Capstone Thesis Presentation – Spring 2025

Ring Trapdoor Functions: A Lattice-Based Framework for Secure Ring Signatures

Bhumika Mittal

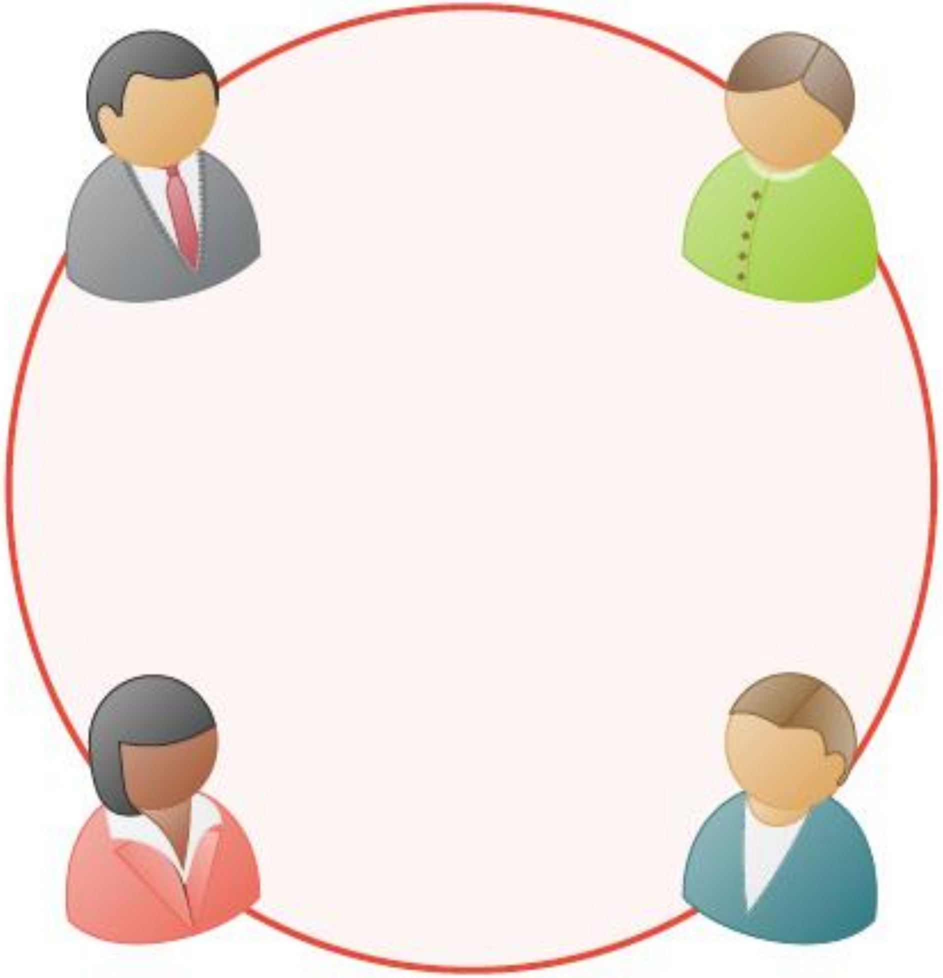
Thesis Supervisors

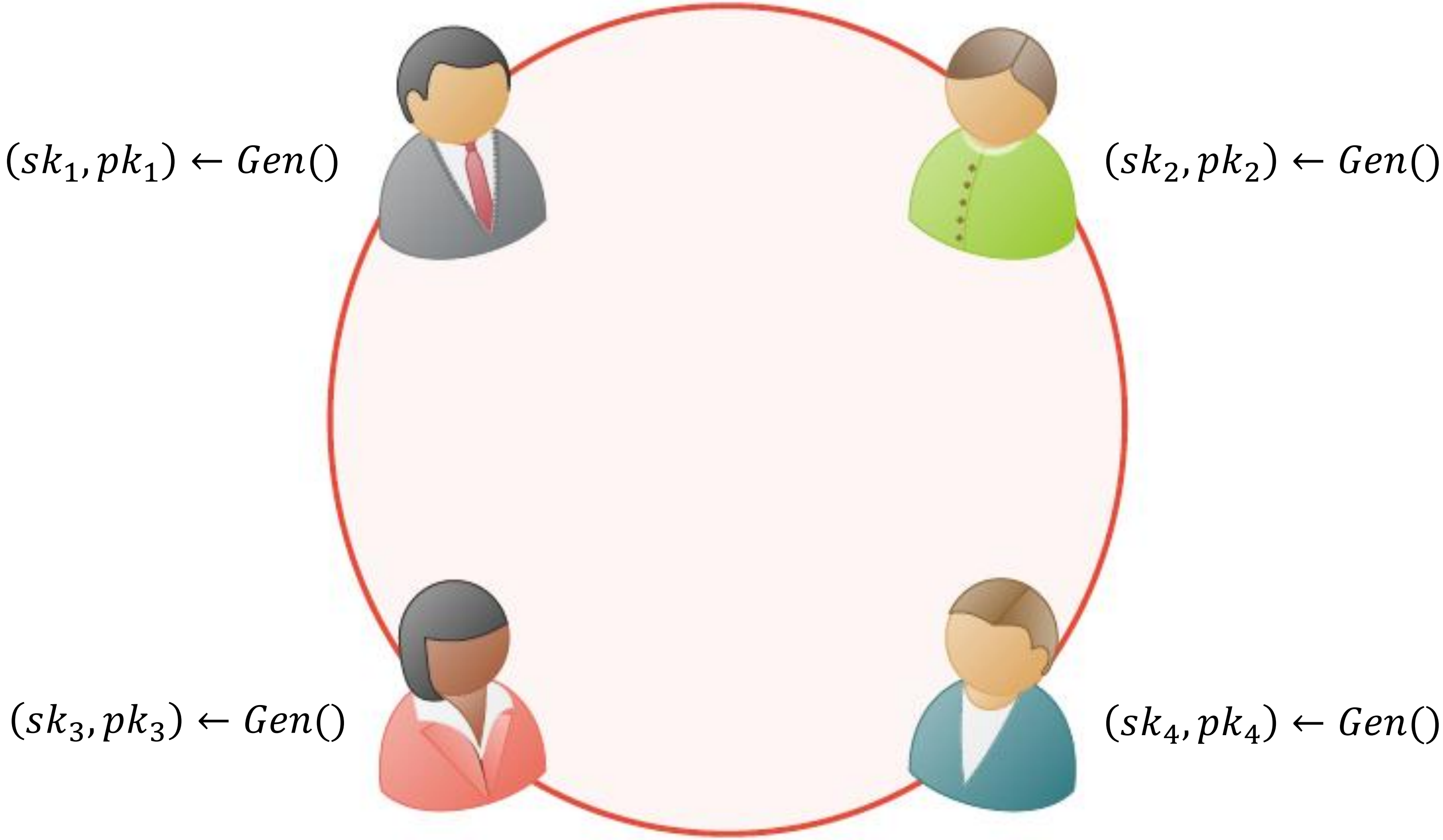
Eike Kiltz,
Professor of Computer Science
Ruhr-Universität Bochum

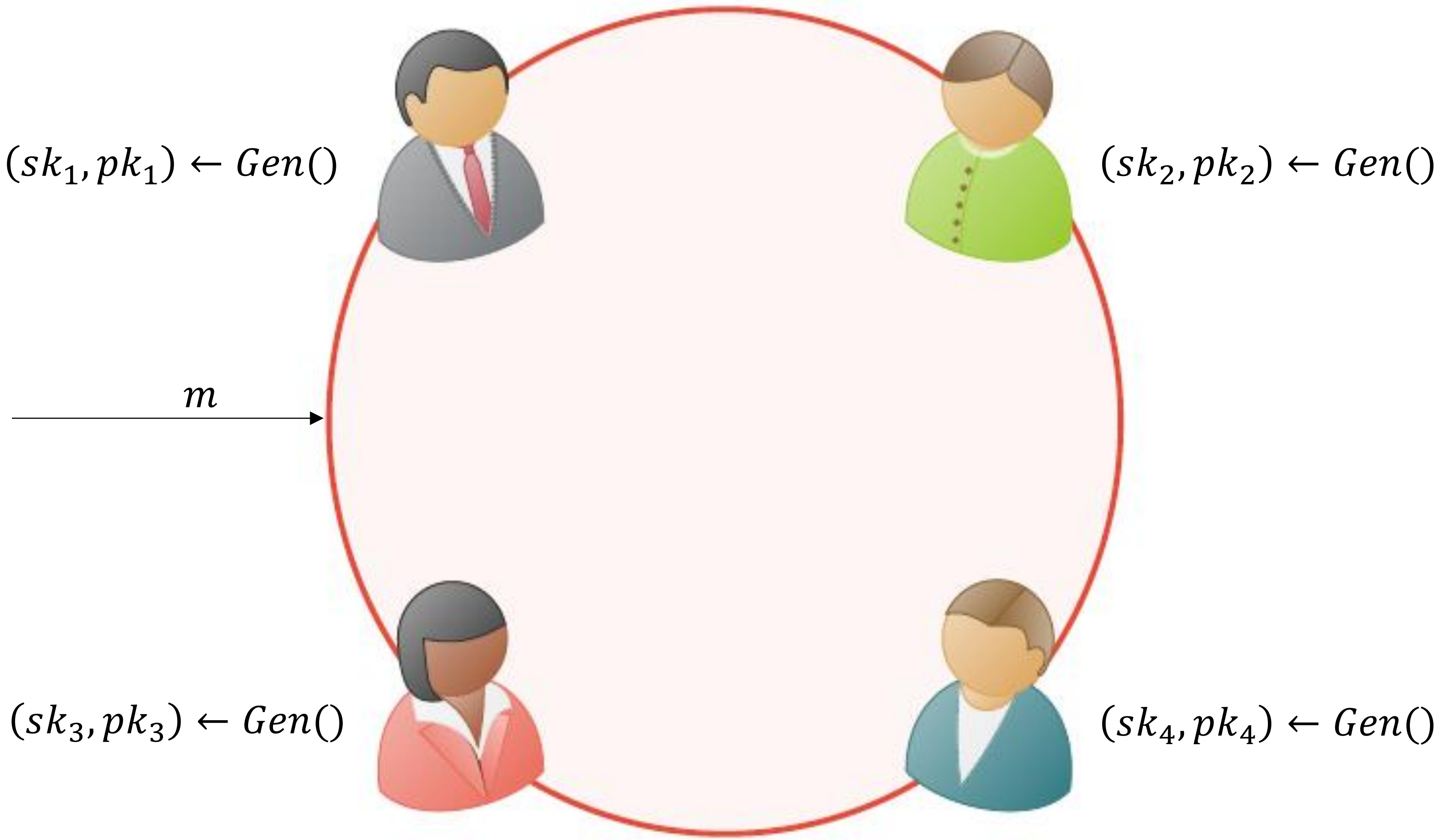
Mahavir Jhawar
Associate Professor of Computer Science
Ashoka University

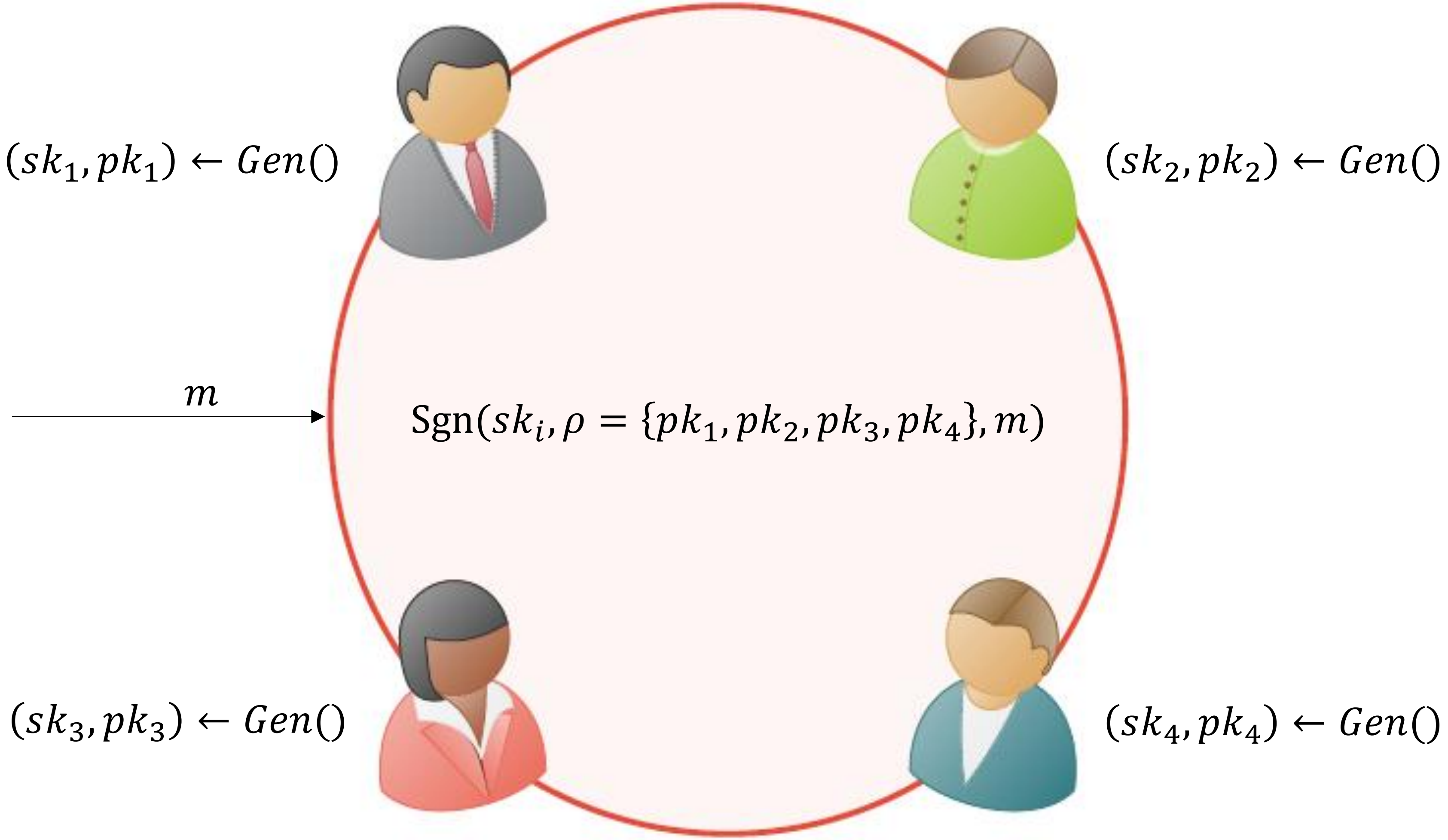
A generic framework for constructing efficient ring signature schemes through novel cryptographic abstractions.

