Trapdoor Hash Functions and Their Applications

Nico Döttling
CISPA Helmholtz Center

Sanjam Garg
UC Berkeley

Yuval Ishai
Technion

Giulio Malavolta
Carnegie Mellon University

Tamer Mour
Weizmann Institute

Rafail Ostrovsky
UC Los Angeles

TPMPC 2019, Bar Ilan University

Trapdoor Hash Functions and Their Applications

Nico Döttling
CISPA Helmholtz Center

Sanjam Garg
UC Berkeley

Yuval Ishai
Technion

Giulio Malavolta
Carnegie Mellon University

Tamer Mour
Weizmann Institute

Rafail Ostrovsky
UC Los Angeles

TPMPC 2019, Bar Ilan University

Setting: Sender-Receiver Computation

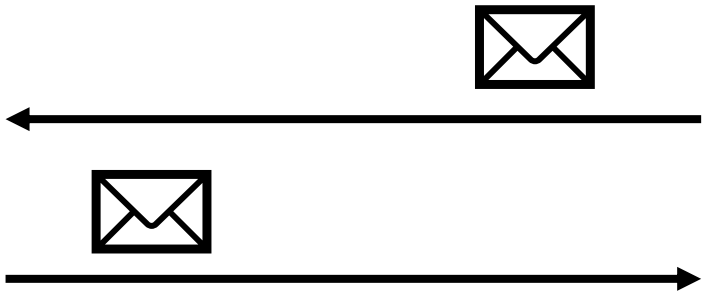
$f: X \times Y \to Z$

Sender

Receiver

$y \in Y$

$x \in X$



communication

Output:

$f(x, y)$

Setting: Sender-Receiver Computation

Focus: **two-message** protocols with **minimum communication**.

Sender

Input: $y \in Y$

receiver message

sender message

Receiver

Input: $x \in X$

Output: $f(x, y)$

Ideal World:

Simple and **optimal** solutions.

Real World (semi-honest security):

Sender and receiver do not trust each other, and want to keep their inputs **private**.

Main question in this work:

Can Secure Protocols be as Efficient as Ideal-World Solutions?

Can Secure Protocols be as Efficient as Ideal-World Solutions?

Setting I.

Sender input is larger than receiver input

$$|y| \gg |x|$$

Example:

String Oblivious Transfer (OT)

$y_0, y_1 \in \{0,1\}^n$ ⟵ $x \in \{0,1\}$

⟶ Output: y_x

In Ideal World, communication dominated by length of **second message** $|f(x,y)| = n$.

Goal

Optimize length of **second message**, i.e.

$$\textbf{download rate} = \frac{|f(x,y)|}{|Second\ Message|}$$

Ideal-World Solution:

download rate = 1

Negative Answer

Secure protocols with **exact** rate 1 **do not exist.**
(even with correlated randomness)

For 2^{λ} security,

$$|second\ message| > n + 2\lambda$$

Best we can hope for:

$$\textbf{download rate} \to 1$$

when $n \to \infty$.

Positive Answer

Protocols with rate approaching 1 using **Trapdoor Hash**.

Results in Setting I: Rate-1 Oblivious Transfer and More

Life before Trapdoor Hash

- Rate - ½ OT is easy to achieve.

- Only known solution for higher rate uses **high-rate homomorphic encryption**.
- Only rate-1 homomorphic encryption scheme: **Damgård-Jurik** scheme based on **DCR**.

Exception: [Gentry-Halevi'19]

	DCR	LWE	QR	DDH
Oblivious Transfer (OT)	**rate-1 [DJ00]**	**rate-½**	**rate-½**	**rate-½**

Life after Trapdoor Hash

	DCR	LWE	QR	DDH
Oblivious Transfer (OT)	**rate-1**	**rate-1**	**rate-1**	**rate-1**
Oblivious Linear-Function Evaluation (OLE)	**rate-1**	**rate-1**	**rate-1**	**rate-1**
Oblivious Matrix-Vector Product (OMV)	**-**	**rate-1**	**rate-1**	**-**

Results in Setting I: Application of Rate-1 OT

1. Single-Server Private Information Retrieval [Kushilevitz-Ostrovsky'97]
First **rate-1** PIR protocols with **polylogarithmic communication** [Ishai-Paskin'07].

	DCR	LWE	QR	DDH
Before TDH	$O(\log^c n)$, **rate-1**	$O(\log^c n)$	$O(2^{\sqrt{\log n}})$ **[KO97]**	$O(2^{\sqrt{\log n}})$ **[KO97]**
After TDH	$O(\log^c n)$, **rate-1**	$O(\log^c n)$, **rate-1**	$O(\log^c n)$, **rate-1**	$O(\log^c n)$, **rate-1**

2. Homomorphic Encryption for Branching Programs
First **semi-compact** homomorphic encryption from **DDH,QR** [Ishai-Paskin'07].

3. Lossy Trapdoor Functions [Peikert-Waters'07]
First **rate-1** constructions.

	DCR	LWE	QR	DDH
Before TDH	**rate-1**	-	-	**constant rate [GGH18]**
After TDH	**rate-1**	**rate-1**	**rate-1**	**rate-1**

Can Secure Protocols be as Efficient as Ideal-World Solutions?

Setting I.

Sender input is larger than receiver input

$$|y| \gg |x|$$

Example:

String Oblivious Transfer (OT)

In Ideal World, communication dominated by length of **second message** $|f(x,y)| = n$.

