about trapdoor permutation (TDP) can be found in [KL14, Chapter 15.1.1]. As a primitive, they were introduced by Yao [Yao82].
2. You can read about RSA-FDH in [KL14, Chapter 13.4.2]. A tighter analysis was done later on by Coron [Cor00].
3. Identification protocols and its connections to signatures are discussed in [KL14, Chapter 13.5.1]. In particular, proof of Theorem 6 can be found in [KL14, Theorem 13.10]. Schnorr's identification protocol for DLP is then covered in [KL14, Chapter 13.5.2]. The protocol is originally from [Sch90].


Jean-Sébastien Coron.

On the exact security of full domain hash.

In Mihir Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 229–235. Springer, Berlin, Heidelberg, August 2000.


Jonathan Katz and Yehuda Lindell.

*Introduction to Modern Cryptography (3rd ed.)*.

Chapman and Hall/CRC, 2014.


Claus-Peter Schnorr.

Efficient identification and signatures for smart cards.

In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 239–252. Springer, New York, August 1990.


Andrew Chi-Chih Yao.

Theory and applications of trapdoor functions (extended abstract).

In *23rd FOCS*, pages 80–91. IEEE Computer Society Press, November 1982.