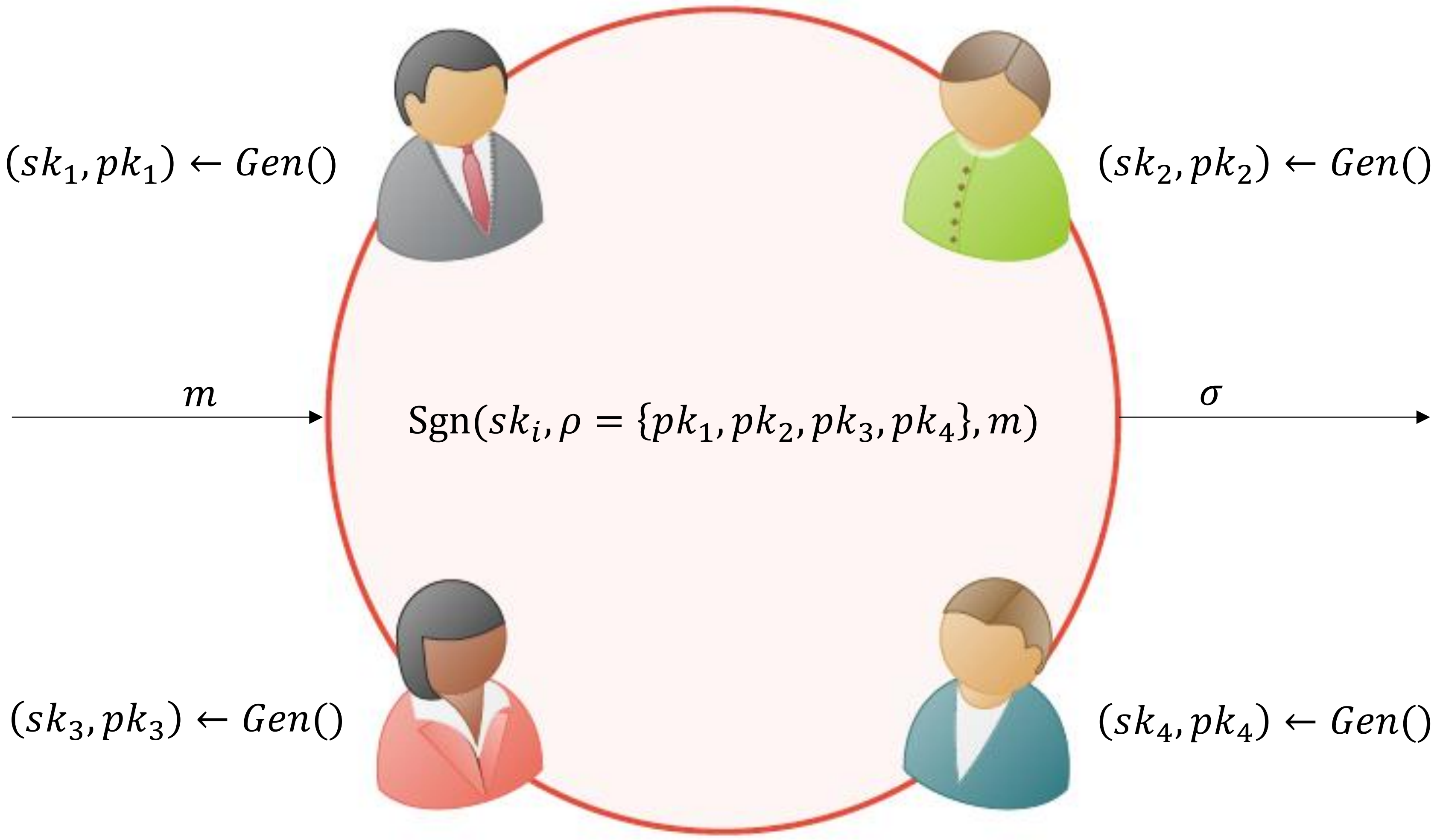
Ring Signatures

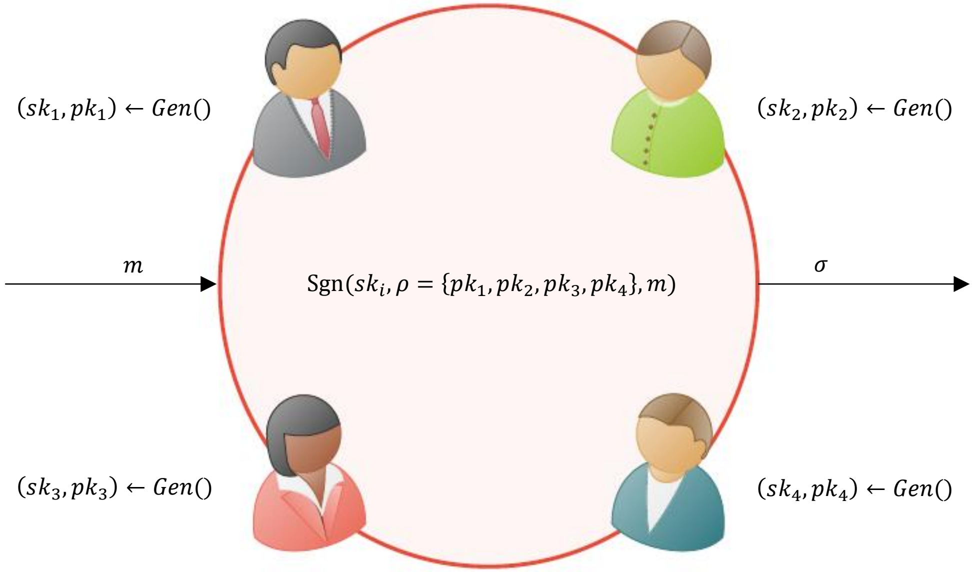



$(sk_1, pk_1) \leftarrow Gen()$
$(sk_2, pk_2) \leftarrow Gen()$
$(sk_3, pk_3) \leftarrow Gen()$
$(sk_4, pk_4) \leftarrow Gen()$

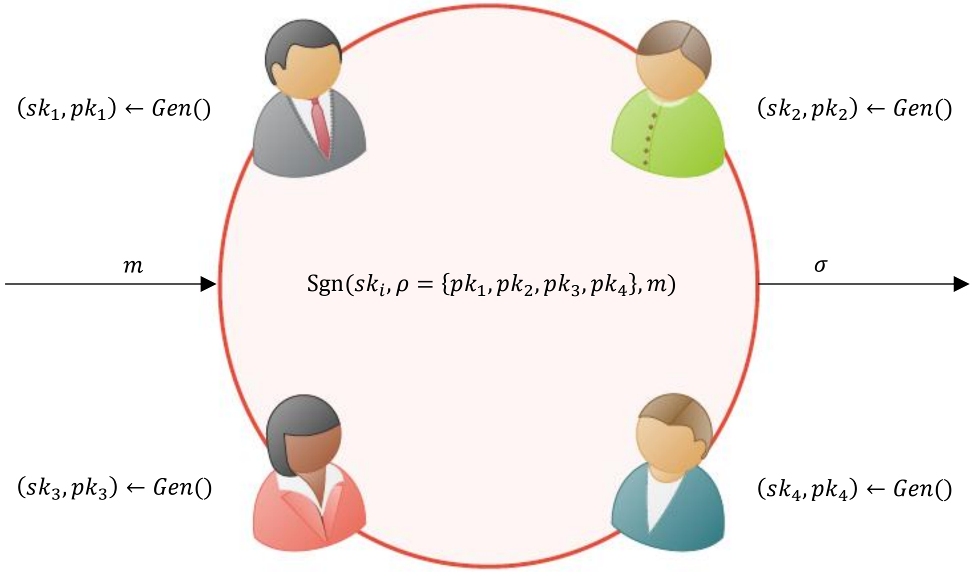

$(sk_1, pk_1) \leftarrow Gen()$
$(sk_2, pk_2) \leftarrow Gen()$
m
$(sk_3, pk_3) \leftarrow Gen()$
$(sk_4, pk_4) \leftarrow Gen()$


$(sk_1, pk_1) \leftarrow Gen()$
$(sk_2, pk_2) \leftarrow Gen()$
m
$\mathrm{Sgn}(sk_i, \rho = \{pk_1, pk_2, pk_3, pk_4\}, m)$
$(sk_3, pk_3) \leftarrow Gen()$
$(sk_4, pk_4) \leftarrow Gen()$


$(sk_1, pk_1) \leftarrow Gen()$
$(sk_2, pk_2) \leftarrow Gen()$
m
$\mathrm{Sgn}(sk_i, \rho = \{pk_1, pk_2, pk_3, pk_4\}, m)$
σ
$(sk_3, pk_3) \leftarrow Gen()$
$(sk_4, pk_4) \leftarrow Gen()$



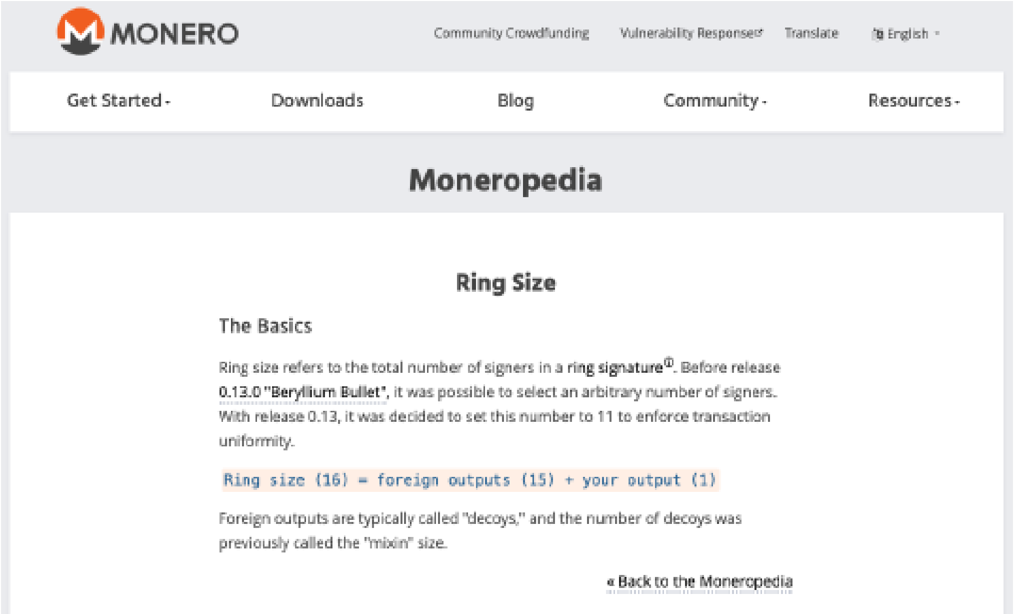
- **Unforgeability**: One sk_i is needed to generate σ.



- **Unforgeability**: One sk_i is needed to generate σ.
- **Anonymity**: Given σ and ρ, it is not possible to identify who signed.

- Originally introduced as a mechanism to protect whistle-blowers.

- Originally introduced as a mechanism to protect whistle-blowers.
- Currently used in electronic voting systems.



MONERO

Community Crowdfunding · Vulnerability Response · Translate · English

Get Started · Downloads · Blog · Community · Resources

Moneropedia

Ring Size

The Basics

Ring size refers to the total number of signers in a ring signature. Before release 0.13.0 "Beryllium Bullet", it was possible to select an arbitrary number of signers. With release 0.13, it was decided to set this number to 11 to enforce transaction uniformity.

```
Ring size (16) = foreign outputs (15) + your output (1)
```

Foreign outputs are typically called "decoys," and the number of decoys was previously called the "mixin" size.

« Back to the Moneropedia

- Originally introduced as a mechanism to protect whistle-blowers.
- Currently used in electronic voting systems.
- Primitive behind cryptocurrencies such as Dash & Monero.

RSig Syntax

A ring signature scheme $RSig$ is given by four algorithms (Stp, Gen, Sgn, Ver):

RSig Syntax

A ring signature scheme $RSig$ is given by four algorithms (Stp, Gen, Sgn, Ver):

- $par \leftarrow Stp(\kappa)$: Given an upper bound, $\kappa > 1$, on the ring size, the probabilistic setup algorithm Stp returns system parameters par, where par defines a message space M. We assume that all algorithms are implicitly given access to the system parameters par.

RSig Syntax

A ring signature scheme $RSig$ is given by four algorithms (Stp, Gen, Sgn, Ver):

- $par \leftarrow Stp(\kappa)$: Given an upper bound, $\kappa > 1$, on the ring size, the probabilistic setup algorithm Stp returns system parameters par, where par defines a message space M. We assume that all algorithms are implicitly given access to the system parameters par.

- $(pk, sk) \leftarrow Gen$: The probabilistic key generation algorithm returns a secret key sk and a corresponding public key pk.

RSig Syntax

A ring signature scheme $RSig$ is given by four algorithms (Stp, Gen, Sgn, Ver):

- $par \leftarrow Stp(\kappa)$: Given an upper bound, $\kappa > 1$, on the ring size, the probabilistic setup algorithm Stp returns system parameters par, where par defines a message space M. We assume that all algorithms are implicitly given access to the system parameters par.

- $(pk, sk) \leftarrow Gen$: The probabilistic key generation algorithm returns a secret key sk and a corresponding public key pk.

- $\sigma \leftarrow Sgn(sk, \rho, m)$: Given a secret key sk, a ring $\rho = \{pk_1, \ldots, pk_\mathrm{k}\}$ such that the public key pk corresponding to sk satisfies $pk \in \rho$ and $k \leq \kappa$, and a message $m \in M$, the probabilistic signing algorithm Sgn returns a signature σ from a signature space S.

RSig Syntax

A ring signature scheme $RSig$ is given by four algorithms (Stp, Gen, Sgn, Ver):

- $par \leftarrow Stp(\kappa)$: Given an upper bound, $\kappa > 1$, on the ring size, the probabilistic setup algorithm Stp returns system parameters par, where par defines a message space M. We assume that all algorithms are implicitly given access to the system parameters par.

- $(pk, sk) \leftarrow Gen$: The probabilistic key generation algorithm returns a secret key sk and a corresponding public key pk.

- $\sigma \leftarrow Sgn(sk, \rho, m)$: Given a secret key sk, a ring $\rho = \{pk_1, \ldots, pk_{\mathrm{k}}\}$ such that the public key pk corresponding to sk satisfies $pk \in \rho$ and $k \leq \kappa$, and a message $m \in M$, the probabilistic signing algorithm Sgn returns a signature σ from a signature space S.

- $b \leftarrow Ver(\sigma, \rho, m)$: Given a signature σ, a ring ρ, and a message m, the deterministic verification algorithm Ver returns a bit $b \in \{0, 1\}$.

RSig Properties—Correctness

RSig Properties—Correctness

RSig is $\delta(\kappa)$-correct if $\forall\kappa \in \mathbb{N}, \{(pk_i, sk_i)\}_{i\in[k]} \in \sup(Gen)$, and for any $i \in [k]$ with $k \le \kappa$,

$$\Pr[\mathrm{Ver}\,(\mathrm{Sgn}(sk_i, \rho, m), \rho, m) \neq 1] \le \delta(\kappa)$$

where $\rho := \{pk_1, \cdots, pk_k\}$.

Unforgeability under Chosen Ring Attack (UF-CRA)

Unforgeability under Chosen Ring Attack (UF-CRA)

Given a target set of public keys $\rho = \{pk_1, pk_2, \ldots, pk_n\}$, there doesn't exist an adversary that can forge a signature σ^* on a message m^* and a ring $\rho^* \subseteq \rho$.

Unforgeability under Chosen Ring Attack (UF-CRA)

Given a target set of public keys $\rho = \{pk_1, pk_2, \ldots, pk_n\}$, there doesn't exist an adversary that can forge a signature σ^* on a message m^* and a ring $\rho^* \subseteq \rho$.

$$\mathsf{Adv}_{\mathrm{RSig},\mathcal{A}}^{(n,\kappa,Q_{\mathsf{Sgn}})\text{-UF-CRA}} := \Pr[(n,\kappa,Q_{\mathsf{Sgn}})\text{-UF-CRA}_{\mathrm{RSig}}(\mathcal{A}) \to 1],$$

Unforgeability under Chosen Ring Attack (UF-CRA)

Given a target set of public keys $\rho = \{pk_1, pk_2, \ldots, pk_n\}$, there doesn't exist an adversary that can forge a signature σ^* on a message m^* and a ring $\rho^* \subseteq \rho$.

$$\mathsf{Adv}_{\mathrm{RSig},\mathcal{A}}^{(n,\kappa,Q_{\mathsf{Sgn}})\text{-UF-CRA}} := \Pr[(n,\kappa,Q_{\mathsf{Sgn}})\text{-UF-CRA}_{\mathrm{RSig}}(\mathcal{A}) \to 1],$$

The adversary is allowed to make **adaptive signing queries** on a message m_i and ring ρ_i, as long as the ring contains at least one of the supplied keys from ρ.

Anonymity under Full Key Exposure

Anonymity under Full Key Exposure

- Any adversary cannot distinguish the real signer from other ring members, even if all their secret keys are exposed.

Anonymity under Full Key Exposure

- Any adversary cannot distinguish the real signer from other ring members, even if all their secret keys are exposed.

- The adversary is given:
  - Public keys,
  - Access to a signing oracle,
  - A message and two indices (before getting secret keys).

Anonymity under Full Key Exposure

- Any adversary cannot distinguish the real signer from other ring members, even if all their secret keys are exposed.

- The adversary is given:
  - Public keys,
  - Access to a signing oracle,
  - A message and two indices (before getting secret keys).

- Then, the adversary receives:
  - Secret keys of all ring members,
  - A challenge signature.

Anonymity under Full Key Exposure

- Any adversary cannot distinguish the real signer from other ring members, even if all their secret keys are exposed.

- The adversary is given:
  - Public keys,
  - Access to a signing oracle,
  - A message and two indices (before getting secret keys).

- Then, the adversary receives:
  - Secret keys of all ring members,
  - A challenge signature.

Guess the index of the real signer.

New Level of Abstraction

New Level of Abstraction

New Level of Abstraction

1. **Foundational Insight**
   Enables fine-grained analysis of ring signature components and their security dependencies.

New Level of Abstraction

1. **Foundational Insight**
   Enables fine-grained analysis of ring signature components and their security dependencies.

2. **Modular Improvement**
   Permits black-box enhancements (e.g., ring size scaling) without protocol redesign.

