Boosting the Security of Blind Signature Schemes

Jonathan Katz
University of Maryland, College Park
jkatz2@gmail.com

Julian Loss*
CISPA Helmholtz Center
for Information Security
lossjulian@gmail.com

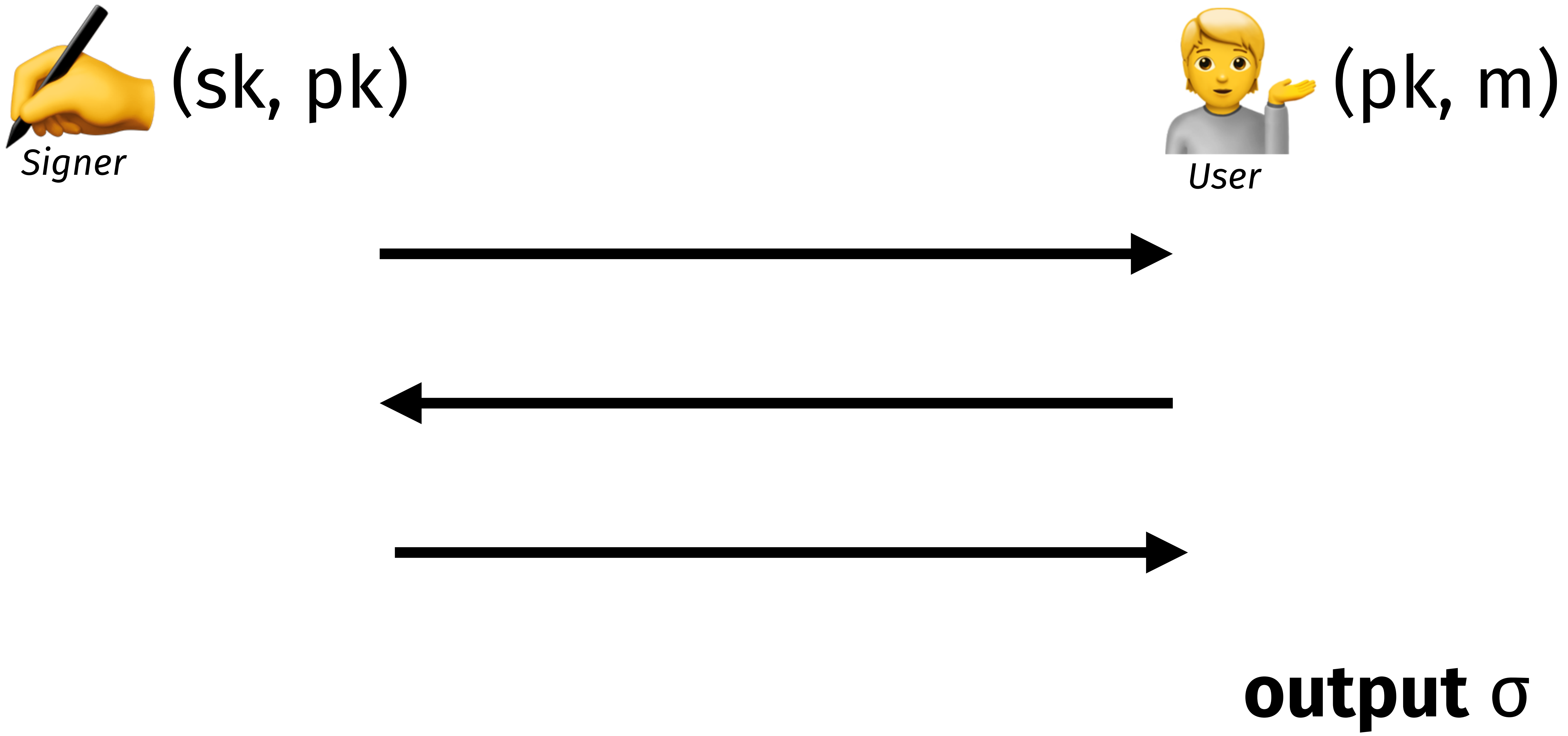
Michael Rosenberg**
University of Maryland, College Park
micro@cs.umd.edu

*Work done while at the University of Maryland, College Park

**Supported by the National Defense Science and Engineering Graduate (NDSEG) fellowship

Blind Signatures

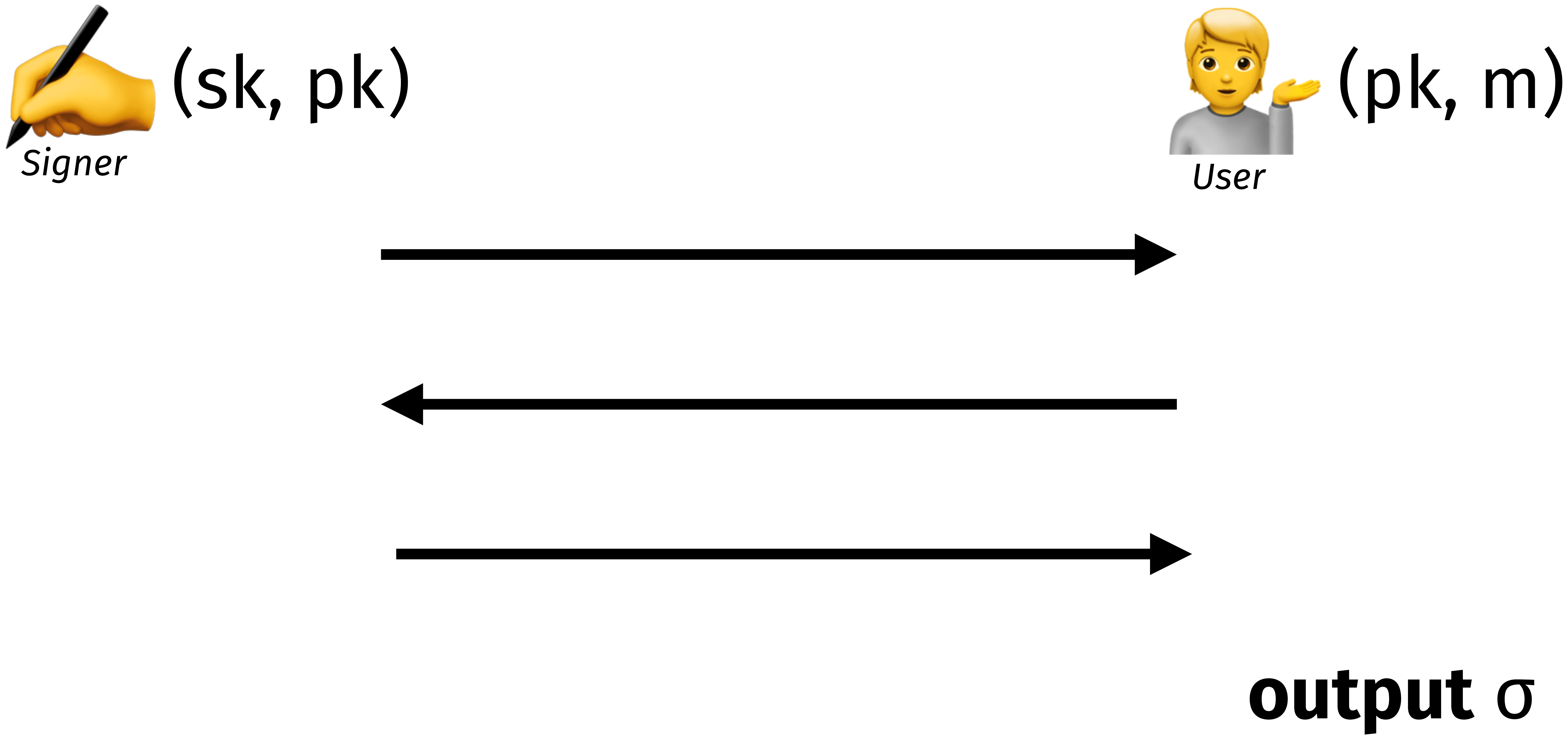
Digital signatures where the signer learns nothing



Blind Signatures

Digital signatures where the signer learns nothing

Blindness The signer can't link any msg-sig pair to a particular execution

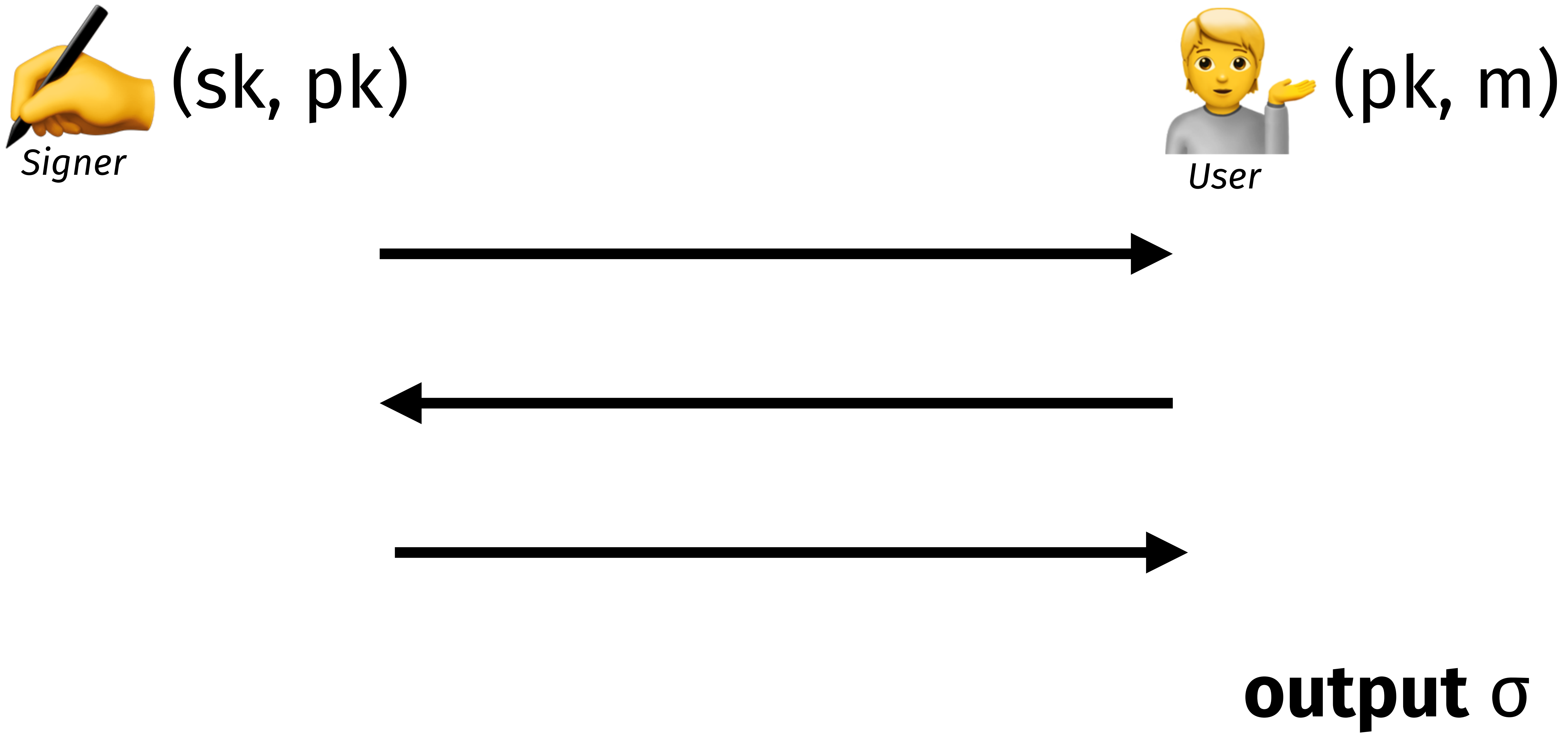


Blind Signatures

Digital signatures where the signer learns nothing

Blindness The signer can't link any msg-sig pair to a particular execution

Unforgeability Given access to the Signer, adversary cannot produce a fresh msg-sig pair



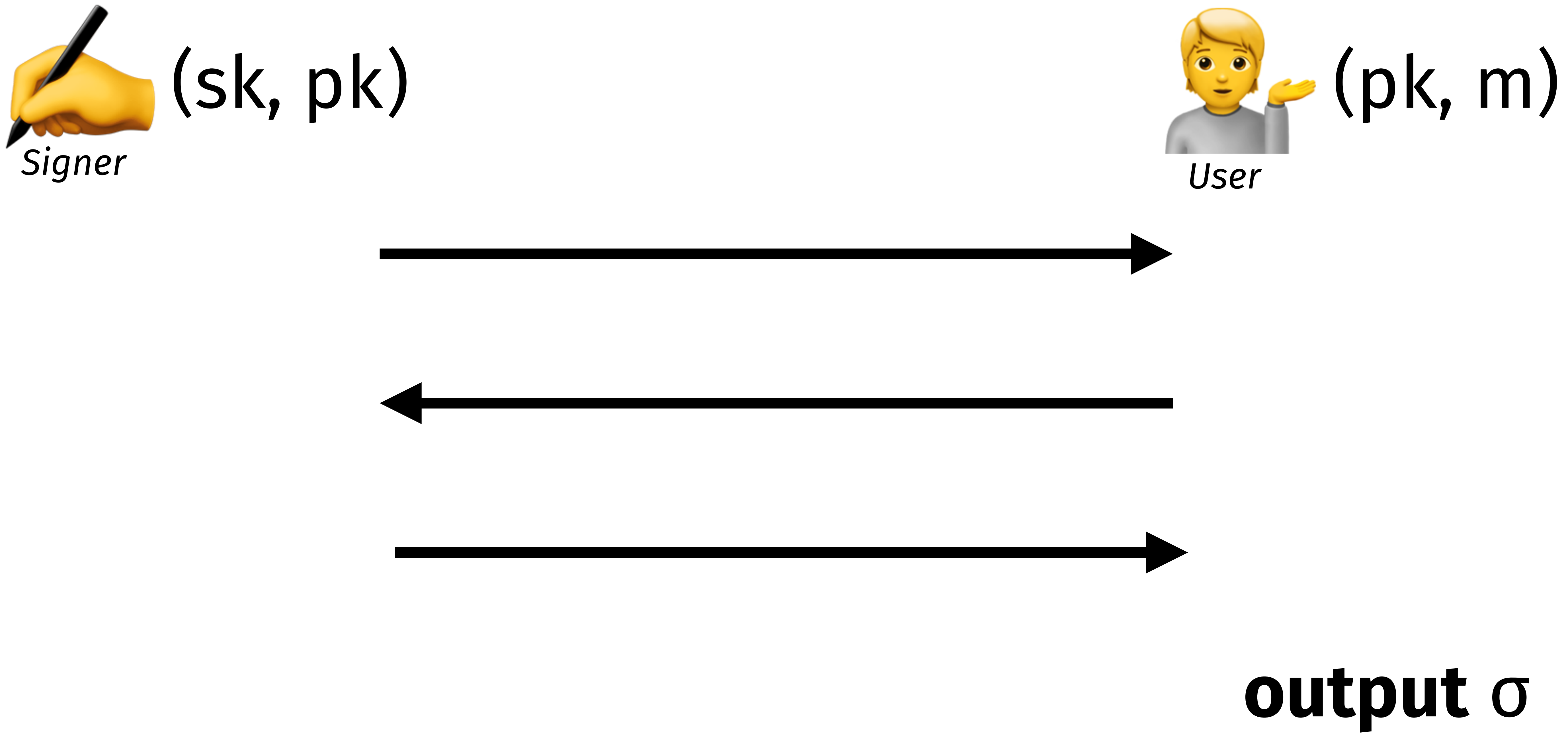
Blind Signatures

Digital signatures where the signer learns nothing

Blindness The signer can't link any msg-sig pair to a particular execution

Unforgeability Given access to the Signer, adversary cannot produce a fresh msg-sig pair

Problem: *Every* pair looks fresh to a blind signer



Blind Signatures

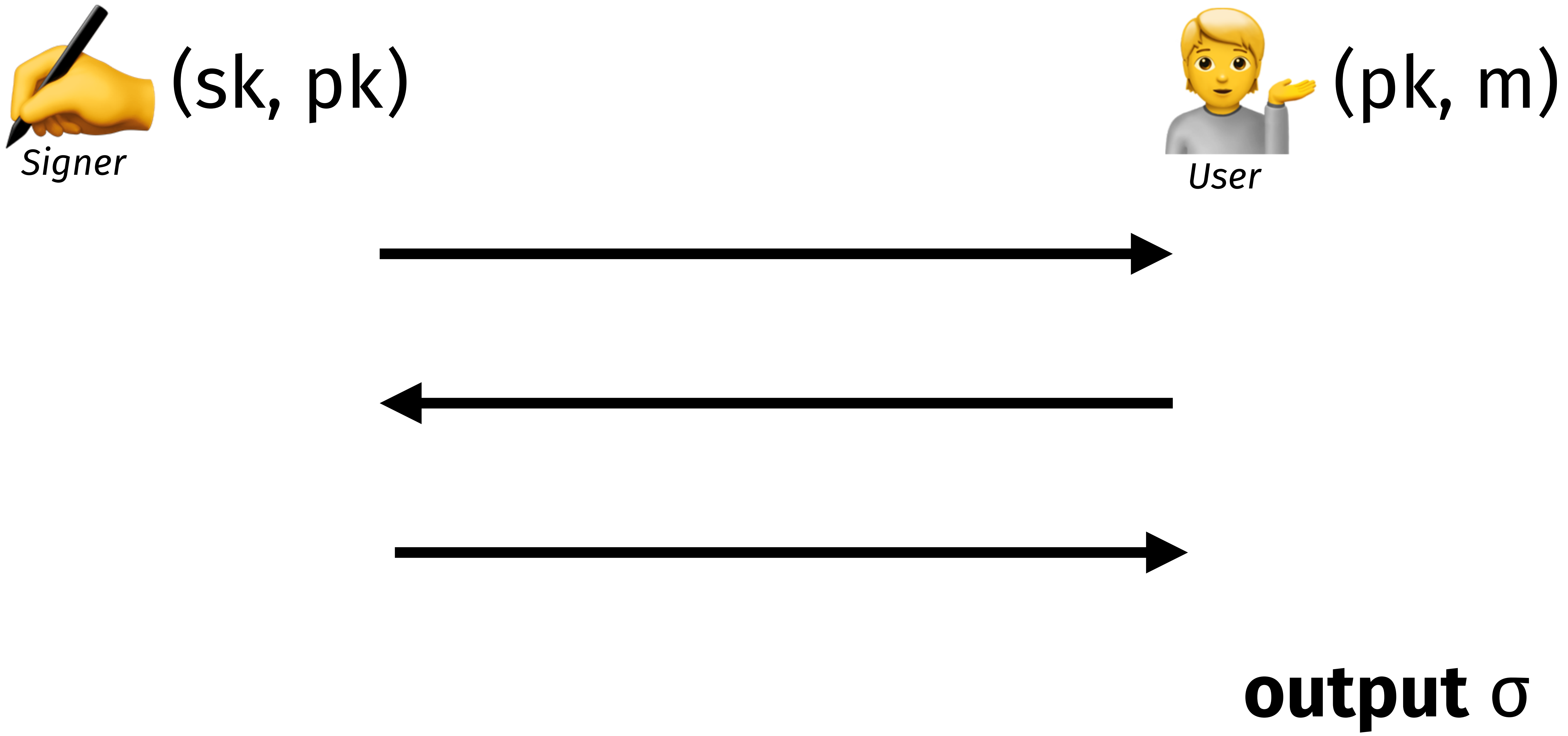
Digital signatures where the signer learns nothing

Blindness The signer can't link any msg-sig pair to a particular execution

Unforgeability Given access to the Signer, adversary cannot produce a fresh msg-sig pair

Problem: *Every* pair looks fresh to a blind signer

One-more-unforgeability (OMUF) Given ℓ msg-sig pairs, cannot produce ℓ+1 distinct pairs



