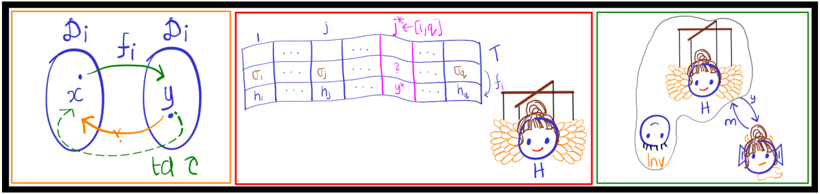


CS783: Theoretical Foundations of Cryptography

Lecture 13 (13/Sep/24)

Instructor: Chethan Kamath

Recall from Last Lecture

- Sub-task 5.a: domain-extension of digital signature/MAC
- Reduces to constructing collision-resistant hash functions

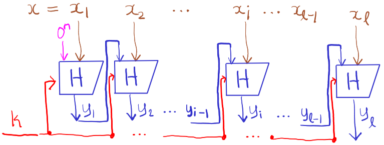
Recall from Last Lecture

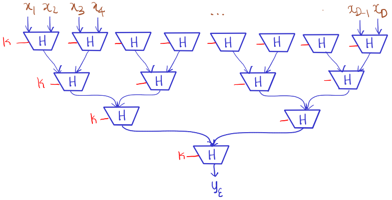
- Sub-task 5.a: domain-extension of digital signature/MAC
- Reduces to constructing collision-resistant hash functions
  - Generic attacks via pigeonhole principle and birthday paradox

Recall from Last Lecture

- Sub-task 5.a: domain-extension of digital signature/MAC

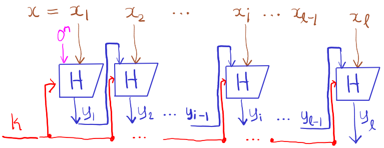
- Reduces to constructing collision-resistant hash functions
  - Generic attacks via pigeonhole principle and birthday paradox
  - Domain extension for compression functions
    - Merkle-Damgård transform
    - Merkle trees

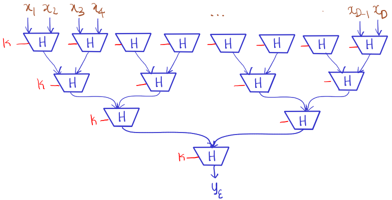




Recall from Last Lecture

- Sub-task 5.a: domain-extension of digital signature/MAC
- Reduces to constructing collision-resistant hash functions
  - Generic attacks via pigeonhole principle and birthday paradox
  - Domain extension for compression functions
    - Merkle-Damgård transform
    - Merkle trees



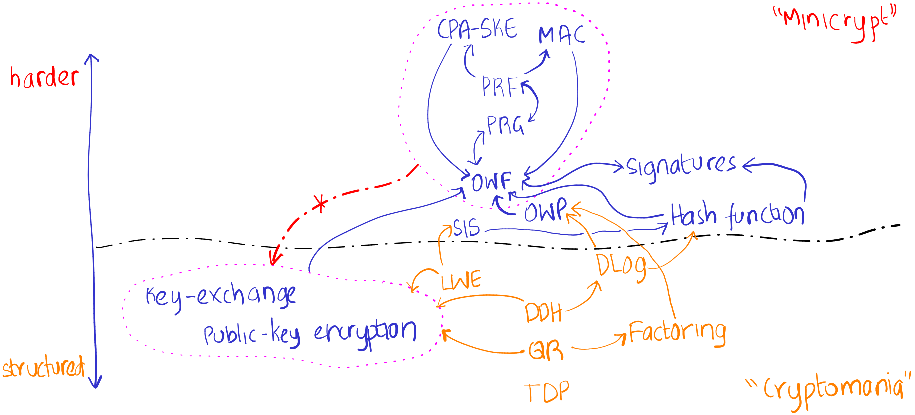


  - Some constructions:
    - Practical/unkeyed: SHA2, MD5
    - Theoretical/keyed: DLog- and SIS-based

Plan for Today's Lecture...

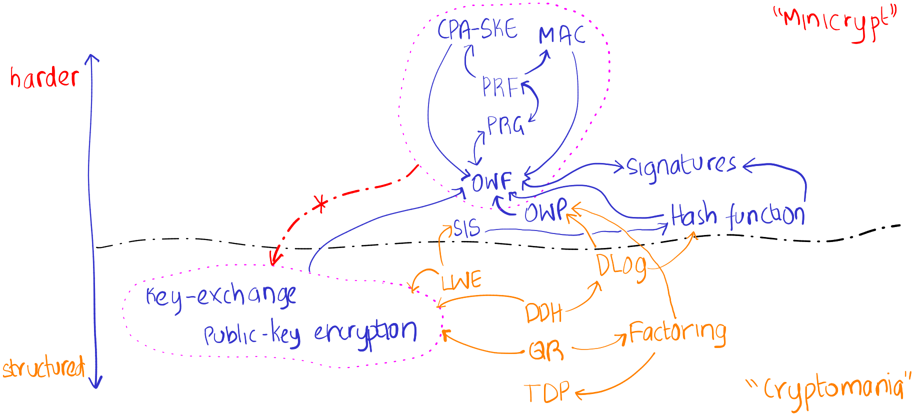
- Motivation: construct efficient signatures

- Motivation: construct efficient signatures



- New primitive: trap-door (one-way) permutation (TDP)
  - Efficient digital signatures from TDP

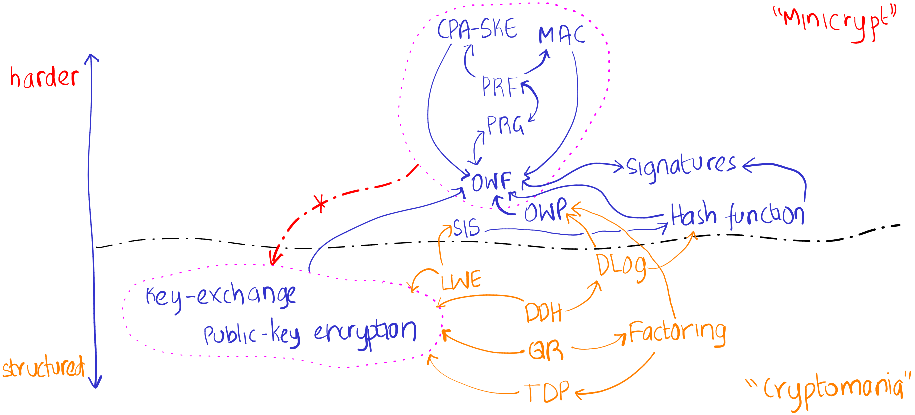
- Motivation: construct efficient signatures



- New primitive: trap-door (one-way) permutation (TDP)
  - Efficient digital signatures from TDP

Plan for Today's Lecture...

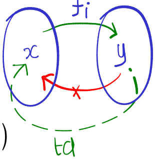
- Motivation: construct efficient signatures



- New primitive: trap-door (one-way) permutation (TDP)
  - Efficient digital signatures from TDP
  - PKE from TDP

Plan for Today's Lecture...