Goal

Optimize length of **second message**, i.e.

$$\textbf{download rate} = \frac{|f(x,y)|}{|Second\ Message|}$$

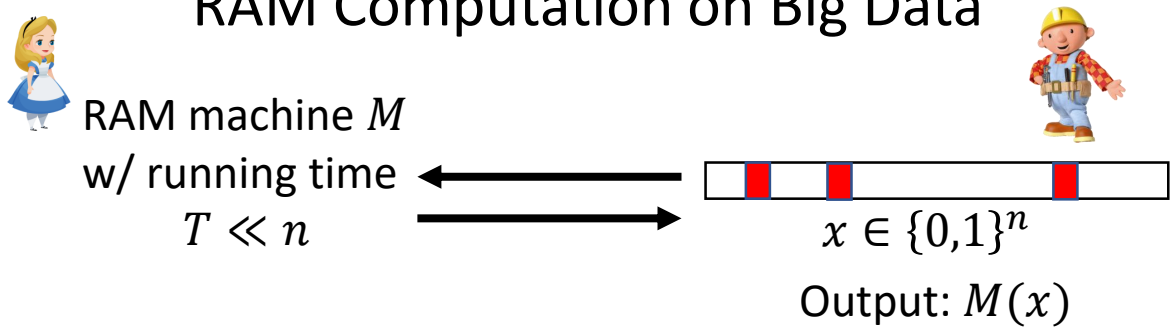
Setting II.

Receiver input is larger than sender input

$$|y| \ll |x|$$

Example:

RAM Computation on Big Data



(DNA analysis, suspects lookup, etc.)

Can Secure Protocols be as Efficient as Ideal-World Solutions?

Setting I.

Sender input is larger than receiver input

$$|y| \gg |x|$$

Example:

String Oblivious Transfer (OT)

In Ideal World, communication dominated by length of **second message** $|f(x,y)| = n$.

Goal

Optimize length of **second message**, i.e.

$$\textbf{download rate} = \frac{|f(x,y)|}{|Second\ Message|}$$

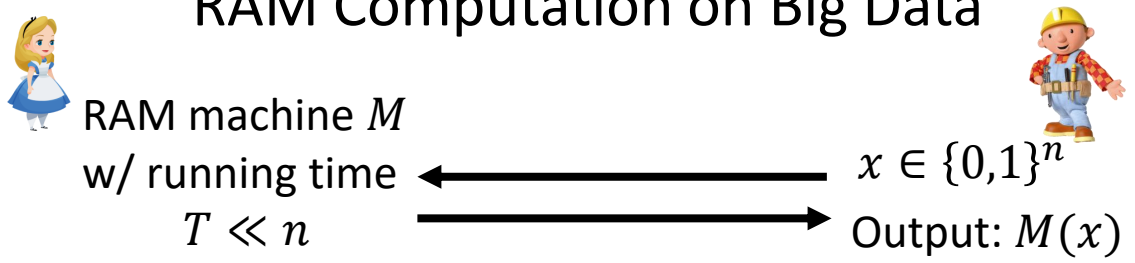
Setting II.

Receiver input is larger than sender input

$$|y| \ll |x|$$

Example:

RAM Computation on Big Data



In Ideal World, communication is $|y| \ll n$:

"much smaller" than n

Goal

Communication **smaller than $|x| = n$.**

Non-trivial for **two-message** protocols.

Results in Setting II: Sublinear Secure RAM Computation

Life before Trapdoor Hash

Only **fully secure** solution is based on **lattice assumptions**.

	Overall Com.	Assumption	Security
Laconic Function Evaluation [QWW18]	$\tilde{O}(T)$	**LWE**	**full**
Laconic OT [CDGGMP17]	$\tilde{O}(T)$	**DDH**	**UMA**

Life after Trapdoor Hash

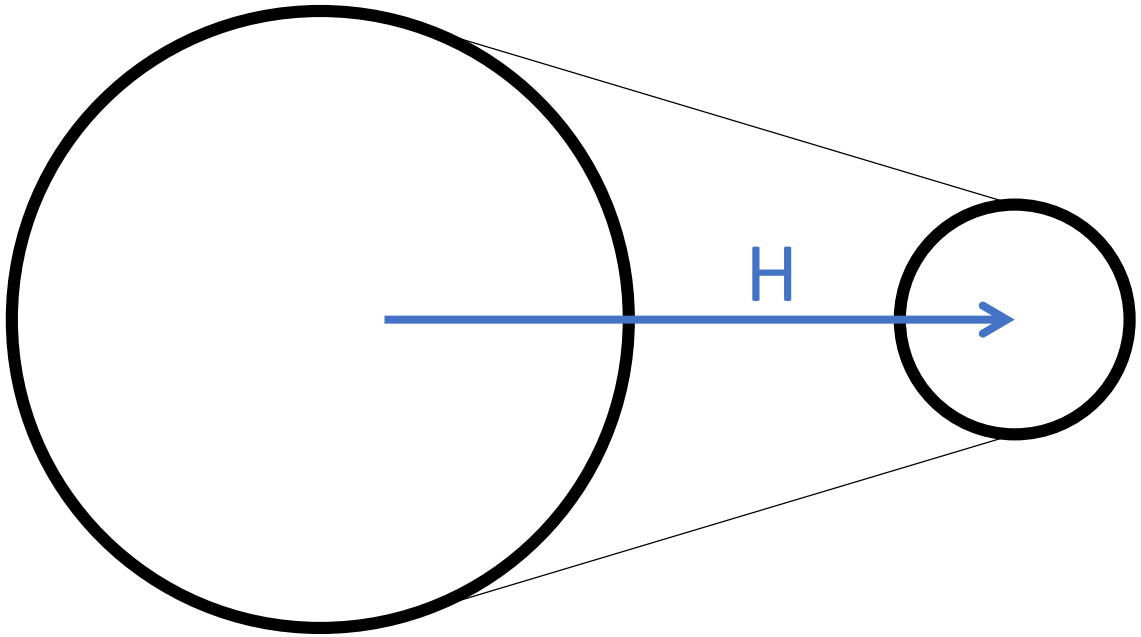
First sublinear two-message protocols under **number-theoretic assumptions**.