Blind Signatures

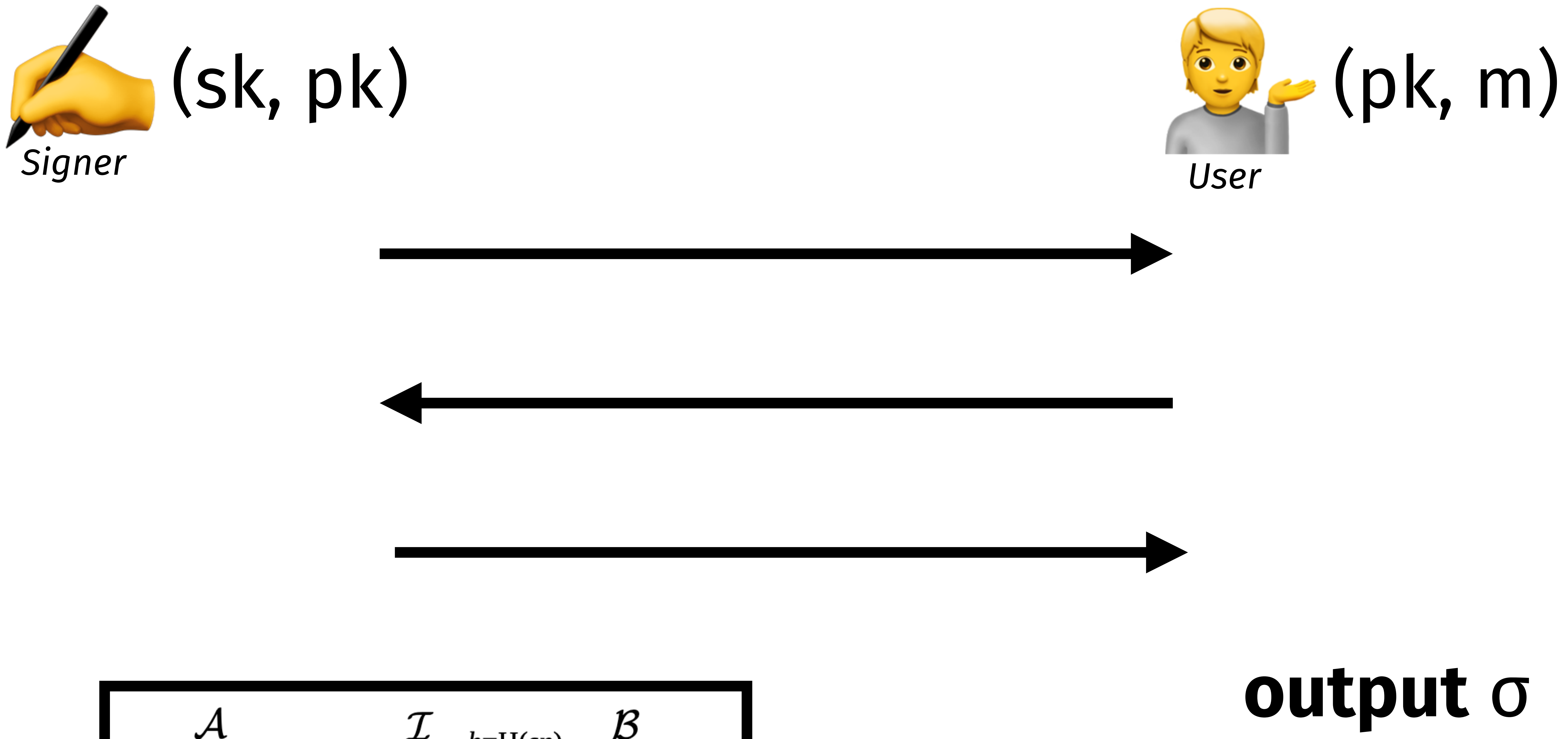
Digital signatures where the signer learns nothing

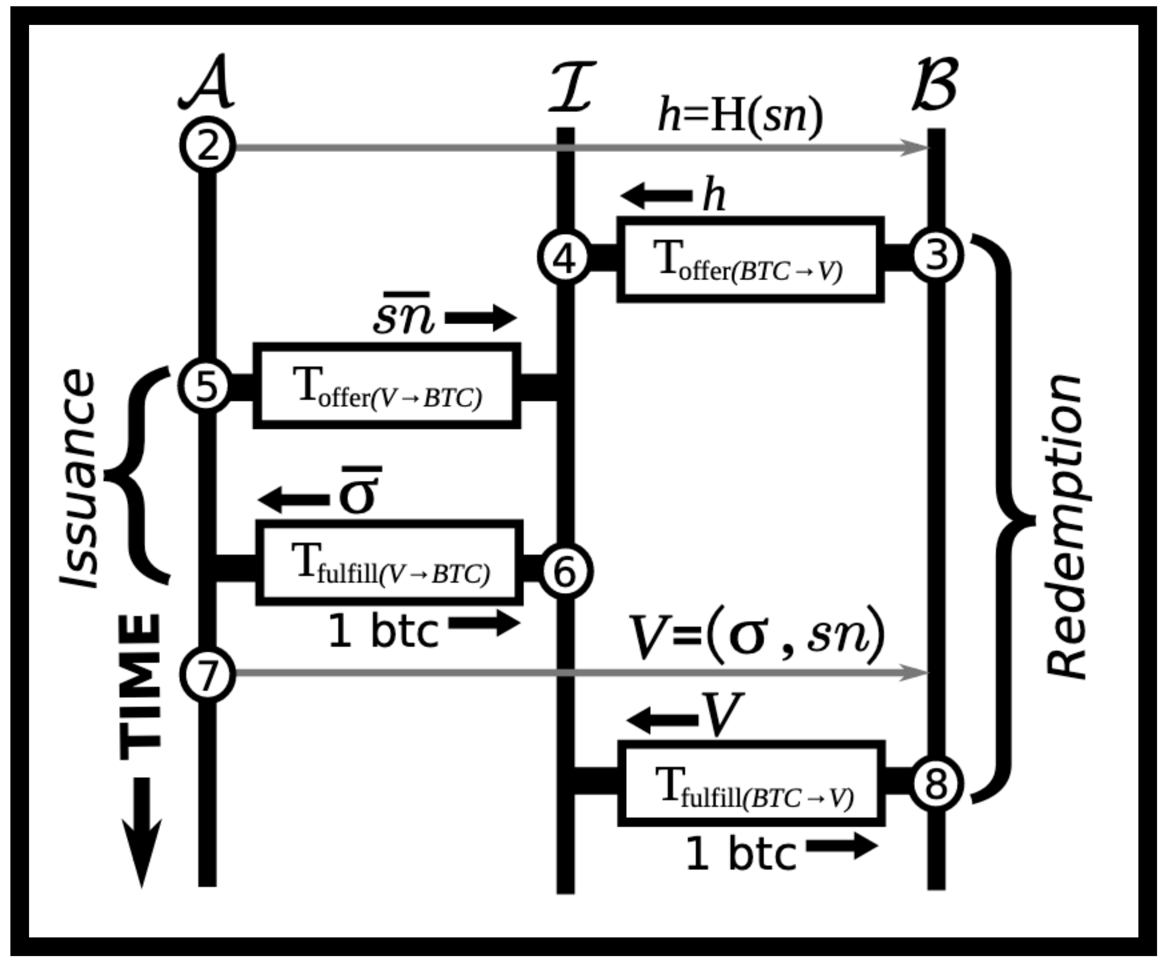
Blindness The signer can't link any msg-sig pair to a particular execution

Unforgeability Given access to the Signer, adversary cannot produce a fresh msg-sig pair

Problem: *Every* pair looks fresh to a blind signer

One-more-unforgeability (OMUF) Given ℓ msg-sig pairs, cannot produce ℓ+1 distinct pairs





Bitcoin Tumbling [HBG16]

Blind Signatures

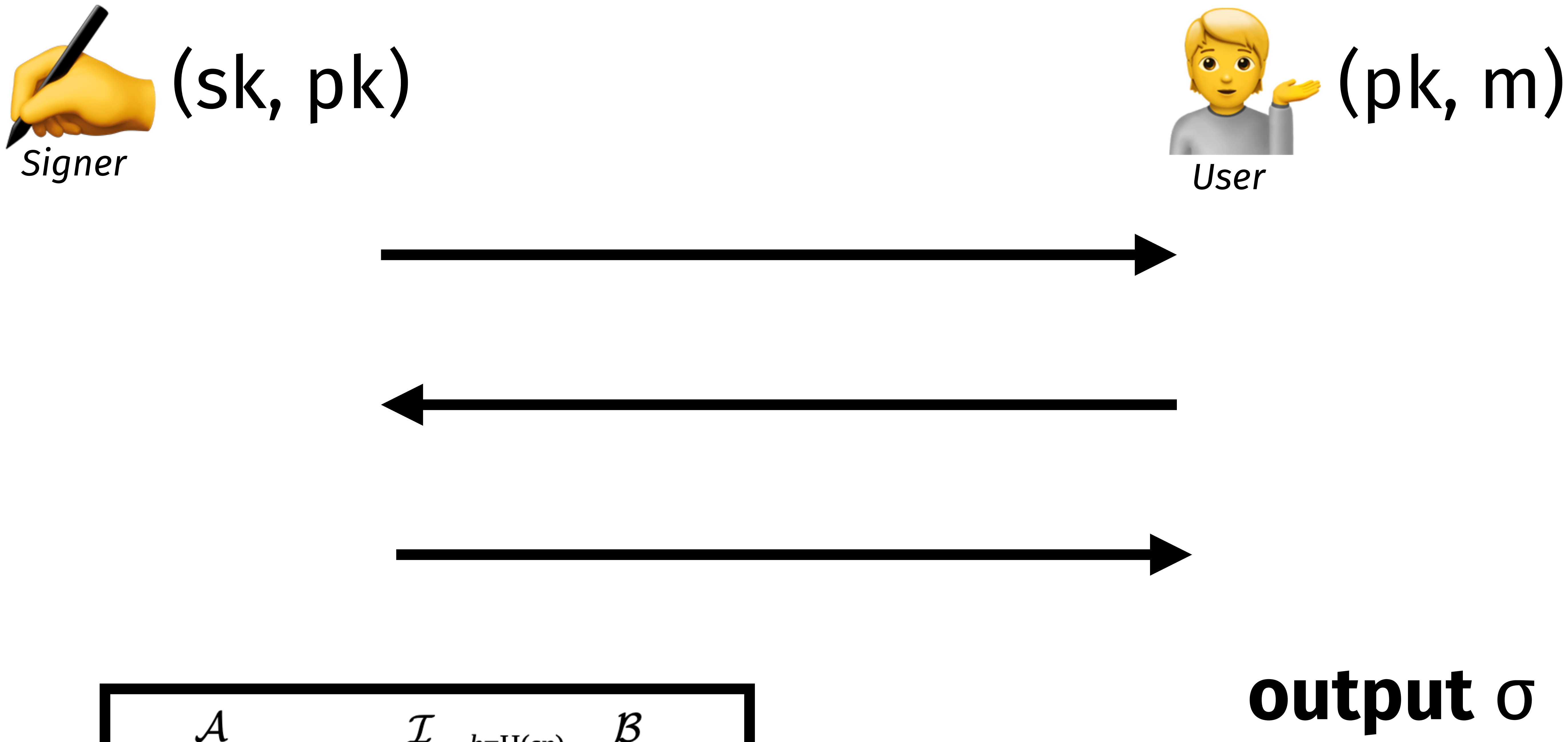
Digital signatures where the signer learns nothing

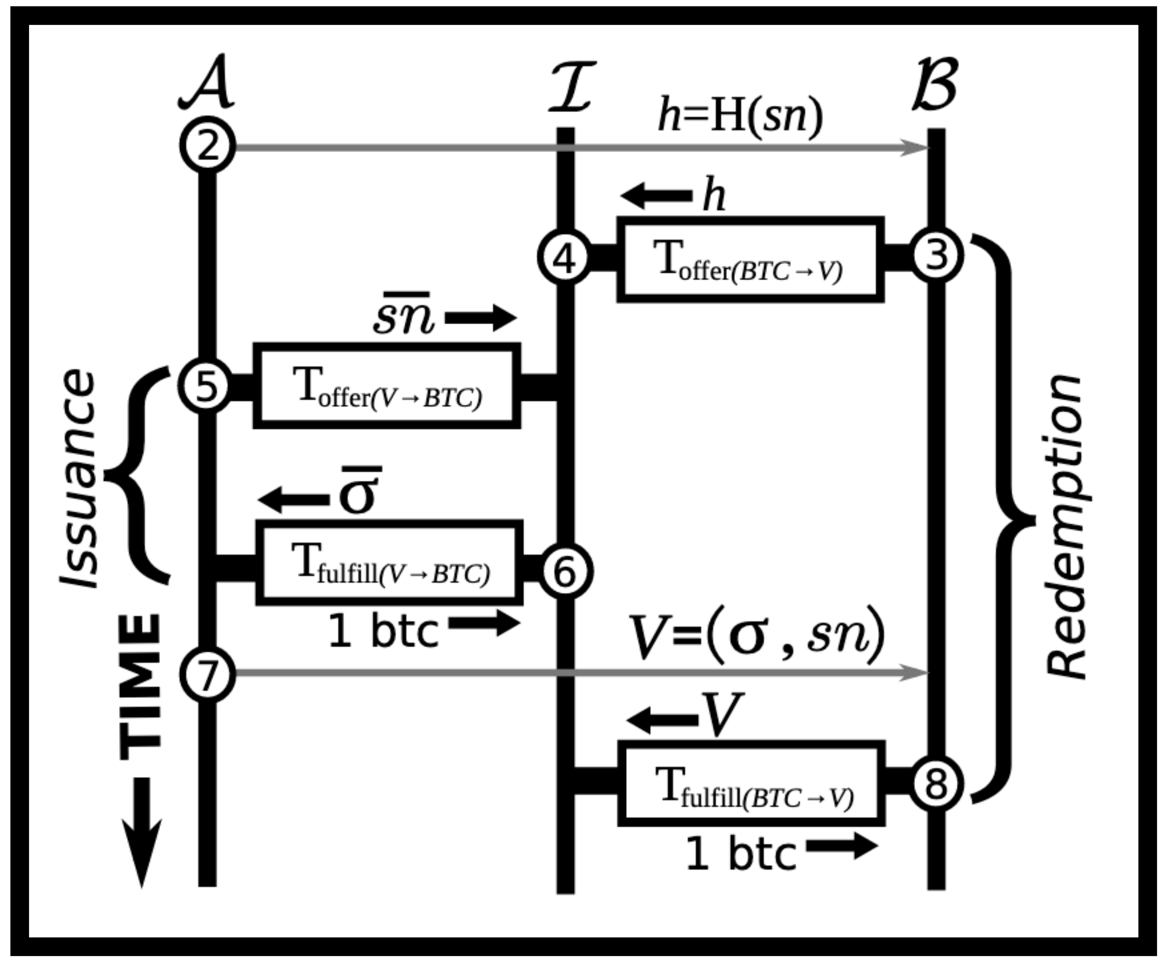
Blindness The signer can't link any msg-sig pair to a particular execution

Unforgeability Given access to the Signer, adversary cannot produce a fresh msg-sig pair

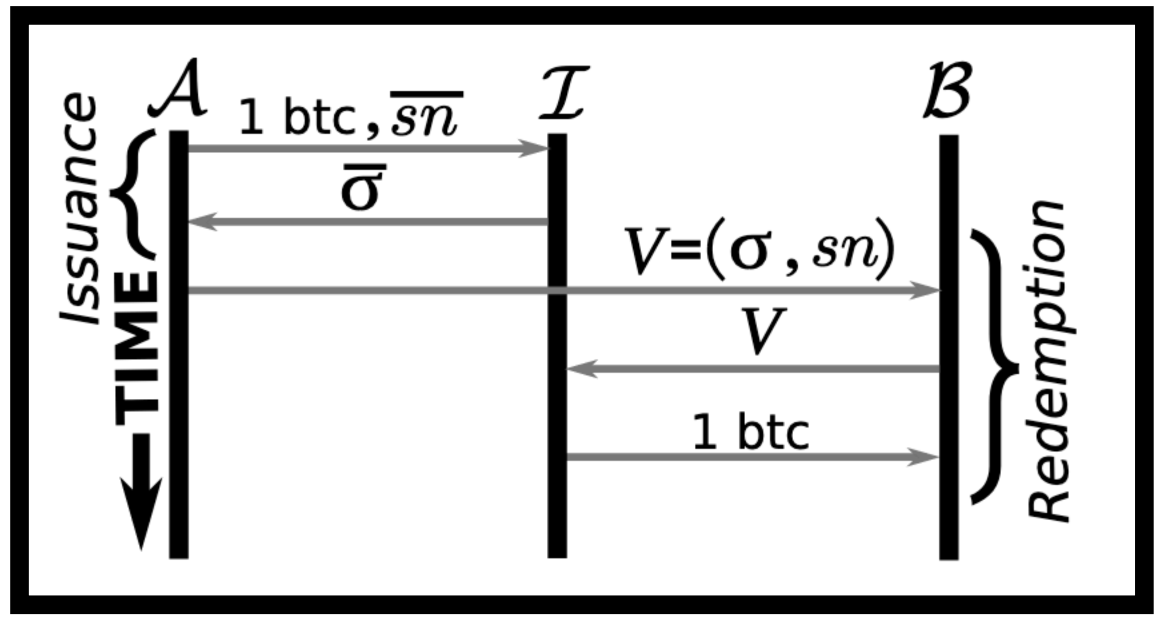
Problem: *Every* pair looks fresh to a blind signer

One-more-unforgeability (OMUF) Given ℓ msg-sig pairs, cannot produce ℓ+1 distinct pairs





Bitcoin Tumbling [HBG16]



Ecash [Chaum83]

Overview of Existing Efficient Blind Signature Schemes

Overview of Existing Efficient Blind Signature Schemes

- Only proven secure for **log-many signatures** [HKL19]

Overview of Existing Efficient Blind Signature Schemes

- Only proven secure for **log-many signatures** [HKL19]
  - Provably insecure for more than $O(\log(\kappa))$ concurrent sessions [BLLOR20]

Overview of Existing Efficient Blind Signature Schemes

- Only proven secure for **log-many signatures** [HKL19]
    - Provably insecure for more than $O(\log(\kappa))$ concurrent sessions [BLLOR20]
- **Nonstandard assumptions:**

Overview of Existing Efficient Blind Signature Schemes

- Only proven secure for **log-many signatures** [HKL19]
  - Provably insecure for more than $O(\log(\kappa))$ concurrent sessions [BLLOR20]
- **Nonstandard assumptions:**
  - Co-DH-inversion [FHS15]

Overview of Existing Efficient Blind Signature Schemes

- Only proven secure for **log-many signatures** [HKL19]
  - Provably insecure for more than $O(\log(\kappa))$ concurrent sessions [BLLOR20]
- **Nonstandard assumptions:**
  - Co-DH-inversion [FHS15]
  - One-more-RSA inversion [BNPS01]

Overview of Existing Efficient Blind Signature Schemes

- Only proven secure for **log-many signatures** [HKL19]
  - Provably insecure for more than $O(\log(\kappa))$ concurrent sessions [BLLOR20]
- **Nonstandard assumptions:**
  - Co-DH-inversion [FHS15]
  - One-more-RSA inversion [BNPS01]
  - Algebraic group model [KLX20]

Overview of Existing Efficient Blind Signature Schemes

- Only proven secure for **log-many signatures** [HKL19]
  - Provably insecure for more than $O(\log(\kappa))$ concurrent sessions [BLLOR20]
- **Nonstandard assumptions:**
  - Co-DH-inversion [FHS15]
  - One-more-RSA inversion [BNPS01]
  - Algebraic group model [KLX20]
- **No concurrency** [BL12, JLO97, KLX20]

Goal

Construct a blind signature scheme in the **ROM**, using **standard number-theoretic assumptions**, that is provably secure after:

1. Poly-many signatures
2. With arbitrary concurrency

Our Contribution

Our Contribution

- A **generic** transformation for
  blind signature schemes:
  log-many sigs → poly-many sigs

Our Contribution

- A **generic** transformation for blind signature schemes: log-many sigs → poly-many sigs
- Resulting blind signature schemes rely only on **DLOG**, **RSA**, or **factoring** in the ROM

Our Contribution

- A **generic** transformation for blind signature schemes: log-many sigs → poly-many sigs
- Resulting blind signature schemes rely only on **DLOG**, **RSA**, or **factoring** in the ROM
- Arbitrary concurrency

Our Contribution

- A **generic** transformation for blind signature schemes: log-many sigs → poly-many sigs
- Resulting blind signature schemes rely only on **DLOG**, **RSA**, or **factoring** in the ROM
- Arbitrary concurrency

Caveats

Our Contribution

- A **generic** transformation for blind signature schemes:
  log-many sigs → poly-many sigs
- Resulting blind signature schemes rely only on **DLOG**, **RSA**, or **factoring** in the ROM
- Arbitrary concurrency

Caveats

- Communication linear in #sessions

Our Contribution

- A **generic** transformation for blind signature schemes: log-many sigs → poly-many sigs
- Resulting blind signature schemes rely only on **DLOG**, **RSA**, or **factoring** in the ROM
- Arbitrary concurrency

Caveats

- Communication linear in #sessions
    - Can be improved to #dishonest users who were caught