1. Trap-Door (One-Way) Permutation (TDP)

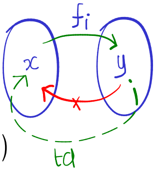


2. Efficient Digital Signatures from TDP (in Random-Oracle Model)

3. Public-Key Encryption from TDP

Plan for Today's Lecture

1 Trap-Door (One-Way) Permutation (TDP)



2 Efficient Digital Signatures from TDP (in Random-Oracle Model)

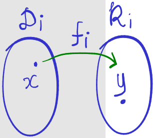
3 Public-Key Encryption from TDP

Recall from Lecture 6: *Collection* of OWFs

Defintion 1 (One-way function (OWF) collection)

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

1. *There is an efficient index-sampling algorithm* Index
2. *Each* f_i *in collection is efficiently computable*



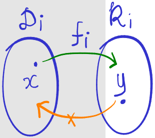
Recall from Lecture 6: *Collection* of OWFs

Defintion 1 (One-way function (OWF) collection)

A collection of functions $\mathsf{f} := \{\mathsf{f}_i : \mathcal{D}_i \to \mathcal{R}_i\}_{i \in \mathcal{I} \subseteq \{0,1\}^*}$ *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:*

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



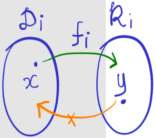
Recall from Lecture 6: *Collection* of OWFs

Defintion 1 (One-way function (OWF) collection)

A collection of functions $\mathsf{f} := \{\mathsf{f}_i : \mathcal{D}_i \to \mathcal{R}_i\}_{i \in \mathcal{I} \subseteq \{0,1\}^*}$ *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:*

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



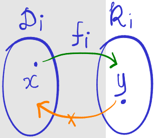
- Recall examples:
  1. Squaring modulo composite $N = pq$: $\mathsf{f}_N(x) := x^2 \bmod N$
  2. Exp. with generator g modulo prime p: $\mathsf{f}_{p,g}(x) := g^x \bmod p$

Recall from Lecture 6: *Collection* of OWFs

Defintion 1 (One-way function (OWF) collection)

A collection of functions $\mathsf{f} := \{\mathsf{f}_i : \mathcal{D}_i \to \mathcal{R}_i\}_{i \in \mathcal{I} \subseteq \{0,1\}^*}$ *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:*



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

- Recall examples:
  1. Squaring modulo composite $N = pq$: $\mathsf{f}_N(x) := x^2 \bmod N$
  2. Exp. with generator g modulo prime p: $\mathsf{f}_{p,g}(x) := g^x \bmod p$

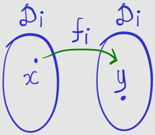
? Describe $\mathcal{I}$, $\mathcal{D}_i$ and $\mathcal{R}_i$ above. How is $i \in \mathcal{I}$ sampled?

OWP Collection *with Trap-Door*

Defintion 2 (Trapdoor (one-way) permutation (TDP) collection)

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

1. *There is an efficient index+trapdoor sampling algorithm* Index

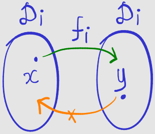


OWP Collection *with Trap-Door*

Defintion 2 (Trapdoor (one-way) permutation (TDP) collection)

A collection of permutations $\mathsf{f} = \{\mathsf{f}_i : \mathcal{D}_i \to \mathcal{D}_i\}_{i \in \mathcal{I} \subseteq \{0,1\}^*}$ *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:*



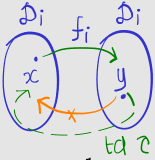
$$p(n) := \Pr_{\substack{(i,\tau) \leftarrow \mathsf{Index}(1^n) \\ x \leftarrow \mathcal{D}_i}}[\mathsf{Inv}(\mathsf{f}_i(x)) \in \mathsf{f}_i^{-1}(\mathsf{f}_i(x))]$$

OWP Collection *with Trap-Door*

Defintion 2 (Trapdoor (one-way) permutation (TDP) collection)

A collection of permutations $\mathsf{f} = \{\mathsf{f}_i : \mathcal{D}_i \to \mathcal{D}_i\}_{i \in \mathcal{I} \subseteq \{0,1\}^*}$ *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:*



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

4. f_i^{-1} *can be efficiently computed given trapdoor* τ *for* i

Candidate TDPs

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

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

N=pq for primes pq

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

Candidate TDPs

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

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

N=pq for primes p,q

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

Quadratic residues mod N

- Rabin TDP $\{f_N : \mathbb{Z}_N^\times[+,+] \to \mathbb{Z}_N^\times[+,+]\}_N$, defined as

$$f_N(x) := x^2 \bmod N$$

  - One-way by hardness of factoring
  - The trapdoor is (p, q)

Candidate TDPs

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

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

N=pq for primes pq

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

Quadratic residues mod N

- Rabin TDP $\{f_N : \mathbb{Z}_N^\times[+,+] \to \mathbb{Z}_N^\times[+,+]\}_N$, defined as

$$f_N(x) := x^2 \bmod N$$

  - One-way by hardness of factoring
  - The trapdoor is (p,q)

Exercise 1

How can we compute $f_N^{-1}(y)$ given (p,q)?

Plan for Today's Lecture

1 Trap-Door (One-Way) Permutation (TDP)

2 Efficient Digital Signatures from TDP (in Random-Oracle Model)

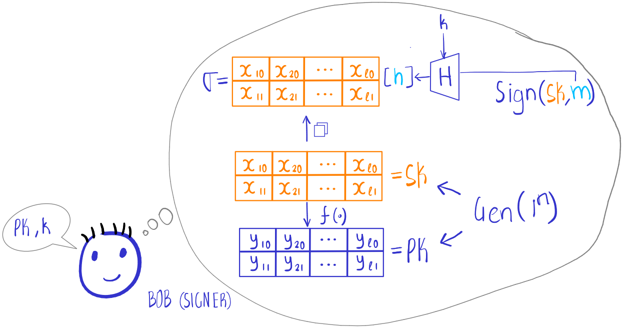
3 Public-Key Encryption from TDP

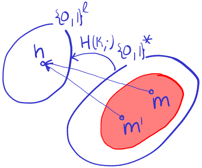
Recall: “Hash-Then-Sign” One-Time Signature

- 1) Compute “hash” $h = \mathsf{H}(\mathsf{k}, m)$ 2) sign h using Lamport's OTS

Recall: "Hash-Then-Sign" One-Time Signature

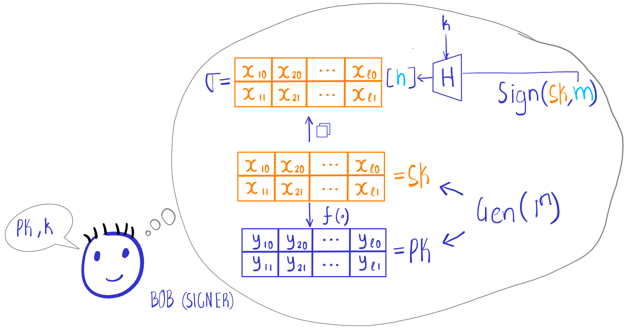
- 1) Compute "hash" $\boldsymbol{h} = \mathrm{H}(\mathrm{k}, \boldsymbol{m})$ 2) sign $\boldsymbol{h}$ using Lamport's OTS

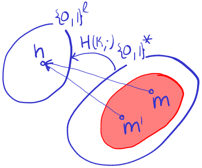




Recall: "Hash-Then-Sign" One-Time Signature

- 1) Compute "hash" $\boldsymbol{h} = \mathrm{H}(\mathrm{k}, \boldsymbol{m})$ 2) sign $\boldsymbol{h}$ using Lamport's OTS

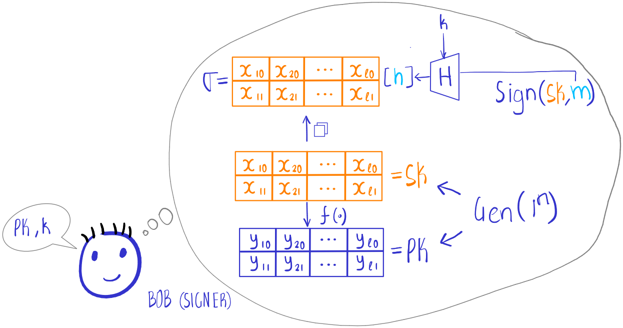


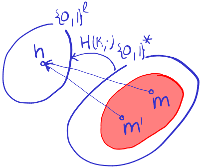


Theorem 1 (Theorem 3, Lecture 12 (rephrased))

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

- 1) Compute "hash" $\boldsymbol{h} = \mathrm{H}(\mathrm{k}, \boldsymbol{m})$ 2) sign $\boldsymbol{h}$ using Lamport's OTS





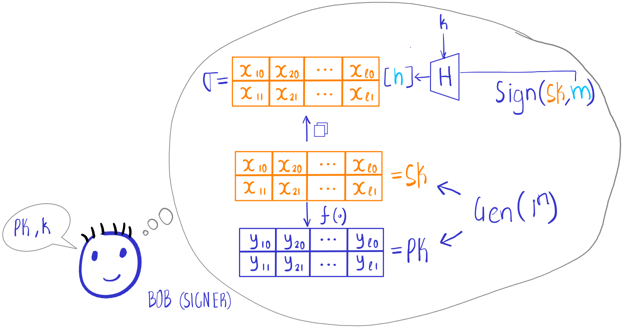
Theorem 1 (Theorem 3, Lecture 12 (rephrased))

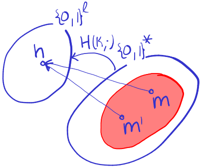
If Lamport's scheme is OTS and H *is CRHF then "hash-then-sign" scheme is a one-time EU-CMA for arbitrarily-long messages.*

- How can a TDP be useful here?