New Level of Abstraction

1. **Foundational Insight**
   Enables fine-grained analysis of ring signature components and their security dependencies.

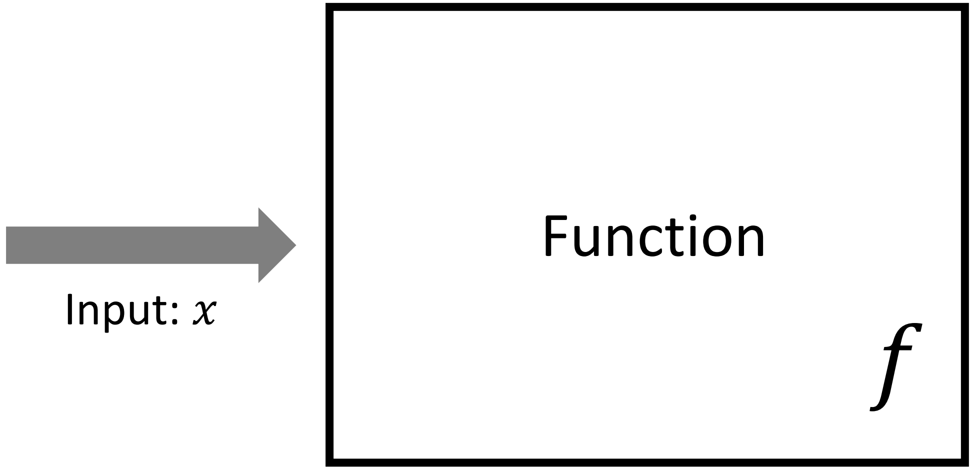
2. **Modular Improvement**
   Permits black-box enhancements (e.g., ring size scaling) without protocol redesign.

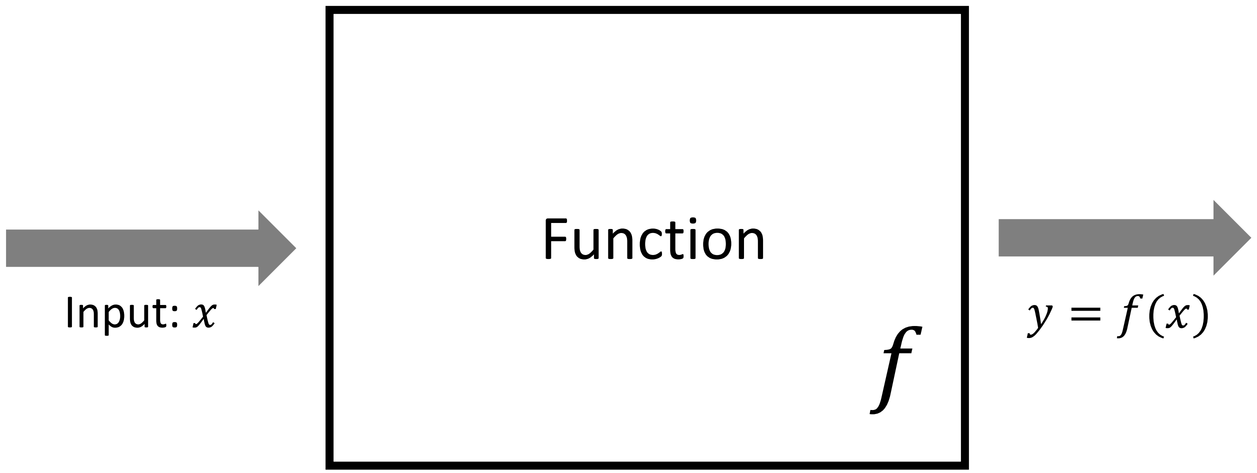
3. **Primitive Reusability**
   Provides building blocks for other anonymity-preserving cryptosystems