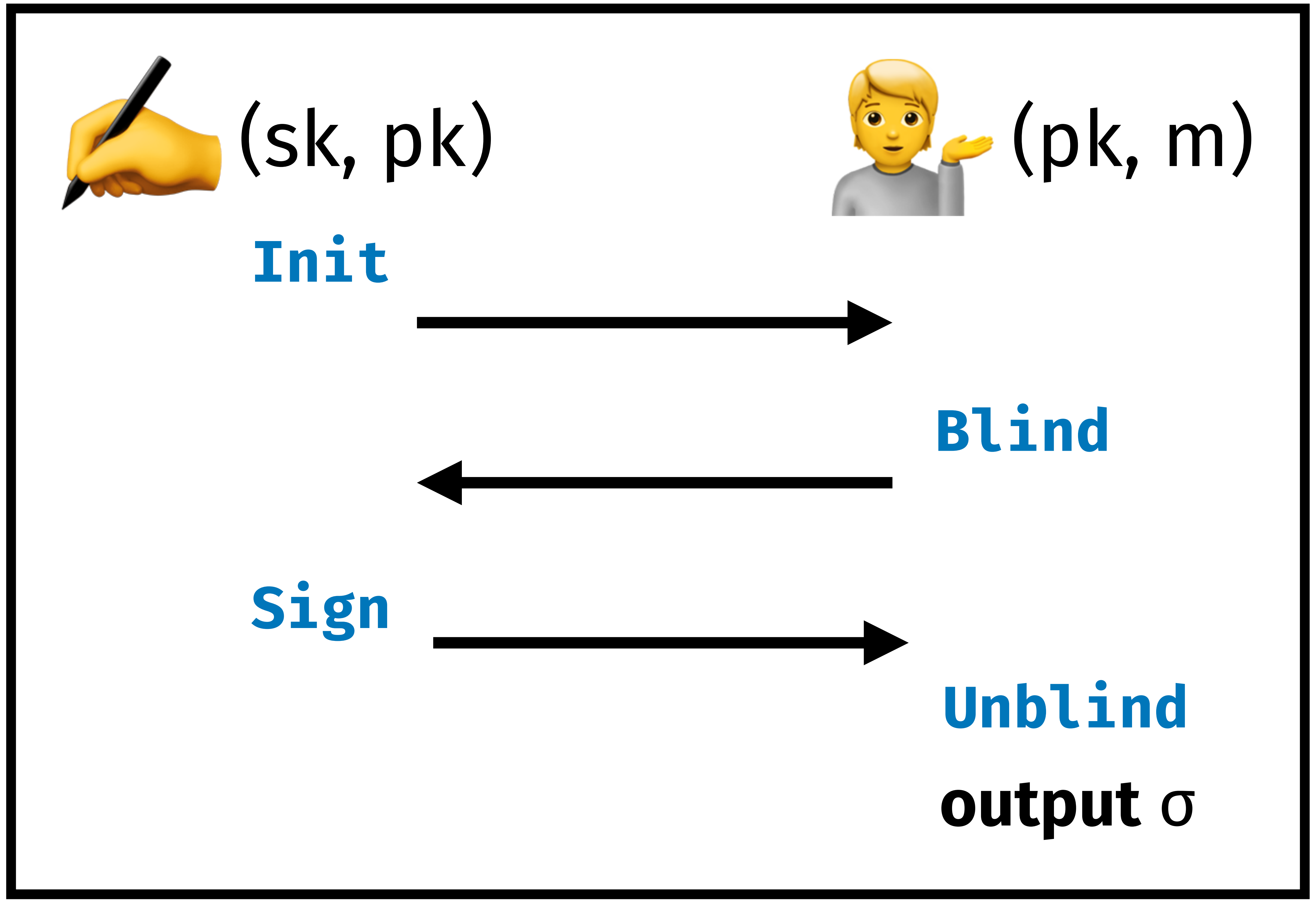
Our Contribution

- A **generic** transformation for blind signature schemes: log-many sigs → poly-many sigs
- Resulting blind signature schemes rely only on **DLOG**, **RSA**, or **factoring** in the ROM
- Arbitrary concurrency

Caveats

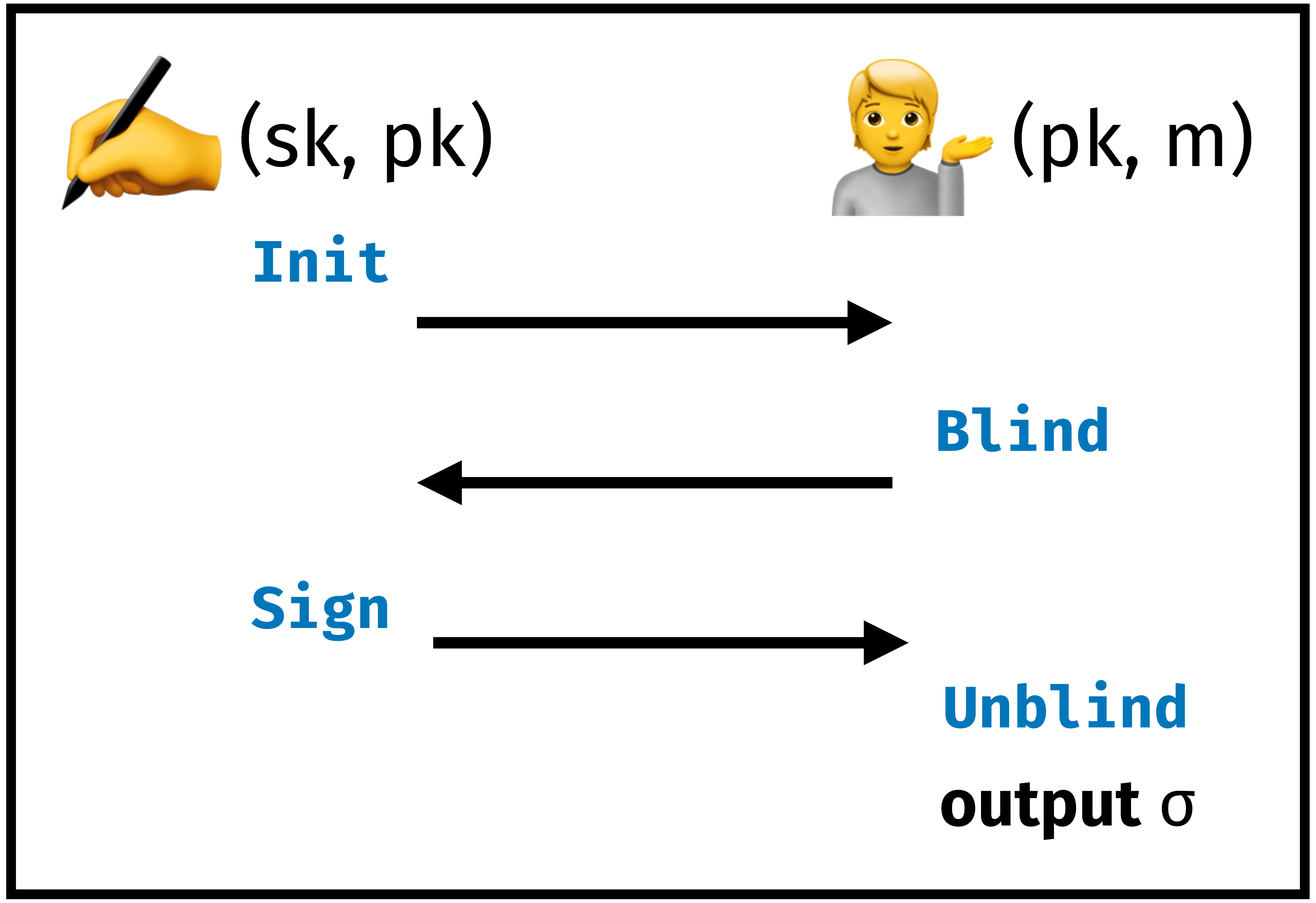
- Communication linear in #sessions
  - Can be improved to #dishonest users who were caught
- ~5000 bit groups