Recall: "Hash-Then-Sign" One-Time Signature

- 1) Compute "hash" $\boldsymbol{h} = \mathrm{H}(\mathrm{k}, \boldsymbol{m})$ 2) sign $\boldsymbol{h}$ using Lamport's OTS





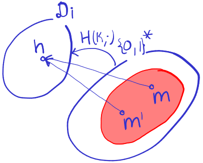
Theorem 1 (Theorem 3, Lecture 12 (rephrased))

If Lamport's scheme is OTS 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

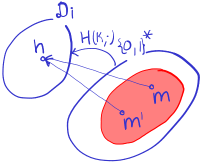
TDP-based Signature: "Hash-then-Invert"

- 1) Compute "hash" $h = \mathsf{H}(\mathsf{k}, m)$ 2) invert h using trapdoor
  - "Full domain" hash function $\mathsf{H} : \mathcal{K} \times \{0,1\}^* \to \mathcal{D}_i$



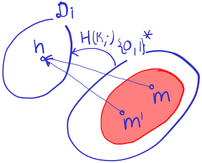
TDP-based Signature: “Hash-then-Invert”

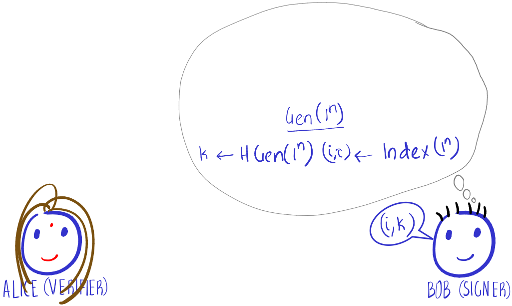
- 1) Compute “hash” $h = \mathsf{H}(\mathsf{k}, m)$ 2) invert h using trapdoor
  - “Full domain” hash function $\mathsf{H} : \mathcal{K} \times \{0,1\}^* \to \mathcal{D}_i$



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

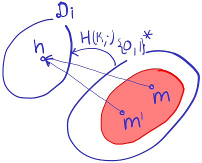
- 1) Compute "hash" $\boldsymbol{h} = \mathrm{H}(\mathrm{k}, \boldsymbol{m})$ 2) invert $\boldsymbol{h}$ using trapdoor
  - "Full domain" hash function $\mathrm{H} : \mathcal{K} \times \{0,1\}^* \to \mathcal{D}_i$

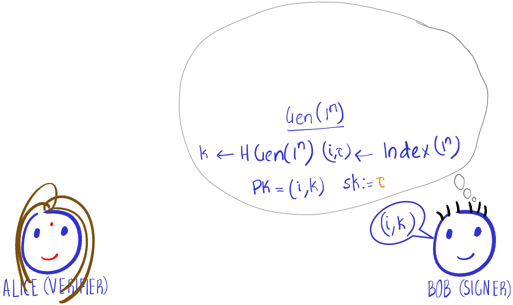




TDP-based Signature: "Hash-then-Invert"

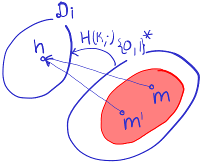
- 1) Compute "hash" $h = \mathrm{H}(\mathrm{k}, m)$ 2) invert h using trapdoor
  - "Full domain" hash function $\mathrm{H} : \mathcal{K} \times \{0,1\}^* \to \mathcal{D}_i$

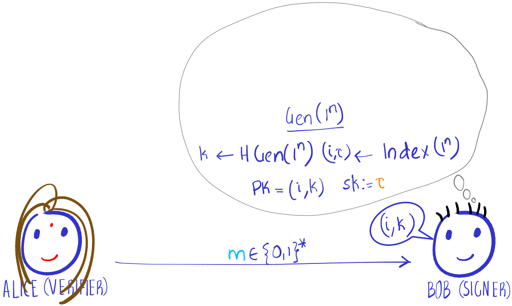




TDP-based Signature: "Hash-then-Invert"

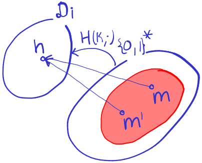
- 1) Compute "hash" $h = \mathrm{H}(\mathrm{k}, m)$ 2) invert h using trapdoor
  - "Full domain" hash function $\mathrm{H} : \mathcal{K} \times \{0,1\}^* \to \mathcal{D}_i$

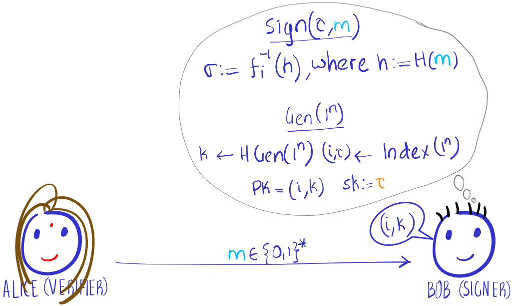




TDP-based Signature: "Hash-then-Invert"

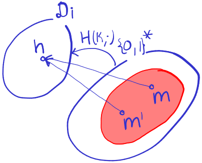
- 1) Compute "hash" $h = \mathrm{H}(\mathrm{k}, m)$ 2) invert h using trapdoor
  - "Full domain" hash function $\mathrm{H} : \mathcal{K} \times \{0,1\}^* \rightarrow \mathcal{D}_i$

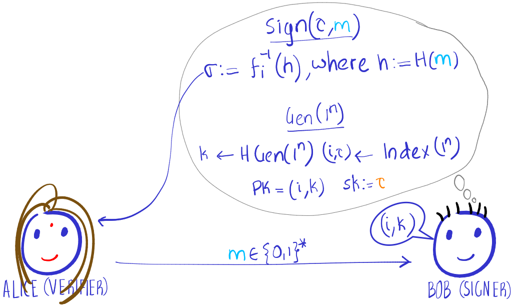




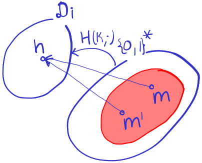
TDP-based Signature: "Hash-then-Invert"

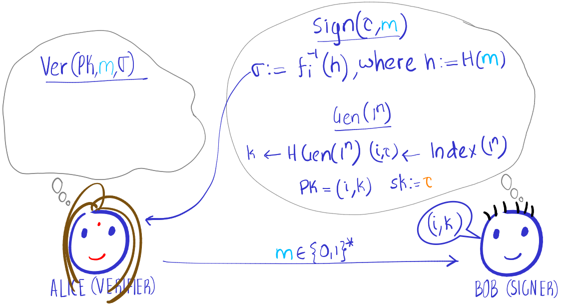
- 1) Compute "hash" $h = \mathsf{H}(\mathsf{k}, m)$ 2) invert h using trapdoor
  - "Full domain" hash function $\mathsf{H} : \mathcal{K} \times \{0,1\}^* \to \mathcal{D}_i$