Trapdoor Functions

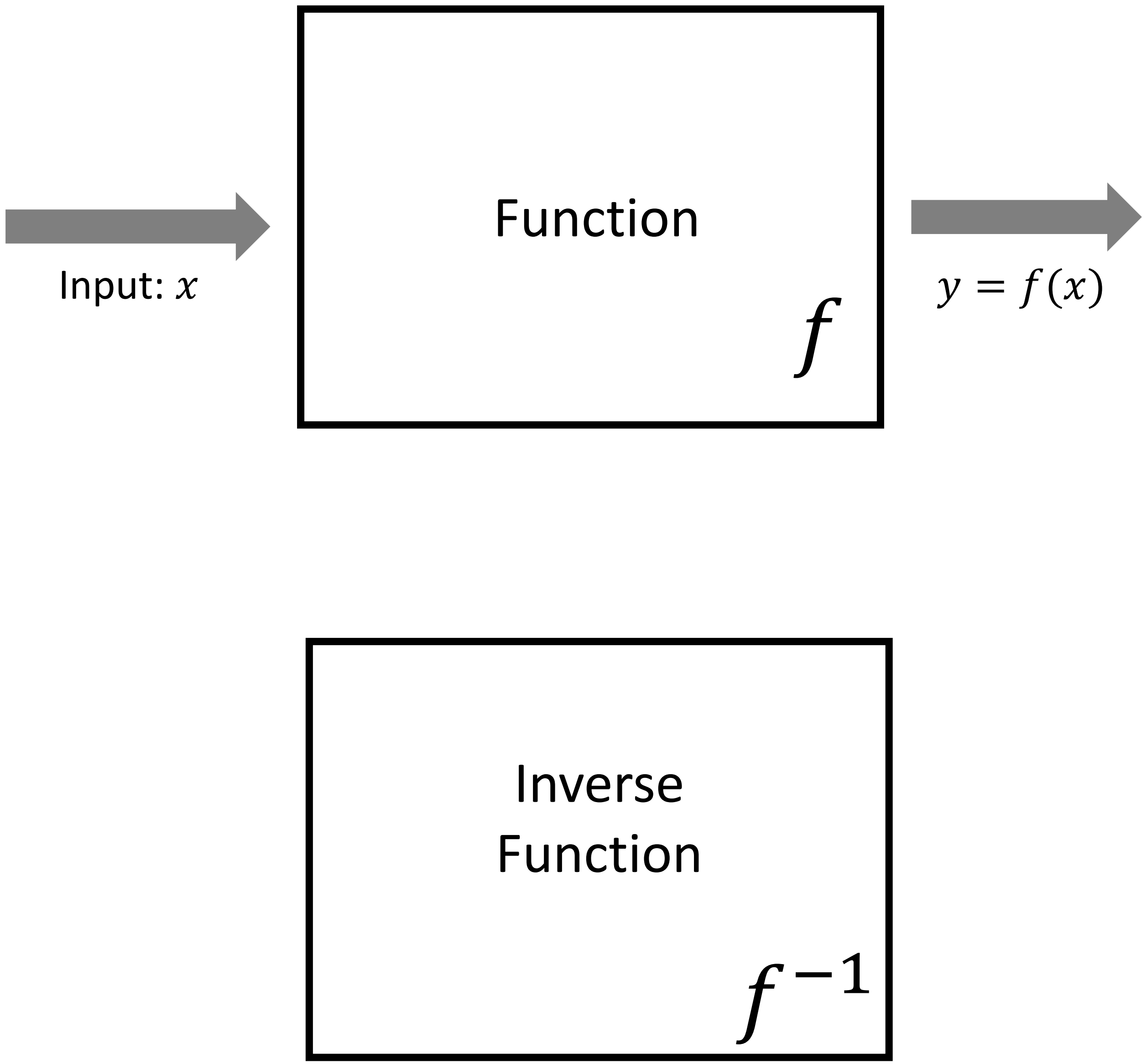
Function

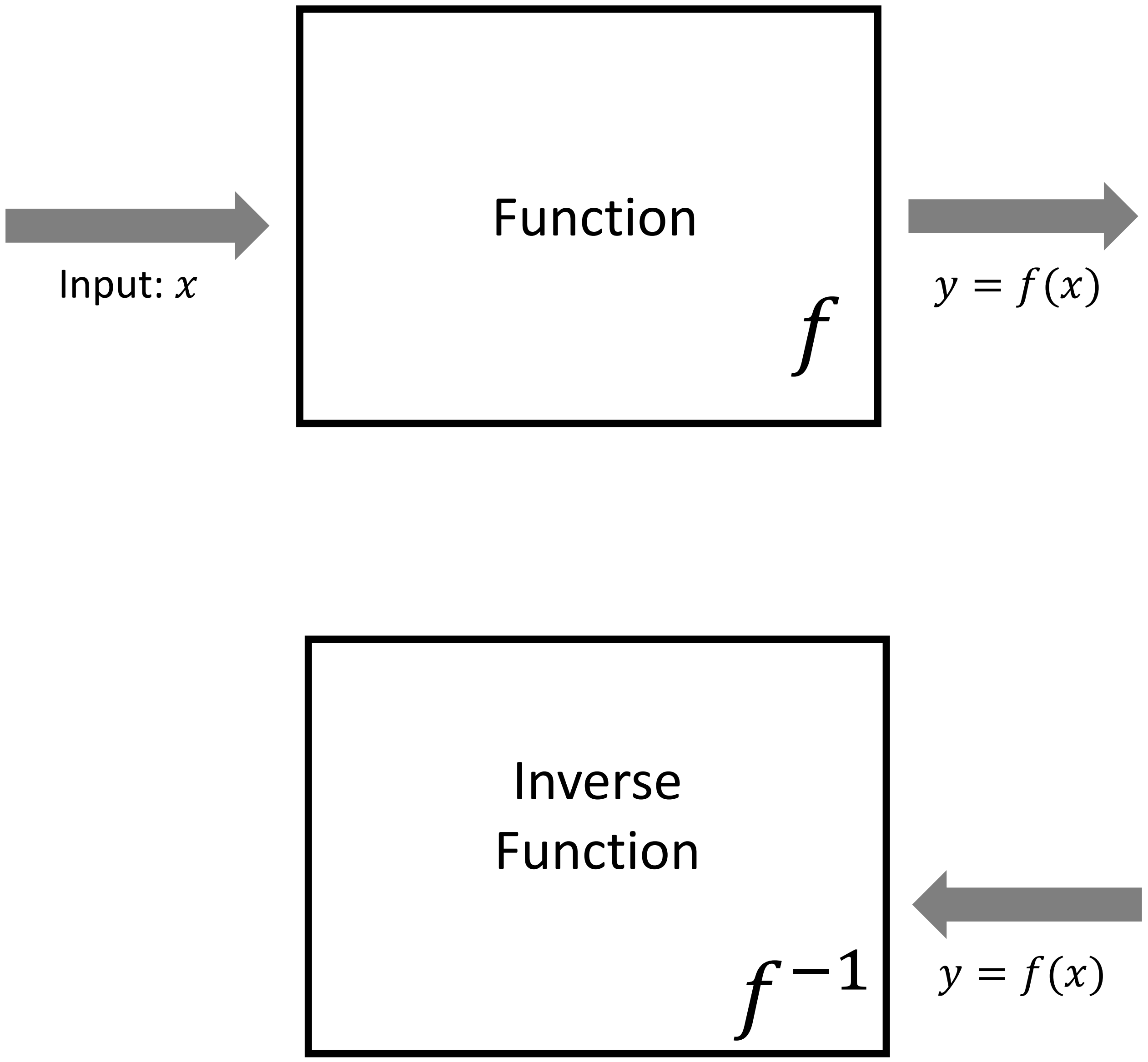
f

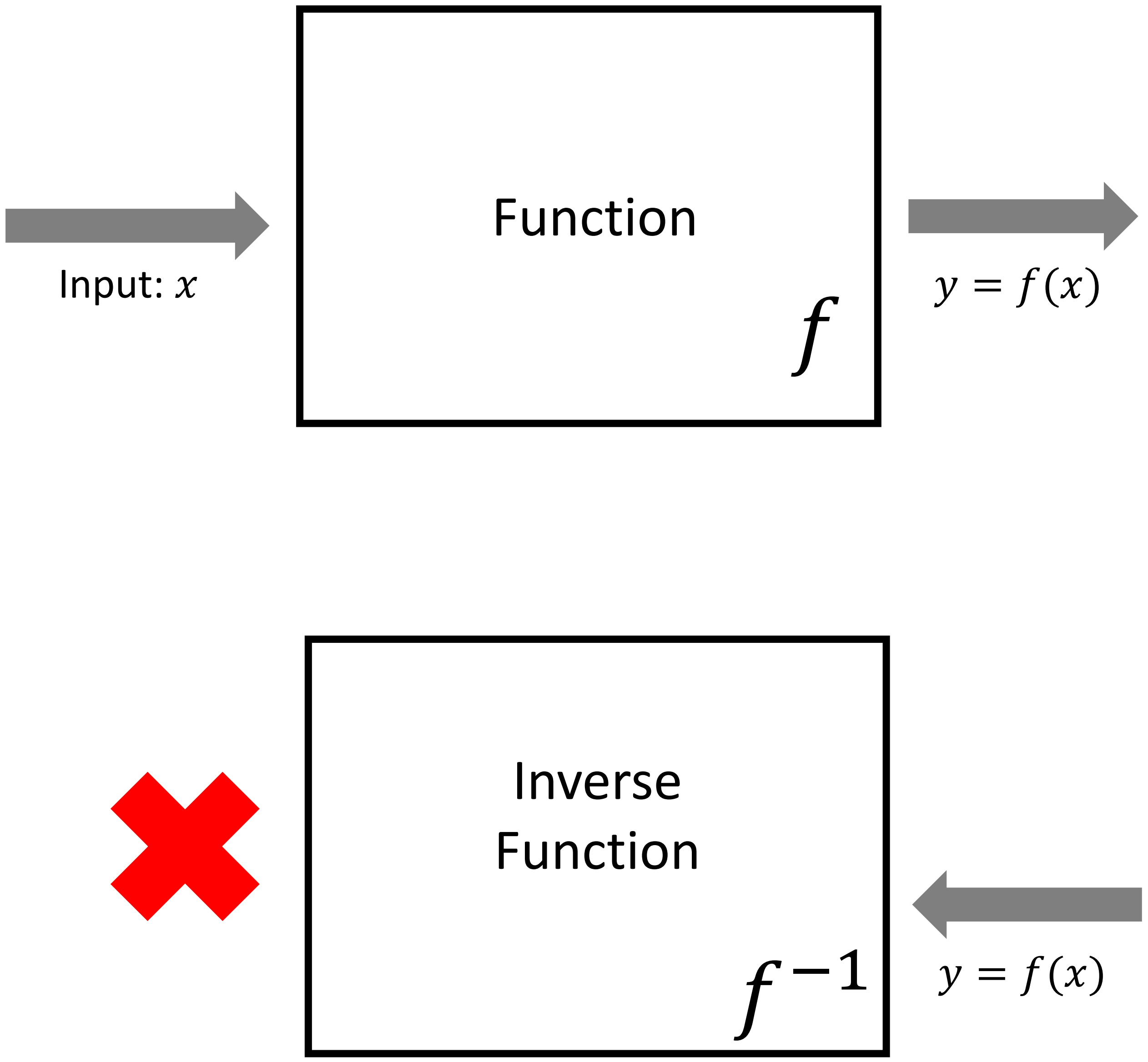

Function
Input: x
f

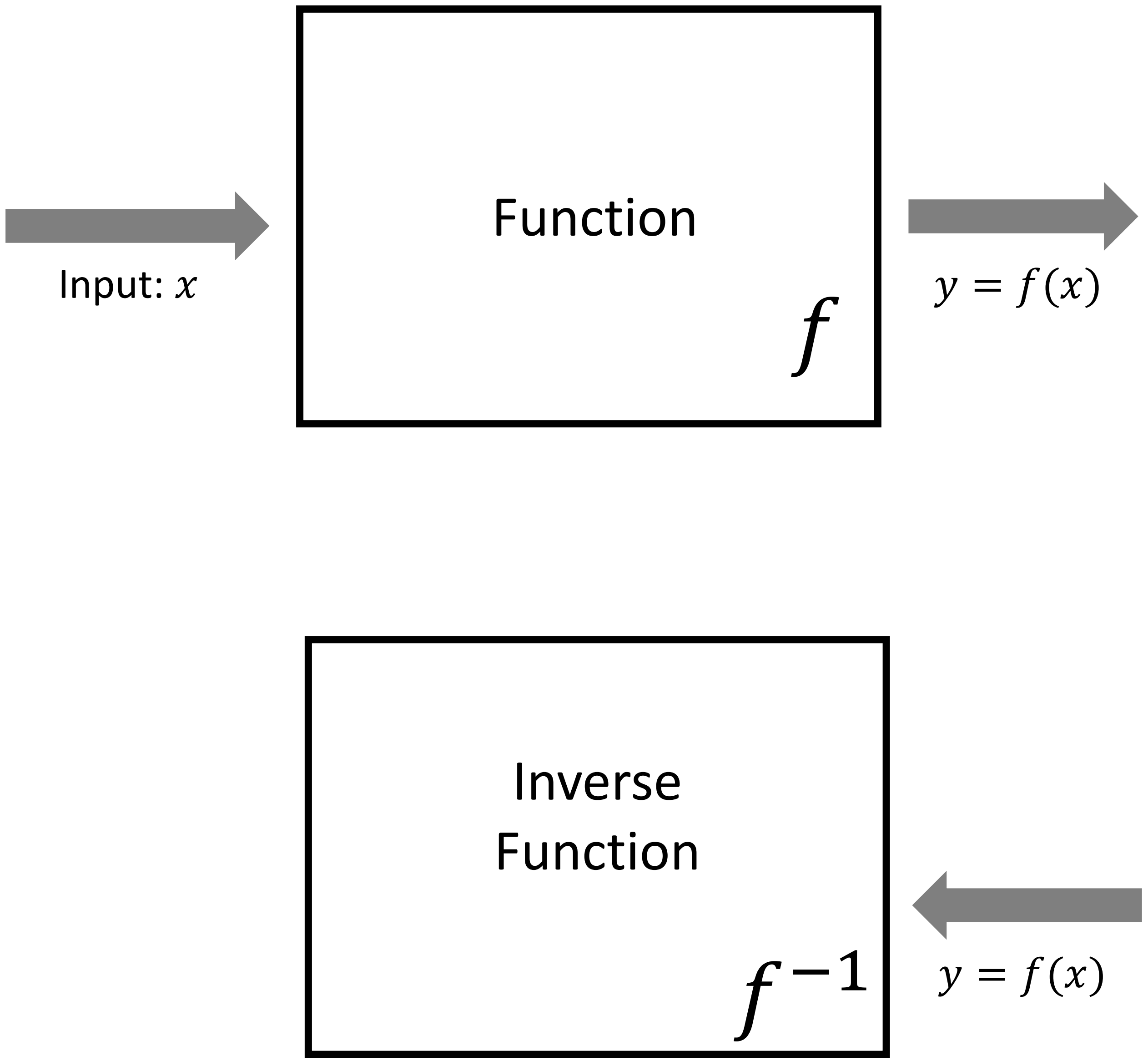


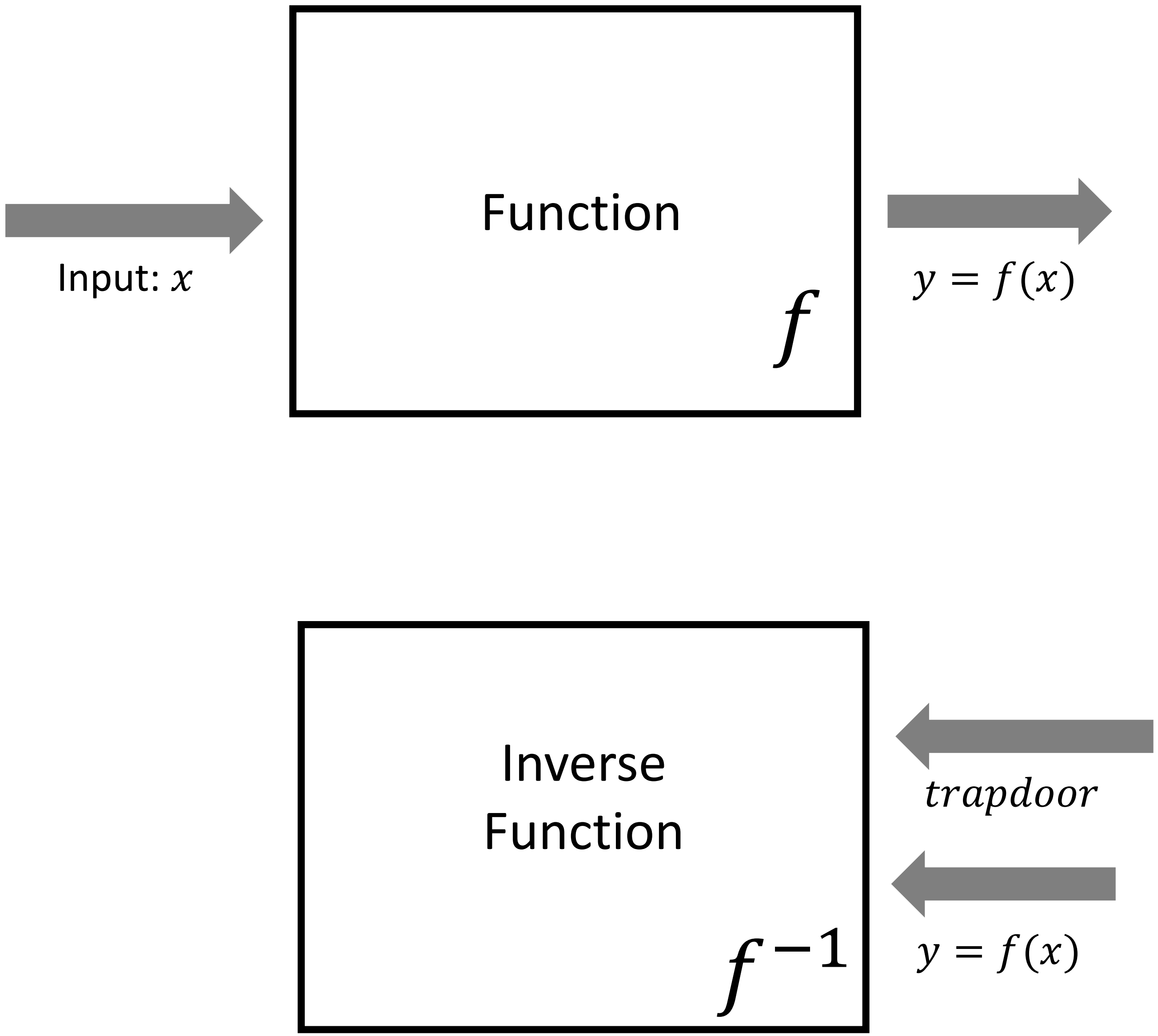
Function
Input: x
f
$y = f(x)$

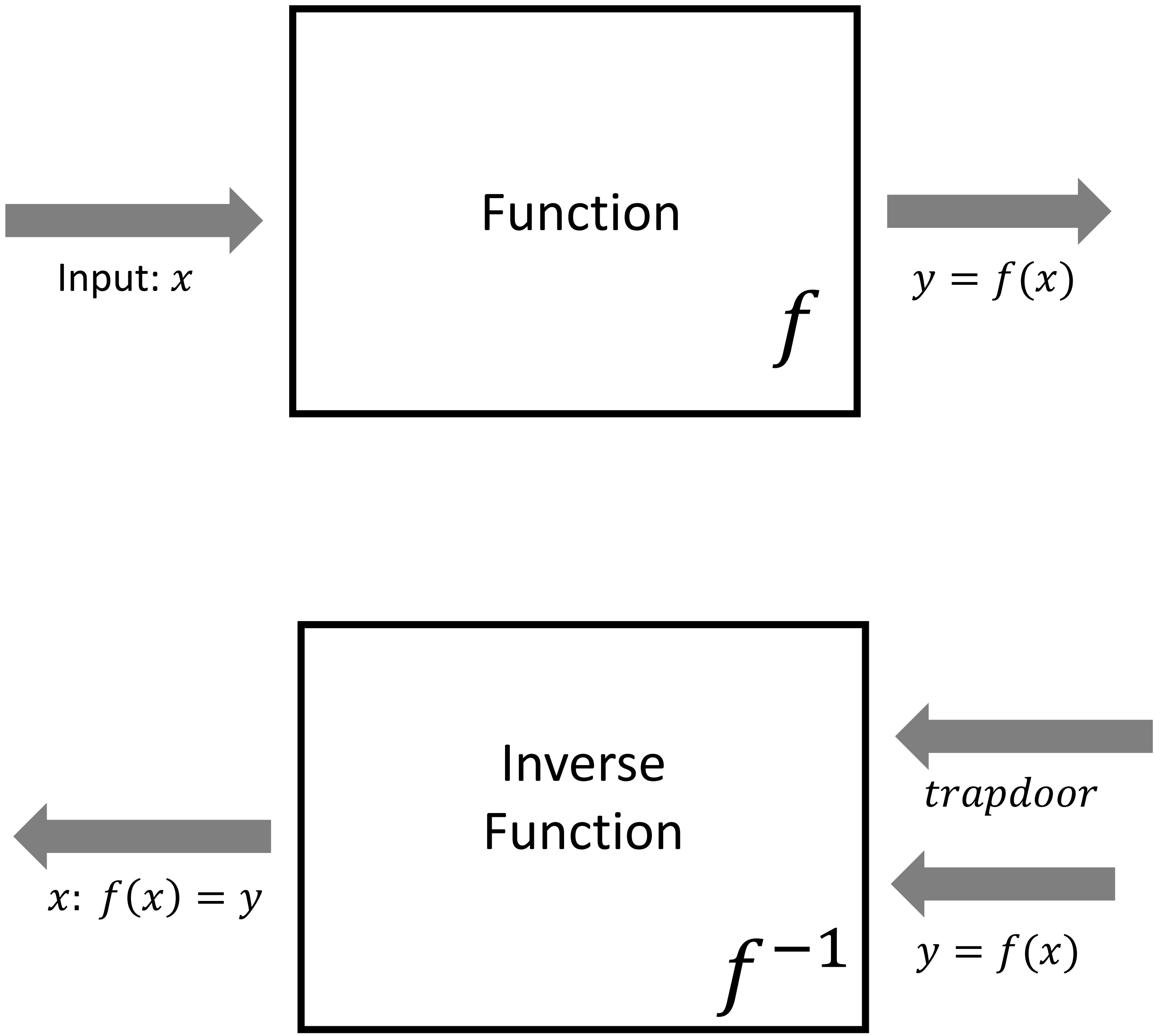

Input: x
Function
f
$y = f(x)$
Inverse
Function
f^{-1}


Function
Input: x
f
$y = f(x)$
Inverse
Function
f^{-1}
$y = f(x)$


Function
Input: x
f
$y = f(x)$
Inverse
Function
f^{-1}
$y = f(x)$

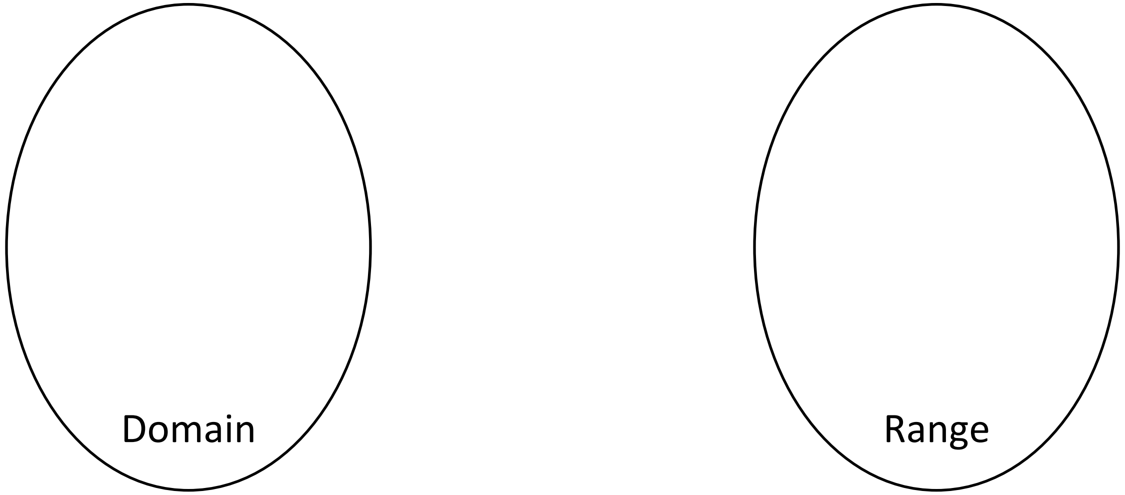

Function
Input: x
f
$y = f(x)$
Inverse
Function
f^{-1}
$y = f(x)$


Input: x
Function
f
$y = f(x)$
Inverse
Function
f^{-1}
$trapdoor$
$y = f(x)$

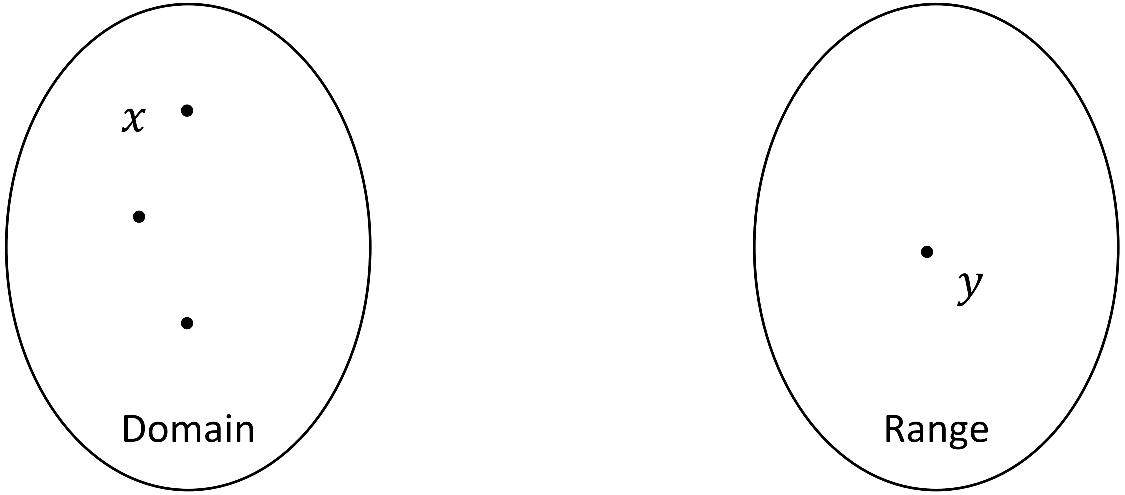

Function
Input: x
f
$y = f(x)$
Inverse
Function
f^{-1}
$trapdoor$
$y = f(x)$
x: $f(x) = y$

