Rotatable Zero Knowledge Sets

Post Compromise Secure Auditable Dictionaries with application to Key Transparency

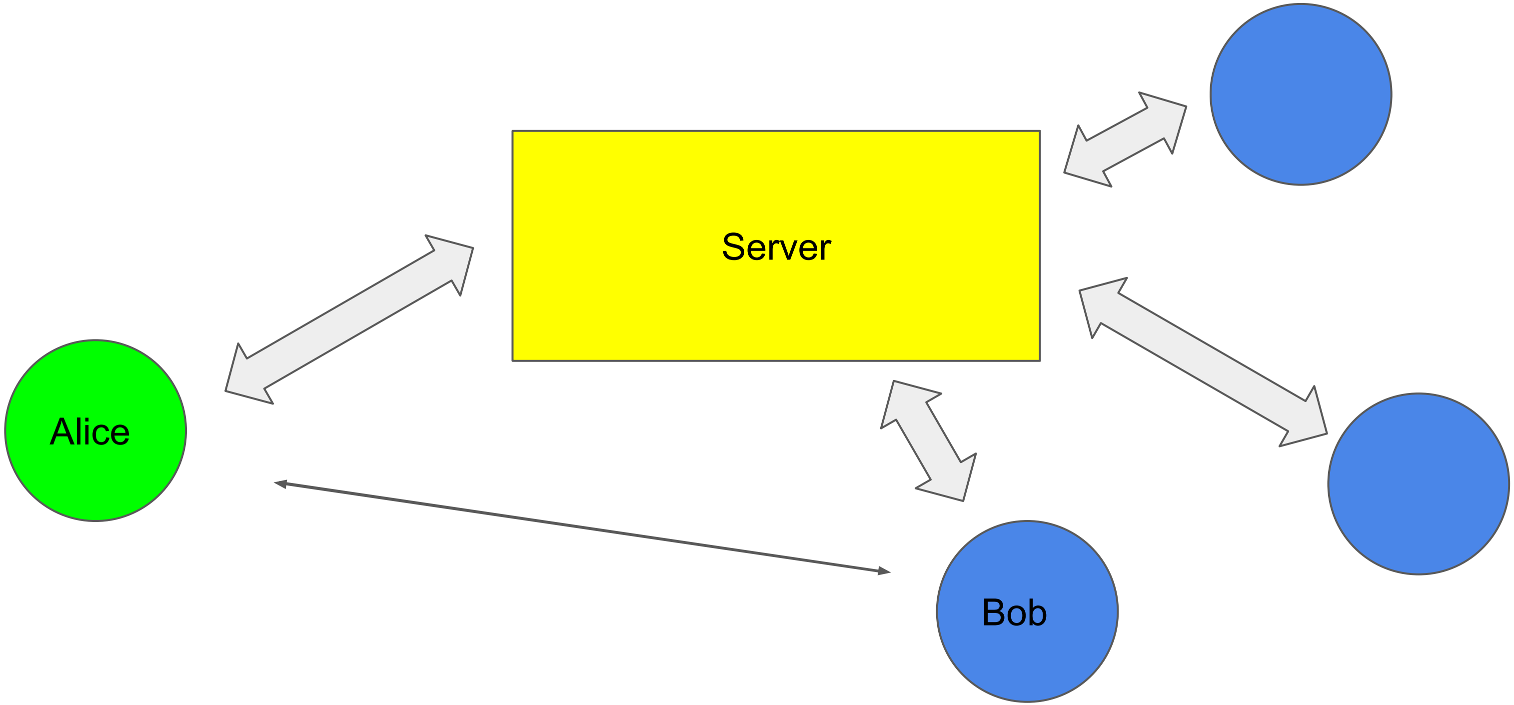
Brian Chen[1], Yevgeniy Dodis[2], Esha Ghosh[3], **Eli Goldin**[2], Balachandar Kesavan[1], Antonio Marcedone[1], Merry Ember Mou[1]

1. Zoom
2. NYU
3. Microsoft Research

Outline

- Motivation (E2EE, Key Transparency, Authenticated Dictionaries)
- Definition of underlying primitive (Rotatable Verifiable Random Function)
- RVRF construction
- Sketch of RVRF zero knowledge proof
- Open Questions

End-to-End Encrypted (E2EE) Communication Systems



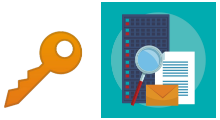
End-to-End Encrypted (E2EE) Communication Systems

Key Transparency/Auditable Dictionaries

Users can verify their public key is stored correctly using proof $\boldsymbol{\pi}$ and commitment **com**