Beginning: Pointcheval's Transform



Blind Okamoto-Schnorr. $O(\log(\kappa))$-OMUF

Beginning: Pointcheval's Transform

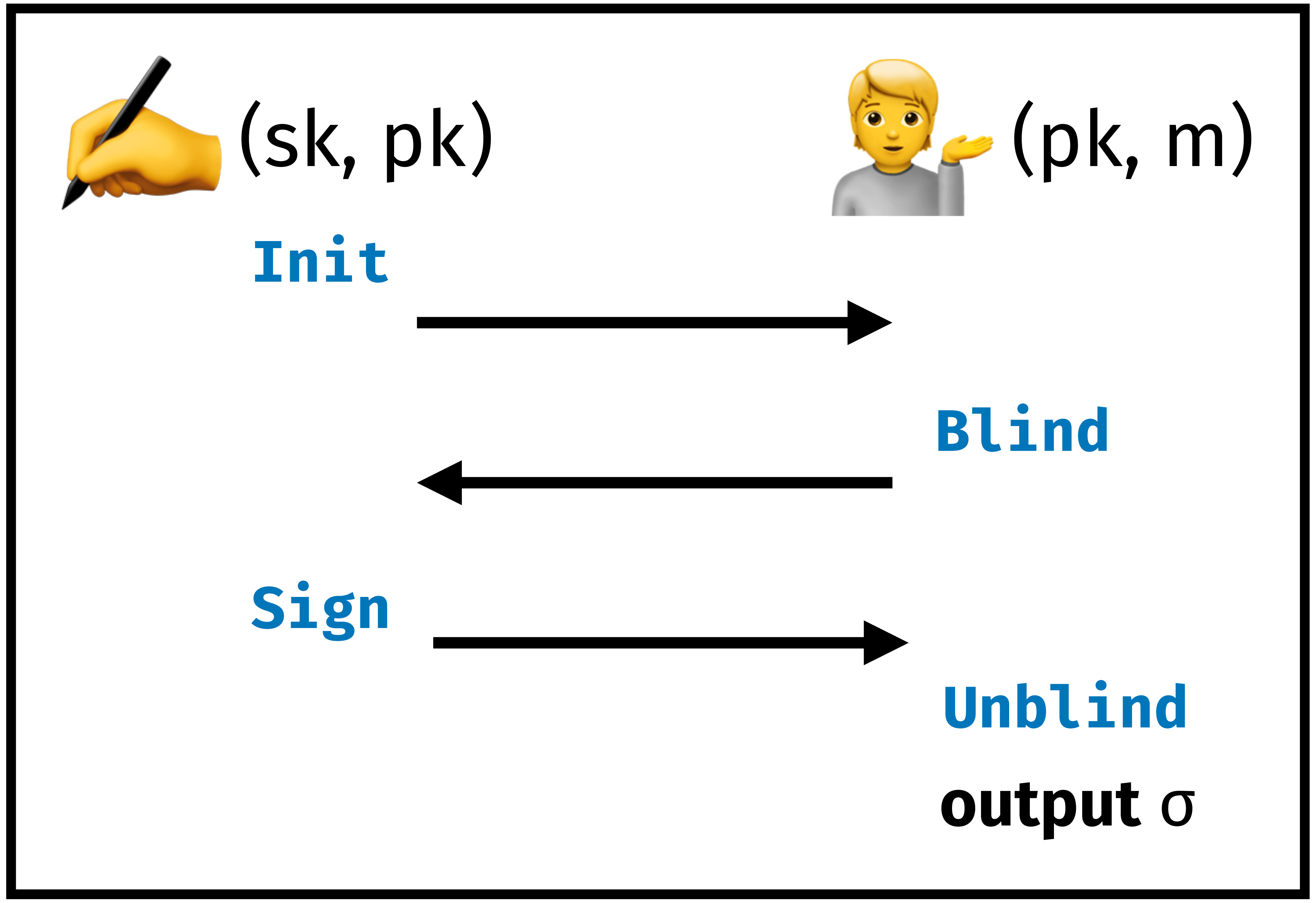


Blind Okamoto-Schnorr. $O(\log(\kappa))$-OMUF

Cut and Choose

$\Rrightarrow$

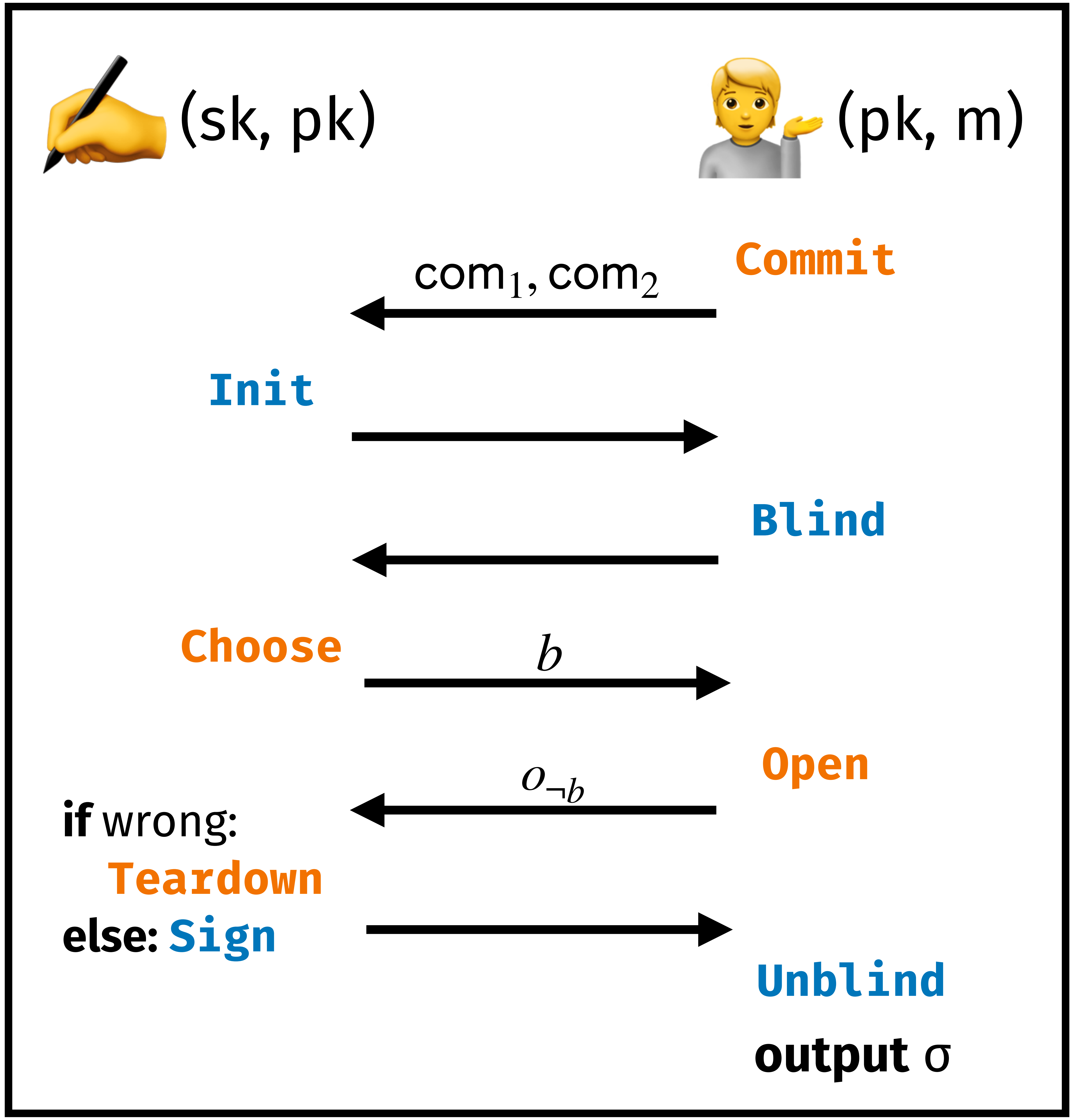
Beginning: Pointcheval's Transform



Blind Okamoto-Schnorr. $O(\log(\kappa))$-OMUF

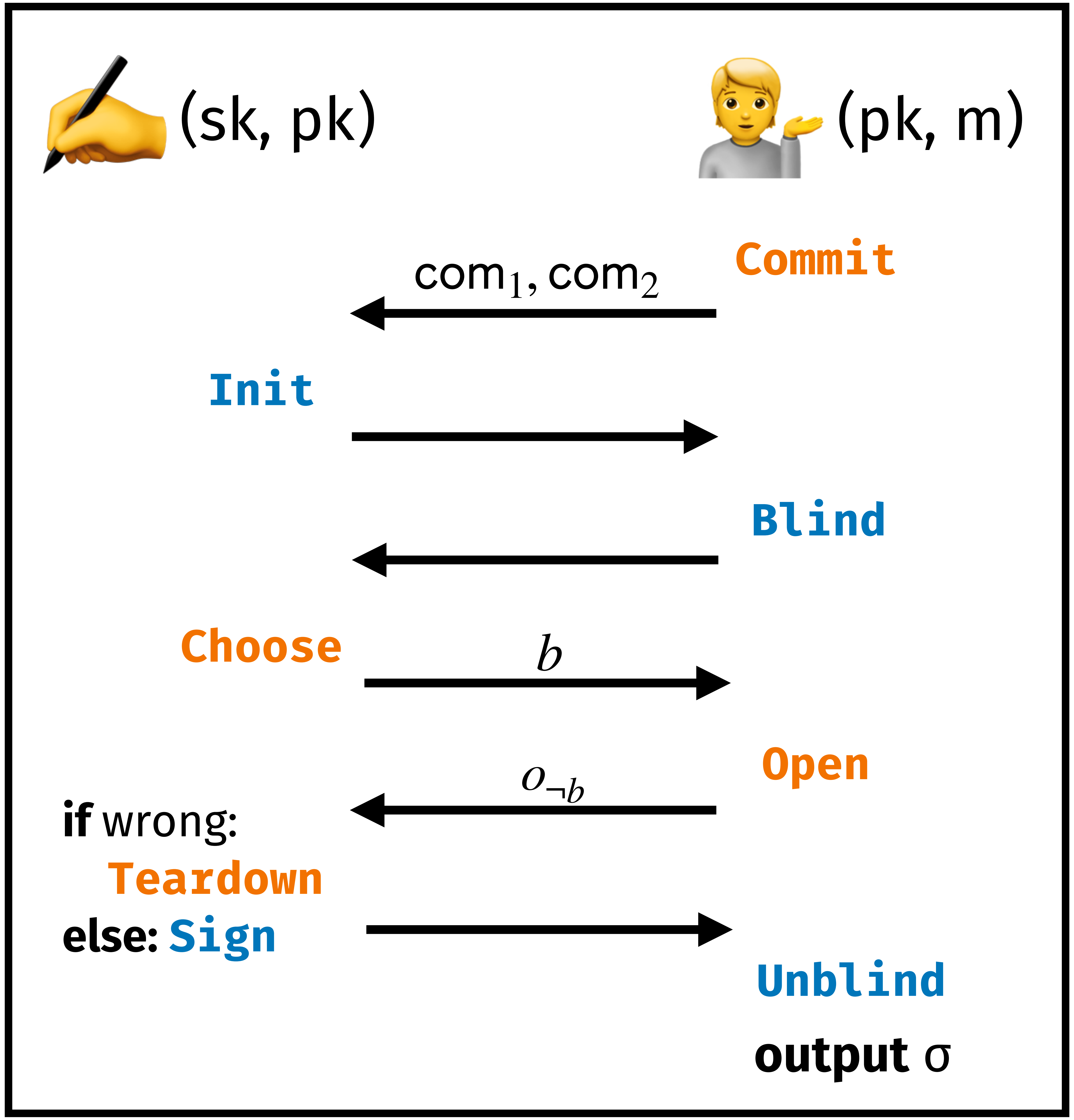
Cut and Choose

$\Rightarrow$



Transformed scheme via [Pointcheval98]. poly(κ)-OMUF

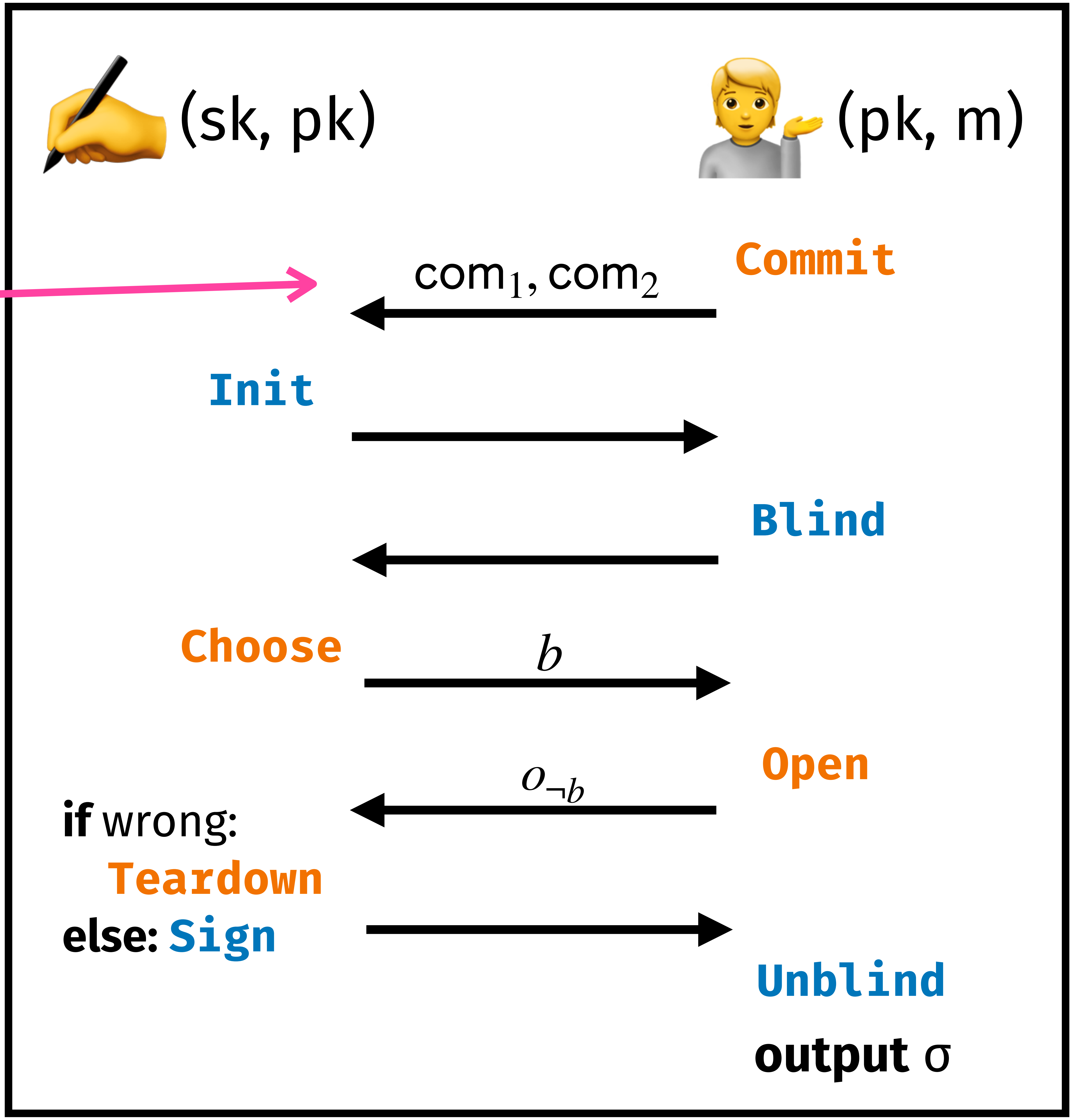
Beginning: Pointcheval's Transform



Transformed scheme via [Pointcheval98]. poly(κ)-OMUF

Beginning: Pointcheval's Transform

User commits to 2 sets of random coins

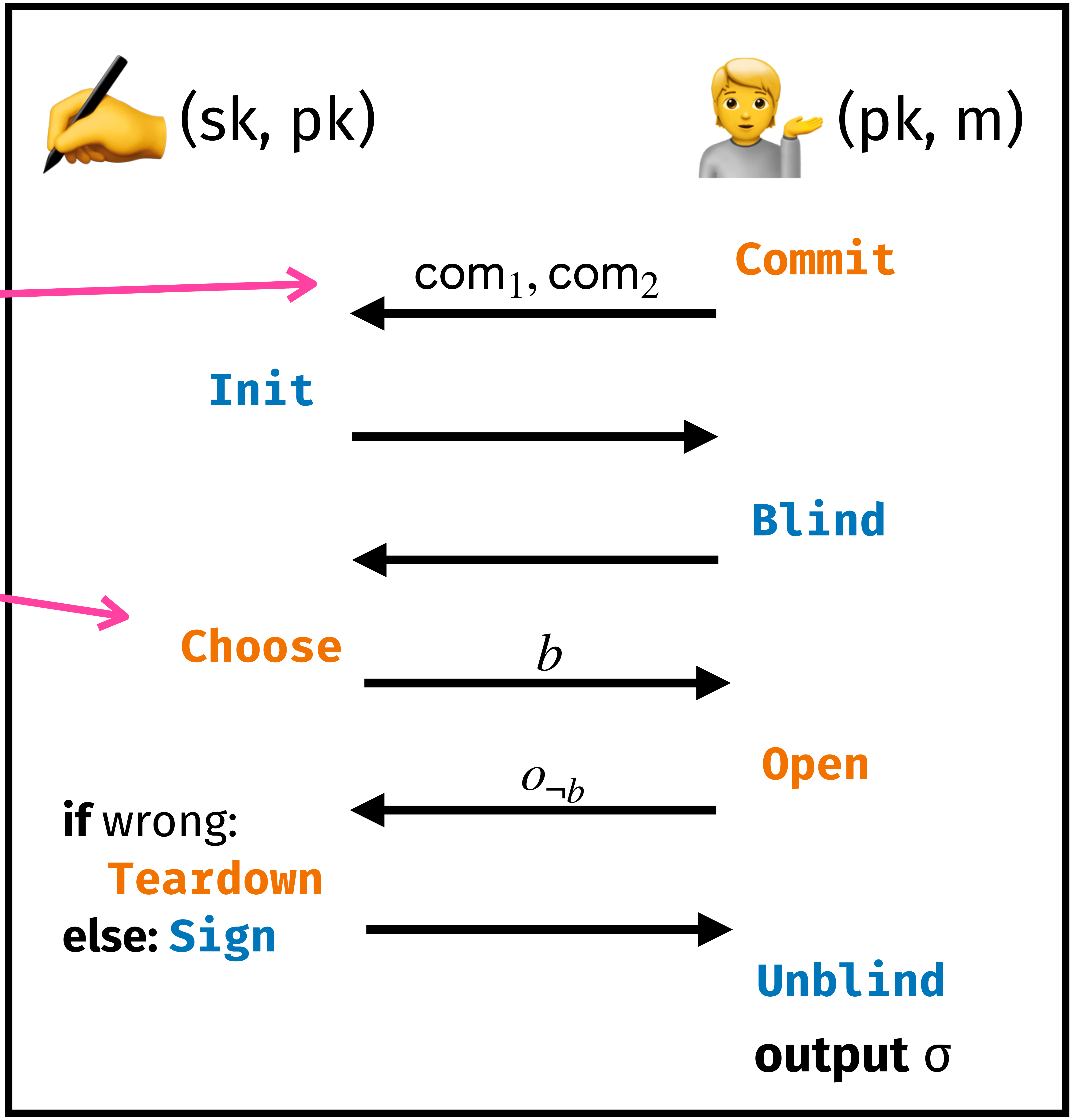


Transformed scheme via [Pointcheval98]. poly(κ)-OMUF

Beginning: Pointcheval's Transform

User commits to 2 sets of random coins

Signer chooses b



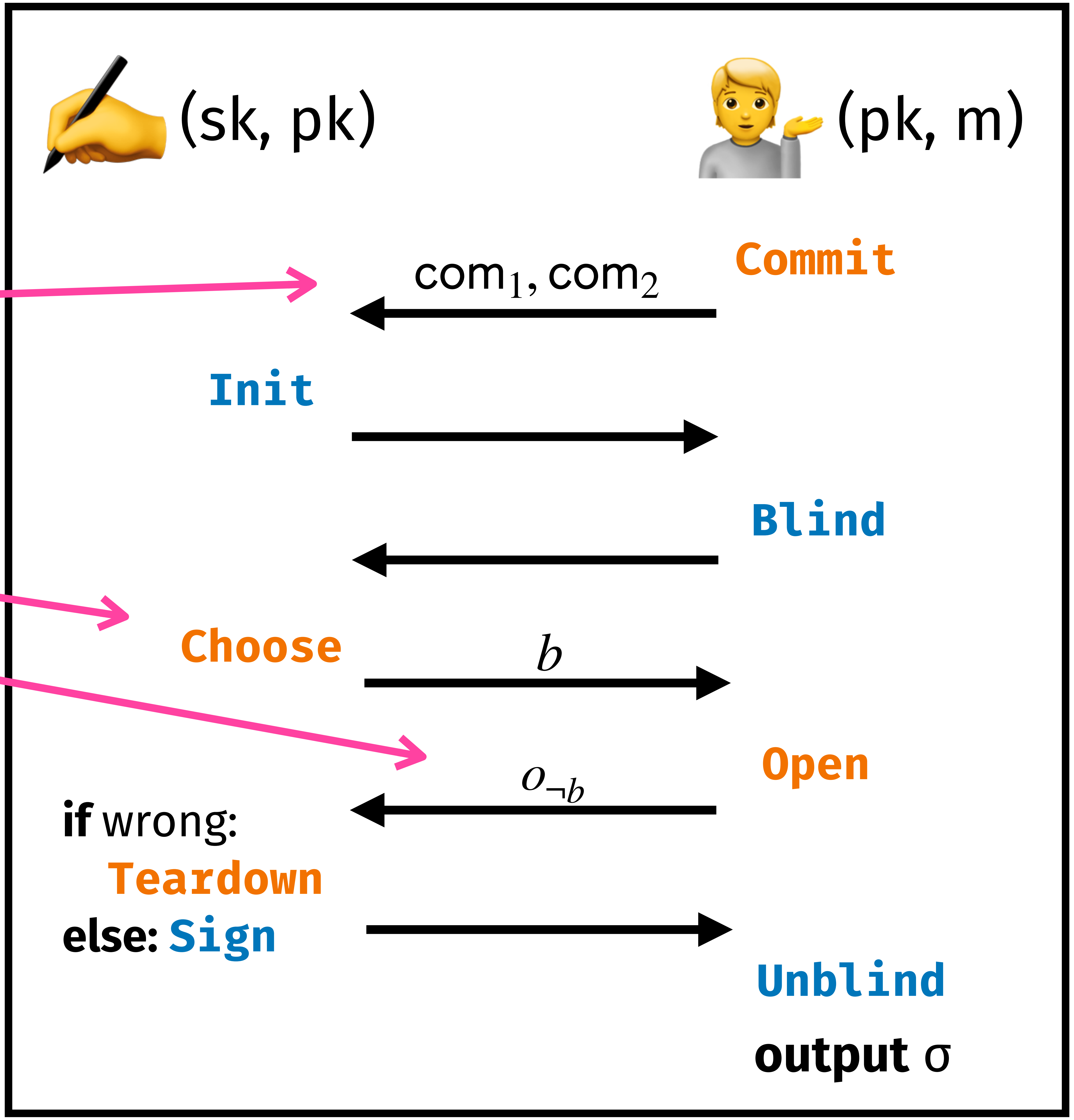
Transformed scheme via [Pointcheval98]. poly(κ)-OMUF

Beginning: Pointcheval's Transform

User commits to 2 sets of random coins

Signer chooses b

User opens the com not chosen



Transformed scheme via [Pointcheval98]. poly(κ)-OMUF

Beginning: Pointcheval's Transform

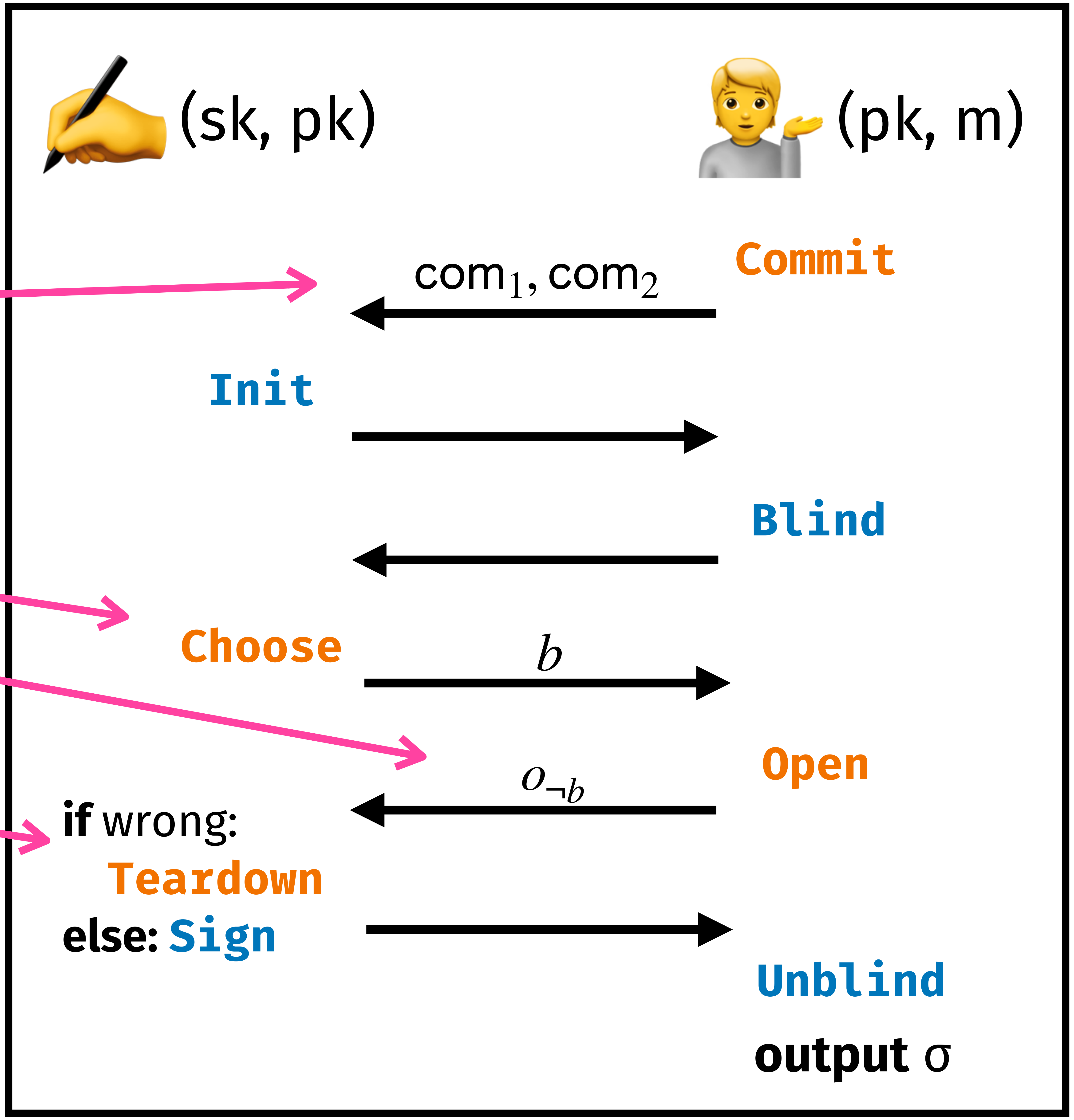
User commits to 2 sets of random coins

Signer chooses b

User opens the com not chosen

Signer checks the opening

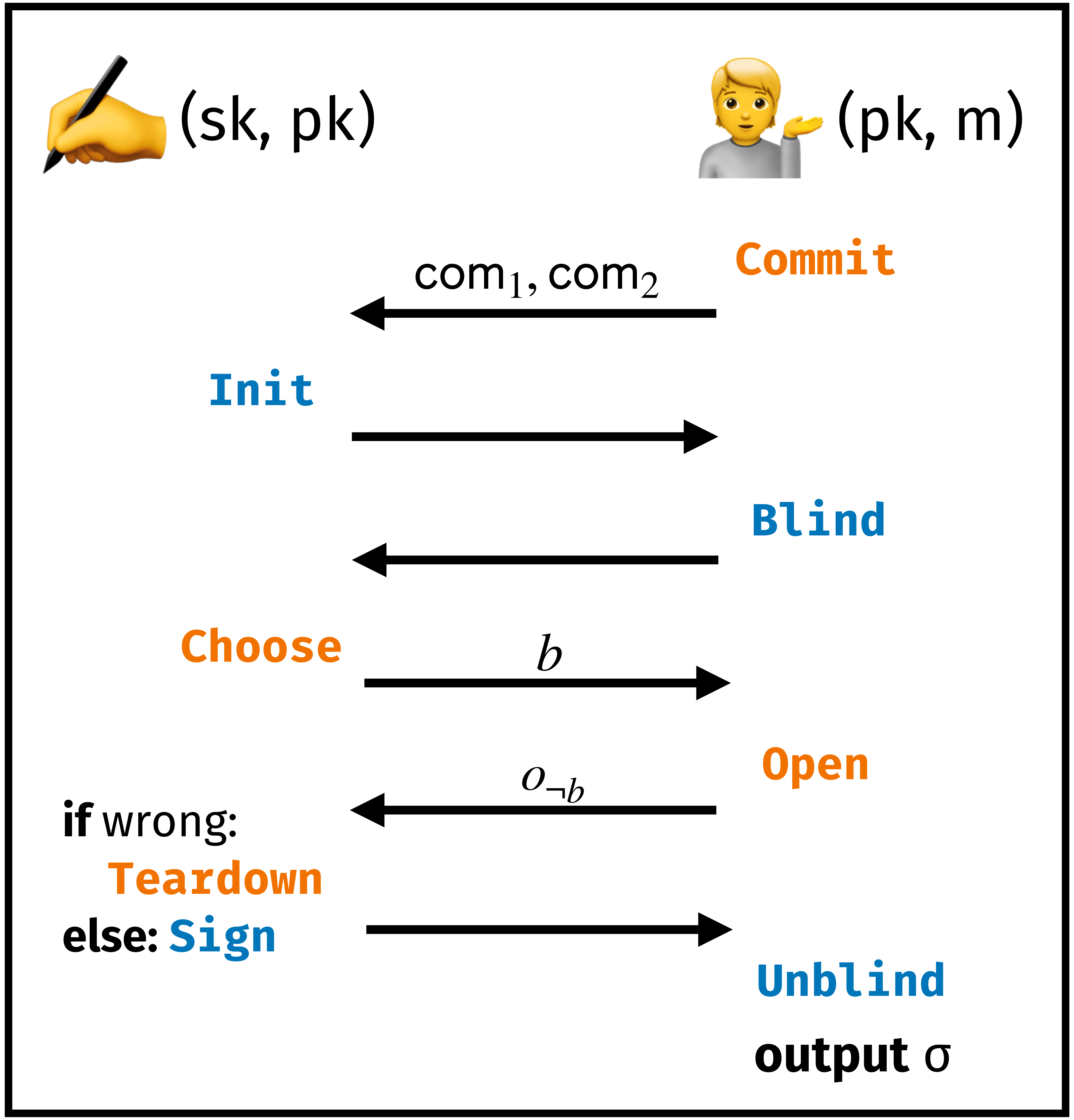
- **Success:** sign
- **Failure:** abort and **refuse to sign anything in the future**



Transformed scheme via [Pointcheval98]. poly(κ)-OMUF

Beginning: Pointcheval's Transform

Intuition

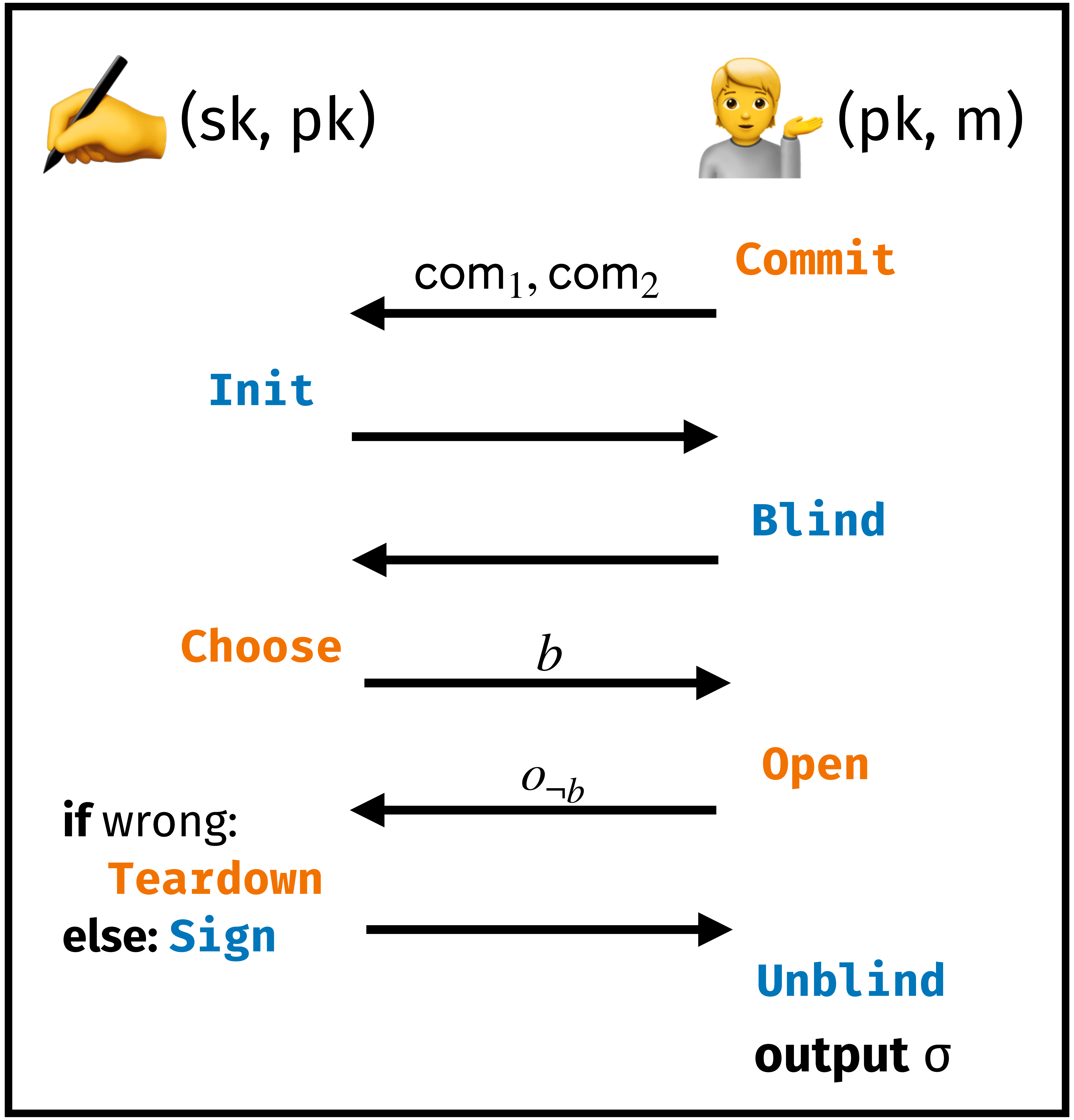


Transformed scheme via [Pointcheval98]. poly(κ)-OMUF

Beginning: Pointcheval's Transform

Intuition

Proof strategy: show that poly-one-more-forgery → log-one-more-forgery in the underlying scheme