Preimage Sampleable Trapdoor Function

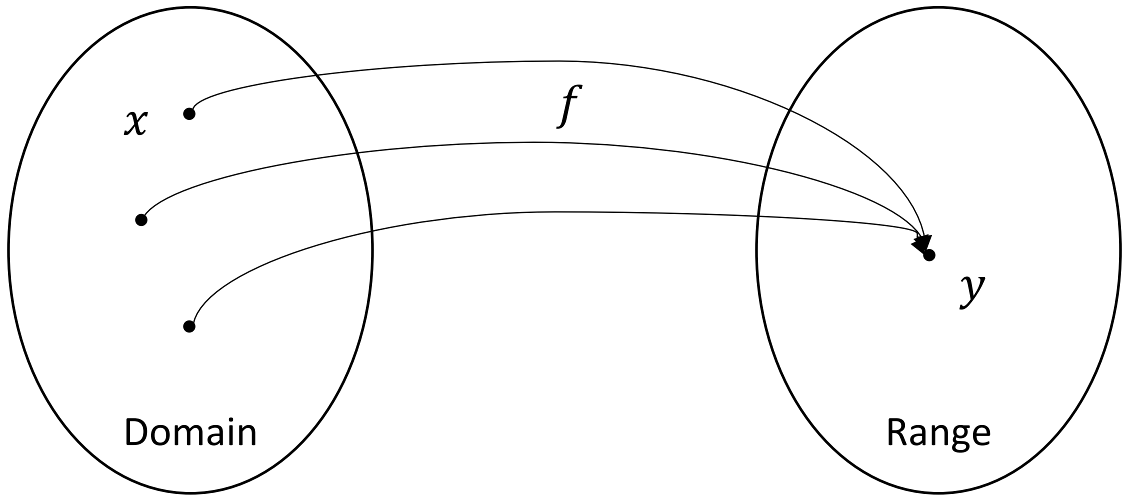
Preimage Sampleable Trapdoor Function



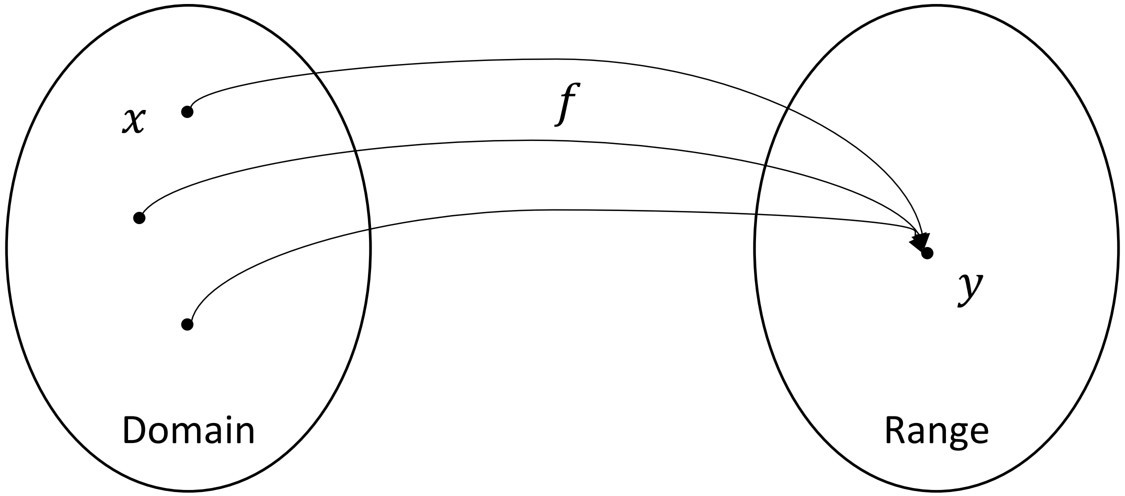
Preimage Sampleable Trapdoor Function



Preimage Sampleable Trapdoor Function

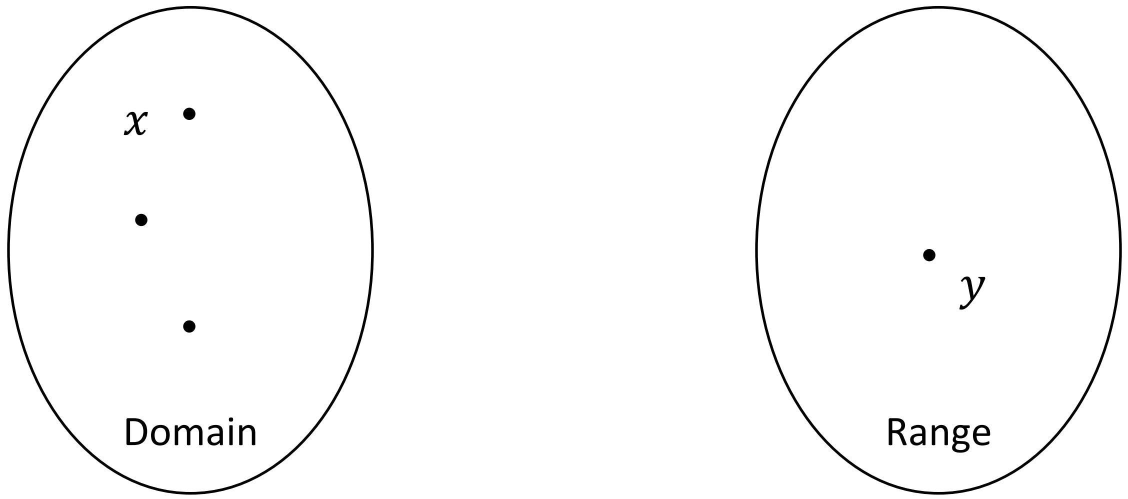


Preimage Sampleable Trapdoor Function



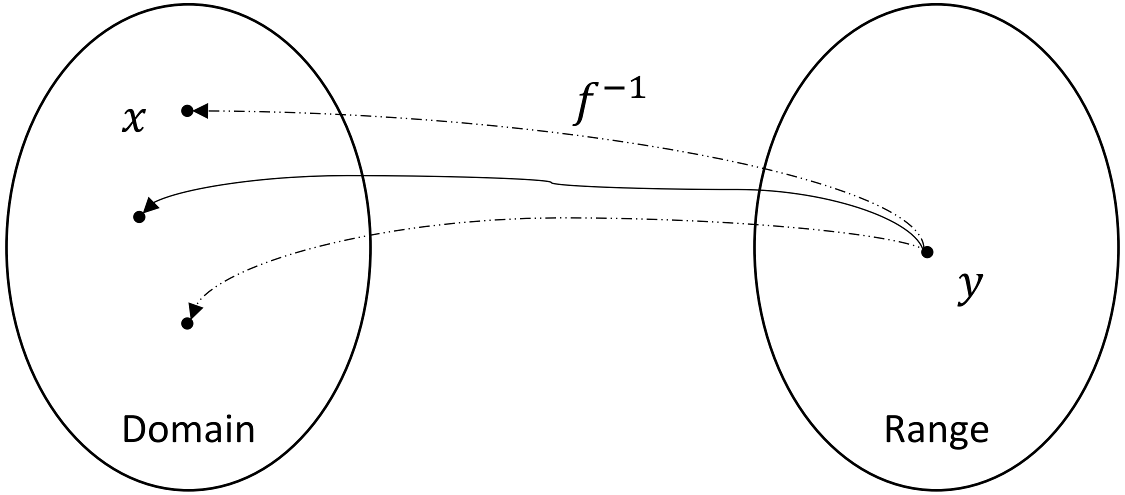
- Sample inputs so their outputs look random.

Preimage Sampleable Trapdoor Function



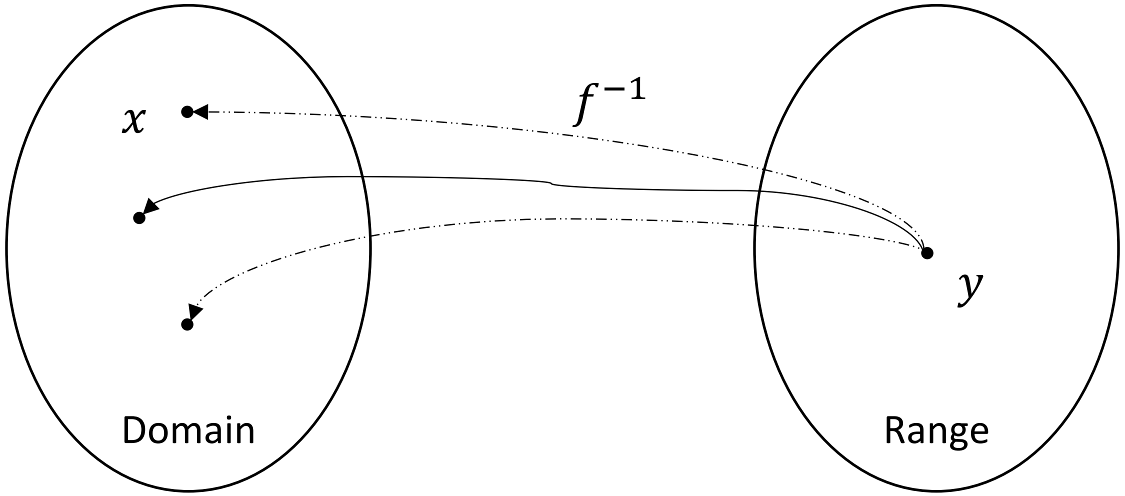
- Sample inputs so their outputs look random.
- Given an output and the trapdoor, produce a matching input that looks like a real, random input.

Preimage Sampleable Trapdoor Function



- Sample inputs so their outputs look random.
- Given an output and the trapdoor, produce a matching input that looks like a real, random input.

Preimage Sampleable Trapdoor Function



- Sample inputs so their outputs look random.
- Given an output and the trapdoor, produce a matching input that looks like a real, random input.
- But without the trapdoor, you can't figure out the input — it's still a one-way function.

Ring Preimage Sampleable Trapdoor Function

RTDF Syntax

RTDF Syntax

An RTDF is defined by five algorithms: $(Stp, Gen, Eval, Inv, Smp)$

RTDF Syntax

An RTDF is defined by five algorithms: $(Stp, Gen, Eval, Inv, Smp)$

- $par \leftarrow Stp(\kappa)$: Given $\kappa > 1$, the setup algorithm returns system parameters par, which define the range R.

RTDF Syntax

An RTDF is defined by five algorithms: $(Stp, Gen, Eval, Inv, Smp)$

- $par \leftarrow Stp(\kappa)$: Given $\kappa > 1$, the setup algorithm returns system parameters par, which define the range R.

- $(a, t) \leftarrow Gen()$: Return
  - a: description of function $f_a\colon D_a \rightarrow R$. Domain D_a is defined by a.
  - t: trapdoor for f_a.

RTDF Syntax

An RTDF is defined by five algorithms: $(Stp, Gen, Eval, Inv, Smp)$

- $par \leftarrow Stp(\kappa)$: Given $\kappa > 1$, the setup algorithm returns system parameters par, which define the range R.
- $(a, t) \leftarrow Gen()$: Return
  - a: description of function $f_a\colon D_a \rightarrow R$. Domain D_a is defined by a.
  - t: trapdoor for f_a.
- $y \leftarrow Eval(\rho = \{a_1, \dots, a_k\}, x)$: Given ring ρ and input $x \in D$, returns $y \in R$ deterministically.

RTDF Syntax

An RTDF is defined by five algorithms: $(Stp, Gen, Eval, Inv, Smp)$

- $par \leftarrow Stp(\kappa)$: Given $\kappa > 1$, the setup algorithm returns system parameters par, which define the range R.

- $(a, t) \leftarrow Gen()$: Return
  - a: description of function f_a: $D_a \rightarrow R$. Domain D_a is defined by a.
  - t: trapdoor for f_a.

- $y \leftarrow Eval(\rho = \{a_1, \dots, a_k\}, x)$: Given ring ρ and input $x \in D$, returns $y \in R$ deterministically.

- $x \leftarrow Inv(t, \rho, y)$: Given trapdoor t, ring ρ, and $y \in R$, returns $x \in D$ probabilistically.

RTDF Syntax

An RTDF is defined by five algorithms: $(Stp, Gen, Eval, Inv, Smp)$

- $par \leftarrow Stp(\kappa)$: Given $\kappa > 1$, the setup algorithm returns system parameters par, which define the range R.

- $(a, t) \leftarrow Gen()$: Return
  - a: description of function f_a: $D_a \rightarrow R$. Domain D_a is defined by a.
  - t: trapdoor for f_a.

- $y \leftarrow Eval(\rho = \{a_1, \ldots, a_k\}, x)$: Given ring ρ and input $x \in D$, returns $y \in R$ deterministically.

- $x \leftarrow Inv(t, \rho, y)$: Given trapdoor t, ring ρ, and $y \in R$, returns $x \in D$ probabilistically.

- $x \leftarrow Smp(\rho = \{a_1, \ldots, a_k\})$: Given ring ρ, returns a sampled $x \in D$.

RTDF Properties

RTDF Properties

1. **Correctness**: If you evaluate a value with the ring function and then invert it using the trapdoor, you get a valid preimage of the output.

RTDF Properties

1. **Correctness**: If you evaluate a value with the ring function and then invert it using the trapdoor, you get a valid preimage of the output.

2. **One-wayness without trapdoor**: Without knowing any trapdoor, it's hard for a ppt adversary to find an input that maps to a given output under the ring function.

RTDF Properties

1. **Correctness**: If you evaluate a value with the ring function and then invert it using the trapdoor, you get a valid preimage of the output.

2. **One-wayness without trapdoor**: Without knowing any trapdoor, it's hard for a ppt adversary to find an input that maps to a given output under the ring function.

3. **Domain Sampling**: An adversary should not be able to tell whether a value came from the ring function’s real input-output process or was just randomly chosen from the output space.

RTDF Properties

1. **Correctness**: If you evaluate a value with the ring function and then invert it using the trapdoor, you get a valid preimage of the output.

2. **One-wayness without trapdoor**: Without knowing any trapdoor, it's hard for a ppt adversary to find an input that maps to a given output under the ring function.

3. **Domain Sampling**: An adversary should not be able to tell whether a value came from the ring function’s real input-output process or was just randomly chosen from the output space.

4. **Preimage Sampling**: An adversary shouldn’t be able to tell whether a preimage was generated using the trapdoor or sampled directly and conditioned to match the output.

RTDF Properties

1. **Correctness**: If you evaluate a value with the ring function and then invert it using the trapdoor, you get a valid preimage of the output.

2. **One-wayness without trapdoor**: Without knowing any trapdoor, it's hard for a ppt adversary to find an input that maps to a given output under the ring function.

3. **Domain Sampling**: An adversary should not be able to tell whether a value came from the ring function's real input-output process or was just randomly chosen from the output space.

4. **Preimage Sampling**: An adversary shouldn't be able to tell whether a preimage was generated using the trapdoor or sampled directly and conditioned to match the output.

5. **Anonymity**: Even if the adversary knows all trapdoors and can make inverse queries, they shouldn't be able to tell which trapdoor was used to invert the function.

One-wayness without trapdoor

Without knowing any trapdoor, it's hard for a ppt adversary to find an input that maps to a given output under the ring function.

One-wayness without trapdoor

$$\mathsf{Adv}^{\mathsf{OW}(\kappa,k)}_{\mathsf{RTDF},\mathcal{A}} := \max_{k} \left(\Pr\left[\mathsf{OW}^{\mathsf{RTDF}}_{\mathcal{A}}(\kappa,k) = 1\right]\right)$$

Without knowing any trapdoor, it's hard for a ppt adversary to find an input that maps to a given output under the ring function.

One-wayness without trapdoor

$$\mathsf{Adv}^{\mathsf{OW}(\kappa,k)}_{\mathsf{RTDF},\mathcal{A}} := \max_{k}\left(\Pr\left[\mathsf{OW}^{\mathsf{RTDF}}_{\mathcal{A}}(\kappa,k)=1\right]\right)$$

Game $\mathsf{OW}^{\mathsf{RTDF}}_{\mathcal{A}}(\kappa,k)$

1: $par \xleftarrow{\$} \mathsf{Stp}(\kappa)$
2: **for** $i \in [k]$ **do**
3: $\quad (a_i, t_i) \xleftarrow{\$} \mathsf{Gen}()$
4: **end for**
5: $\rho \leftarrow \{a_1, \ldots, a_k\}$
6: $y \xleftarrow{\$} \mathcal{R}$
7: $x' \xleftarrow{\$} \mathcal{A}(par, \rho, y)$
8: **return** 1 if $\mathsf{Eval}(\rho, x') = y$, else 0

Without knowing any trapdoor, it's hard for a ppt adversary to find an input that maps to a given output under the ring function.

Domain Sampling

An adversary should not be able to tell whether a value came from the ring function's real input-output process or was just randomly chosen from the output space.

Domain Sampling

$$\mathsf{Adv}^{\mathsf{DS}(n,\kappa)}_{\mathsf{RTDF},\mathcal{A}} := \left|\Pr\left[\mathsf{DS}^{\mathsf{RTDF}}_{\mathcal{A}}(n,\kappa) = 1\right] - \frac{1}{2}\right|$$

An adversary should not be able to tell whether a value came from the ring function's real input-output process or was just randomly chosen from the output space.

Domain Sampling

$$\mathsf{Adv}^{\mathsf{DS}(n,\kappa)}_{\mathsf{RTDF},\mathcal{A}} := \left|\Pr\left[\mathsf{DS}^{\mathsf{RTDF}}_{\mathcal{A}}(n,\kappa) = 1\right] - \frac{1}{2}\right|$$

Game $\mathsf{DS}^{\mathsf{RTDF}}_{\mathcal{A}}(n,\kappa)$

1: $par \xleftarrow{\$} \mathsf{Stp}(\kappa)$
2: **for** $i \in [n]$ **do**
3: $\quad (a_i, t_i) \xleftarrow{\$} \mathsf{Gen}()$
4: **end for**
5: $b \xleftarrow{\$} \{0,1\}$
6: $b' \xleftarrow{\$} \mathcal{A}^{\mathsf{Sample}}(par, \{a_1, \ldots, a_n\})$
7: **return** $[b' = b]$

An adversary should not be able to tell whether a value came from the ring function's real input-output process or was just randomly chosen from the output space.