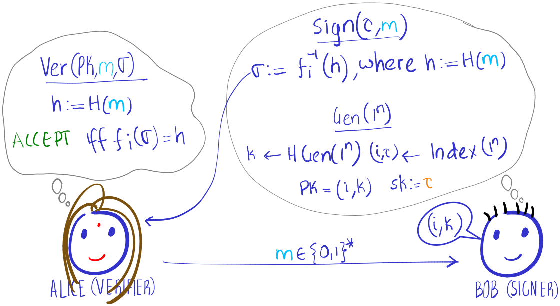
- 1) Compute "hash" $\boldsymbol{h} = \mathrm{H}(\mathrm{k}, \boldsymbol{m})$ 2) invert $\boldsymbol{h}$ using trapdoor
  - "Full domain" hash function $\mathrm{H} : \mathcal{K} \times \{0,1\}^* \rightarrow \mathcal{D}_i$

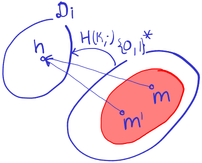




TDP-based Signature: "Hash-then-Invert"

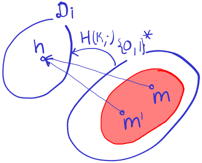
- 1) Compute "hash" $\boldsymbol{h} = \mathrm{H}(\mathrm{k}, \boldsymbol{m})$ 2) invert $\boldsymbol{h}$ using trapdoor
  - "Full domain" hash function $\mathrm{H} : \mathcal{K} \times \{0,1\}^* \to \mathcal{D}_i$

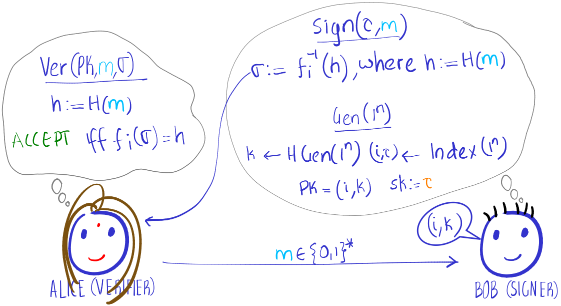




TDP-based Signature: "Hash-then-Invert"

- 1) Compute "hash" $h = \mathsf{H}(\mathsf{k}, m)$ 2) invert h using trapdoor
  - "Full domain" hash function $\mathsf{H} : \mathcal{K} \times \{0,1\}^* \to \mathcal{D}_i$





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

$$\mathsf{f}_{N,e}(x) := x^e \bmod N$$

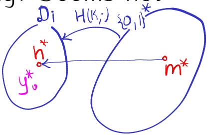
  - Public key: (N, e) and description of H
  - Signatures: one element of $\mathbb{Z}_N^\times$
  - Signing/verification: one exponentiation + hash evaluation

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

- Is H being CRHF sufficient to prove security?

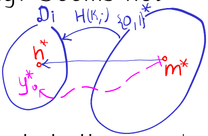
Let's Prove Security of "Hash-then-Invert"...

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



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

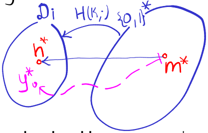
- Is H being CRHF sufficient to prove security? Seems not
- Problem:
  1. Need to invert a *particular challenge* $\boldsymbol{y}^*$
  2. Forger could forger *any* message $\boldsymbol{m}^*$



- Want: exploit H to "link" forgery $(\sigma^*, \boldsymbol{m}^*)$ and challenge $\boldsymbol{y}^*$

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

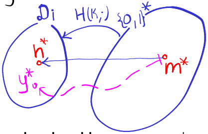
- Is H being CRHF sufficient to prove security? Seems not
- Problem:
  1. Need to invert a *particular challenge* $\boldsymbol{y}^*$
  2. Forger could forger *any* message $\boldsymbol{m}^*$



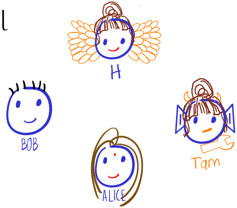
- Want: exploit H to "link" forgery $(\sigma^*, \boldsymbol{m}^*)$ and challenge $\boldsymbol{y}^*$
- "Solution": prove security in *random-oracle* model

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* $\boldsymbol{y}^*$
  2. Forger could forger *any* message $\boldsymbol{m}^*$

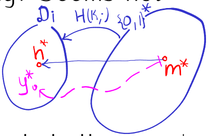


- Want: exploit H to "link" forgery $(\sigma^*, \boldsymbol{m}^*)$ and challenge $\boldsymbol{y}^*$
- "Solution": prove security in *random-oracle* model
  - Idealised model where H is a *random function*
    - All parties have *oracle* access to H

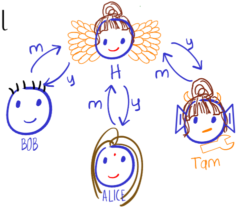


Let's Prove Security of "Hash-then-Invert"...

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

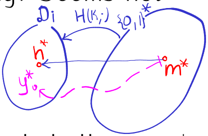


- Want: exploit H to "link" forgery $(\sigma^*, \boldsymbol{m}^*)$ and challenge $\boldsymbol{y}^*$
- "Solution": prove security in *random-oracle* model
  - Idealised model where H is a *random function*
    - All parties have *oracle* access to H

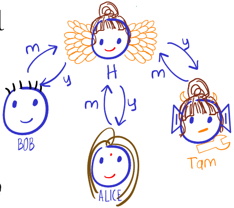


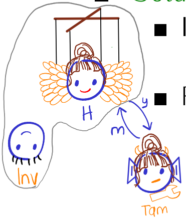
Let's Prove Security of "Hash-then-Invert"...

- Is H being CRHF sufficient to prove security? Seems not
- Problem:



  1. Need to invert a *particular challenge* $\boldsymbol{y}^*$
  2. Forger could forger *any* message $\boldsymbol{m}^*$
- Want: exploit H to "link" forgery $(\sigma^*, \boldsymbol{m}^*)$ and challenge $\boldsymbol{y}^*$
- "Solution": prove security in *random-oracle* model

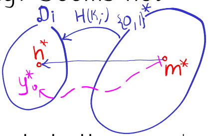




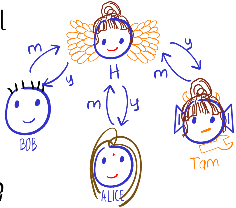
  - Idealised model where H is a *random function*
    - All parties have *oracle* access to H
  - Reduction may "control" H (programming):
    - Constructs H by on-the-fly/lazy sampling
    - A "fresh" query $\boldsymbol{m} \in \{0,1\}^*$ replied with $\boldsymbol{y} \leftarrow \mathcal{D}_I$
    - A "repeat" query $\boldsymbol{m}$ responded *consistently* with $\boldsymbol{y}$ (in table)

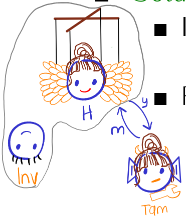
Let's Prove Security of "Hash-then-Invert"...

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



- Want: exploit H to "link" forgery $(\sigma^*, \boldsymbol{m}^*)$ and challenge $\boldsymbol{y}^*$
- "Solution": prove security in *random-oracle* model
  - Idealised model where H is a *random function*
    - All parties have *oracle* access to H
  - Reduction may "control" 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)





- Warning:
  - Only heuristic security guarantee
  - Adversary could exploit specific implementation of H

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

Theorem 2

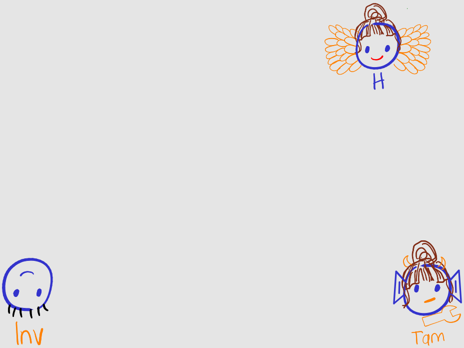
If f is a TDP and $\boldsymbol{H}$ is a random oracle then "hash-then-invert" is EU-CMA for arbitrarily-long messages.

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

Theorem 2

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

Proof sketch: plug and pray via random oracle programming.