Private Laconic OT (through TDH)	$O(T \cdot \sqrt{n})$	**DDH**	**full**
	$O(T \cdot \sqrt[3]{n})$	**SXDH+**	**full**

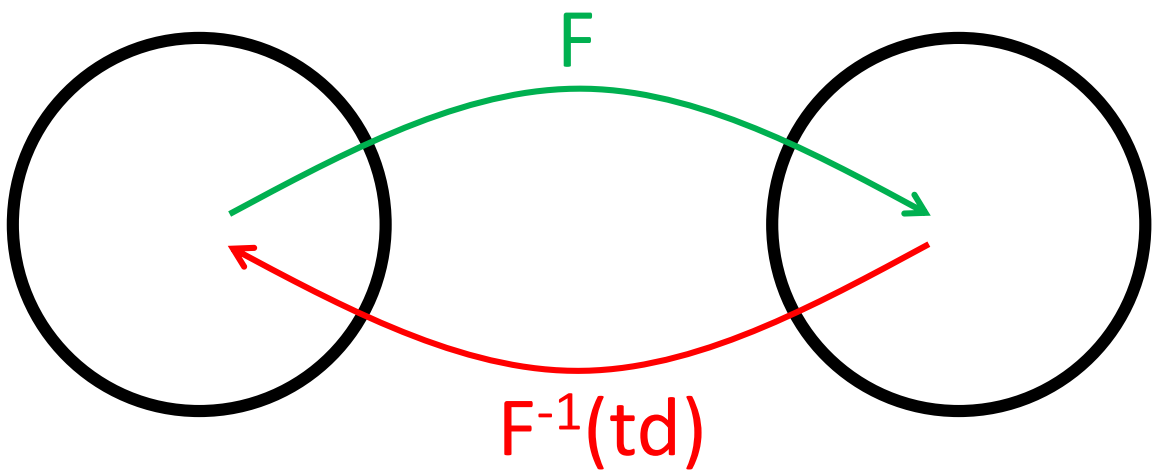
Open Question (already): can we close the efficiency gap?