Alice	MIIBCg
Bob	6Sj/6L
Carol	+xGZ/w

Identity

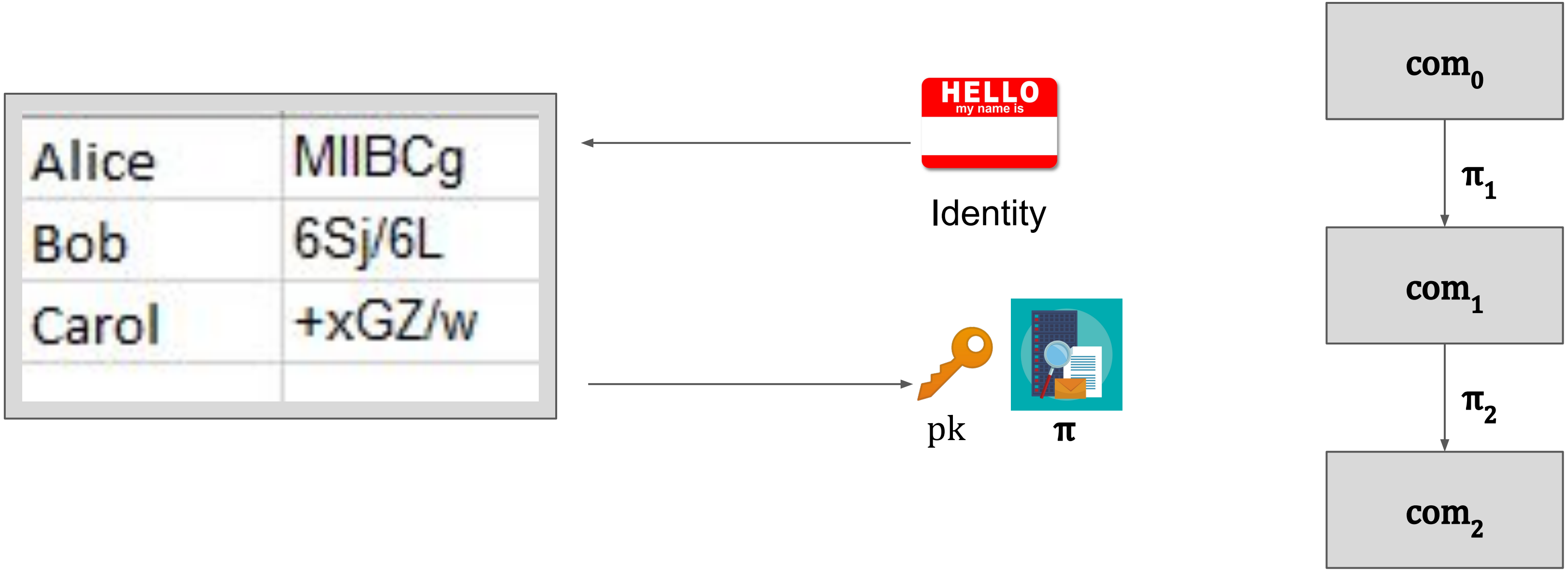


pk $\boldsymbol{\pi}$

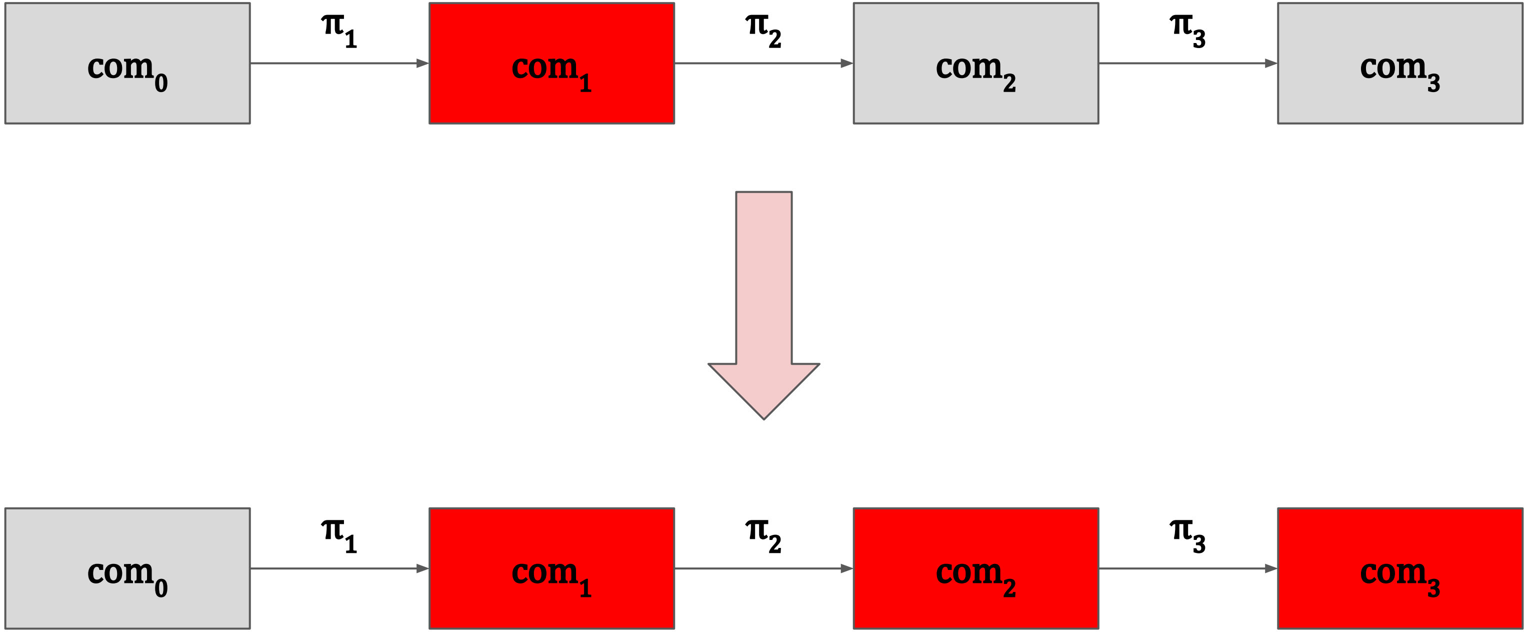
$\mathbf{com}_0$ → $\mathbf{com}_1$ → $\mathbf{com}_2$