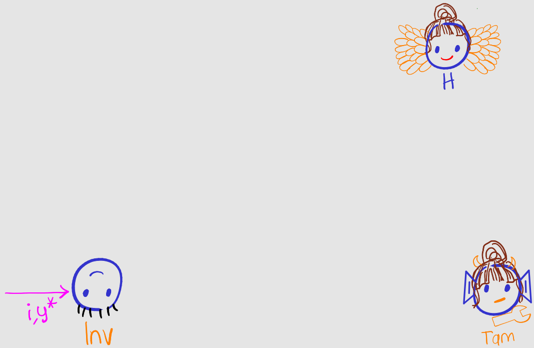


Let's Prove Security of "Hash-then-Invert"...

Theorem 2

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

Proof sketch: plug and pray via random oracle programming.

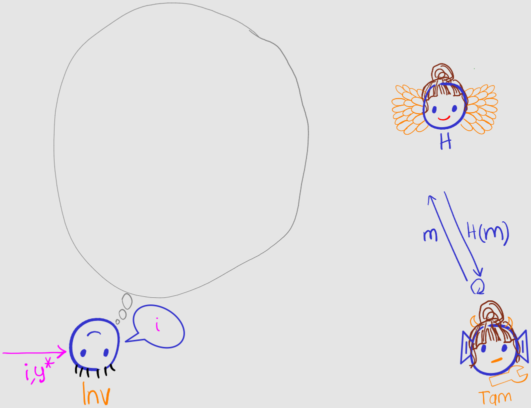


Let's Prove Security of "Hash-then-Invert"...

Theorem 2

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

Proof sketch: plug and pray via random oracle programming.

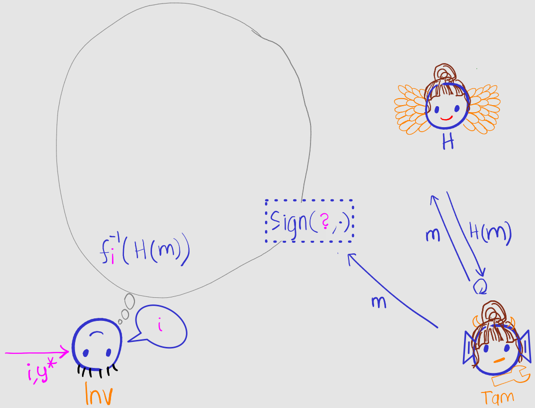


Let's Prove Security of "Hash-then-Invert"...

Theorem 2

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

Proof sketch: plug and pray via random oracle programming.

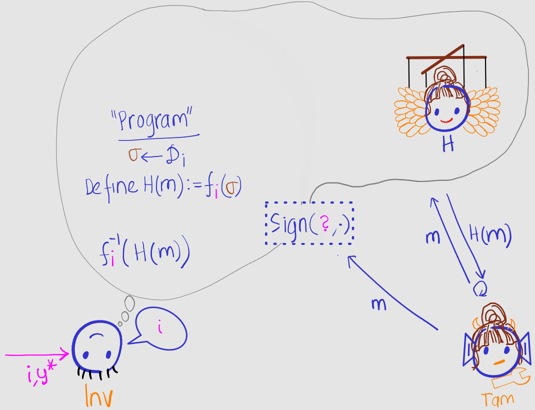


Let's Prove Security of "Hash-then-Invert"...

Theorem 2

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

Proof sketch: plug and pray via random oracle programming.

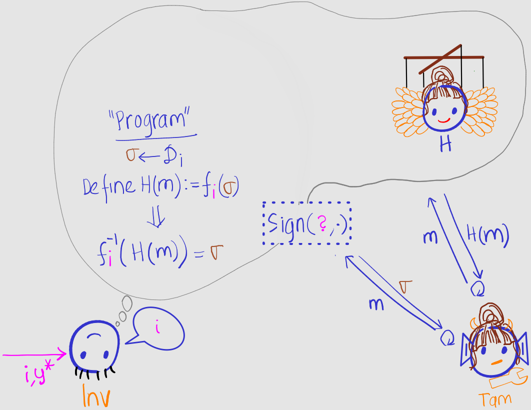


Let's Prove Security of "Hash-then-Invert"...

Theorem 2

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

Proof sketch: plug and pray via random oracle programming.

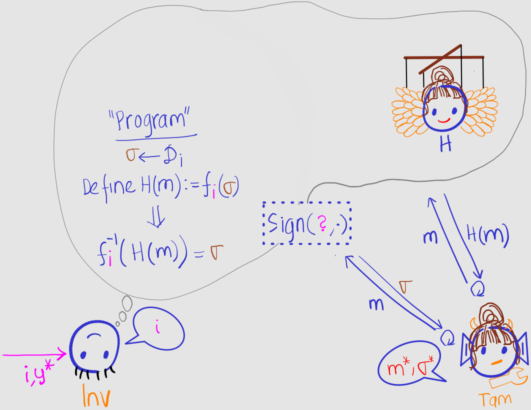


Let's Prove Security of "Hash-then-Invert"...

Theorem 2

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

Proof sketch: plug and pray via random oracle programming.

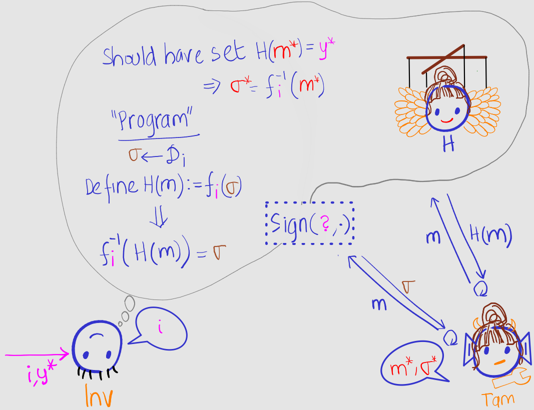


Let's Prove Security of "Hash-then-Invert"...

Theorem 2

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

Proof sketch: plug and pray via random oracle programming.

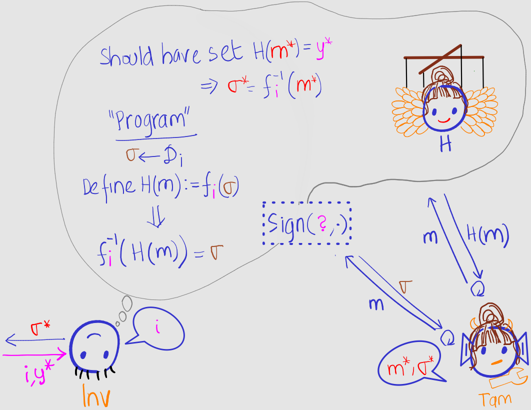


Let's Prove Security of "Hash-then-Invert"...

Theorem 2

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

Proof sketch: plug and pray via random oracle programming.

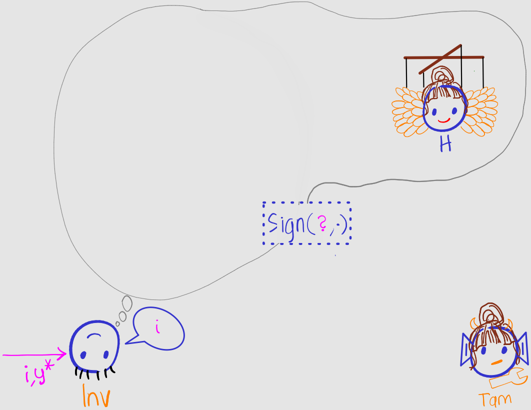


Let's Prove Security of "Hash-then-Invert"...

Theorem 2

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

Proof sketch: plug and pray via random oracle programming.



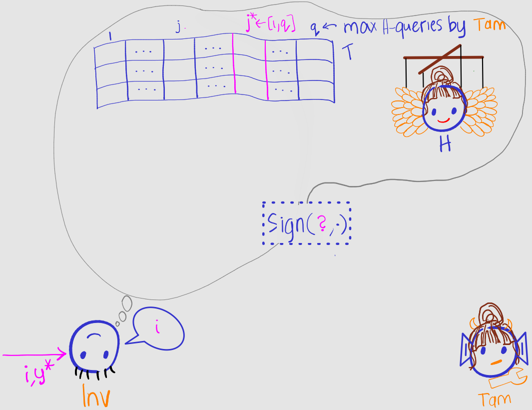
□

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

Theorem 2

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

Proof sketch: plug and pray via random oracle programming.