Domain Sampling

$$\mathsf{Adv}^{\mathsf{DS}(n,\kappa)}_{\mathsf{RTDF},\mathcal{A}} := \left|\Pr\left[\mathsf{DS}^{\mathsf{RTDF}}_{\mathcal{A}}(n,\kappa)=1\right]-\frac{1}{2}\right|$$

Game $\mathsf{DS}^{\mathsf{RTDF}}_{\mathcal{A}}(n,\kappa)$

$par \xleftarrow{\$} \mathsf{Stp}(\kappa)$
for $i \in [n]$ **do**
  $(a_i, t_i) \xleftarrow{\$} \mathsf{Gen}()$
end for
$b \xleftarrow{\$} \{0,1\}$
$b' \xleftarrow{\$} \mathcal{A}^{\mathsf{Sample}}(par, \{a_1,\dots,a_n\})$
return $[b' = b]$

Oracle $\mathsf{Sample}(\rho = \{a_{i_1},\dots,a_{i_k}\})$

if $k > \kappa$ **or** $\exists a(a \in \rho \wedge a \notin \{a_1,\dots,a_n\})$ **then**
  return $\perp$
end if
if $b = 0$ **then**
  $x \xleftarrow{\$} \mathsf{Smp}(\rho)$
  $y \leftarrow \mathsf{Eval}(\rho, x)$
else
  $y \xleftarrow{\$} \mathcal{R}$
end if
return y

An adversary should not be able to tell whether a value came from the ring function's real input-output process or was just randomly chosen from the output space.

Preimage Sampling

An adversary shouldn't be able to tell whether a preimage was generated using the trapdoor or sampled directly and conditioned to match the output.

Preimage Sampling

$$\mathsf{Adv}^{\mathsf{PS}(n,\kappa)}_{\mathsf{RTDF},\mathcal{A}} := \left|\Pr\left[\mathsf{PS}^{\mathsf{RTDF}}_{\mathcal{A}}(n,\kappa) = 1\right] - \frac{1}{2}\right|$$

An adversary shouldn't be able to tell whether a preimage was generated using the trapdoor or sampled directly and conditioned to match the output.

Preimage Sampling

$$\mathsf{Adv}^{\mathsf{PS}(n,\kappa)}_{\mathsf{RTDF},\mathcal{A}} := \left|\Pr\left[\mathsf{PS}^{\mathsf{RTDF}}_{\mathcal{A}}(n,\kappa) = 1\right] - \frac{1}{2}\right|$$

Game $\mathsf{PS}^{\mathsf{RTDF}}_{\mathcal{A}}(n,\kappa)$

$par \xleftarrow{\$} \mathsf{Stp}(\kappa)$
for $i \in [n]$ **do**
    $(a_i, t_i) \xleftarrow{\$} \mathsf{Gen}()$
end for
$b \xleftarrow{\$} \{0,1\}$
$b' \xleftarrow{\$} \mathcal{A}^{\mathsf{Challenge}}(par, \{a_1, \ldots, a_n\})$
return $[b' = b]$

An adversary shouldn't be able to tell whether a preimage was generated using the trapdoor or sampled directly and conditioned to match the output.

Preimage Sampling

$$\mathsf{Adv}^{\mathsf{PS}(n,\kappa)}_{\mathsf{RTDF},\mathcal{A}} := \left|\Pr\left[\mathsf{PS}^{\mathsf{RTDF}}_{\mathcal{A}}(n,\kappa) = 1\right] - \frac{1}{2}\right|$$

Game $\mathsf{PS}^{\mathsf{RTDF}}_{\mathcal{A}}(n,\kappa)$

$par \xleftarrow{\$} \mathsf{Stp}(\kappa)$
for $i \in [n]$ **do**
  $(a_i, t_i) \xleftarrow{\$} \mathsf{Gen}()$
end for
$b \xleftarrow{\$} \{0,1\}$
$b' \xleftarrow{\$} \mathcal{A}^{\mathsf{Challenge}}(par, \{a_1, \ldots, a_n\})$
return $[b' = b]$

Oracle $\mathsf{Challenge}(\rho = \{a_{i_1}, \ldots, a_{i_k}\}, y \in \mathcal{R})$

if $k > \kappa$ **or** $\exists a(a \in \rho \wedge a \notin \{a_1, \ldots, a_n\})$ **then**
  return $\perp$
end if
if $b = 0$ **then**
  $t \xleftarrow{\$} \{\mu(a) \mid a \in \rho\}$
  $x \xleftarrow{\$} \mathsf{Inv}(t, \rho, y)$
else
  Sample $x \xleftarrow{\$} \mathsf{Smp}(\rho)$ until $f_\rho(x) = y$
end if
return x

An adversary shouldn't be able to tell whether a preimage was generated using the trapdoor or sampled directly and conditioned to match the output.

Anonymity

Even if the adversary knows all trapdoors and can make inverse queries, they shouldn't be able to tell which trapdoor was used to invert the function.

Anonymity

$$\mathsf{Adv}_{\mathrm{RTDF},\mathcal{A}}^{(n,\kappa,Q_{\mathsf{OInv}})\text{-MC-Ano}} := \left|\Pr[(n,\kappa,Q_{\mathsf{OInv}})\text{-MC-Ano}_{\mathrm{RTDF}}(\mathcal{A}) \to 1] - \frac{1}{2}\right|$$

Even if the adversary knows all trapdoors and can make inverse queries, they shouldn't be able to tell which trapdoor was used to invert the function.

Anonymity

$$\mathsf{Adv}_{\mathrm{RTDF},\mathcal{A}}^{(n,\kappa,Q_{\mathsf{OInv}})\text{-MC-Ano}} := \left|\Pr[(n,\kappa,Q_{\mathsf{OInv}})\text{-MC-Ano}_{\mathrm{RTDF}}(\mathcal{A}) \to 1] - \frac{1}{2}\right|$$

Game $(n,\kappa,Q_{\mathrm{OInv}})\text{-MC-Ano}_{\mathrm{RTDF}}(\mathcal{A})$

$par \xleftarrow{\$} \mathrm{Stp}(\kappa)$
for $i \in [n]$ **do**
    $(a_i, t_i) \xleftarrow{\$} \mathrm{Gen}$
end for
$b \xleftarrow{\$} \{0,1\}$
$b' \xleftarrow{\$} \mathcal{A}^{\mathrm{OInv}}(par, (a_1,t_1),\ldots,(a_n,t_n))$
return $[b = b']$

Even if the adversary knows all trapdoors and can make inverse queries, they shouldn't be able to tell which trapdoor was used to invert the function.

Anonymity

$$\mathsf{Adv}^{(n,\kappa,Q_{\mathsf{OInv}})\text{-MC-Ano}}_{\mathrm{RTDF},\mathcal{A}} := \left|\Pr[(n,\kappa,Q_{\mathsf{OInv}})\text{-MC-Ano}_{\mathrm{RTDF}}(\mathcal{A}) \to 1] - \frac{1}{2}\right|$$

Game $(n,\kappa,Q_{\mathrm{OInv}})\text{-MC-Ano}_{\mathrm{RTDF}}(\mathcal{A})$

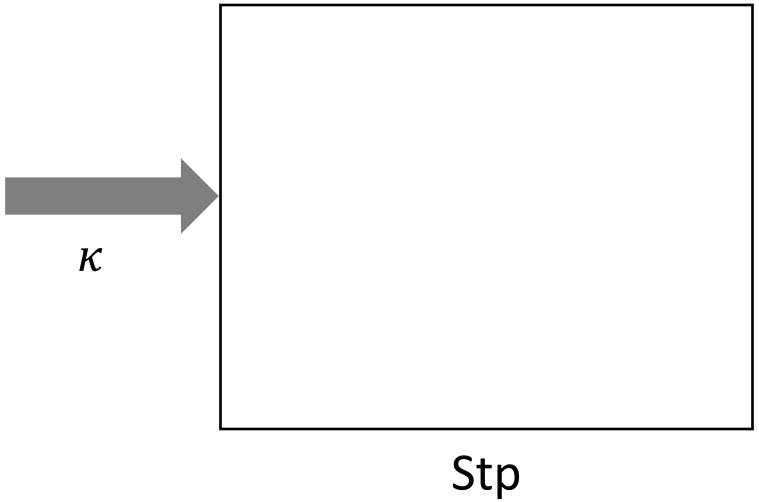
$par \xleftarrow{\$} \mathrm{Stp}(\kappa)$
for $i \in [n]$ **do**
  $(a_i, t_i) \xleftarrow{\$} \mathrm{Gen}$
end for
$b \xleftarrow{\$} \{0,1\}$
$b' \xleftarrow{\$} \mathcal{A}^{\mathrm{OInv}}(par, (a_1,t_1), \ldots, (a_n,t_n))$
return $[b = b']$

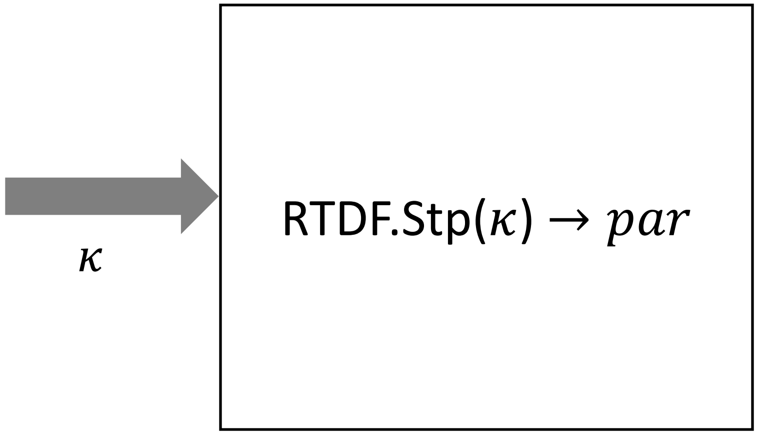
Oracle $\mathrm{OInv}(i_0 \in [n], i_1 \in [n], \rho = \{a_1, \ldots, a_{k \le n}\}, x)$

if $(\rho \subseteq \{a_1, \ldots, a_n\}) \wedge (a_{i_0} \in \rho) \wedge (a_{i_1} \in \rho)$ **then**
  $y \leftarrow \mathrm{Eval}(\rho, x)$
  return y
else
  return $\perp$
end if

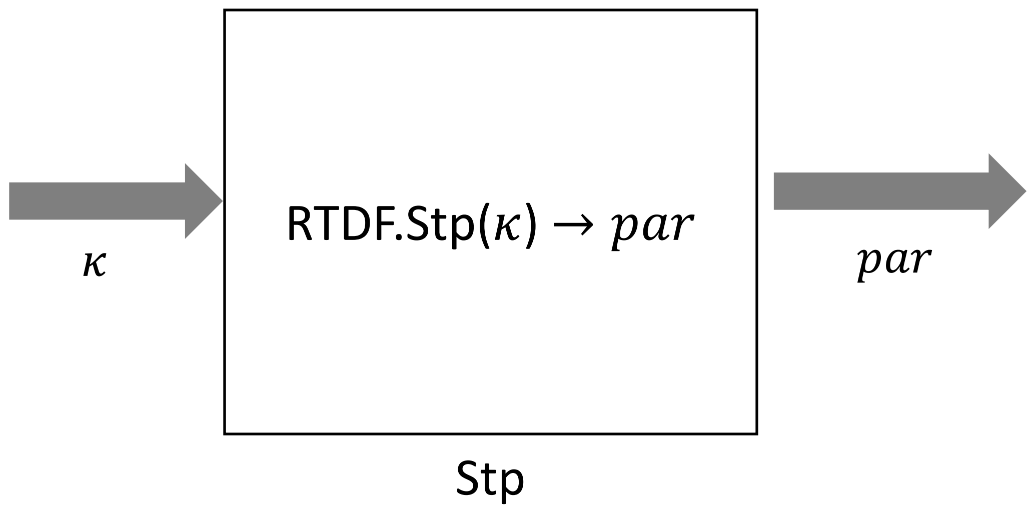
Even if the adversary knows all trapdoors and can make inverse queries, they shouldn't be able to tell which trapdoor was used to invert the function.

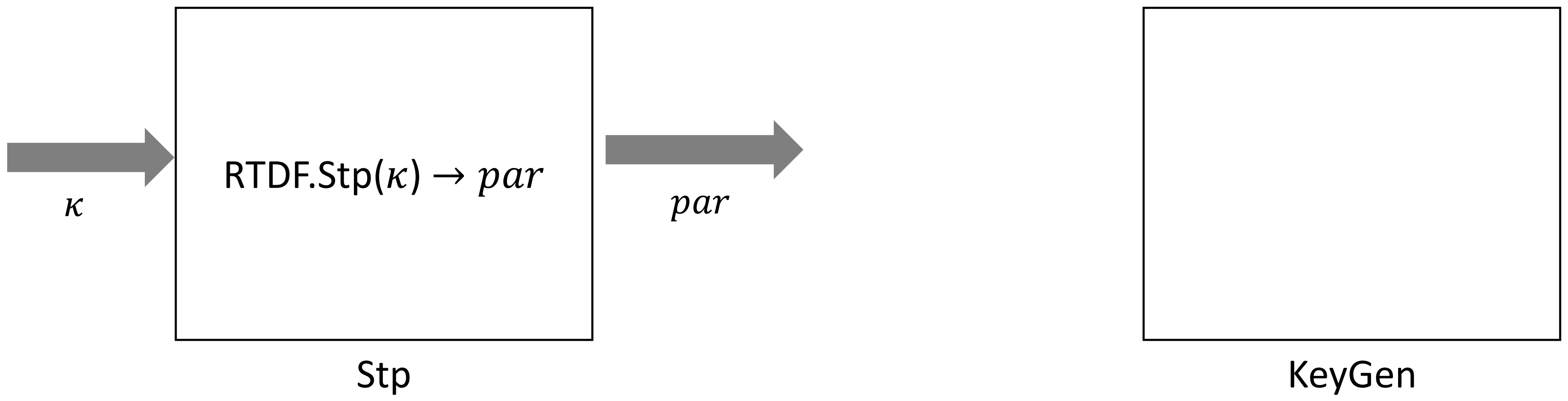
RTDF → RSig


κ
Stp



Stp


κ
RTDF.Stp(κ) $\rightarrow par$
par
Stp


κ
RTDF.Stp$(\kappa) \rightarrow par$
par
Stp
KeyGen

κ

RTDF.Stp(κ) $\rightarrow par$

par

Stp

1. Stp(κ) $\rightarrow par$

KeyGen

κ

RTDF.Stp(κ) $\rightarrow par$

par

Stp

1. Stp(κ) $\rightarrow par$
2. RTDF.Gen() $\rightarrow (a, t)$

KeyGen

κ

RTDF.Stp(κ) $\rightarrow par$

par

Stp

1. Stp(κ) $\rightarrow par$
2. RTDF.Gen() $\rightarrow (a, t)$
3. $pk \coloneqq a$

KeyGen

κ

RTDF.Stp(κ) $\rightarrow par$

par

Stp

1. Stp(κ) $\rightarrow par$
2. RTDF.Gen() $\rightarrow (a, t)$
3. $pk \coloneqq a$
4. s$k \coloneqq t$

KeyGen

κ → RTDF.Stp(κ) $\rightarrow par$ → par

Stp

1. Stp(κ) $\rightarrow par$
2. RTDF.Gen() $\rightarrow (a, t)$
3. $pk \coloneqq a$
4. sk $\coloneqq t$

→ (pk, sk)

KeyGen

κ

RTDF.Stp(κ) $\to par$

par

Stp

1. Stp(κ) $\to par$
2. RTDF.Gen() $\to (a, t)$
3. $pk \coloneqq a$
4. sk $\coloneqq t$

(pk, sk)

KeyGen

Sign

κ

RTDF.Stp(κ) $\rightarrow par$

par

Stp

1. Stp(κ) $\rightarrow par$
2. RTDF.Gen() $\rightarrow (a, t)$
3. $pk \coloneqq a$
4. sk $\coloneqq t$

(pk, sk)

KeyGen

sk

ρ

m

Sign

κ → **Stp** → par

RTDF.Stp$(\kappa) \rightarrow par$

Stp

$\rightarrow (pk, sk)$

1. Stp$(\kappa) \rightarrow par$
2. RTDF.Gen() $\rightarrow (a, t)$
3. $pk \coloneqq a$
4. sk $\coloneqq t$

KeyGen

Inputs: sk, ρ, m

1. $H(m) \rightarrow y \in R$

Sign

κ

RTDF.Stp(κ) $\rightarrow$ par

par

Stp

1. Stp(κ) $\rightarrow$ par
2. RTDF.Gen() $\rightarrow$ (a, t)
3. $pk \coloneqq a$
4. sk $\coloneqq$ t

(pk, sk)

KeyGen

sk

ρ

m

1. $H(m) \rightarrow y \in R$
2. RTDF.Inv(sk, ρ, y) $\rightarrow$ x

Sign

κ

RTDF.Stp(κ) $\rightarrow par$

par

Stp

1. Stp(κ) $\rightarrow par$
2. RTDF.Gen() $\rightarrow (a, t)$
3. $pk \coloneqq a$
4. sk $\coloneqq t$

(pk, sk)