Key Transparency/Auditable Dictionaries

Auditors can verify commitments are updated correctly using π_i



Privacy Totally Lost on Corruption



Our Contribution

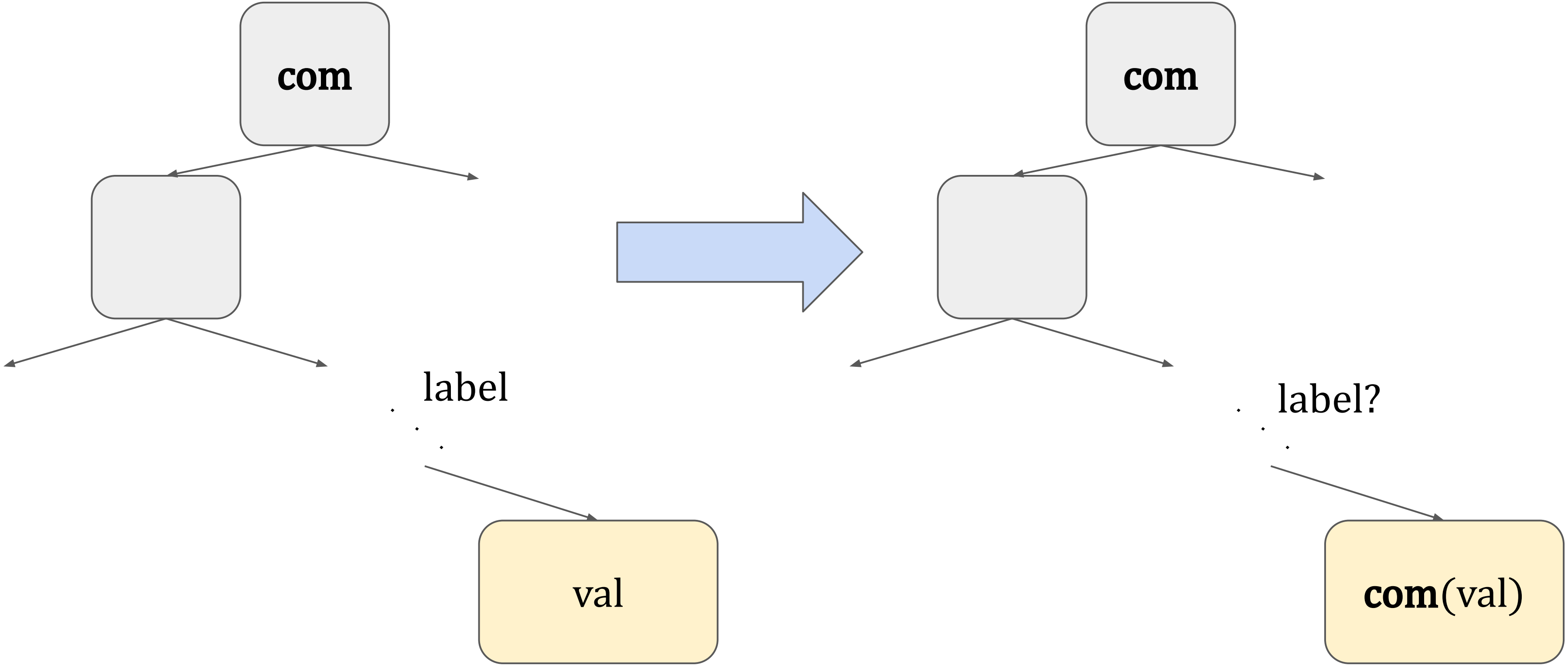
Post-Compromise Security (PCS)

Modeling: Rotatable Zero Knowledge Sets (RZKS)

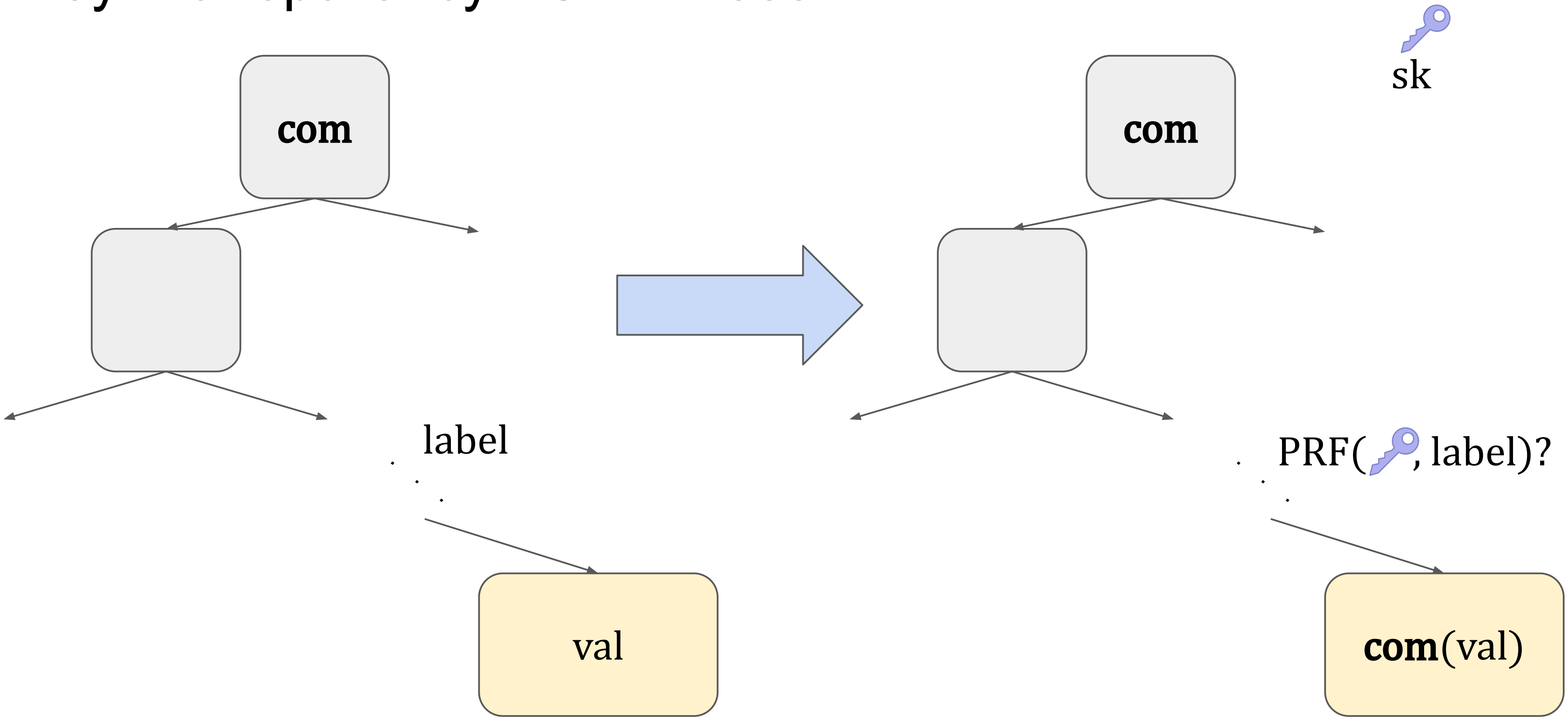
Rotatable Verifiable Random Functions (RVRF)