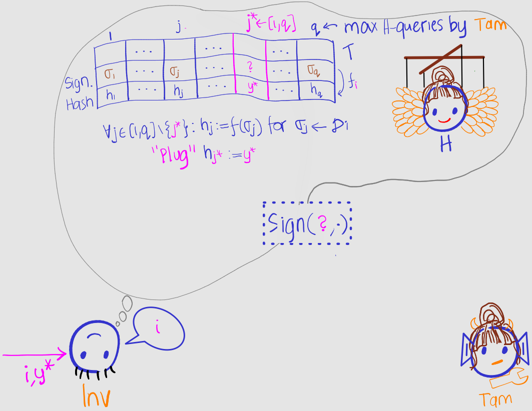
□

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

Theorem 2

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

Proof sketch: plug and pray via random oracle programming.

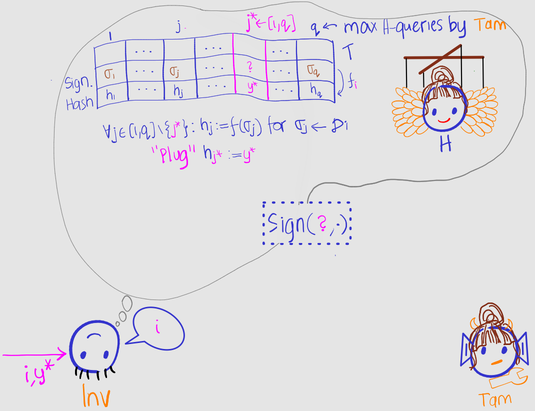


Let's Prove Security of "Hash-then-Invert"...

Theorem 2

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

Proof sketch: plug and pray via random oracle programming.



Reduction strategy

- Plug: embed challenge y^* as hash of Tam's j^*-th H-query

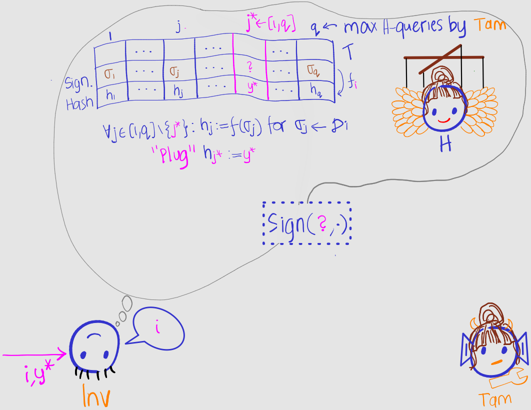
□

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

Theorem 2

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

Proof sketch: plug and pray via random oracle programming.



Reduction strategy

- Plug: embed challenge y^* as hash of Tam's j^*-th H-query
- Assume Tam always queries H(m) before Sign(?,m).

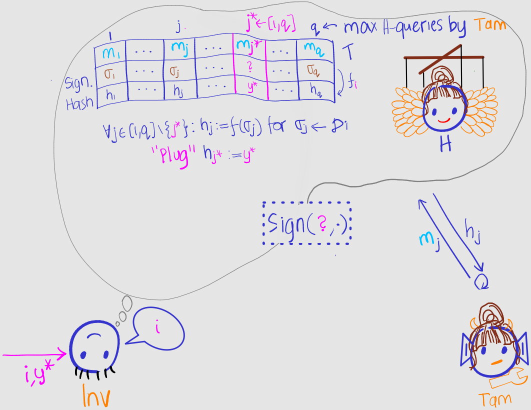
□

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

Theorem 2

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

Proof sketch: plug and pray via random oracle programming.



Reduction strategy

- Plug: embed challenge y^* as hash of Tam's j^*-th H-query
- Assume Tam always queries $H(m)$ before Sign(?,m).
- Answer j-th fresh H-query m_j with h_j

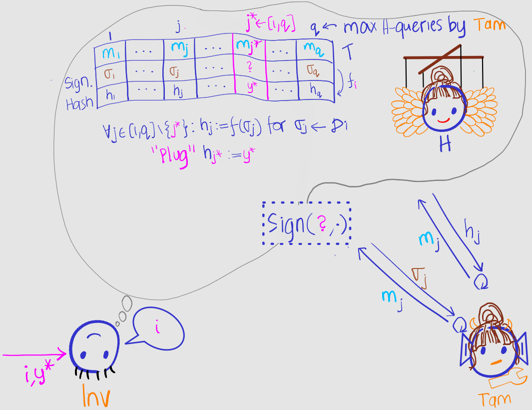
$\square$

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

Theorem 2

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

Proof sketch: plug and pray via random oracle programming.



Reduction strategy

- Plug: embed challenge y^* as hash of Tam's j^*-th H-query
- Assume Tam always queries $H(m)$ before Sign(?, m).
- Answer j-th fresh H-query m_j with h_j
- Pray 1: Answer Sign query for $m_j, j \neq j^*$ with σ_j. If $j = j^*$ ABORT,

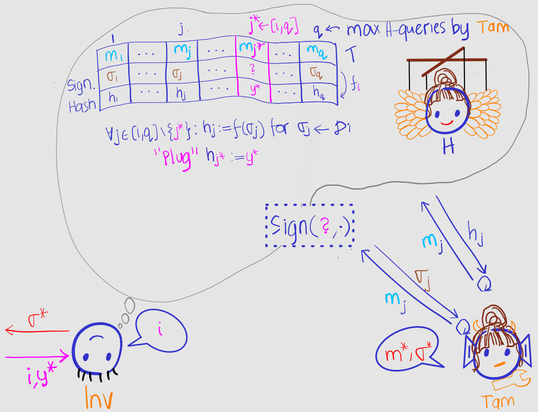
□

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

Theorem 2

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

Proof sketch: plug and pray via random oracle programming.



Reduction strategy

- Plug: embed challenge y^* as hash of Tam's j^*-th H-query
- Assume Tam always queries H(m) before Sign(?,m).
- Answer j-th fresh H-query m_j with h_j
- Pray 1: Answer Sign query for $m_j, j \neq j^*$ with σ_j. If $j = j^*$ ABORT$_1$
- Pray 2: If $m^* \neq m_{j^*}$ ABORT$_2$; else o/p σ^*

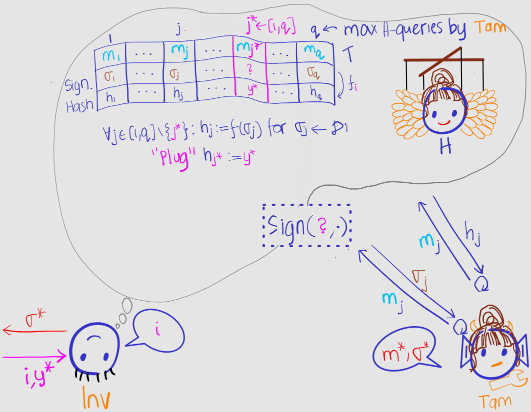
□

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

Theorem 2

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

Proof sketch: plug and pray via random oracle programming.



Reduction strategy

- Plug: embed challenge y^* as hash of Tam's j^*-th H-query
- Assume Tam always queries $H(m)$ before $\mathrm{Sign}(?, m)$.
- Answer j-th fresh H-query m_j with h_j
- Pray 1: Answer Sign query for $m_j, j \neq j^*$ with σ_j. If $j = j^*$ ABORT$_1$
- Pray 2: If $m^* \neq m_{j^*}$ ABORT$_2$; else o/p σ^*

Analysis: similar to Lamport's $\square$

1 Trap-Door (One-Way) Permutation (TDP)

2 Efficient Digital Signatures from TDP (in Random-Oracle Model)

3 Public-Key Encryption from TDP

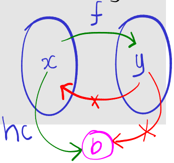
Let's Construct CPA-Secure PKE from TDP...