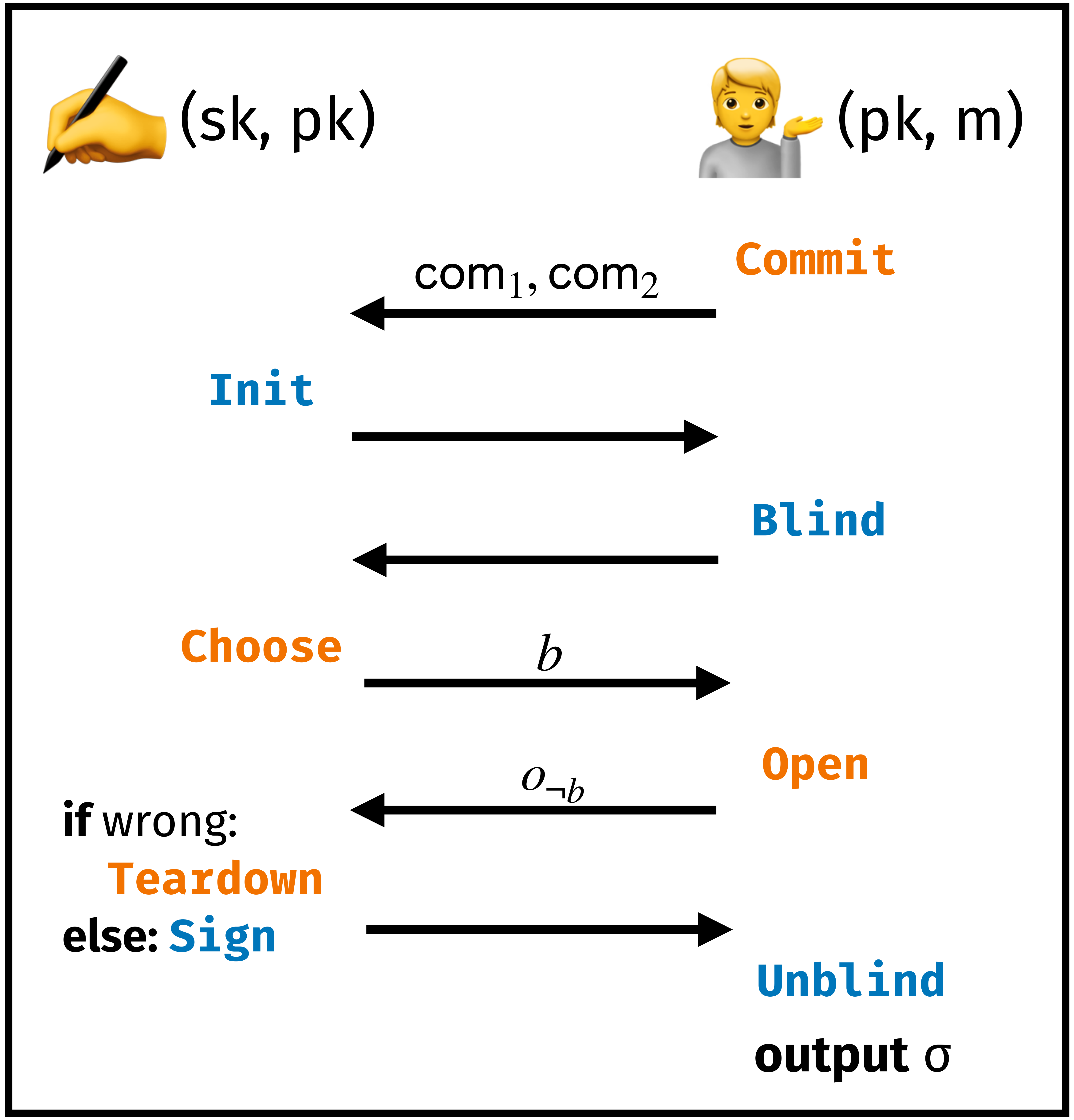
Transformed scheme via [Pointcheval98]. poly(κ)-OMUF

Beginning: Pointcheval's Transform

Intuition

Proof strategy: show that poly-one-more-forgery → log-one-more-forgery in the underlying scheme

Limit calls by **simulating the signer** when user acts honestly



Transformed scheme via [Pointcheval98]. poly(κ)-OMUF

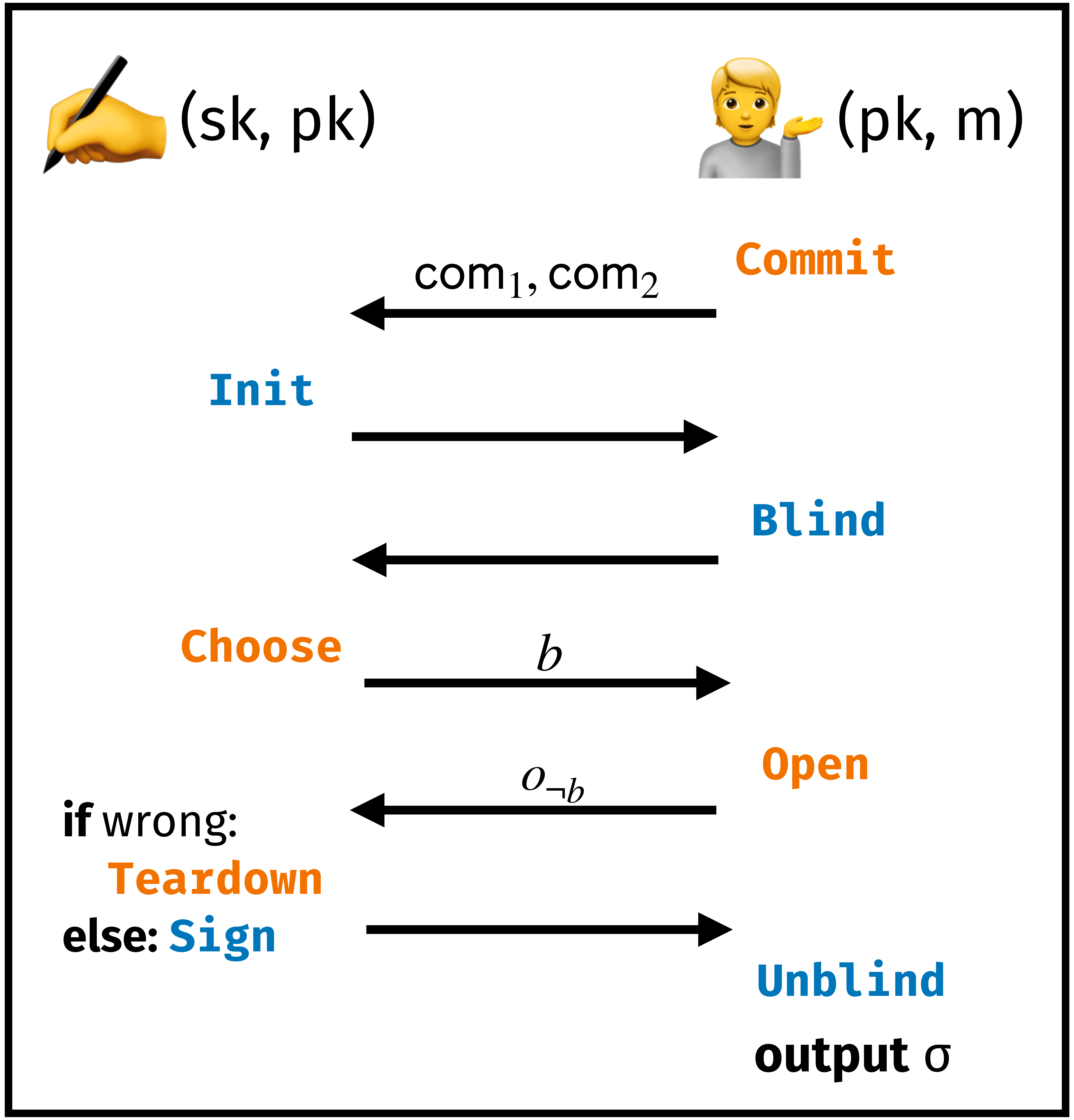
Beginning: Pointcheval's Transform

Intuition

Proof strategy: show that poly-one-more-forgery → log-one-more-forgery in the underlying scheme

Limit calls by **simulating the signer** when user acts honestly

How do you force the user to not cheat?



Transformed scheme via [Pointcheval98]. poly(κ)-OMUF

Beginning: Pointcheval's Transform

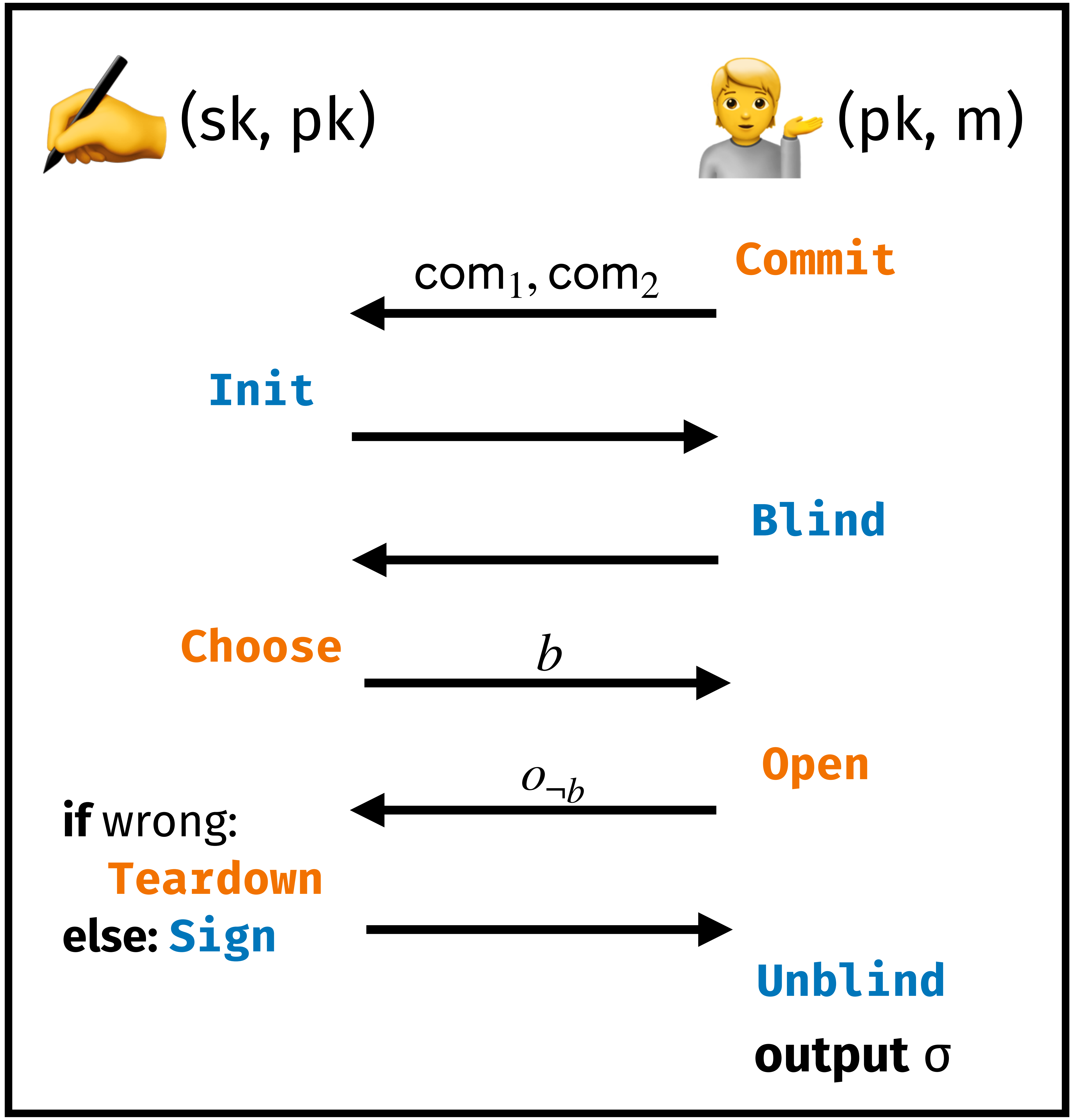
Intuition

Proof strategy: show that poly-one-more-forgery → log-one-more-forgery in the underlying scheme

Limit calls by **simulating the signer** when user acts honestly

How do you force the user to not cheat?

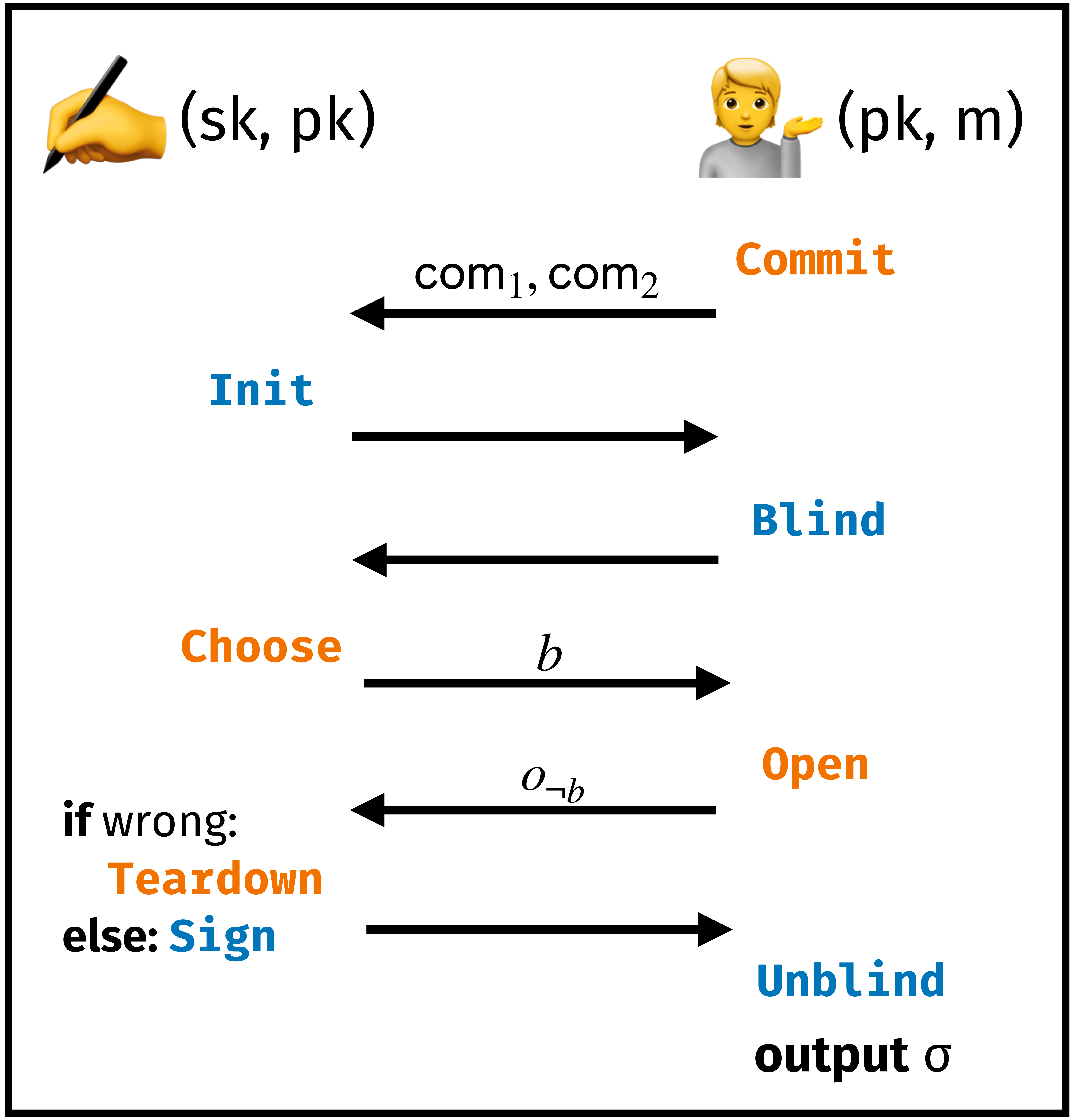
Cut and choose: make two commitments, and later open one at random: $\Pr[\text{cheat}] = 1/2$