(Also: extension proofs, stronger soundness)

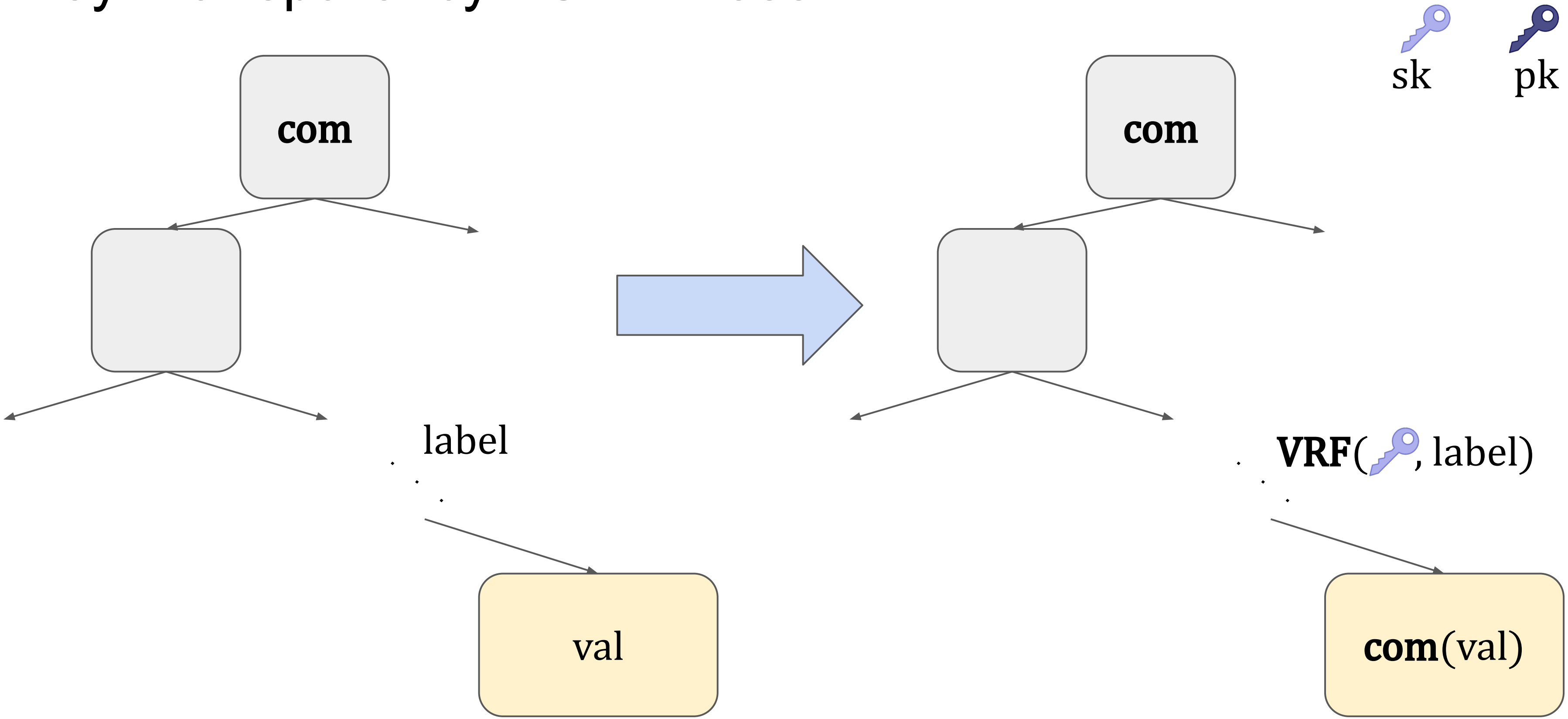
Key Transparency - SEEMless



Key Transparency - SEEMless



Key Transparency - SEEMless



VRF

KeyGen → sk pk

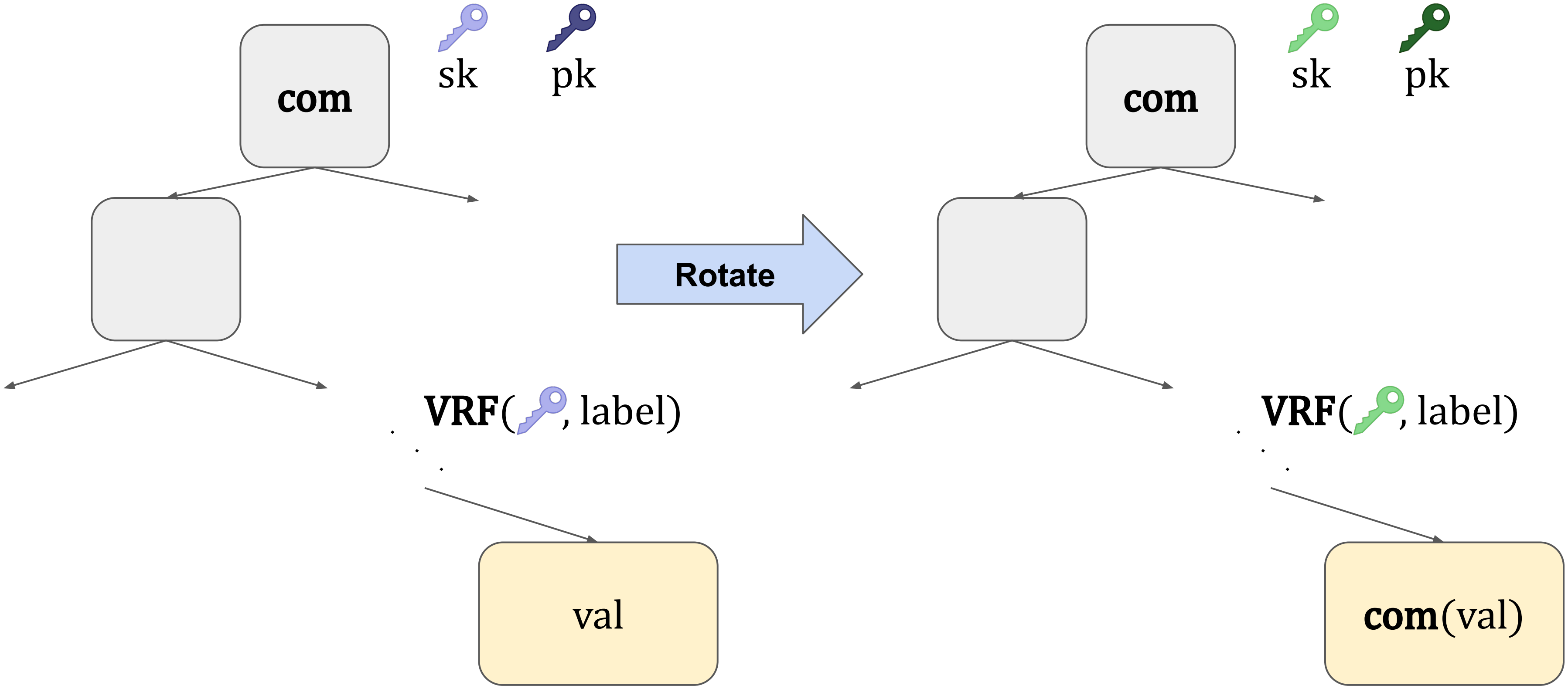
Query(, x) → (x , π)