KeyGen

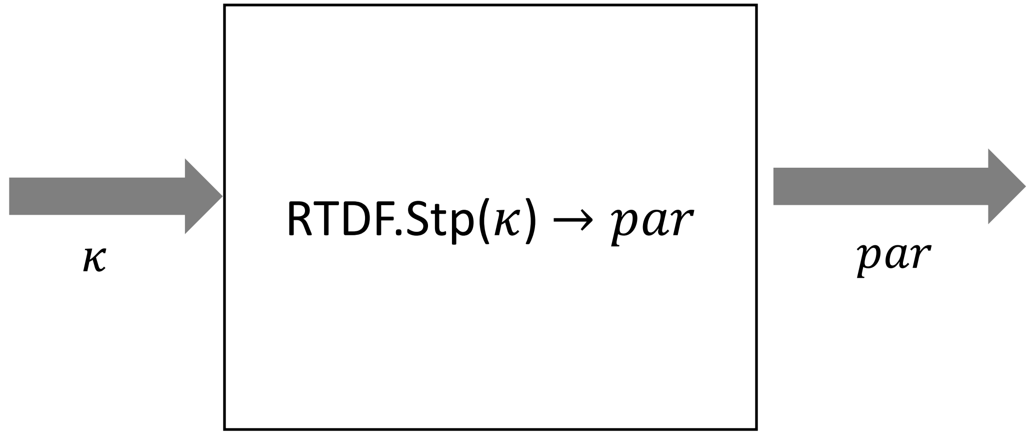
sk

ρ

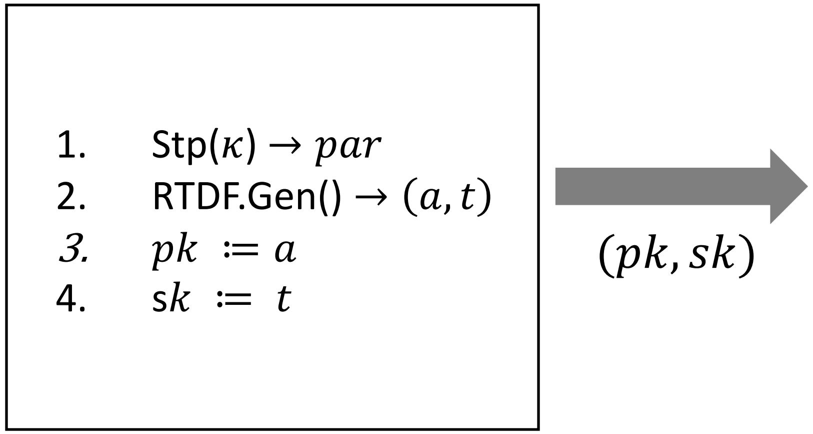
m

1. $H(m) \rightarrow y \in R$
2. RTDF.Inv(sk, ρ, y) $\rightarrow x$
3. $\sigma \coloneqq x$

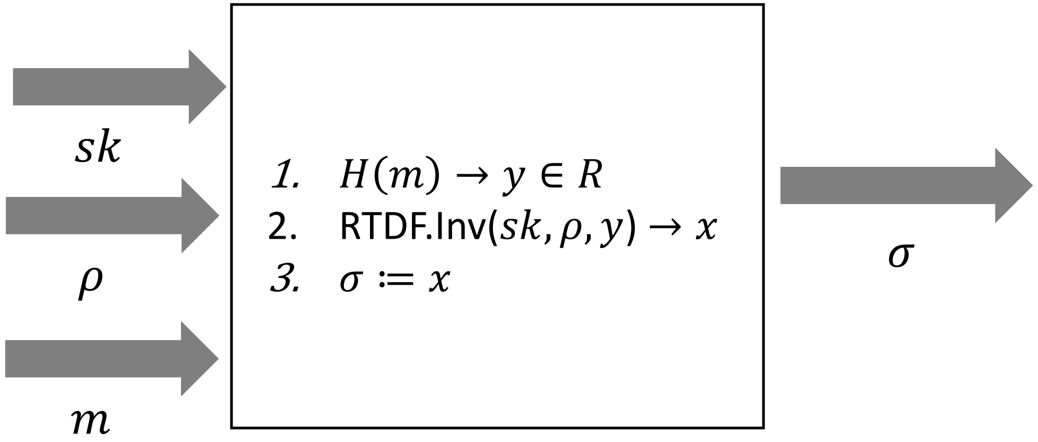
Sign



Stp



KeyGen



Sign

κ

RTDF.Stp(κ) $\rightarrow par$

par

Stp

1. Stp(κ) $\rightarrow par$
2. RTDF.Gen() $\rightarrow (a, t)$
3. $pk \coloneqq a$
4. sk $\coloneqq t$

(pk, sk)

KeyGen

sk

ρ

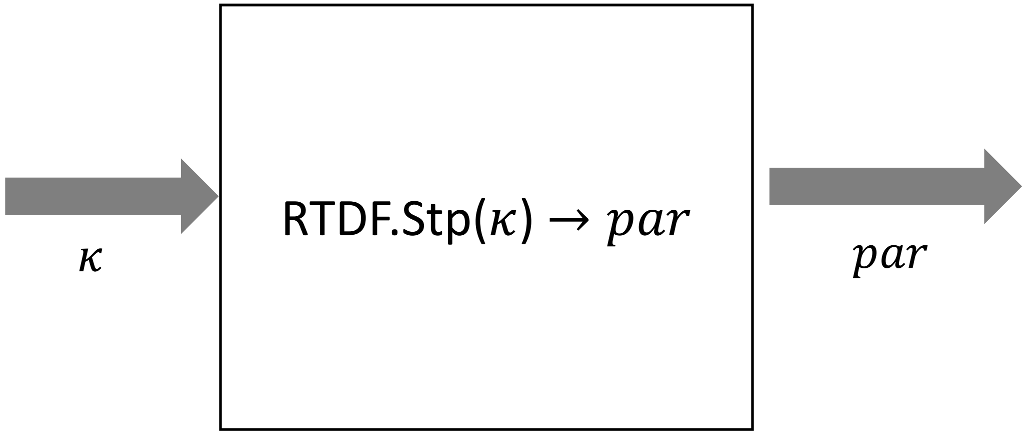
m

1. $H(m) \rightarrow y \in R$
2. RTDF.Inv(sk, ρ, y) $\rightarrow x$
3. $\sigma \coloneqq x$