- Recall from Lecture 6:

Defintion 3 (Definition 2, Lecture 6)

A predicate $\mathsf{hc} : \{0,1\}^n \to \{0,1\}$ *is hard-core for a function family* $\mathsf{f}_n : \{0,1\}^n \to \{0,1\}^m$, *if for every PPT predictor* P, *the following is negligible*

$$\delta(n) := \Pr_{x \leftarrow \{0,1\}^n}[\mathsf{P}(\mathsf{f}(x)) = \mathsf{hc}(x)] - 1/2$$



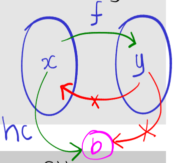
Let's Construct CPA-Secure PKE from TDP...

- Recall from Lecture 6:

Defintion 3 (Definition 2, Lecture 6)

A predicate $\mathsf{hc} : \{0,1\}^n \to \{0,1\}$ *is hard-core for a function family* $\mathsf{f}_n : \{0,1\}^n \to \{0,1\}^m$*, if for every PPT predictor* P*, the following is negligible*

$$\delta(n) := \Pr_{x \leftarrow \{0,1\}^n}[\mathsf{P}(\mathsf{f}(x)) = \mathsf{hc}(x)] - 1/2$$



Theorem 3 (Goldreich-Levin Theorem (Theorem 3, Lecture 6))

For a OWP f*, let* $\mathsf{f}'(x,r) := (\mathsf{f}(x), r)$*. Then* $\mathsf{hc}(x,r) := \langle x, r \rangle_2$ *is a hard-core predicate for* f'*.*

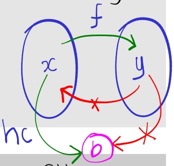
Let's Construct CPA-Secure PKE from TDP...

- Recall from Lecture 6:

Defintion 3 (Definition 2, Lecture 6)

A predicate $\mathsf{hc} : \{0,1\}^n \to \{0,1\}$ *is hard-core for a function family* $\mathsf{f}_n : \{0,1\}^n \to \{0,1\}^m$, *if for every PPT predictor* P, *the following is negligible*

$$\delta(n) := \Pr_{x \leftarrow \{0,1\}^n}[\mathsf{P}(\mathsf{f}(x)) = \mathsf{hc}(x)] - 1/2$$



Theorem 3 (Goldreich-Levin Theorem (Theorem 3, Lecture 6))

For a OWP f, *let* $\mathsf{f}'(x, r) := (\mathsf{f}(x), r)$. *Then* $\mathsf{hc}(x, r) := \langle x, r \rangle_2$ *is a hard-core predicate for* f'.

Exercise 2

Extend Goldreich-Levin theorem for TDP $\mathsf{f} = \{\mathsf{f}_i : \mathcal{D}_i \to \mathcal{D}_i\}_{i \in \mathcal{I}}$

Let's Construct CPA-Secure PKE from TDP...

- Let $\mathsf{f} = \{\mathsf{f}_i : \mathcal{D}_i \to \mathcal{D}_i\}_{i \in \mathcal{I}}$ be a TDP and hc be a HCP for f

? How do you construct PKE?

Let's Construct CPA-Secure PKE from TDP...

- Let $\mathsf{f} = \{\mathsf{f}_i : \mathcal{D}_i \to \mathcal{D}_i\}_{i \in \mathcal{I}}$ be a TDP and hc be a HCP for f

? How do you construct PKE?

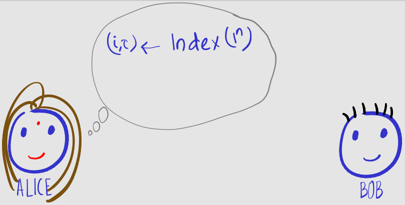
Construction 1 (PKE $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}) \leftarrow$ TDP $\mathsf{f}_i : \mathcal{D}_i \to \mathcal{D}_i$)

Let's Construct CPA-Secure PKE from TDP...

- Let $f = \{f_i : \mathcal{D}_i \to \mathcal{D}_i\}_{i \in \mathcal{I}}$ be a TDP and hc be a HCP for f

How do you construct PKE?

Construction 1 (PKE $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}) \leftarrow$ TDP $f_i : \mathcal{D}_i \to \mathcal{D}_i$)

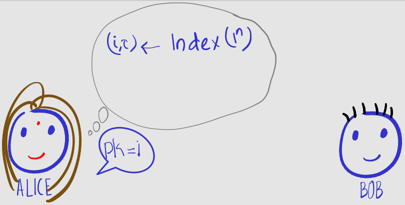


Let's Construct CPA-Secure PKE from TDP...

- Let $f = \{f_i : \mathcal{D}_i \to \mathcal{D}_i\}_{i \in \mathcal{I}}$ be a TDP and hc be a HCP for f

? How do you construct PKE?

Construction 1 (PKE $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}) \leftarrow$ TDP $f_i : \mathcal{D}_i \to \mathcal{D}_i$)

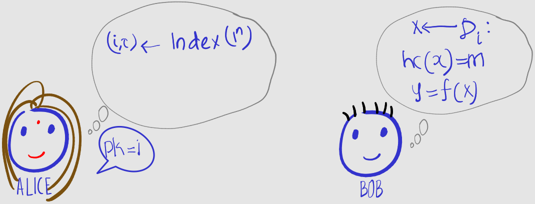


Let's Construct CPA-Secure PKE from TDP...

- Let $f = \{f_i : \mathcal{D}_i \to \mathcal{D}_i\}_{i \in \mathcal{I}}$ be a TDP and hc be a HCP for f

? How do you construct PKE?

Construction 1 (PKE $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}) \leftarrow$ TDP $f_i : \mathcal{D}_i \to \mathcal{D}_i$)

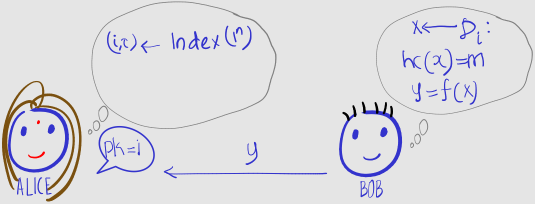


Let's Construct CPA-Secure PKE from TDP...

- Let $f = \{f_i : \mathcal{D}_i \to \mathcal{D}_i\}_{i \in \mathcal{I}}$ be a TDP and hc be a HCP for f

? How do you construct PKE?

Construction 1 (PKE $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}) \leftarrow$ TDP $f_i : \mathcal{D}_i \to \mathcal{D}_i$)

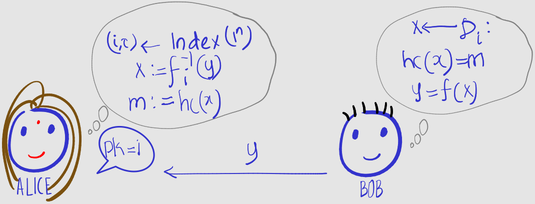


Let's Construct CPA-Secure PKE from TDP...

- Let $f = \{f_i : \mathcal{D}_i \to \mathcal{D}_i\}_{i \in \mathcal{I}}$ be a TDP and hc be a HCP for f

? How do you construct PKE?

Construction 1 (PKE $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}) \leftarrow$ TDP $f_i : \mathcal{D}_i \to \mathcal{D}_i$)

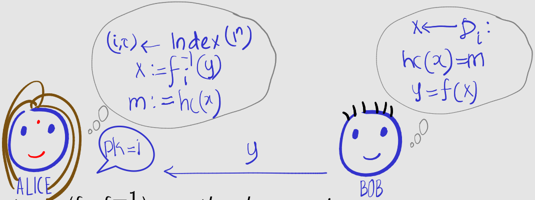


Let's Construct CPA-Secure PKE from TDP...

- Let $\mathsf{f} = \{\mathsf{f}_i : \mathcal{D}_i \to \mathcal{D}_i\}_{i \in \mathcal{I}}$ be a TDP and hc be a HCP for f

? How do you construct PKE?