Verify(, x, y, π) → 0/1

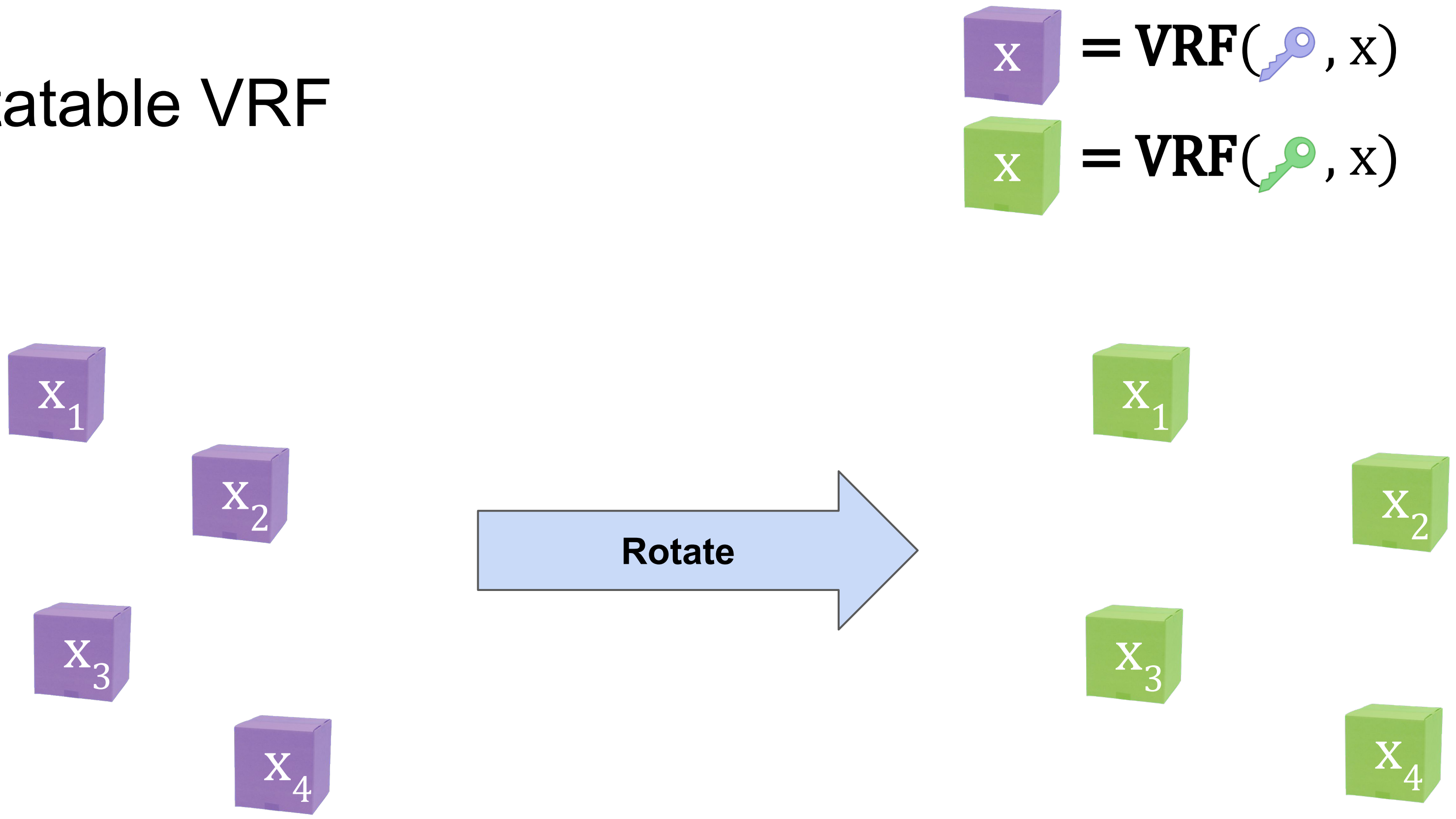
Uniqueness: Verify → 1 iff y = x

Pseudorandomness x looks random

Key Rotation



Rotatable VRF



Rotatable VRF

KeyGen →  sk ,  pk

Query(, x) → (

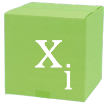
, π)