Transformed scheme via [Pointcheval98]. poly(κ)-OMUF

Beginning: Pointcheval's Transform

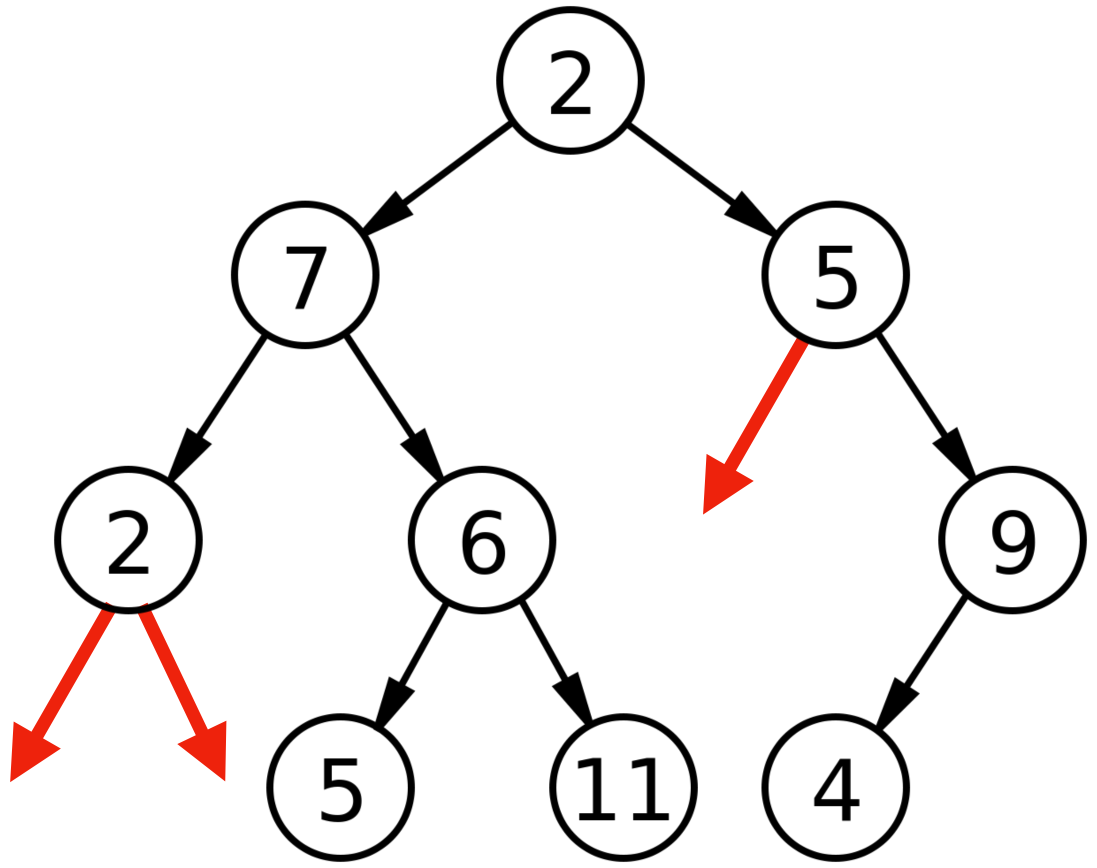
The Upshot

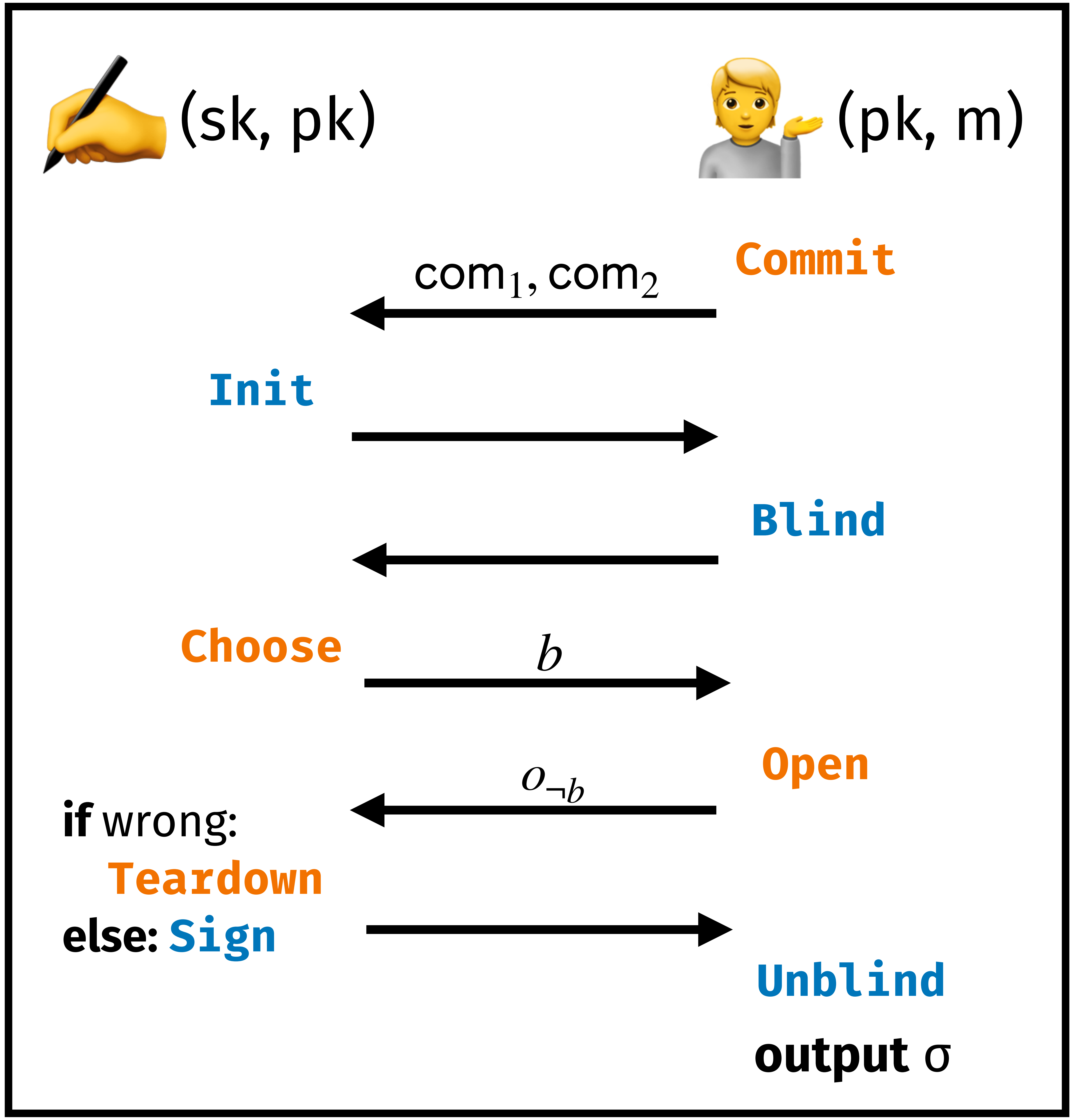


Transformed scheme via [Pointcheval98]. poly(κ)-OMUF

Beginning: Pointcheval's Transform

The Upshot

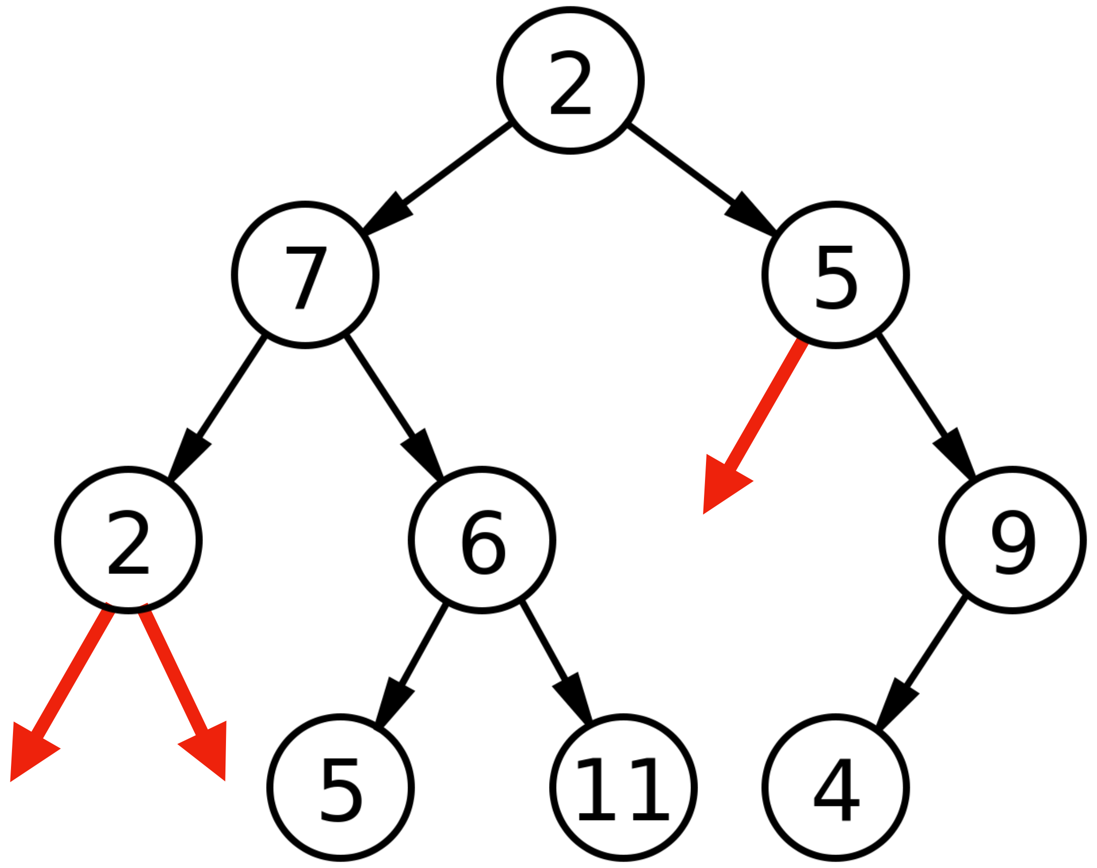




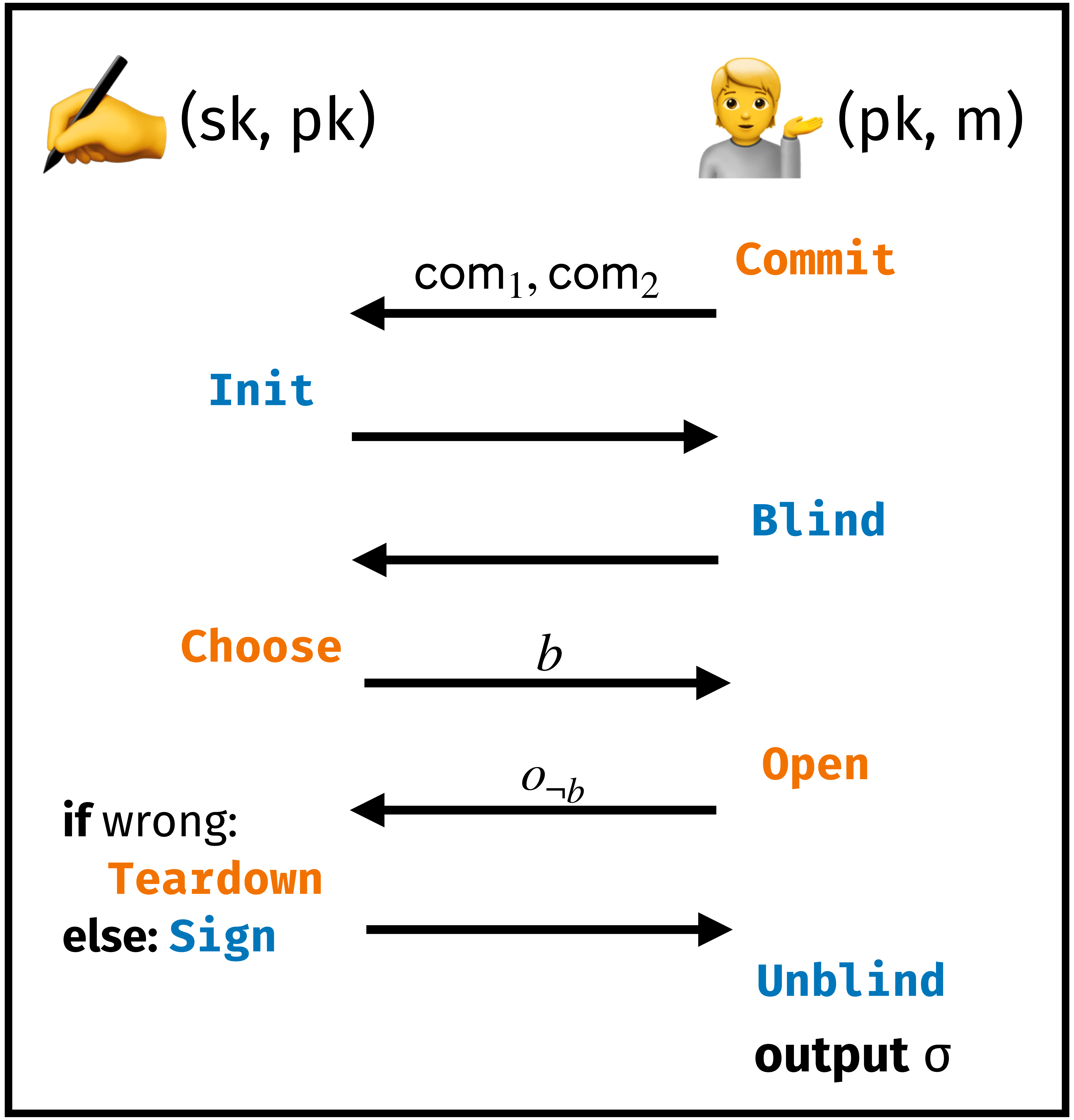
Transformed scheme via [Pointcheval98]. poly(κ)-OMUF

Beginning: Pointcheval's Transform

The Upshot



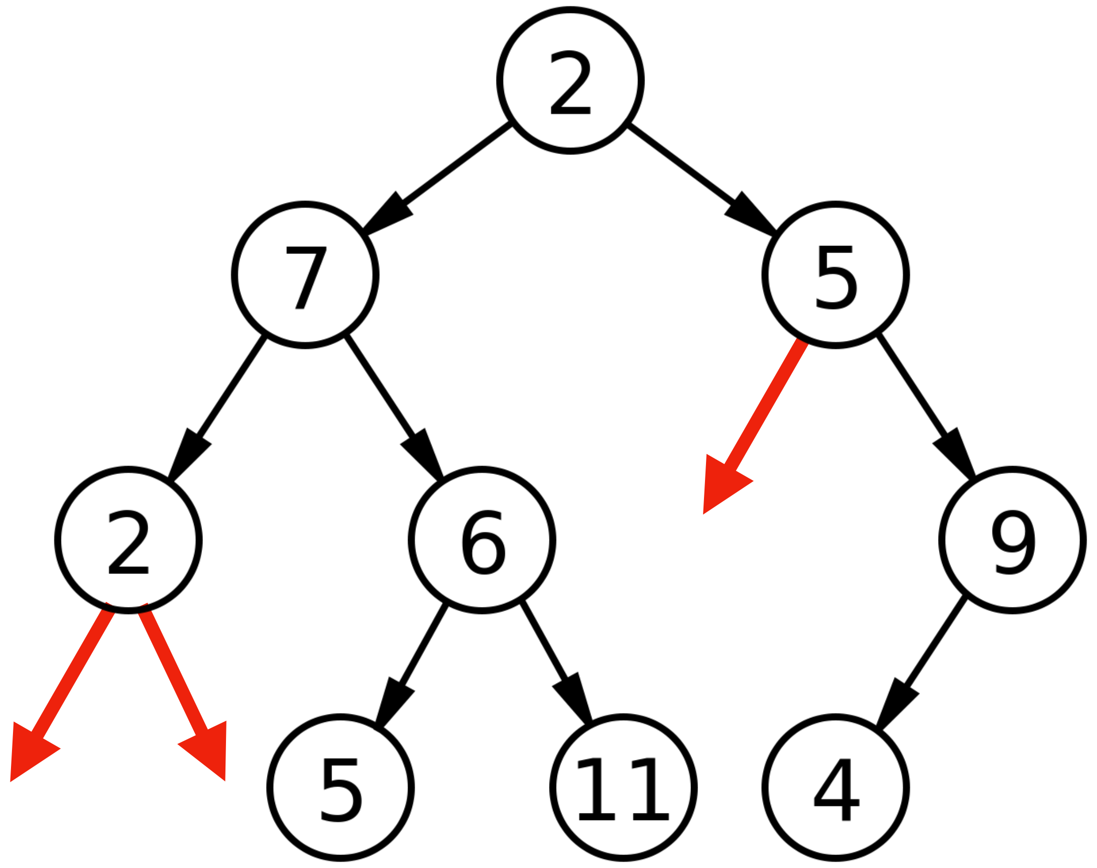
Since Okamoto-Schnorr is $\log(\kappa)$-OMUF,
the **transformed scheme is** $\mathrm{poly}(\kappa)$**-OMUF**



Transformed scheme via [Pointcheval98]. $\mathrm{poly}(\kappa)$-OMUF

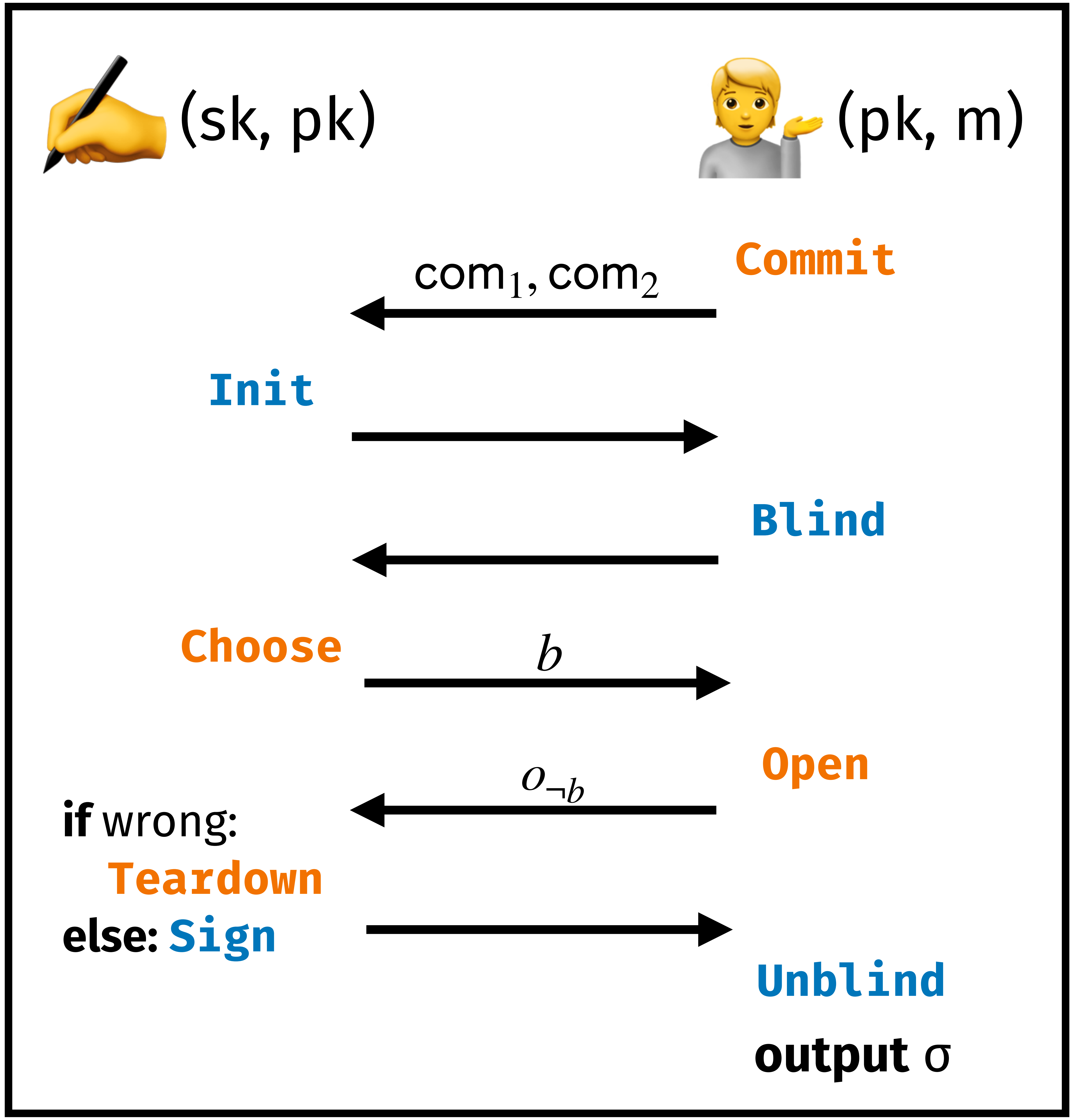
Beginning: Pointcheval's Transform

The Upshot



Since Okamoto-Schnorr is $\log(\kappa)$-OMUF, the **transformed scheme is** $\mathrm{poly}(\kappa)$**-OMUF**

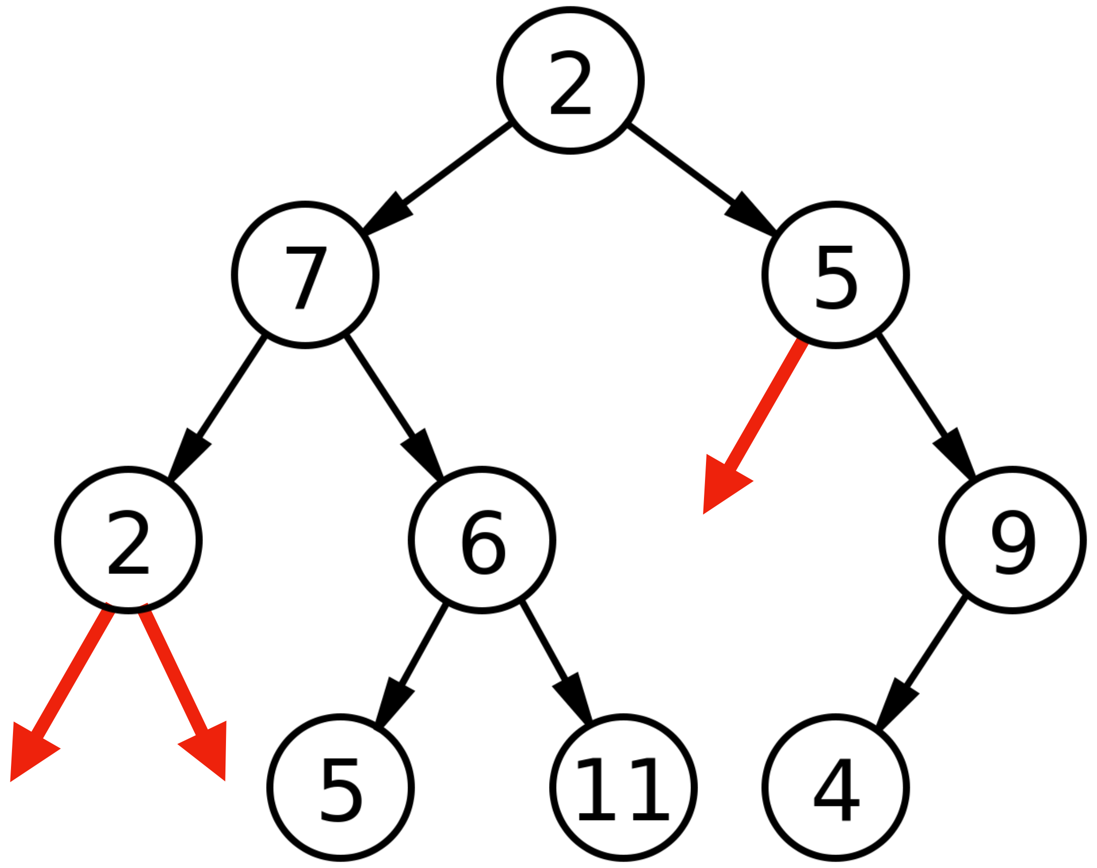
Caveats:



Transformed scheme via [Pointcheval98]. $\mathrm{poly}(\kappa)$-OMUF

Beginning: Pointcheval's Transform

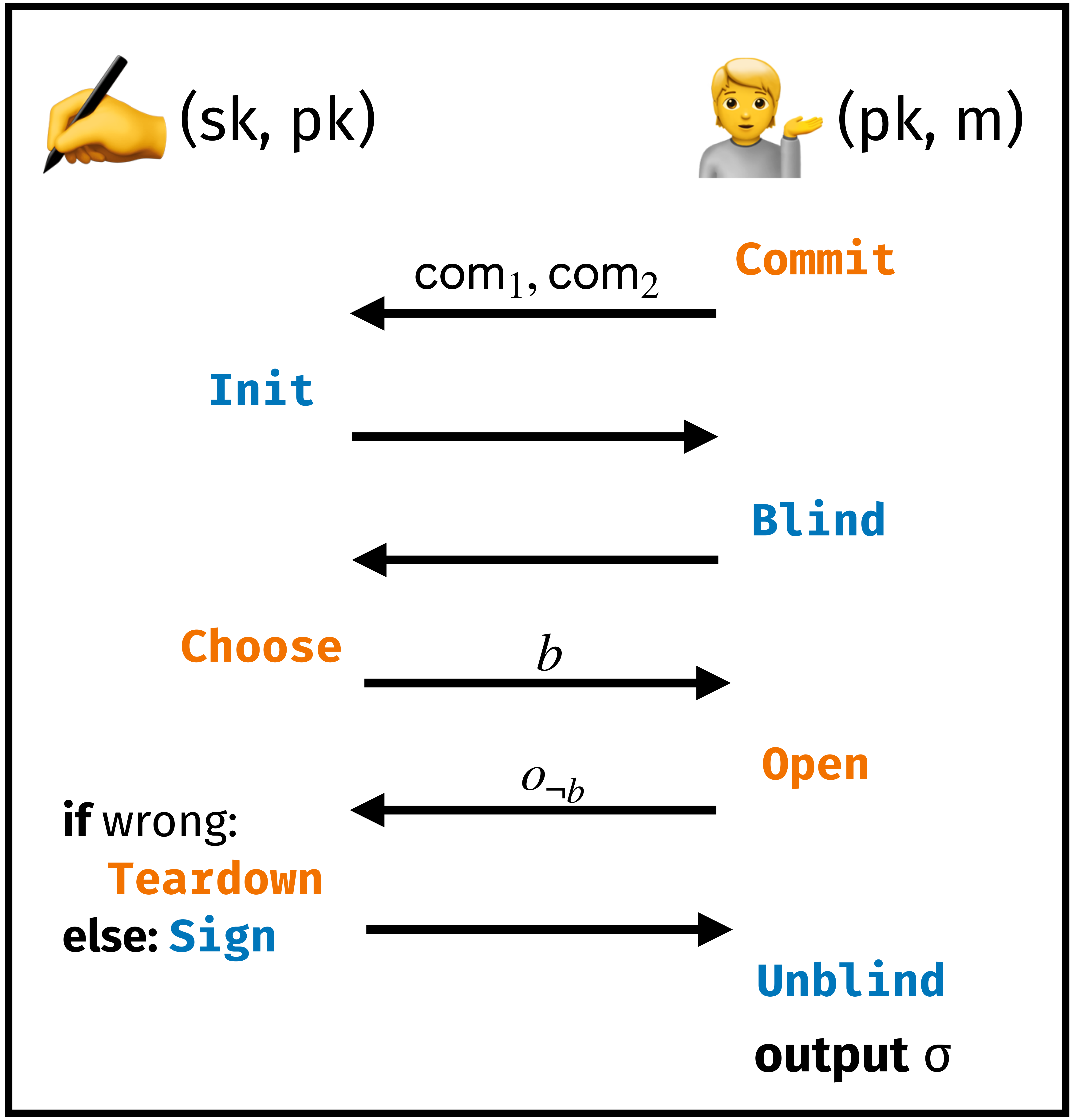
The Upshot



Since Okamoto-Schnorr is $\log(\kappa)$-OMUF, the **transformed scheme is** $\text{poly}(\kappa)$**-OMUF**