Trapdoor Hash Functions

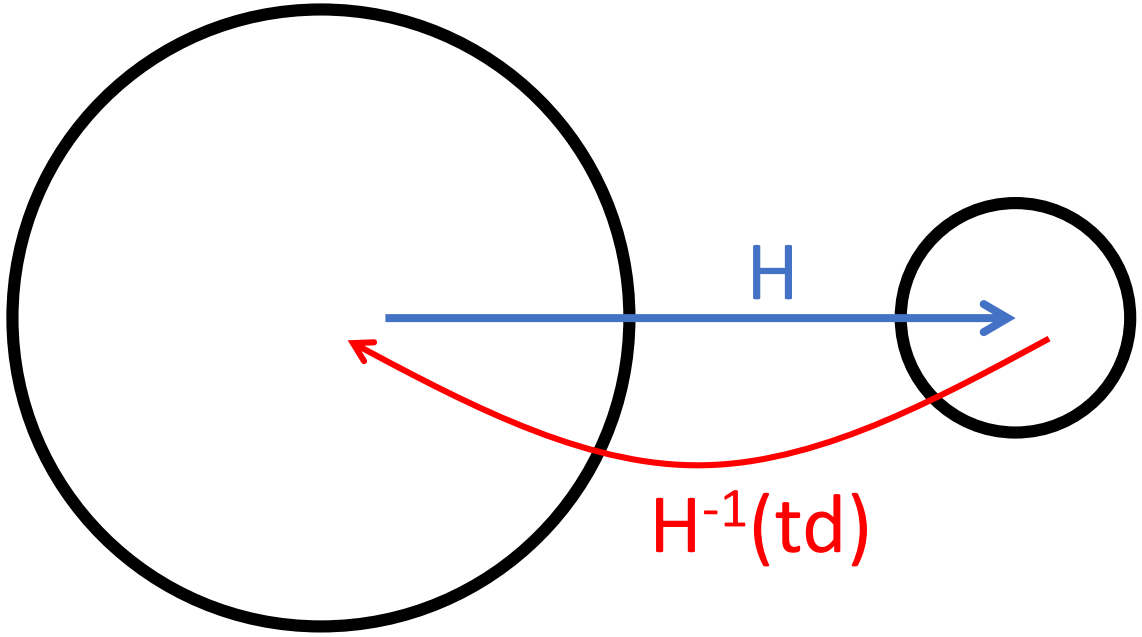
Hash Functions



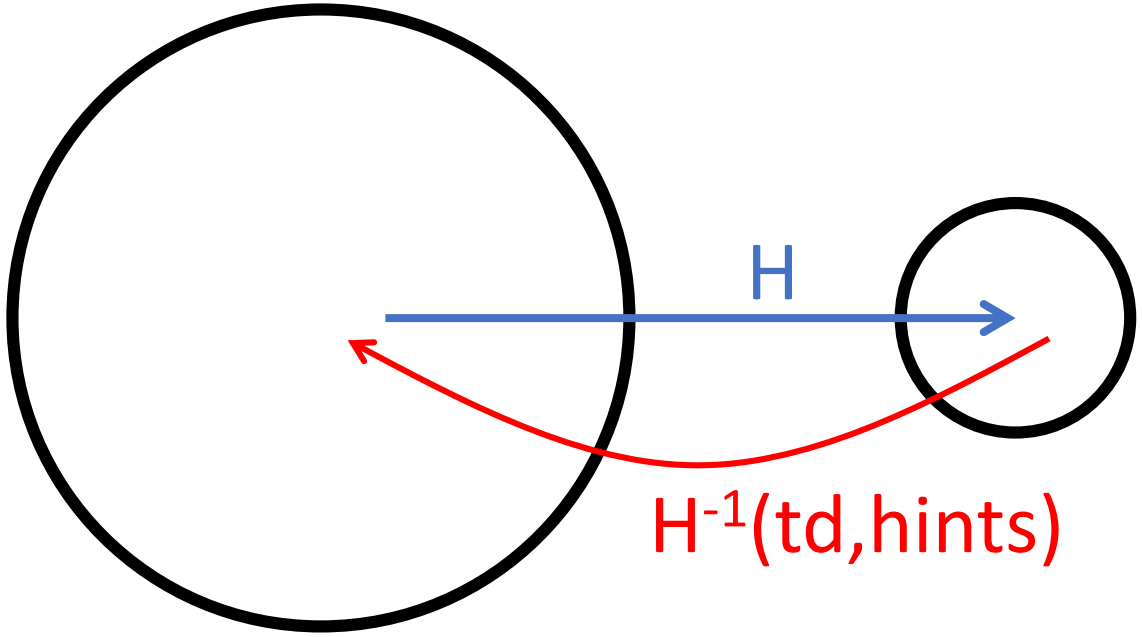
Trapdoor Functions



Trapdoor Hash Functions



Trapdoor Hash Functions



Defining Trapdoor Hash

Alice

Bob

The Hash Function:

Input Privacy: h hides x.

h

h

"I want to learn $x[i]$, give me some hints"

key

Key Generation:

(key, trapdoor)← **G**(i)

Index Privacy: key hides i.

The "Hinting" Function:

e

Decoding:

$x[i] \leftarrow$ **D**(trapdoor, h, e)

Efficiency: small hints, i.e. **high rate**

Defining Trapdoor Hash

Alice

Bob

The Hash Function:

Input Privacy: h hides x.

h

h

"I want to learn $x[i_1, \ldots, i_m]$, give me some hints"

$\text{key}_1, \ldots, \text{key}_m$

Key Generation:

$(\text{key}_j, \text{trapdoor}_j) \leftarrow \mathbf{G}(i_j)$

Index Privacy: key hides i.

The "Hinting" Function:

$e_1, \ldots, e_m$

Decoding:

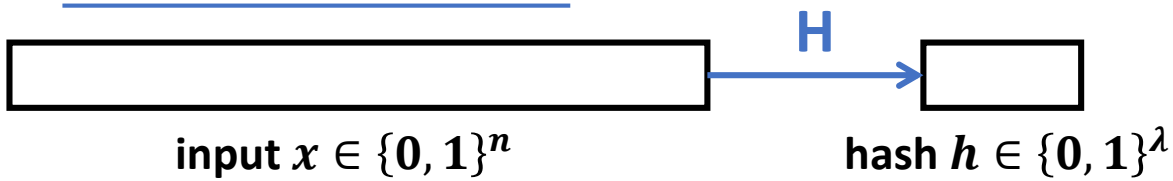
$x[i_j] \leftarrow \mathbf{D}(\text{trapdoor}_j, h, e_j)$

Efficiency: small hints, i.e. **high rate**

Defining Trapdoor Hash

TDH = (H,G,E,D)

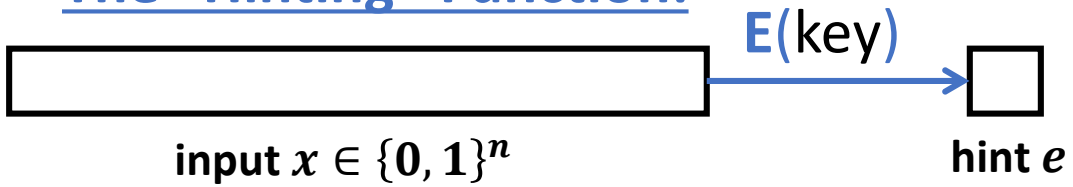
The Hash Function:



Key Generation:

(key, trapdoor)← **G**(i)

The "Hinting" Function:



Decoding:

$x[i] \leftarrow$ **D**(trapdoor, h, e)

Input Privacy: h hides x.

Index Privacy: key hides i.

Rate:

$$\frac{\mathbf{1}}{|\boldsymbol{e}|}$$

Optimally, **rate = 1**.