Verify(, x, y, π) → 0/1

Rotate(, x_1, ..., x_k) →  sk ,  pk , π_{rot}

VerRotate(, , y_1, ..., y_k, y_1', ..., y_k') → 0/1

VerRotate → 1 iff y_i = 

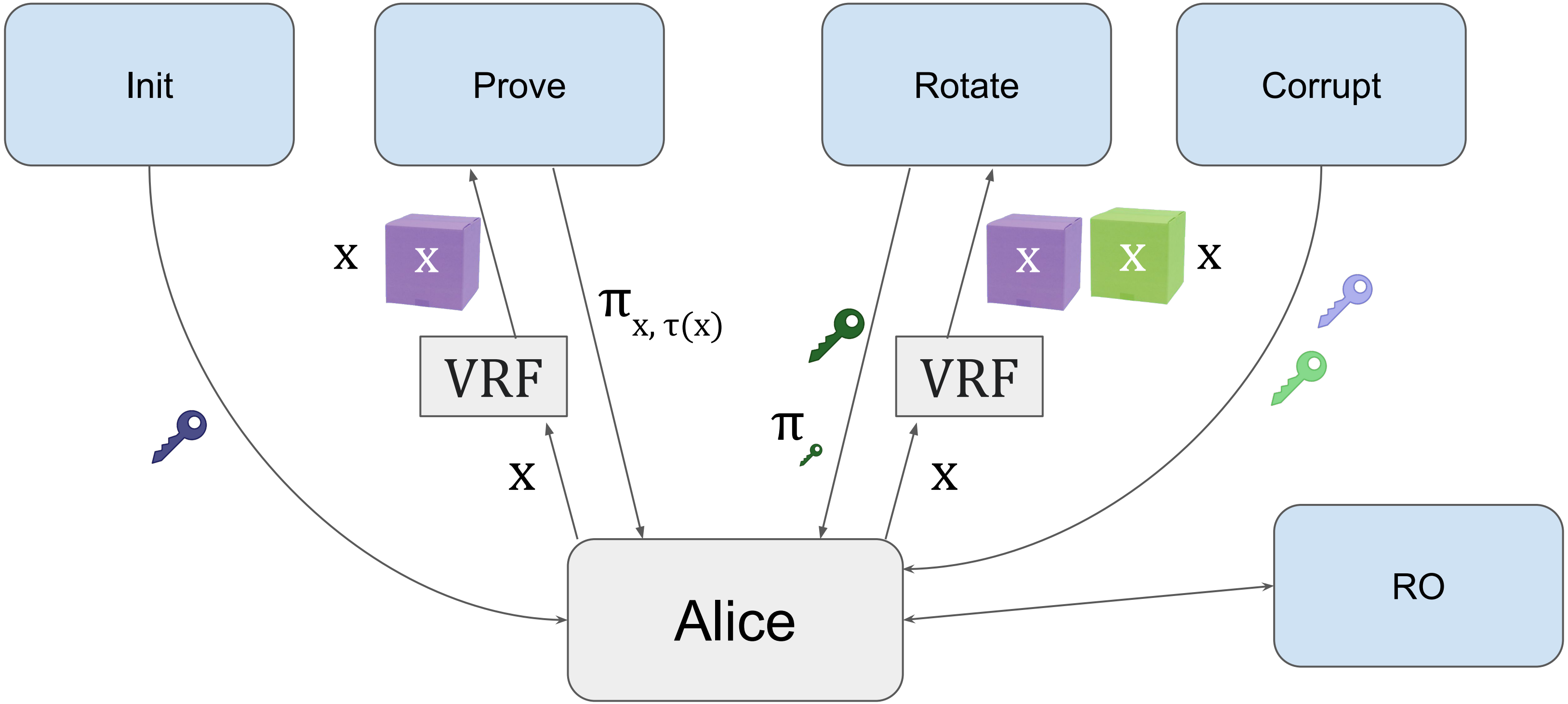
and y_i' = 

Rotatable VRF Security

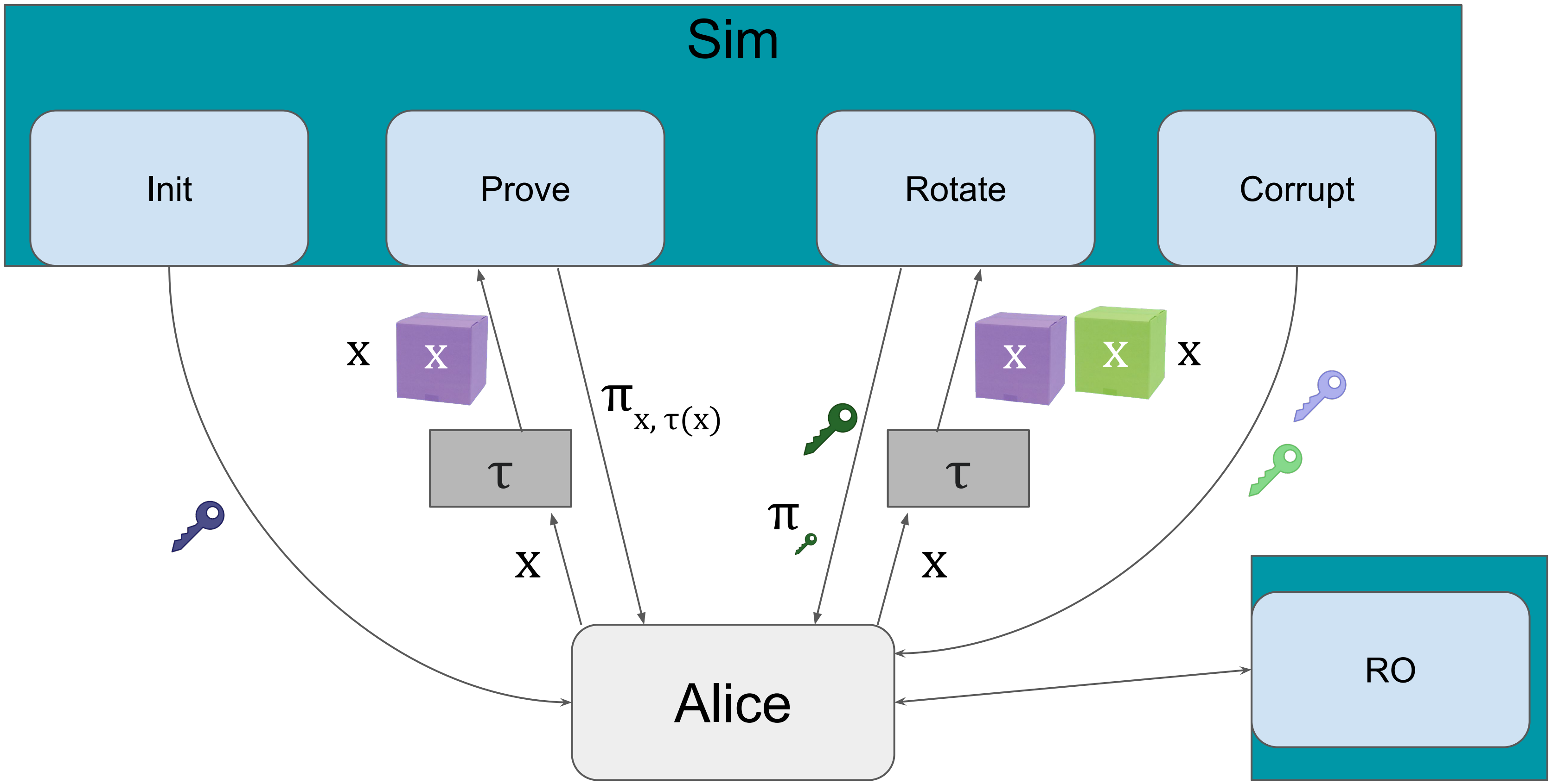
Uniqueness $\Rightarrow$ Extractability

Pseudorandomness $\Rightarrow$ Zero-Knowledge

Rotatable VRF Security - Zero Knowledge



Rotatable VRF Security - Zero Knowledge



DDH Tuples

(g, h, g^a, h^a)

[ChaumPederson93]: ZK proof

Standard VRF [GRPV22]

$\mathrm{F} : \mathrm{M} \rightarrow \mathrm{G}$ (random oracle) g (group generator)

$\mathrm{KeyGen} \rightarrow \mathrm{sk}, \mathrm{pk} = \mathrm{g}^{\mathrm{sk}}$

$\mathrm{VRF}(\mathrm{sk}, \mathrm{x}) = \mathrm{F}(\mathrm{x})^{\mathrm{sk}}$

Standard VRF - Query

$\mathrm{pk} = \mathrm{g}^{\mathrm{sk}}$ and $\mathrm{VRF}(\mathrm{sk}, \mathrm{x}) = \mathrm{F}(\mathrm{x})^{\mathrm{sk}}$

$\mathrm{Query}(\mathrm{sk}, \mathrm{x})$:

$\mathrm{y} = \mathrm{VRF}(\mathrm{sk}, \mathrm{x})$,

$\pi =$ proof $(\mathrm{g}, \mathrm{F}(\mathrm{x}), \mathrm{pk}, \mathrm{y})$ is a DDH-tuple

Why?

$\mathbf{pk} = \mathbf{g}^{\mathbf{sk}} \Longrightarrow \mathbf{y} = \mathbf{F}(\mathbf{x})^{\mathbf{sk}}$

Standard VRF - Rotate?