Caveats:

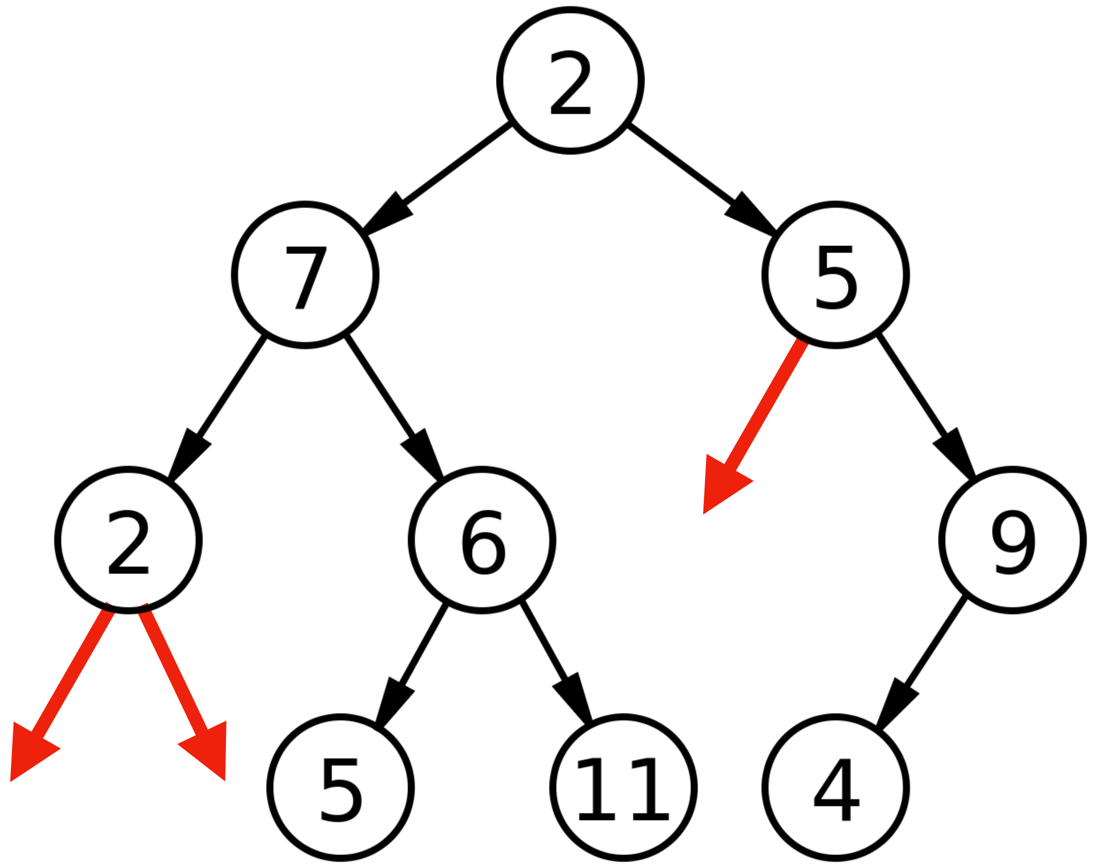
- Parallel, not arbitrarily concurrent



Transformed scheme via [Pointcheval98]. $\text{poly}(\kappa)$-OMUF

Beginning: Pointcheval's Transform

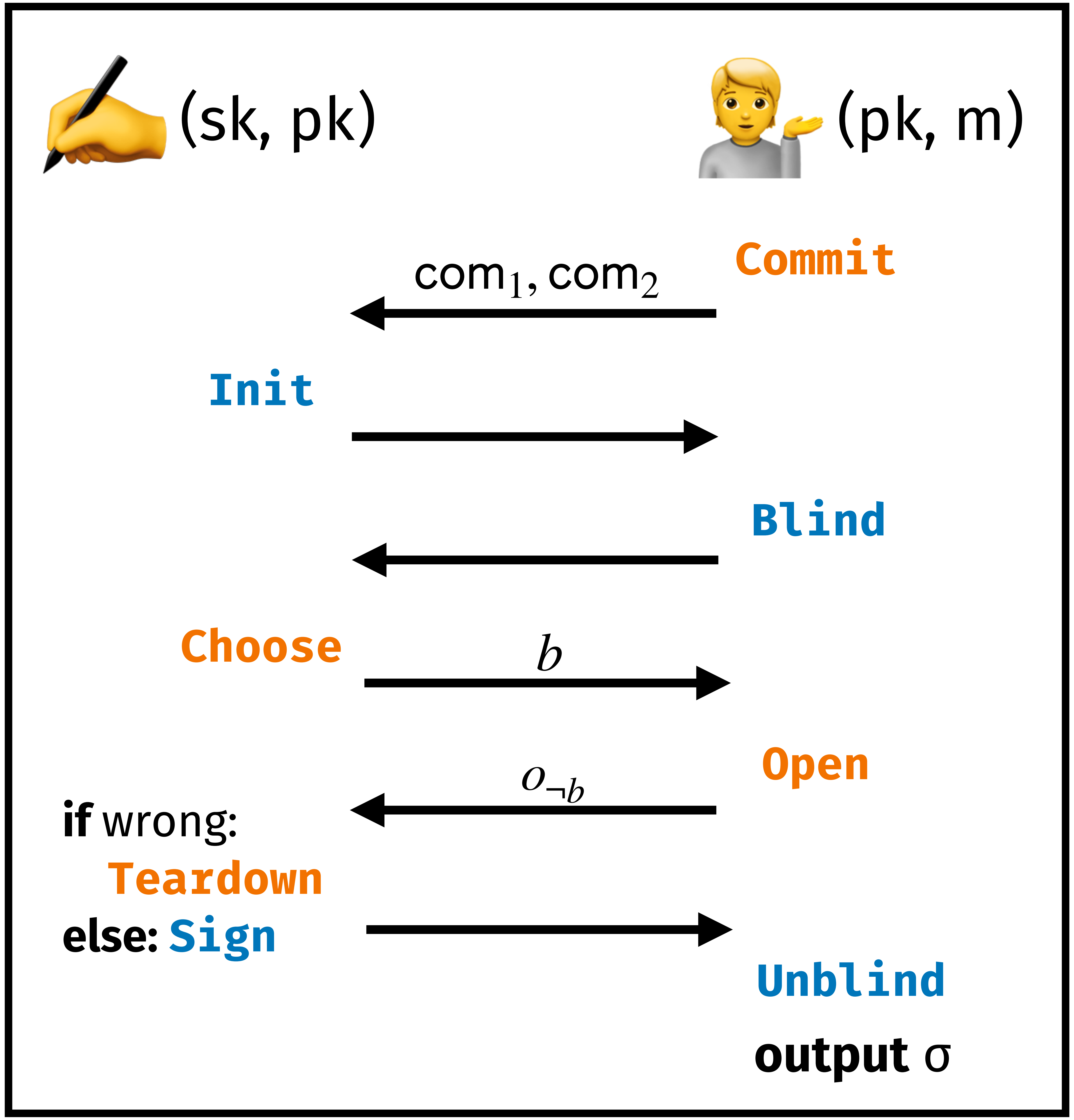
The Upshot



Since Okamoto-Schnorr is $\log(\kappa)$-OMUF, the **transformed scheme is** $\text{poly}(\kappa)$**-OMUF**

Caveats:

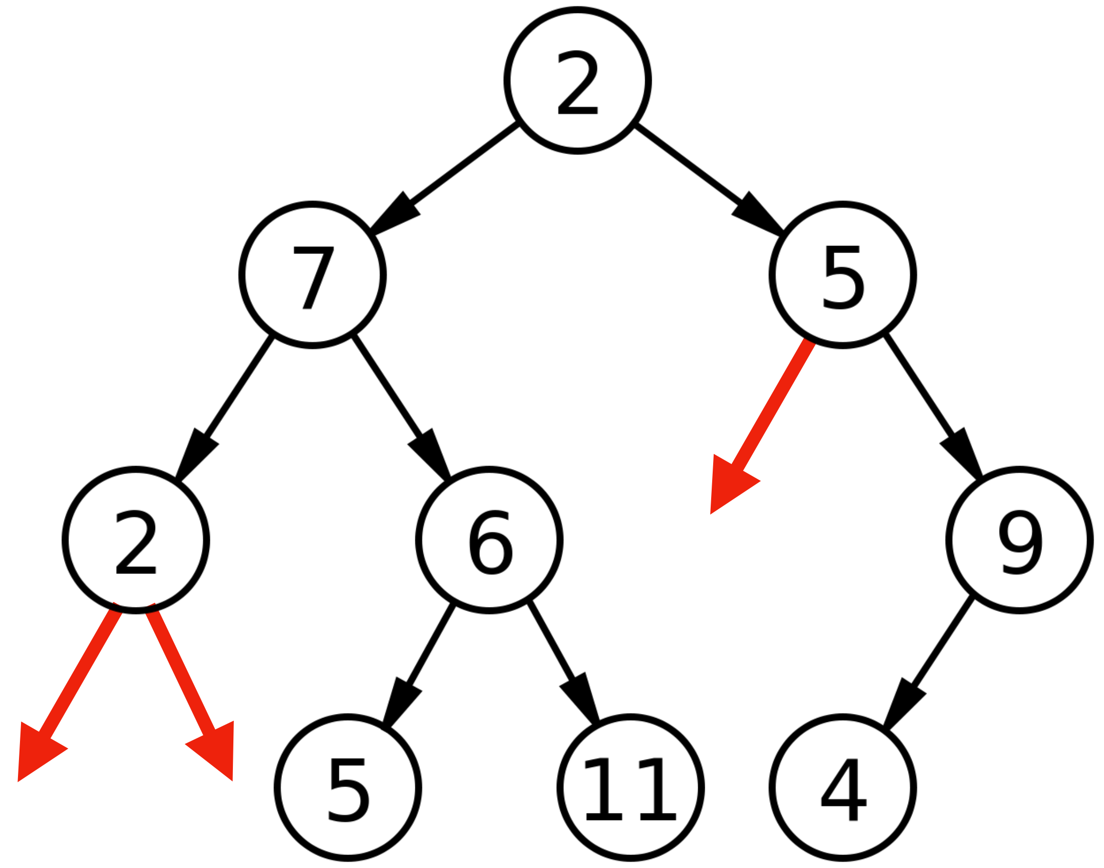
- Parallel, not arbitrarily concurrent
- Teardown on cheat detection



Transformed scheme via [Pointcheval98]. $\text{poly}(\kappa)$-OMUF

Beginning: Pointcheval's Transform

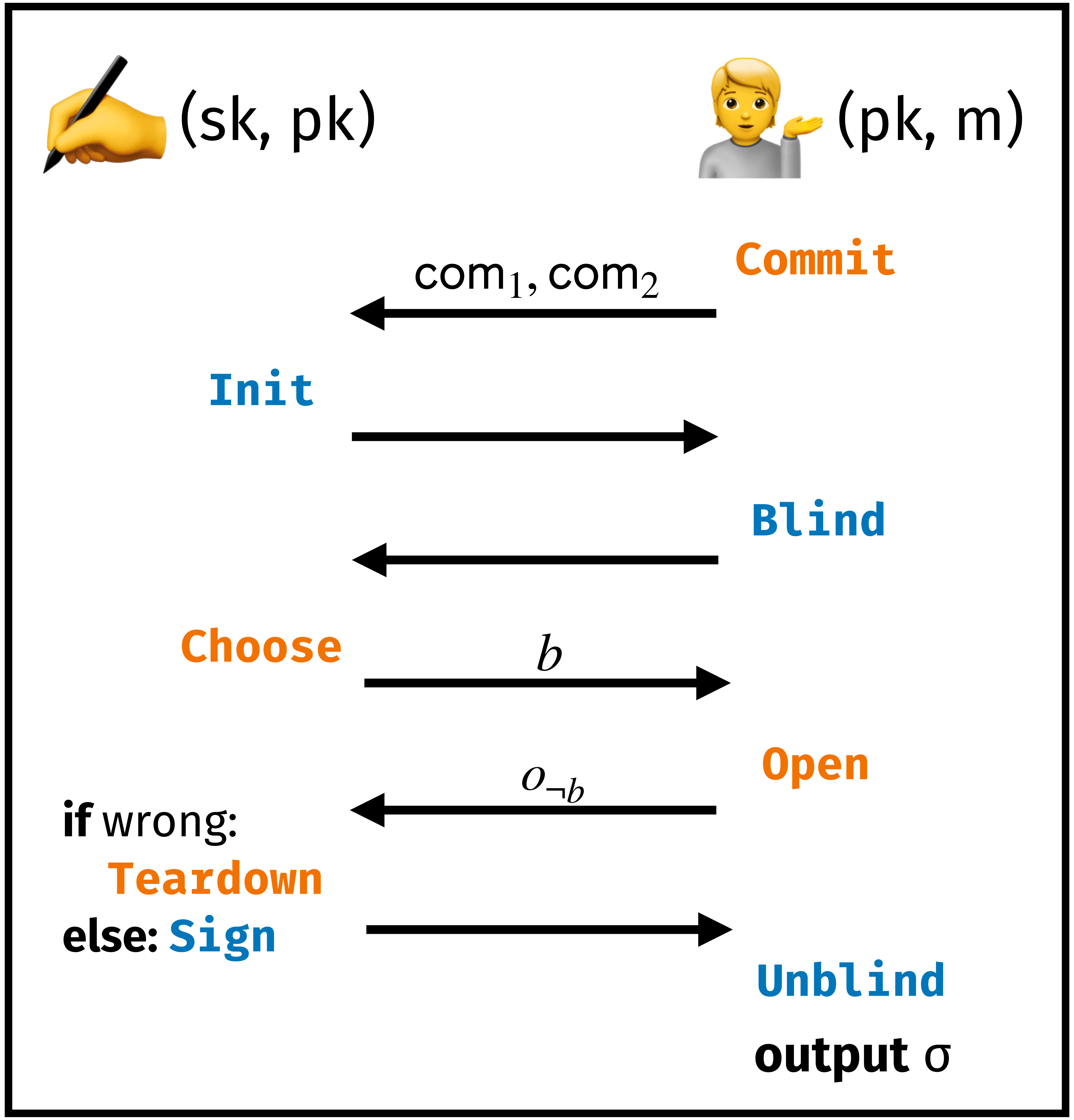
The Upshot



Since Okamoto-Schnorr is $\log(\kappa)$-OMUF, the **transformed scheme is** $\mathrm{poly}(\kappa)$**-OMUF**

Caveats:

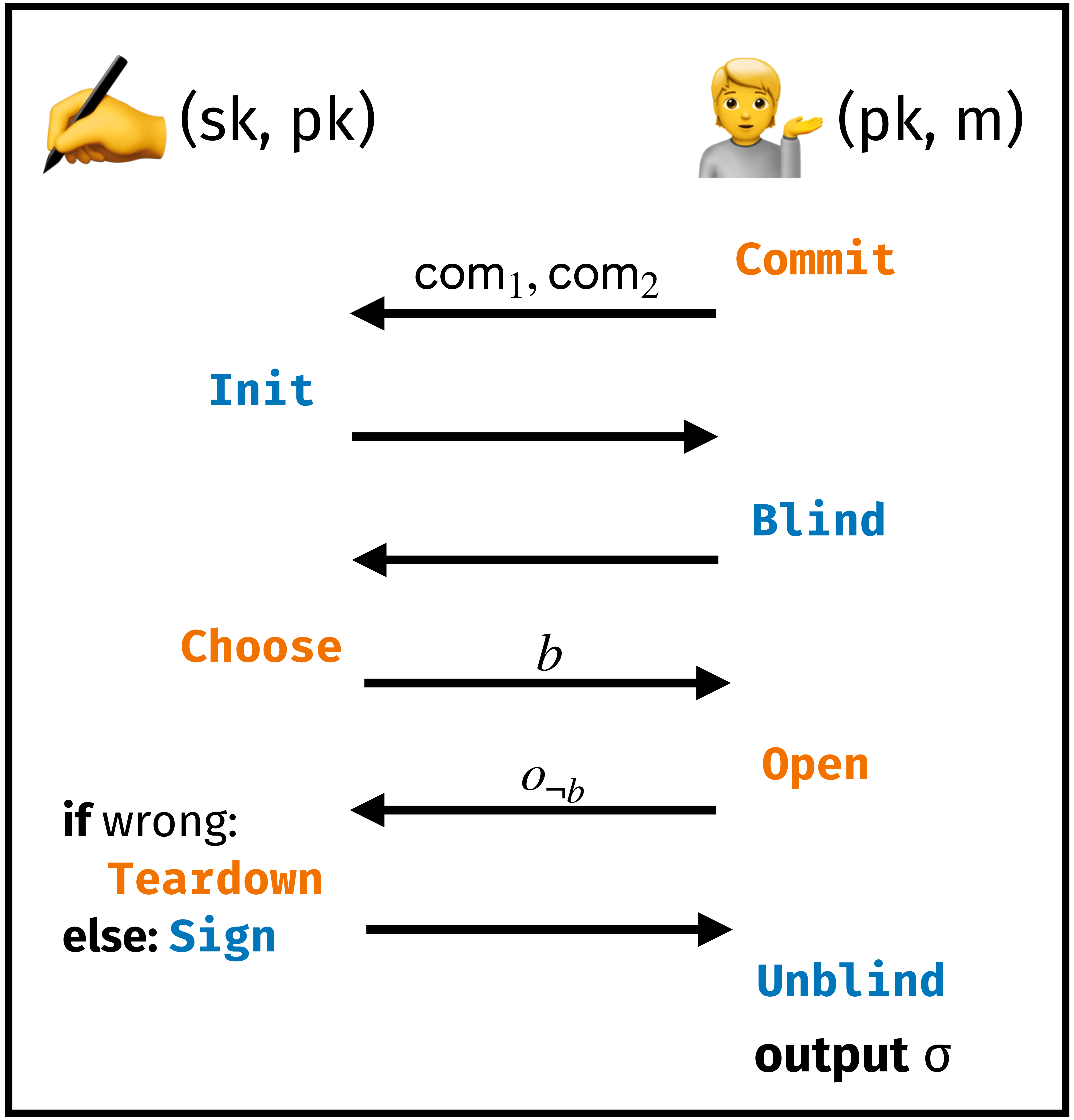
- Parallel, not arbitrarily concurrent
- Teardown on cheat detection
- Not generic