Main technical contribution:

Rate-1 TDH from **DDH,QR,LWE,DCR**

We also consider TDH for general functions of x.

Trapdoor Hash from DDH

Similar technique used to construct **IBE [DG17], laconic OT [CDGGMP17]** and **Trapdoor Functions [GH18,GGH19]**.

- **Multiplicative abelian group $\mathbb{G}$ of prime order p, with a public generator $g \in \mathbb{G}$.**
  For all $g_1, g_2 \in \mathbb{G}$,

$$\boldsymbol{g_1 \cdot g_2 = g_2 \cdot g_1 \in \mathbb{G}}$$

- **The DDH (Decisional Diffie-Hellman) assumption**
  For **uniform** $\mathrm{a}, \mathrm{b}, \mathrm{c} \in \mathbb{Z}_p$ and an element $g \in \mathbb{G}$,

$$\left(\boldsymbol{g^a, g^b, g^{ab}}\right) \equiv \left(\boldsymbol{g^a, g^b, g^c}\right)$$

Trapdoor Hash from DDH

Alice

public parameters

$2 \times n$ uniform group elements

$$\begin{pmatrix} g_{1,0} & g_{2,0} & g_{3,0} & \cdots & g_{n,0} \\ g_{1,1} & g_{2,1} & g_{3,1} & \cdots & g_{n,1} \end{pmatrix} \in \mathbb{G}^{2\times n}$$

Bob

input $\boldsymbol{x} \in \{\mathbf{0}, \mathbf{1}\}^{\boldsymbol{n}}$

Trapdoor Hash from DDH

Alice

public parameters

$$\begin{pmatrix} g_{1,0} & g_{2,0} & g_{3,0} & \cdots & g_{n,0} \\ g_{1,1} & g_{2,1} & g_{3,1} & \cdots & g_{n,1} \end{pmatrix}$$

Bob

Hash Function:

$$H(x) = \prod_j g_{j,x[j]}$$

$h = H(x)$ →

$$\begin{pmatrix} g_{1,0} & \boldsymbol{g_{2,0}} & \boldsymbol{g_{3,0}} & \cdots & g_{n,0} \\ \boldsymbol{g_{1,1}} & g_{2,1} & g_{3,1} & \cdots & \boldsymbol{g_{n,1}} \end{pmatrix}$$

1	0	0	⋯	1

H → $h \in \mathbb{G}$

input $\boldsymbol{x \in \{0,1\}^n}$

Input Privacy: statistical*.

Trapdoor Hash from DDH

Alice

public parameters

$$\begin{pmatrix} g_{1,0} & g_{2,0} & g_{3,0} & \cdots & g_{n,0} \\ g_{1,1} & g_{2,1} & g_{3,1} & \cdots & g_{n,1} \end{pmatrix}$$

Bob

Hash Function:

$$H(x) = \prod_j g_{j,x[j]}$$

$$h = H(x) \longrightarrow$$

Key Generation:

"I want to learn $x[i]$"

$$key = \begin{pmatrix} \tilde{g}_{1,0} & \tilde{g}_{2,0} & \tilde{g}_{3,0} & \cdots & \tilde{g}_{n,0} \\ \tilde{g}_{1,1} & \tilde{g}_{2,1} & \tilde{g}_{3,1} & \cdots & \tilde{g}_{n,1} \end{pmatrix} \longleftarrow$$

trapdoor: uniform $t \in \mathbb{Z}_p$

$$key = \begin{pmatrix} g_{1,0}^t & \cdots & g_{i,0}^t & \cdots & g_{n,0}^t \\ g_{1,1}^t & \cdots & g_{i,1}^t \cdot g & \cdots & g_{n,1}^t \end{pmatrix}$$

Index Privacy: assuming DDH,

$$\begin{pmatrix} \tilde{g}_{1,0} & \tilde{g}_{2,0} & \tilde{g}_{3,0} & \cdots & \tilde{g}_{n,0} \\ \tilde{g}_{1,1} & \tilde{g}_{2,1} & \tilde{g}_{3,1} & \cdots & \tilde{g}_{n,1} \end{pmatrix} \equiv \text{uniform matrix in } \mathbb{G}^{2\times n}$$

Trapdoor Hash from DDH

Alice

public parameters

$$\begin{pmatrix} g_{1,0} & g_{2,0} & g_{3,0} & \cdots & g_{n,0} \\ g_{1,1} & g_{2,1} & g_{3,1} & \cdots & g_{n,1} \end{pmatrix}$$

Bob

Hash Function:

$$H(x) = \prod_j g_{j,x[j]}$$

$$h = H(x)$$

Key Generation:

"I want to learn $x[i]$"

$$key = \begin{pmatrix} \tilde{g}_{1,0} & \tilde{g}_{2,0} & \tilde{g}_{3,0} & \cdots & \tilde{g}_{n,0} \\ \tilde{g}_{1,1} & \tilde{g}_{2,1} & \tilde{g}_{3,1} & \cdots & \tilde{g}_{n,1} \end{pmatrix}$$

trapdoor: uniform $t \in \mathbb{Z}_p$

$$key = \begin{pmatrix} g_{1,0}^t & \cdots & g_{i,0}^t & \cdots & g_{n,0}^t \\ g_{1,1}^t & \cdots & g_{i,1}^t \cdot g & \cdots & g_{n,1}^t \end{pmatrix}$$