$\mathrm{pk} = \mathrm{g}^{\mathrm{sk}}$ and $\mathrm{VRF}(\mathrm{sk}, \mathrm{x}) = \mathrm{F}(\mathrm{x})^{\mathrm{sk}}$

$\mathrm{Rotate}(\mathrm{sk}, \mathrm{x})$:

Choose random exponent a_{sk}

$\mathrm{sk} * \mathrm{a}_{\mathrm{sk}} \rightarrow \mathrm{sk'}$

$\mathrm{g}^{\mathrm{sk'}} = \mathrm{pk}^{\mathrm{a}} \rightarrow \mathrm{pk'}$

Rotatable VRF - Rotate

$\mathrm{pk} = \mathrm{g}^{\mathrm{sk}}$ and $\mathrm{VRF}(\mathrm{sk}, \mathrm{x}) = \mathrm{F}(\mathrm{x})^{\mathrm{sk}}$

$\mathrm{Rotate}(\mathrm{sk}, \mathrm{x})$:

$\mathrm{sk'} = \mathrm{sk} * \mathrm{a}, \mathrm{pk'} = \mathrm{pk}^{\mathrm{a}}, \mathrm{y} = \mathrm{VRF}(\mathrm{sk}, \mathrm{x}), \mathrm{y'} = \mathrm{VRF}(\mathrm{sk}, \mathrm{x'})$

$\pi =$ proof $(\mathrm{pk}, \mathrm{y}, \mathrm{pk'}, \mathrm{y'})$ is a DDH-tuple

Why?

$\mathbf{y} = \mathbf{VRF}(\mathbf{sk}, \mathbf{x}), \mathbf{pk'} = \mathbf{pk}^{\mathbf{a}} \Rightarrow \mathbf{y'} = \mathbf{y}^{\mathbf{a}} = \mathbf{F}(\mathbf{x})^{\mathbf{sk} * \mathbf{a}} = \mathbf{VRF}(\mathbf{sk'}, \mathbf{x})$

Rotatable VRF - Zero Knowledge

Ignoring rotations:

DDH Assumption + programmable ROM → Zero Knowledge

Idea: program $\mathrm{F}(\mathrm{x})$ so $\mathrm{y} = \mathrm{F}(\mathrm{x})^{\mathrm{sk}}$

Rotatable VRF - Zero Knowledge

With corruptions:
$\mathrm{pk} = \mathrm{g}^{\mathrm{sk}}$ commits to sk!

If we give $(\mathrm{pk}, \mathrm{pk}')$, commit to $(\mathrm{sk}, \mathrm{sk}')$
need $\mathrm{F}(\mathrm{x})^{\mathrm{sk}} = \mathrm{y}$ and $\mathrm{F}(\mathrm{x})^{\mathrm{sk}'} = \mathrm{y}'$

May be impossible! But does it lead to an attack?

Rotatable VRF - Zero Knowledge

With corruptions:

$\mathrm{pk} = \mathrm{g}^{\mathrm{sk}}$ commits to sk!

If we give $(\mathrm{pk}, \mathrm{pk}')$, commit to $(\mathrm{sk}, \mathrm{sk}')$

need $\mathrm{F}(\mathrm{x})^{\mathrm{sk}} = \mathrm{y}$ and $\mathrm{F}(\mathrm{x})^{\mathrm{sk}'} = \mathrm{y}'$

May be impossible! But does it lead to an attack? **LIKELY NOT**

Rotatable VRFs - Zero Knowledge

Solution: Stronger idealized models

Shoup's Generic Group Model (GGM)

$$x \longrightarrow \boxed{\mathrm{EXP}} \longrightarrow g^x$$

$$g^a, g^b \longrightarrow \boxed{*} \longrightarrow g^{a+b} = \mathrm{EXP}(\mathrm{EXP}^{-1}(a) + \mathrm{EXP}^{-1}(b))$$

Rotatable VRFs - Zero Knowledge

Key trick: in GGM, giving $\mathrm{pk} = \mathrm{g}^{\mathrm{sk}}$ does not commit to sk.

Don't need to define discrete logs until corruption

Rotatable VRFs - Zero Knowledge (Lazy GGM)

Query	Group Element	Discrete Log
$F(x)$	7314153	B_x
pk_1	1531678	A_1
pk_2	9817532	A_1*A_2
$VRF_1(x)$	1253278	B_x*A_1
$VRF_2(x)$	0982436	$B_x*A_1*A_2$
$2*pk_1*VRF_1(x)$	4732814	$2+A_1+B_x*A_1$

Rotatable VRFs - Zero Knowledge (Lazy GGM)

Query	Group Element	Discrete Log
$F(x)$	7314153	B_x
pk_1	1531678	A_1
pk_2	9817532	A_1*A_2
$VRF_1(x)$	1253278	B_x*A_1
$VRF_2(x)$	0982436	$B_x*A_1*A_2$
$2*pk_1*VRF_1(x)$	4732814	$2+A_1+B_x*A_1$

Corrupt

Query	Group Element	Discrete Log
$F(x)$	7314153	B_x
pk_1	1531678	153
pk_2	9817532	102
$VRF_1(x)$	1253278	$153*B_x$
$VRF_2(x)$	0982436	$102*B_x$
$2*pk_1*VRF_1(x)$	4732814	$155+102*B_x$

Future Work

1. Do we need GGM in order to achieve RZKS/rotatable VRFs? Why or why not?
2. Can we use similar GGM programming techniques on other “non-committing” primitives?
3. RZKS auditors verify “append-only” property here. Are there other useful properties auditors could verify using similar techniques?