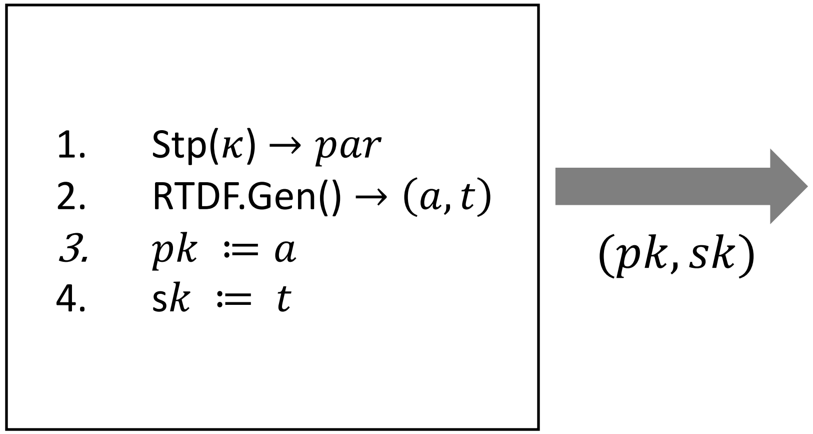
σ

Sign

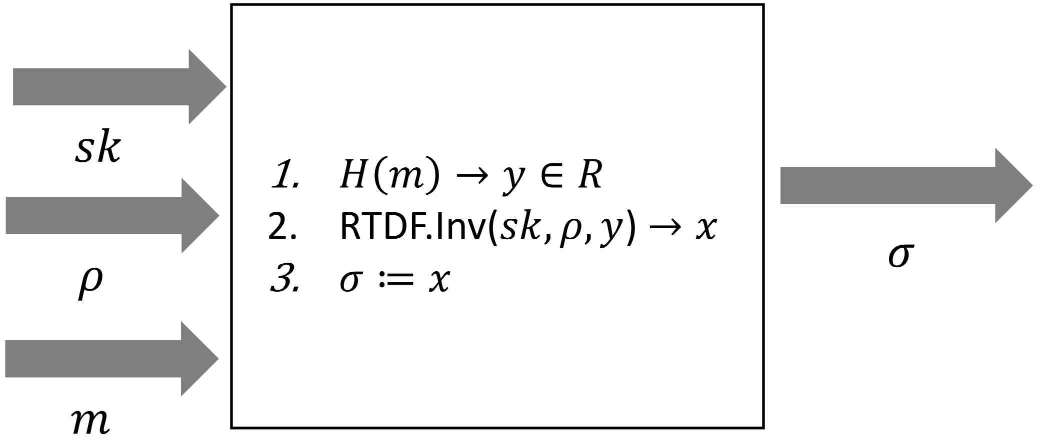
Sign



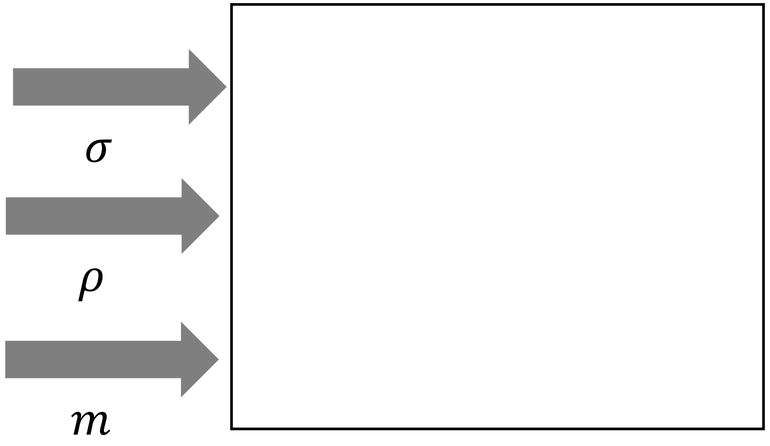
Stp



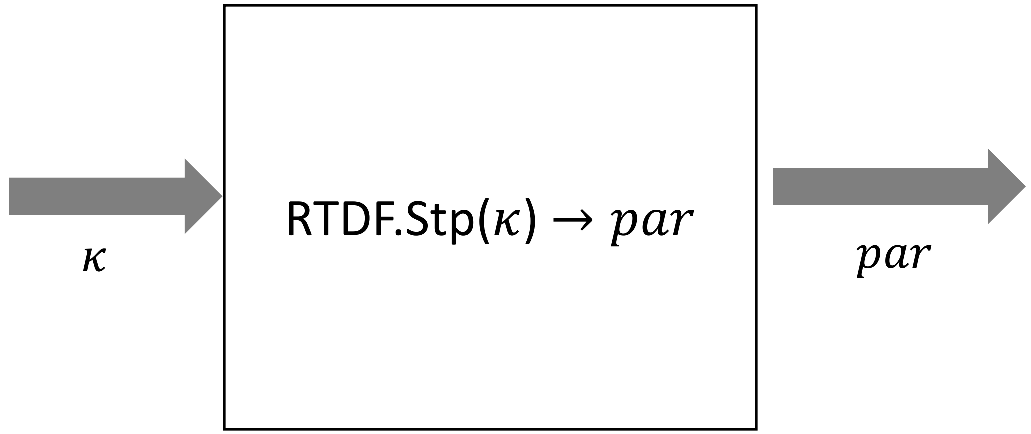
KeyGen



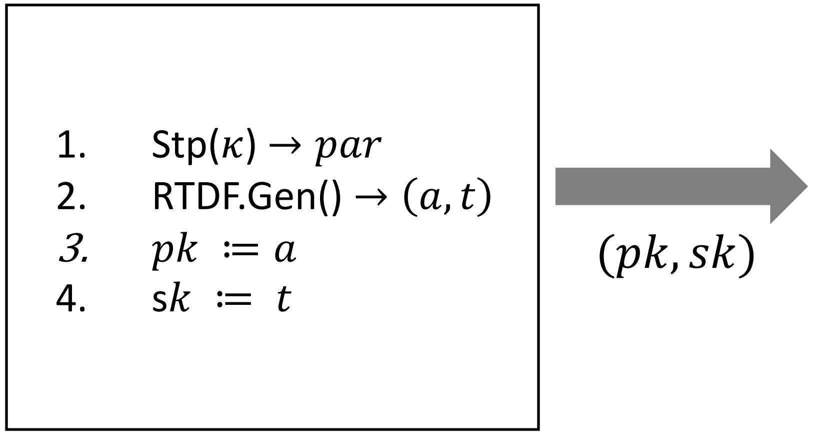
Sign



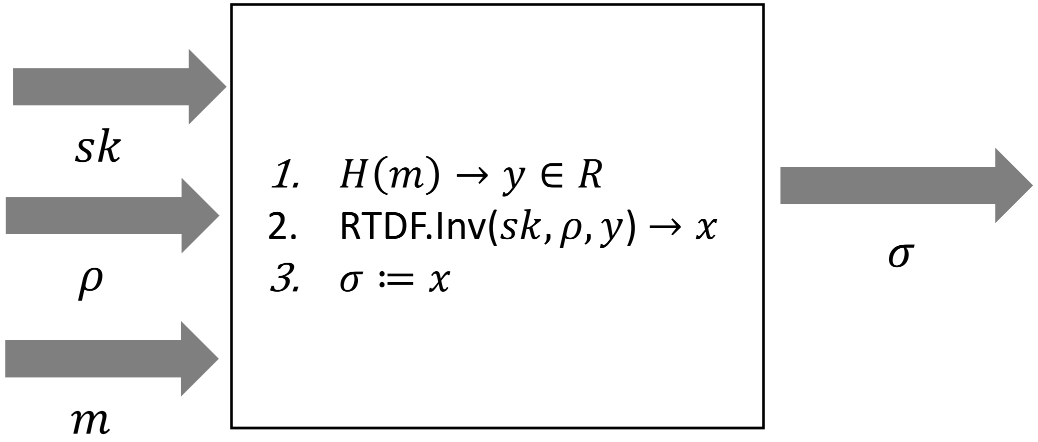
Sign



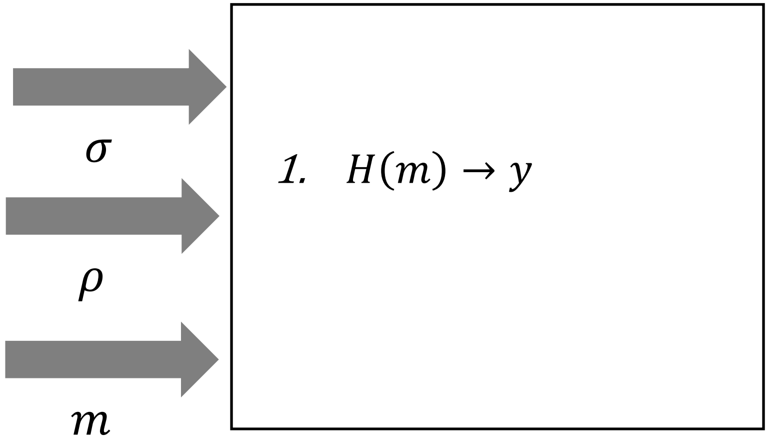
Stp



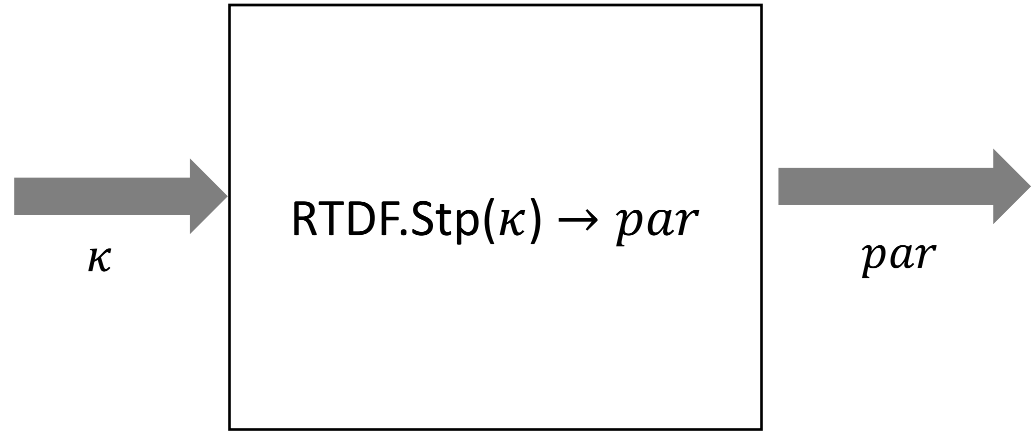
KeyGen



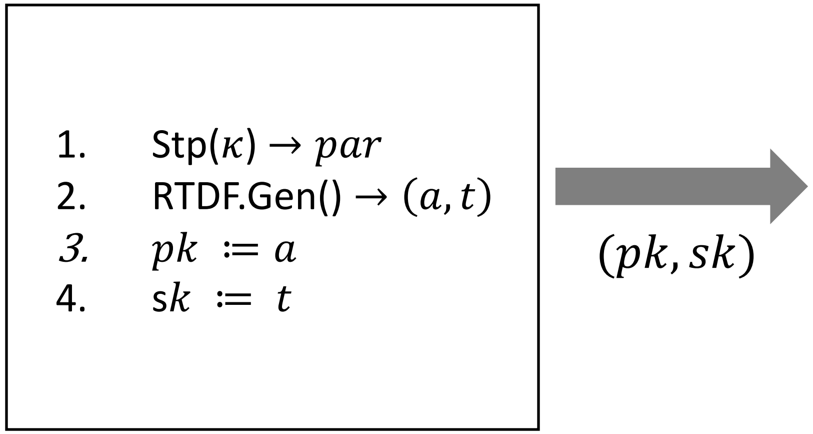
Sign



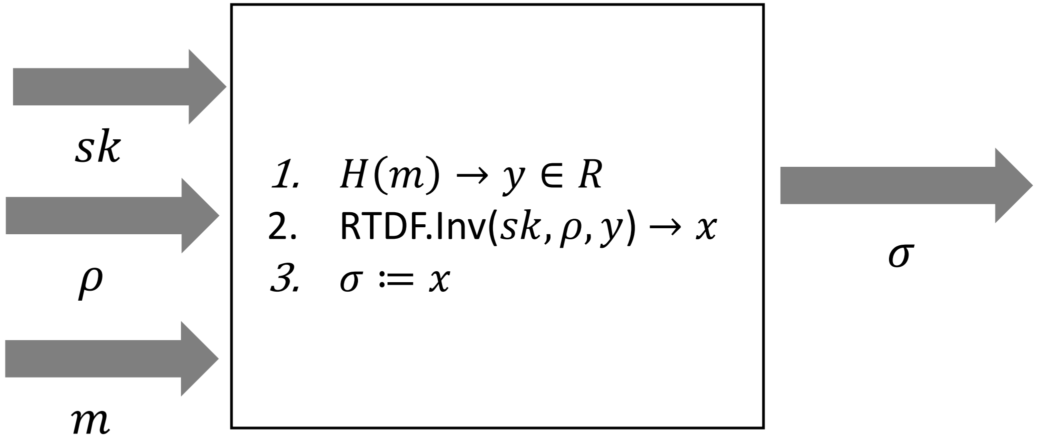
Sign



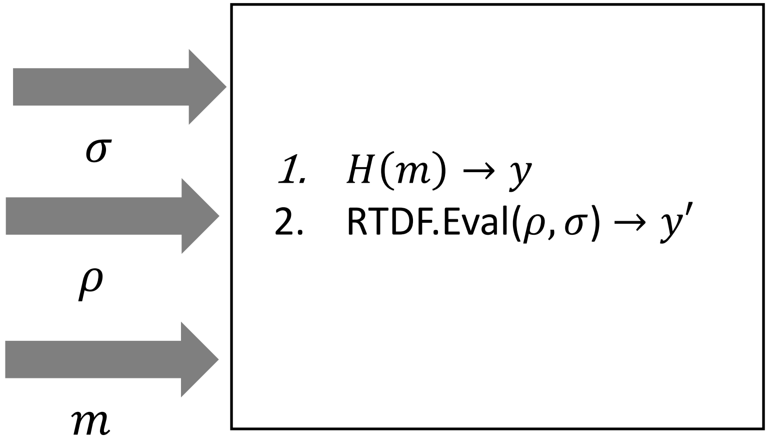
Stp



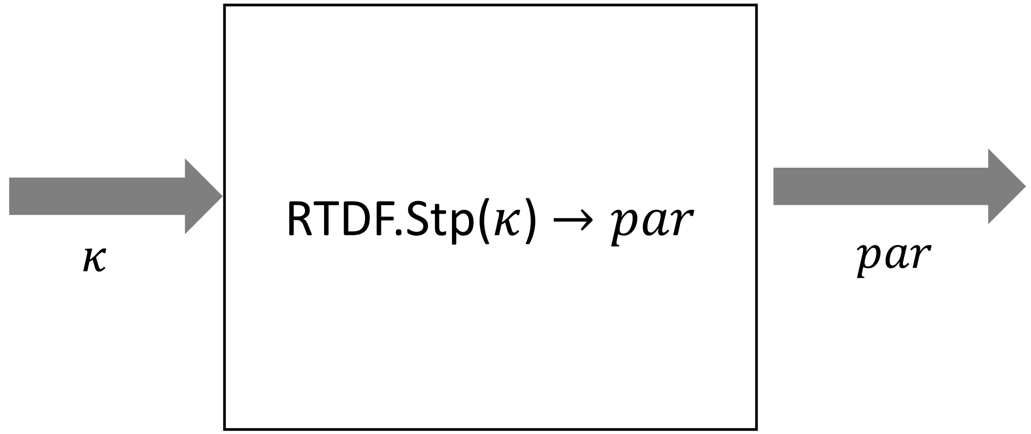
KeyGen



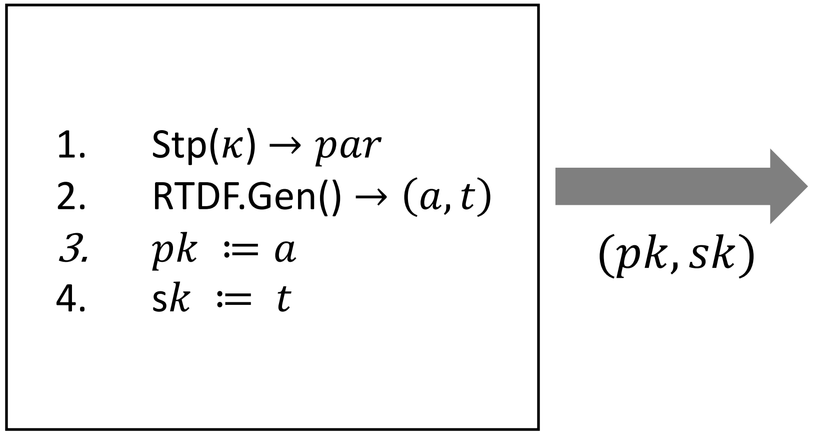
Sign



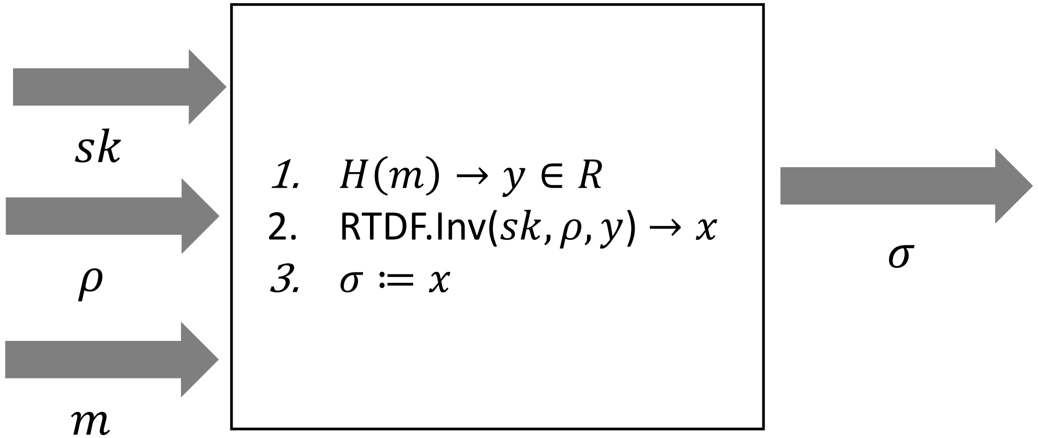
Sign



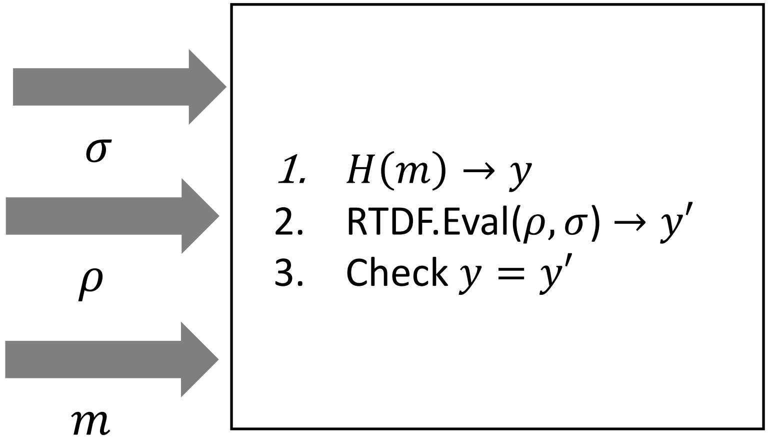
Stp



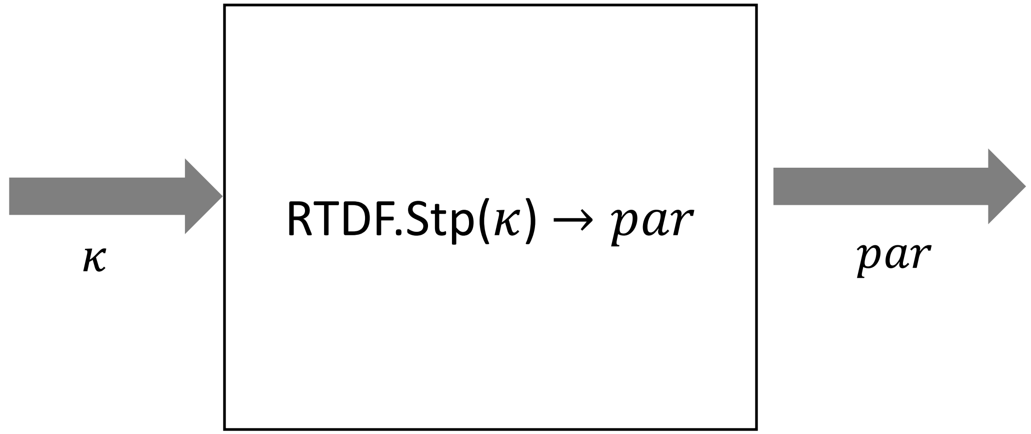
KeyGen



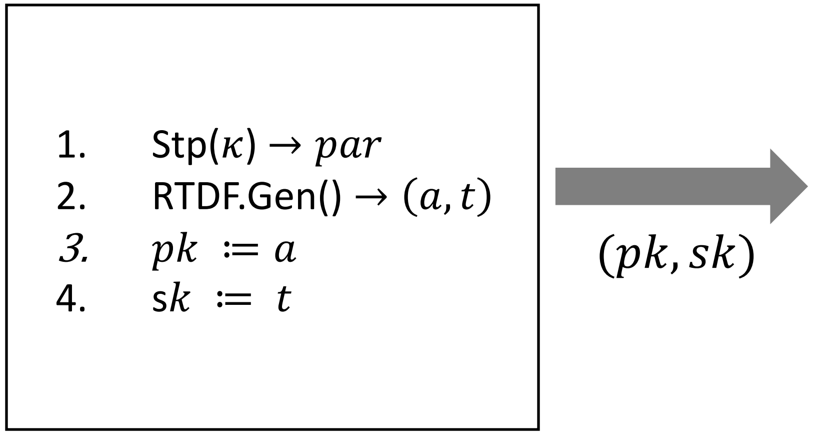
Sign



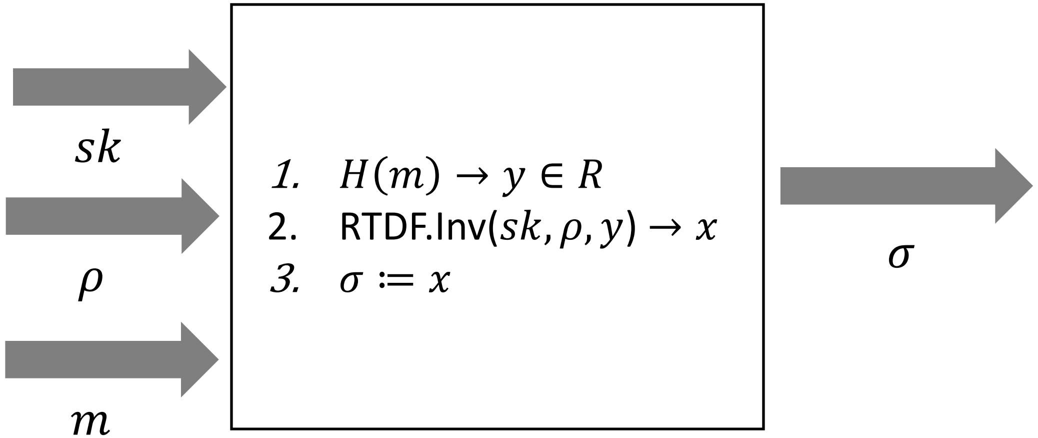
Sign



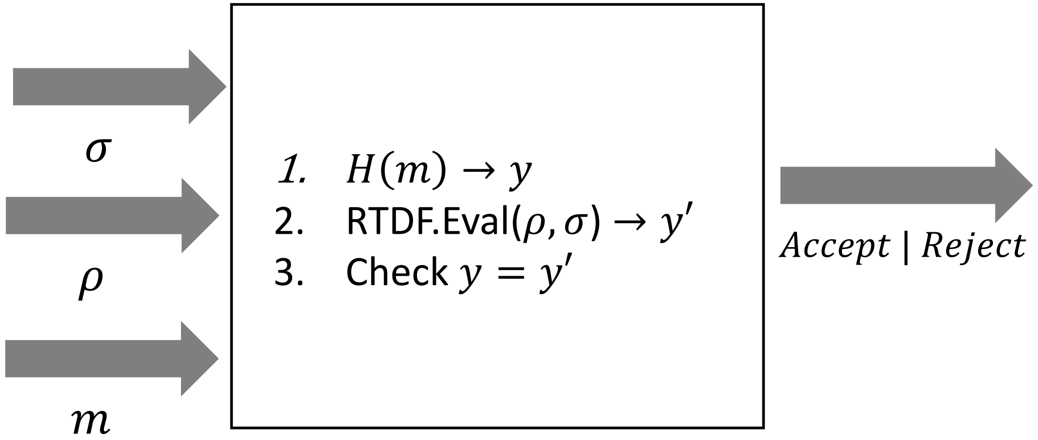
Stp



KeyGen



Sign



Sign

Summary

- Introduced a novel abstraction—Ring Trapdoor Functions (RTDFs)—to construct efficient ring signature schemes, especially in lattice-based settings.

Summary

- Introduced a novel abstraction—Ring Trapdoor Functions (RTDFs)—to construct efficient ring signature schemes, especially in lattice-based settings.

- Clearly defined RTDF syntax and security properties, supported by game-based concrete security proofs.

Summary

- Introduced a novel abstraction—Ring Trapdoor Functions (RTDFs)—to construct efficient ring signature schemes, especially in lattice-based settings.

- Clearly defined RTDF syntax and security properties, supported by game-based concrete security proofs.

- RTDF framework provides a more flexible and expressive model than the earlier BK10 definition, enabling support for advanced constructions like GANDALF.

Summary

- Introduced a novel abstraction—Ring Trapdoor Functions (RTDFs)—to construct efficient ring signature schemes, especially in lattice-based settings.
- Clearly defined RTDF syntax and security properties, supported by game-based concrete security proofs.
- RTDF framework provides a more flexible and expressive model than the earlier BK10 definition, enabling support for advanced constructions like GANDALF.
- RTDF allows improvements (e.g., scaling ring size) at the primitive level, propagating to ring signatures without redesign.

Summary

- Introduced a novel abstraction—Ring Trapdoor Functions (RTDFs)—to construct efficient ring signature schemes, especially in lattice-based settings.
- Clearly defined RTDF syntax and security properties, supported by game-based concrete security proofs.
- RTDF framework provides a more flexible and expressive model than the earlier BK10 definition, enabling support for advanced constructions like GANDALF.
- RTDF allows improvements (e.g., scaling ring size) at the primitive level, propagating to ring signatures without redesign.
- RTDF serves as a building block for other anonymity-preserving primitives, promoting primitive reusability in cryptographic design.