Hinting:

$$E(key, x) = \prod_j \tilde{g}_{j,x[j]}$$

$$\begin{pmatrix} \tilde{g}_{1,0} & \tilde{g}_{2,0} & \tilde{g}_{3,0} & \cdots & \tilde{g}_{n,0} \\ \tilde{g}_{1,1} & \tilde{g}_{2,1} & \tilde{g}_{3,1} & \cdots & \tilde{g}_{n,1} \end{pmatrix}$$

1	0	0	⋯	1

input $x \in \{0,1\}^n$

E

$$e = E(key, x)$$

$$e \in \mathbb{G}$$

Rate:

$$\frac{1}{|e|} = \frac{1}{\lambda}$$

Trapdoor Hash from DDH

Alice

public parameters

$$\begin{pmatrix} g_{1,0} & g_{2,0} & g_{3,0} & \cdots & g_{n,0} \\ g_{1,1} & g_{2,1} & g_{3,1} & \cdots & g_{n,1} \end{pmatrix}$$

Bob

Hash Function:

$$H(x) = \prod_j g_{j,x[j]}$$

$$h = H(x)$$

Key Generation:

"I want to learn $x[i]$"

$$key = \begin{pmatrix} \tilde{g}_{1,0} & \tilde{g}_{2,0} & \tilde{g}_{3,0} & \cdots & \tilde{g}_{n,0} \\ \tilde{g}_{1,1} & \tilde{g}_{2,1} & \tilde{g}_{3,1} & \cdots & \tilde{g}_{n,1} \end{pmatrix}$$

trapdoor: uniform $t \in \mathbb{Z}_p$

$$key = \begin{pmatrix} g_{1,0}^t & \cdots & g_{i,0}^t & \cdots & g_{n,0}^t \\ g_{1,1}^t & \cdots & g_{i,1}^t \cdot g & \cdots & g_{n,1}^t \end{pmatrix}$$

Hinting:

$$E(key, x) = \prod_j \tilde{g}_{j,x[j]}$$

$$\begin{pmatrix} \tilde{g}_{1,0} & \tilde{g}_{2,0} & \tilde{g}_{3,0} & \cdots & \tilde{g}_{n,0} \\ \tilde{g}_{1,1} & \tilde{g}_{2,1} & \tilde{g}_{3,1} & \cdots & \tilde{g}_{n,1} \end{pmatrix}$$

1	0	0	⋯	1

input $x \in \{0,1\}^n$

E

$$e = E(key, x)$$

$$e \in \mathbb{G}$$

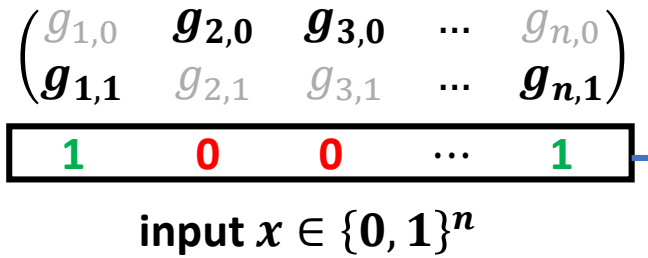
How to recover $x[i]$ given

- The **hash** h
- The **hint** e
- The **trapdoor** t ?

Trapdoor Hash at The Bar: Decoding

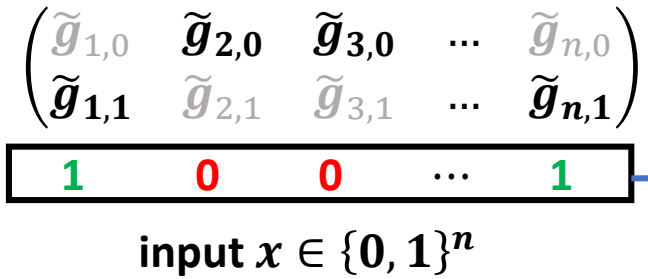
Observation: given that

$$\begin{pmatrix} \tilde{g}_{1,0} & \tilde{g}_{2,0} & \tilde{g}_{3,0} & \cdots & \tilde{g}_{n,0} \\ \tilde{g}_{1,1} & \tilde{g}_{2,1} & \tilde{g}_{3,1} & \cdots & \tilde{g}_{n,1} \end{pmatrix} = \begin{pmatrix} g_{1,0}^t & \cdots & g_{i,0}^t & \cdots & g_{n,0}^t \\ g_{1,1}^t & \cdots & g_{i,1}^t \cdot g & \cdots & g_{n,1}^t \end{pmatrix}$$



H

$$h = \prod_j g_{j,x[j]}$$



E

$$e = \prod_j \tilde{g}_{j,x[j]} = \prod_j g_{j,x[j]}^t \cdot g^{x[i]} = h^t \cdot g^{x[i]}$$

Trapdoor Hash from DDH

Alice

public parameters

$$\begin{pmatrix} g_{1,0} & g_{2,0} & g_{3,0} & \cdots & g_{n,0} \\ g_{1,1} & g_{2,1} & g_{3,1} & \cdots & g_{n,1} \end{pmatrix}$$

Bob

Hash Function:

$$H(x) = \prod_j g_{j,x[j]}$$

$$h = H(x) \longrightarrow$$

Key Generation:

"I want to learn $x[i]$"

$$key = \begin{pmatrix} \tilde{g}_{1,0} & \tilde{g}_{2,0} & \tilde{g}_{3,0} & \cdots & \tilde{g}_{n,0} \\ \tilde{g}_{1,1} & \tilde{g}_{2,1} & \tilde{g}_{3,1} & \cdots & \tilde{g}_{n,1} \end{pmatrix} \longleftarrow$$

trapdoor: uniform $t \in \mathbb{Z}_p$

$$\boldsymbol{key} = \begin{pmatrix} g_{1,0}^t & \cdots & g_{i,0}^t & \cdots & g_{n,0}^t \\ g_{1,1}^t & \cdots & \boldsymbol{g_{i,1}^t \cdot g} & \cdots & g_{n,1}^t \end{pmatrix}$$

Hinting:

$$E(key, x) = \prod_j \tilde{\boldsymbol{g}}_{j,x[j]}$$

$$e = E(key, x) \longrightarrow$$

Decoding: using hash h and trapdoor t

$$\boldsymbol{e = h^t} \quad \rightarrow \boldsymbol{x[i] = 0}$$

$$\boldsymbol{e = h^t \cdot g} \quad \rightarrow \boldsymbol{x[i] = 1}$$

Rate-$1/\lambda$ Trapdoor Hash

- For applications in **Setting II (Sublinear Secure RAM Computation)**:
  Rate-$1/\lambda$ TDH **is sufficient***.

- For applications in **Setting I (Rate-1 OT)**:
  We need **Rate-1 TDH**, i.e. TDH where the hint is a single bit.

Rate-1 Trapdoor Hash from DDH

Alice

Bob

Hinting:

$$e = \prod_i \tilde{\boldsymbol{g}}_{i,x[i]}$$

$x[i] = 0$ → $\boldsymbol{h^t} \in \mathbb{G}$

$x[i] = 1$ → $\boldsymbol{h^t \cdot g} \in \mathbb{G}$

$$e = E(key, x)$$

Decoding: using hash h and trapdoor t

e

$= h^t$? → $\boldsymbol{x[i] = 0}$

$= h^t \cdot g$? → $\boldsymbol{x[i] = 1}$

Rate-1 Trapdoor Hash from DDH

Alice

Bob

Hinting:

$$e = \prod_i \tilde{\boldsymbol{g}}_{i,x[i]}$$

$x[i] = 0$ ↙ $\boldsymbol{h^t} \in \mathbb{G}$

$x[i] = 1$ ↘ $\boldsymbol{h^t \cdot g} \in \mathbb{G}$

$LSB(e)$ →

Decoding: using hash h and trapdoor t

$LSB(e)$

$= LSB(h^t)\,?$ ↙ $\boldsymbol{x[i] = 0}$

$= LSB(h^t g)?$ ↘ $\boldsymbol{x[i] = 1}$

Attempt I:

Send **LSB** of e.

Fails when $LSB(h^t) = LSB(h^t g)$ is **equal**: happens with probability $1/2$.

Rate-1 Trapdoor Hash from DDH

Alice

Bob

Hinting:

$$e = \prod_i \tilde{\boldsymbol{g}}_{i,x[i]}$$

$x[i] = 0$ → $\boldsymbol{h^t} \in \mathbb{G}$

$x[i] = 1$ → $\boldsymbol{h^t \cdot g} \in \mathbb{G}$

$\Phi(e)$ →

Decoding: using hash h and trapdoor t

$\Phi(e)$

$= \Phi(h^t)$? → $\boldsymbol{x[i] = 0}$

$= \Phi(h^t g)$? → $\boldsymbol{x[i] = 1}$

Attempt I:

Send **LSB** of e.

Fails when $LSB(h^t) = LSB(h^t g)$ is **equal**: happens with probability $1/2$.

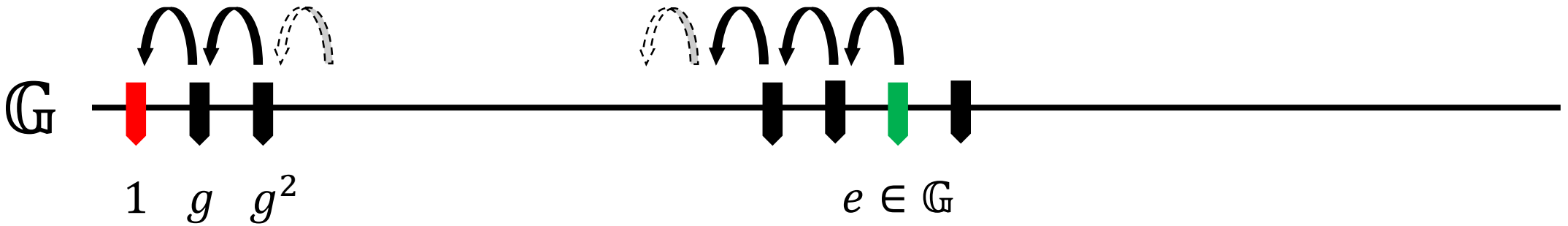
Goal: an encoding $\Phi: \mathbb{G} \to \{0,1\}$ such that with high probability