Construction 1 (PKE $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}) \leftarrow$ TDP $\mathsf{f}_i : \mathcal{D}_i \to \mathcal{D}_i$)



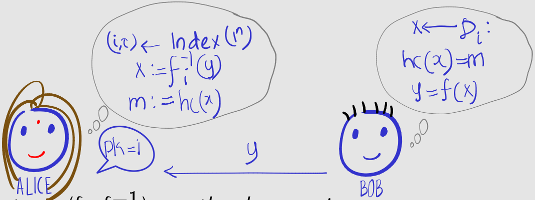
- *Alice: set up* $(\mathsf{f}_i, \mathsf{f}_i^{-1})$ *as the key pair*
- *Bob: to encrypt a bit* m*, sample* $x \leftarrow \mathcal{D}_i$ *such that* $\mathsf{hc}(x) = m$ *and send* $y = \mathsf{f}_i'(x)$ *as "hint"*
- *Alice: to decrypt, compute* $x := \mathsf{f}_i^{-1}(y)$ *and output* $\mathsf{hc}(x)$

Let's Construct CPA-Secure PKE from TDP...

- Let $\mathsf{f} = \{\mathsf{f}_i : \mathcal{D}_i \to \mathcal{D}_i\}_{i \in \mathcal{I}}$ be a TDP and hc be a HCP for f

? How do you construct PKE?

Construction 1 (PKE $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}) \leftarrow$ TDP $\mathsf{f}_i : \mathcal{D}_i \to \mathcal{D}_i$)



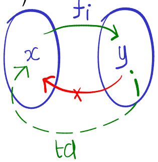
- *Alice: set up* $(\mathsf{f}_i, \mathsf{f}_i^{-1})$ *as the key pair*
- *Bob: to encrypt a bit* m, *sample* $x \leftarrow \mathcal{D}_i$ *such that* $\mathsf{hc}(x) = m$ *and send* $y = \mathsf{f}'_i(x)$ *as "hint"*
- *Alice: to decrypt, compute* $x := \mathsf{f}_i^{-1}(y)$ *and output* $\mathsf{hc}(x)$

Theorem 4

If f' *is a TDP then* Π *is IND-CPA secure.*

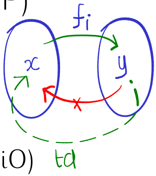
To Recap Today's Lecture

- Introduced a new primitive: trap-door permutation (TDP)
  - Motivation: efficient signature schemes



To Recap Today's Lecture

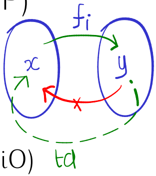
- Introduced a new primitive: trap-door permutation (TDP)
  - Motivation: efficient signature schemes



- Two candidate TDPs: RSA and Rabin TDP
  - Not many candidates known (Paillier TDP, TDP from iO)

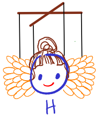
To Recap Today's Lecture

- Introduced a new primitive: trap-door permutation (TDP)
  - Motivation: efficient signature schemes



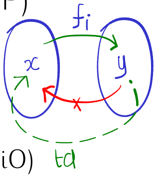
- Two candidate TDPs: RSA and Rabin TDP
  - Not many candidates known (Paillier TDP, TDP from iO)

- Efficient signature via "hash-then-invert" paradigm
  - Proof in random-oracle model: plug and pray + programming
  - RSA-PKCS#1 standard based on RSA-FDH



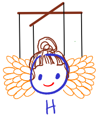
To Recap Today's Lecture

- Introduced a new primitive: trap-door permutation (TDP)
  - Motivation: efficient signature schemes



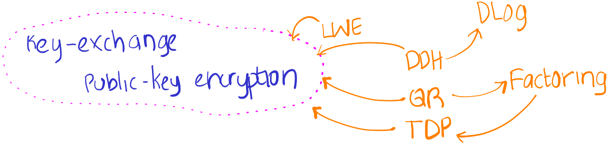
- Two candidate TDPs: RSA and Rabin TDP
  - Not many candidates known (Paillier TDP, TDP from iO)

- Efficient signature via "hash-then-invert" paradigm
  - Proof in random-oracle model: plug and pray + programming
  - RSA-PKCS#1 standard based on RSA-FDH



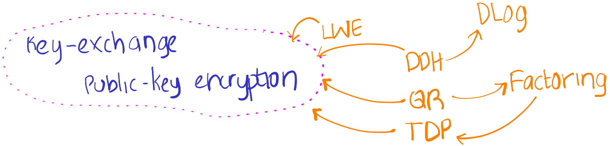
- PKE from TDP:
  - New PKE based on RSA assumption
    - Not same as (textbook) RSA encryption

- We learnt: secure communication in the public-key setting
- Cryptographic primitives encountered: key-exchange, public-key encryption, signature, hash function, TDP
- Hardness assumptions: Factoring, DLog, QR, LWE, RSA

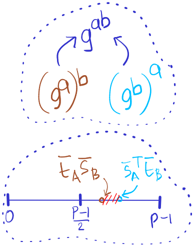


To Recap This Module

- We learnt: secure communication in the public-key setting
- Cryptographic primitives encountered: key-exchange, public-key encryption, signature, hash function, TDP
- Hardness assumptions: Factoring, DLog, QR, LWE, RSA

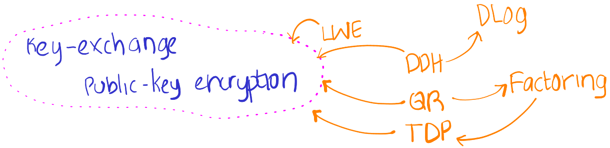


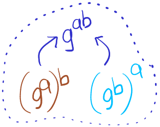
- Key conceptual takeaway: structure vs. hardness

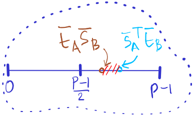


To Recap This Module

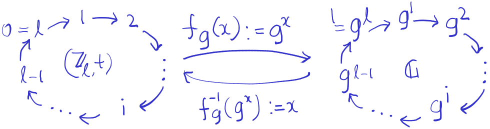
- We learnt: secure communication in the public-key setting
- Cryptographic primitives encountered: key-exchange, public-key encryption, signature, hash function, TDP
- Hardness assumptions: Factoring, DLog, QR, LWE, RSA

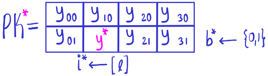






- Key conceptual takeaway: structure vs. hardness





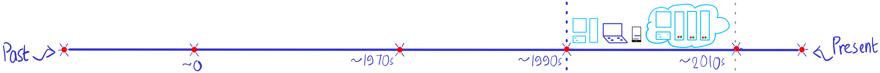
- Key tools: groups, random self-reducibility, plug and pray

MODULE 3
(Secure comp.)

- Zero-knowledge proofs
- Private computation of functions (MPC)
- Private outsourcing: fully-homomorphic encryption
- Verifiable outsourcing: SNARGs

Ubiquity of computing

**



References

1 [KL14, §13.3 and §15.1] for details of this lecture.
2 Trap-door permutations were introduced in [DH76]. Yao [Yao82] who showed how to construct PKE using TDPs.
3 The random oracle model was proposed in [FS87]. But it was in [BR93] that it was shown how it can be fully exploited. E.g., the random-oracle-based of "hash-then-invert" construction is from there.

Mihir Bellare and Phillip Rogaway.
Random oracles are practical: A paradigm for designing efficient protocols.
In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93*, pages 62–73. ACM Press, November 1993.

Whitfield Diffie and Martin E. Hellman.
New directions in cryptography.
IEEE Trans. Inf. Theory, 22(6):644–654, 1976.

Amos Fiat and Adi Shamir.
How to prove yourself: Practical solutions to identification and signature problems.
In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987.

Jonathan Katz and Yehuda Lindell.
Introduction to Modern Cryptography (3rd ed.).
Chapman and Hall/CRC, 2014.

Andrew Chi-Chih Yao.
Theory and applications of trapdoor functions (extended abstract).
In *23rd FOCS*, pages 80–91. IEEE Computer Society Press, November 1982.