$$\Phi(h^t) \neq \Phi(h^t g)$$

Rate-1 Trapdoor Hash from DDH

Goal: an encoding $\Phi: \mathbb{G} \to \{0,1\}$ such that with high probability

$$\Phi(h^t) \neq \Phi(h^t g)$$



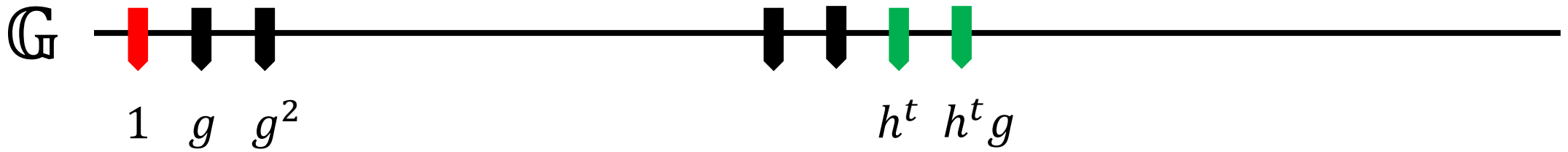
Attempt II:

$\Phi(e) =$ LSB of **number of steps to reach 1**

Rate-1 Trapdoor Hash from DDH

Goal: an encoding $\Phi: \mathbb{G} \to \{0,1\}$ such that with high probability

$$\Phi(h^t) \neq \Phi(h^t g)$$



Attempt II:

$$\Phi(e) = \text{LSB of } \textbf{number of steps to reach 1}$$

Clearly, $\Phi(h^t) \neq \Phi(h^t g)$ for all $t \in \mathbb{Z}_p$.

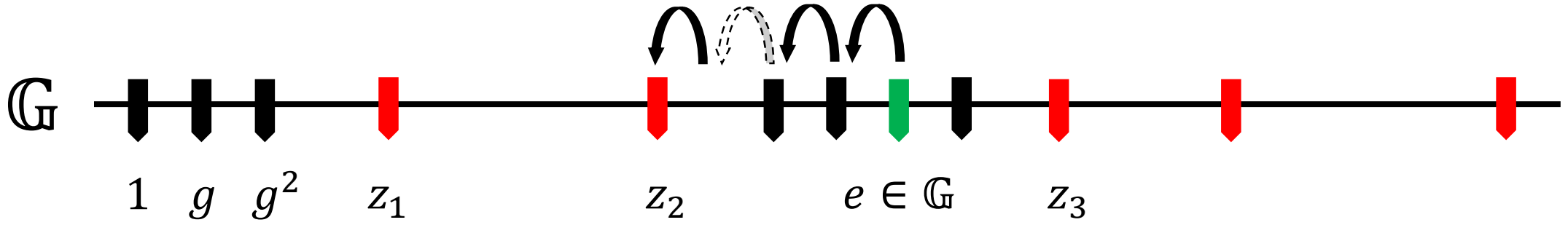
Problem: **not efficient** (this is DLOG).

Rate-1 Trapdoor Hash from DDH

Goal: an encoding $\Phi: \mathbb{G} \to \{0,1\}$ such that with high probability

$$\Phi(h^t) \neq \Phi(h^t g)$$

Idea: distributed discrete log [BGI16,DKK18]



Attempt III: using a PRF, define many random "**reference points**": $\boldsymbol{z}$ **s.t.** $\boldsymbol{PRF(z) = 0}$.

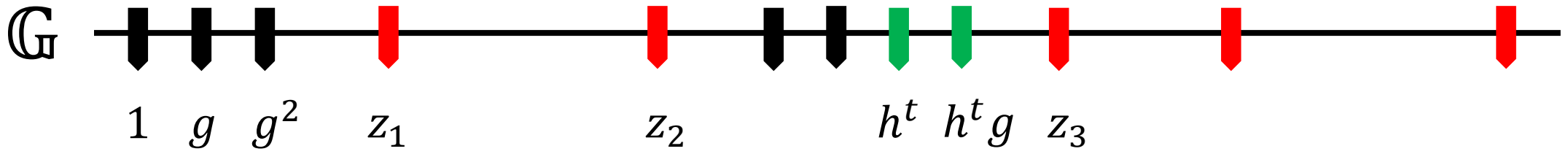
$\Phi(e) =$ LSB of **number of steps to reach the closest reference point**

Rate-1 Trapdoor Hash from DDH

Goal: an encoding $\Phi: \mathbb{G} \to \{0,1\}$ such that with high probability

$$\Phi(h^t) \neq \Phi(h^t g)$$

Idea: distributed discrete log [BGI16,DKK18]



Attempt III: using a PRF, define many random "**reference points**": $\boldsymbol{z}$ **s.t.** $\boldsymbol{PRF(z) = 0}$.

$\Phi(e) =$ LSB of **number of steps to reach the closest reference point**

If the reference points are **dense enough**, this is **efficient**.
If the reference points are **sparse enough**, $\Phi(h^t) \neq \Phi(h^t g)$ **with high probability**.

Conclusion and Further Research

We introduce **Trapdoor Hash**:

- Simple primitive with **constructions from various standard assumption**.
- Powerful primitive with **significant applications** in communication-efficient protocols.

Applications in **Setting I: Rate-Optimal Protocols for OT and Other Functions**:

- Constructions from new standard assumptions.
- More **general functionalities**?
- Constructions from **general assumptions**?

Applications in **Setting II: Sublinear Protocols for Secure RAM Computation**:

- First constructions from number-theoretic assumptions.
- Can we get constructions **as efficient as lattice-based schemes**?

Other applications of trapdoor hash or similar techniques?

Thanks for Listening!