Transformed scheme via [Pointcheval98]. $\mathrm{poly}(\kappa)$-OMUF

Our Transform

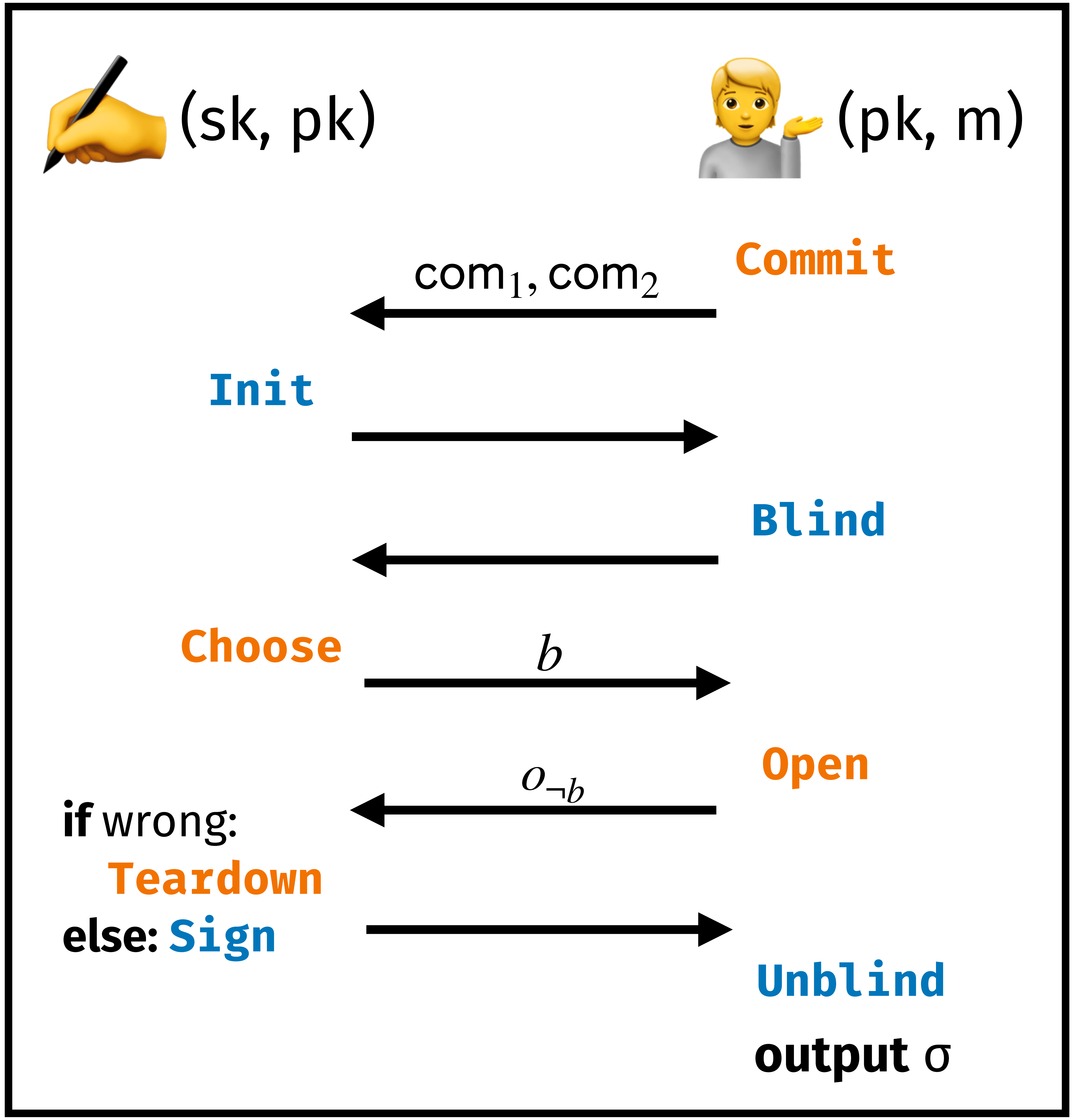
Construction



Our Transform

Construction

Protocol depends on how many sessions have begun

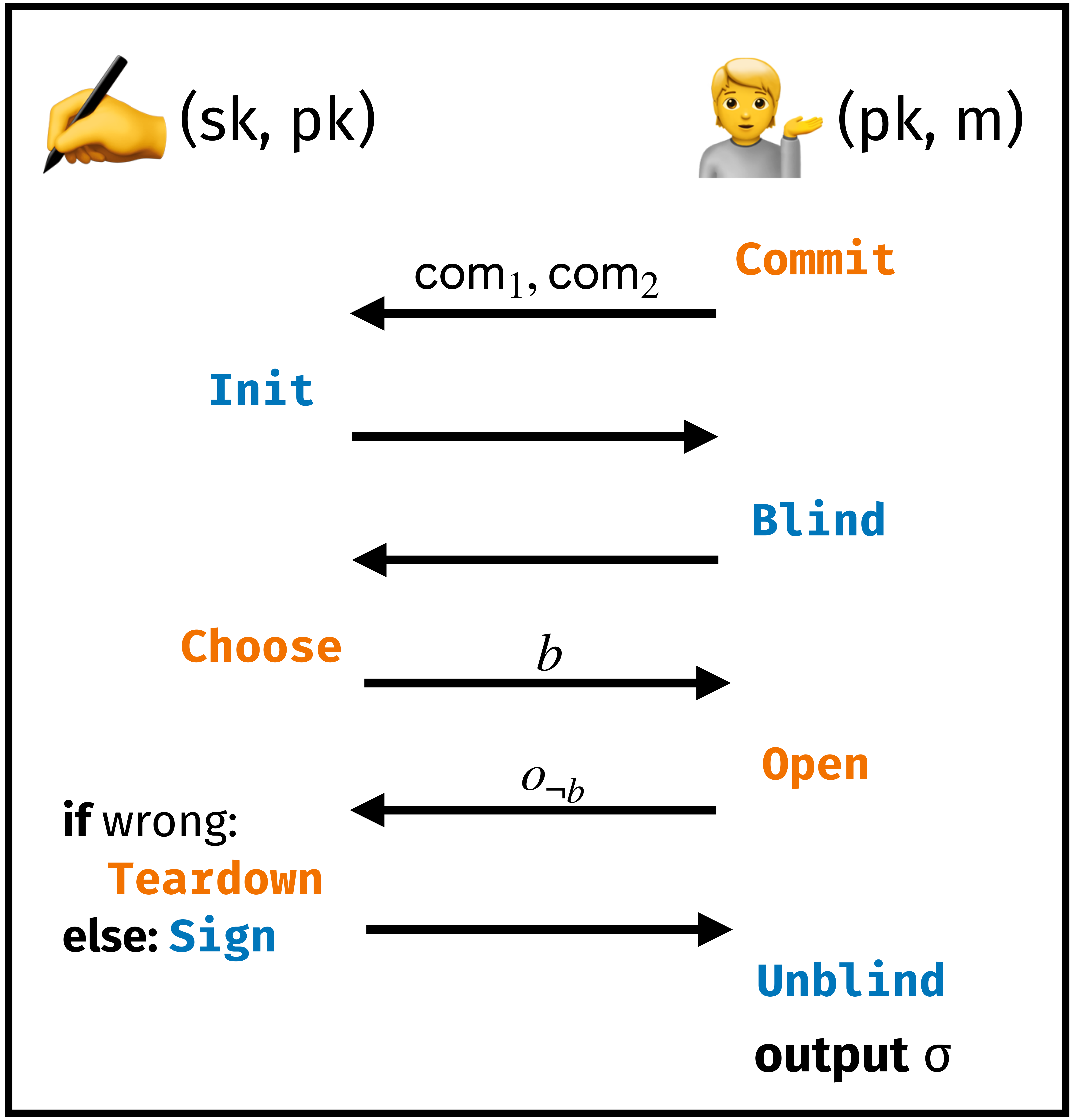


Our Transform

Construction

Protocol depends on how many sessions have begun

In session N:



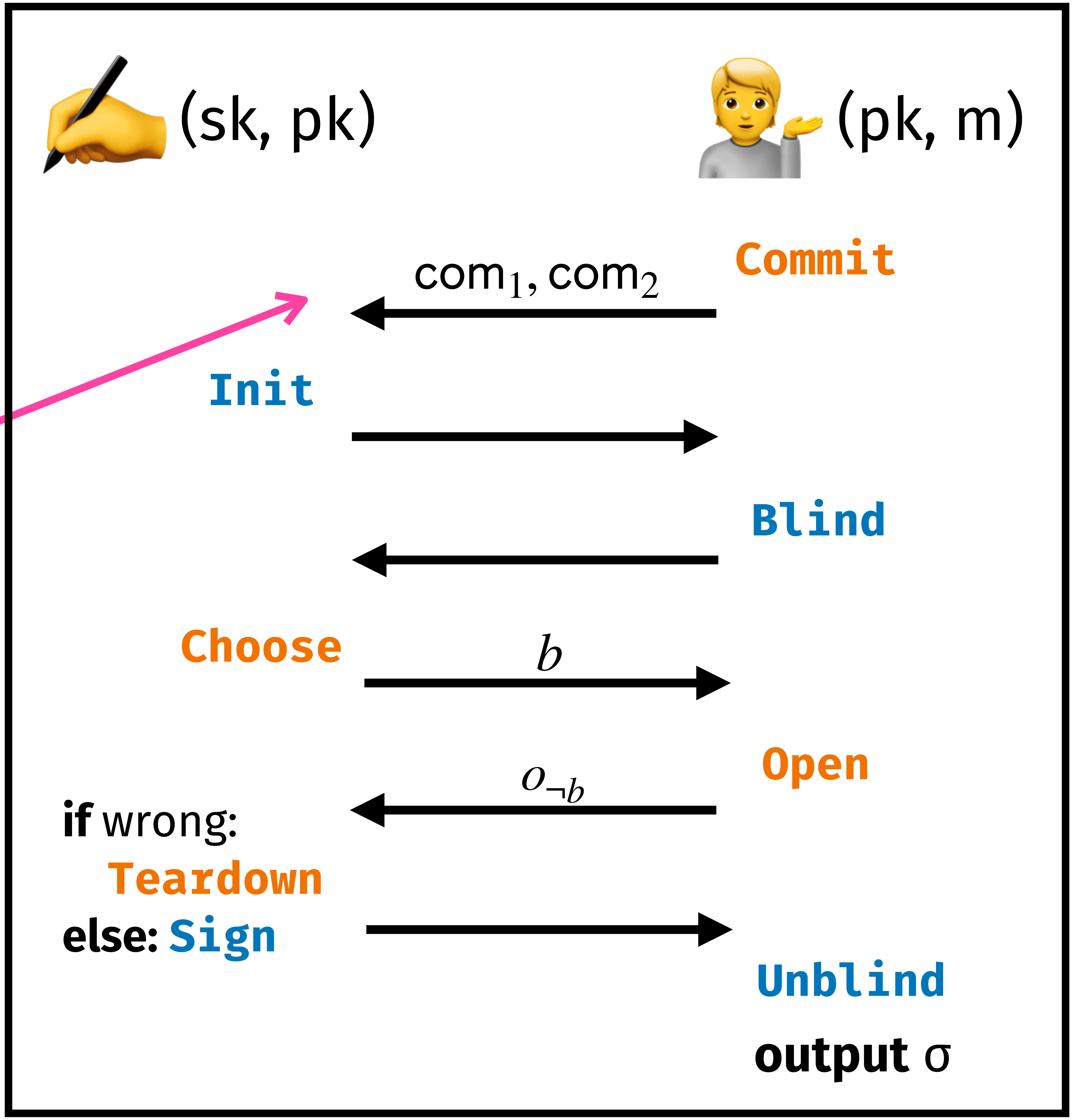
Our Transform

Construction

Protocol depends on how many sessions have begun

In session N:

User commits to N sets of random coins



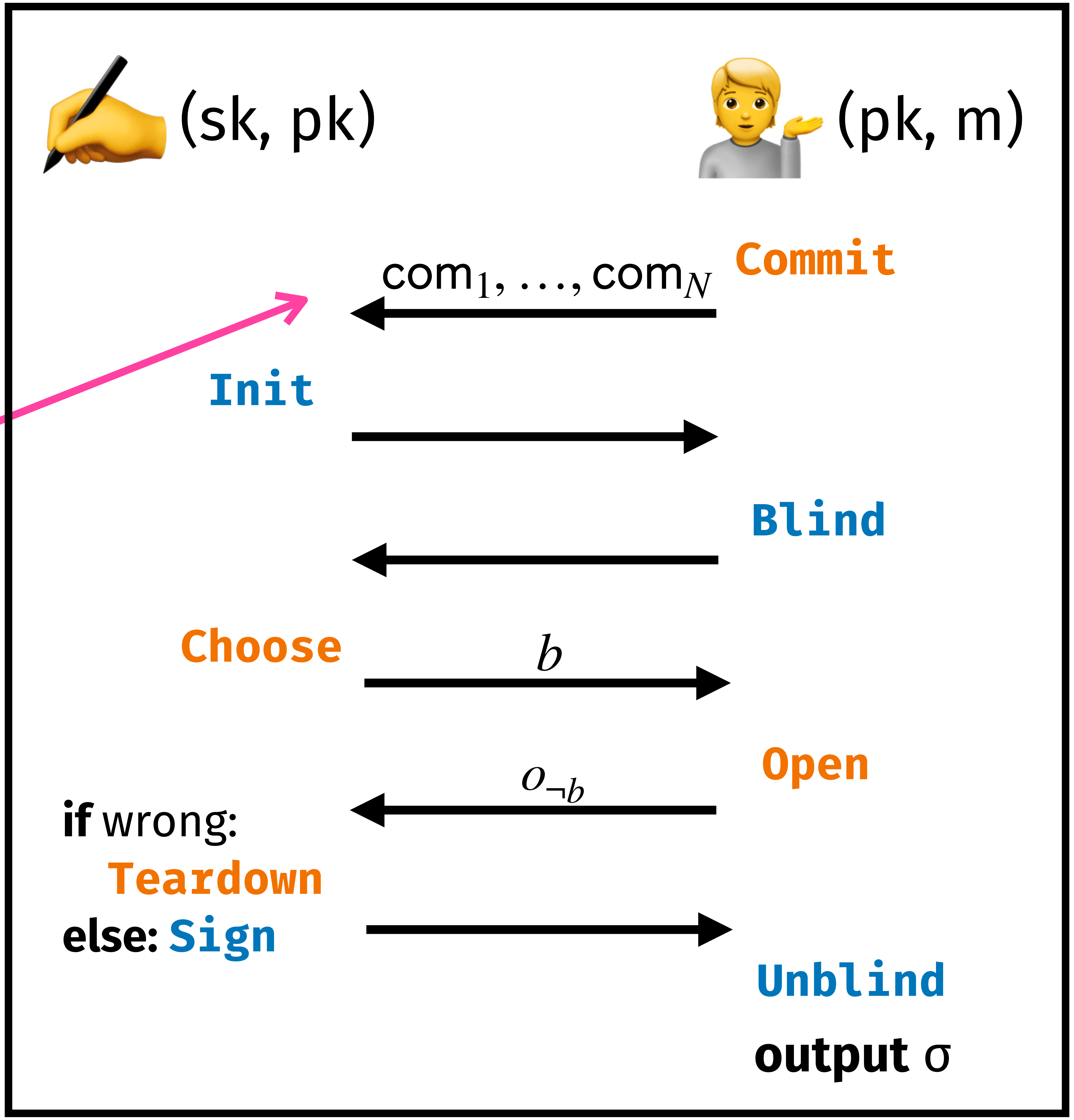
Our Transform

Construction

Protocol depends on how many sessions have begun

In session N:

User commits to N sets of random coins



Our Transform

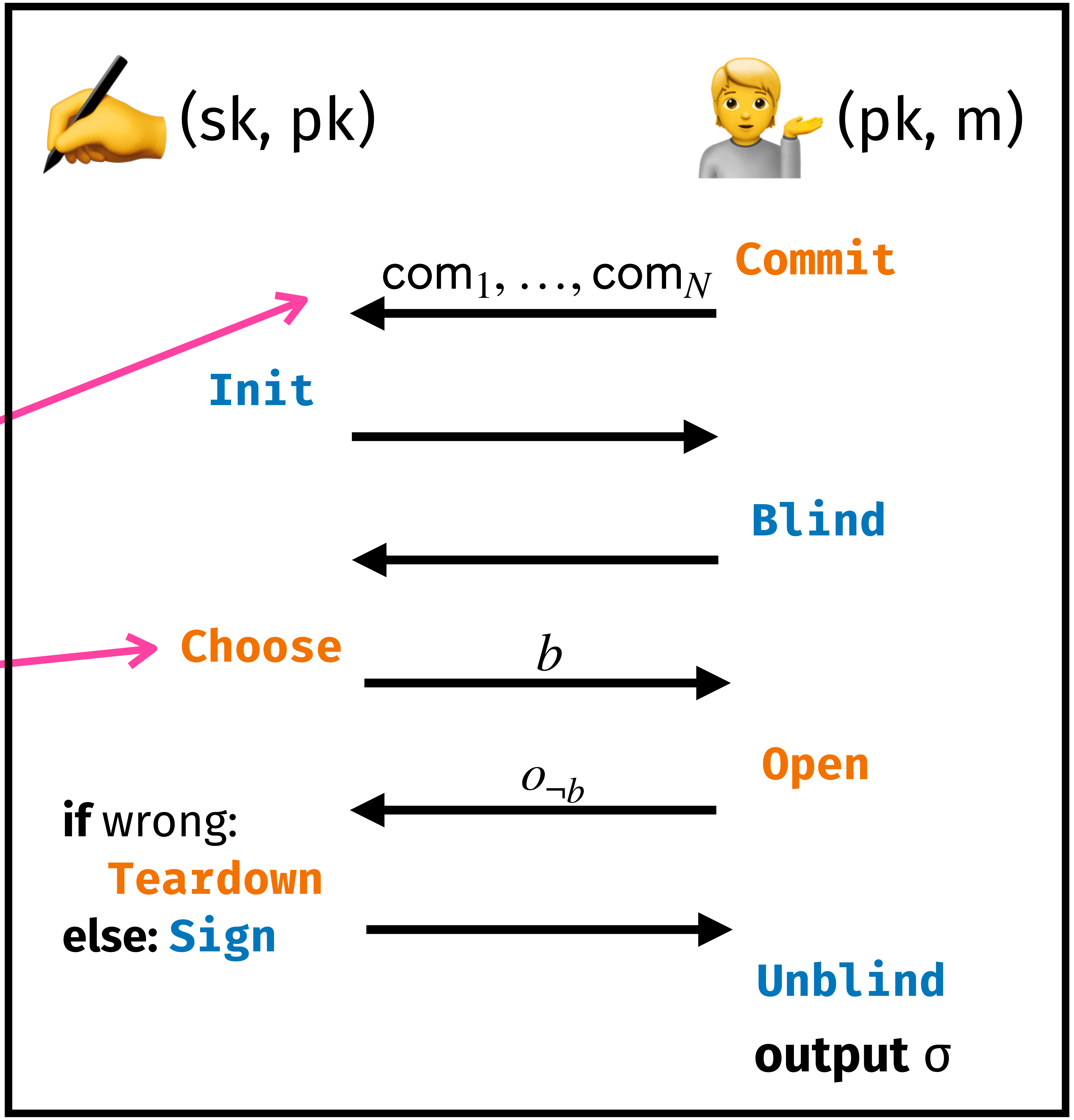
Construction

Protocol depends on how many sessions have begun

In session N:

User commits to N sets of random coins

Signer picks an I



Our Transform

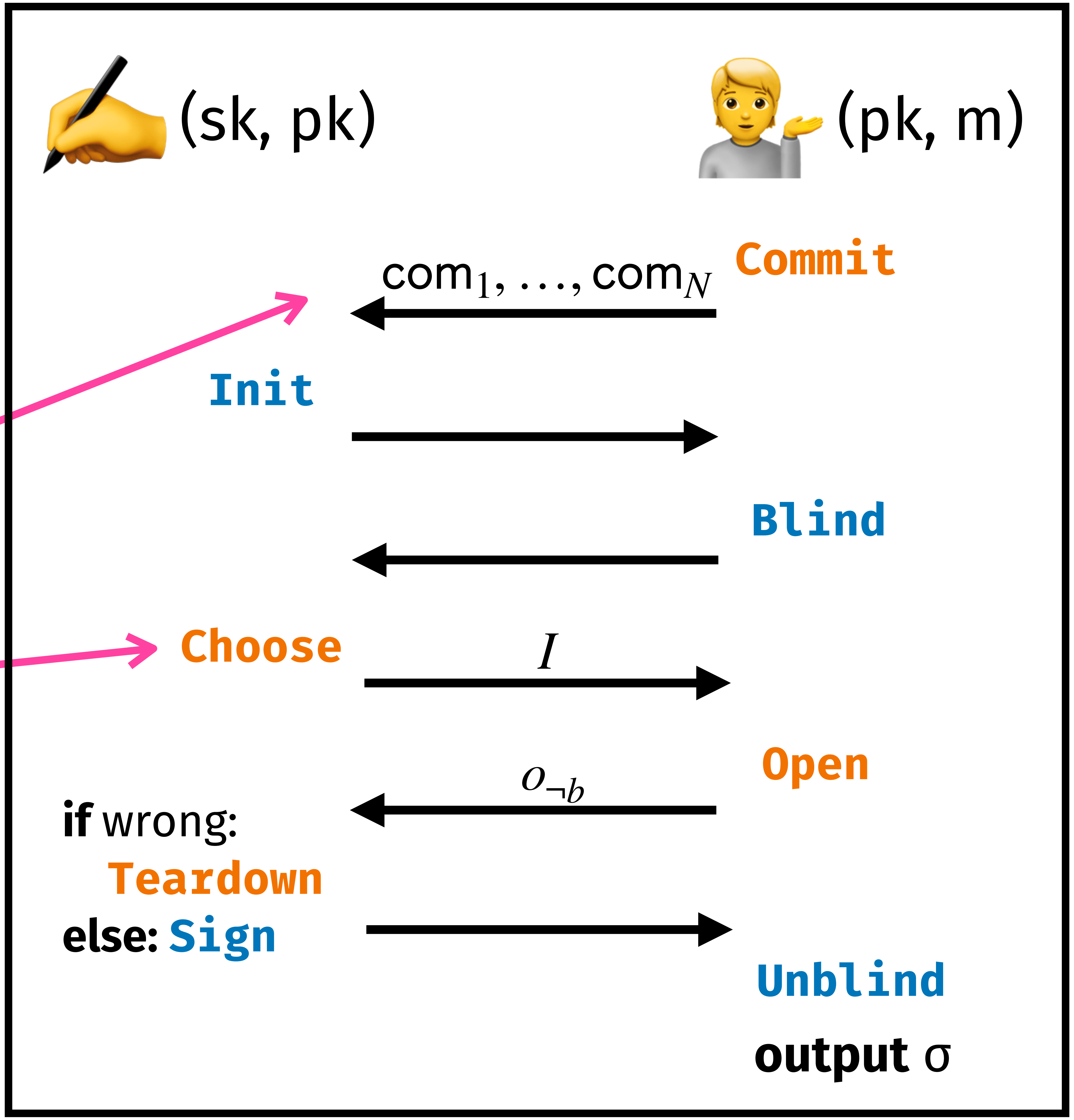
Construction

Protocol depends on how many sessions have begun

In session N:

User commits to N sets of random coins

Signer picks an I



Our Transform

Construction

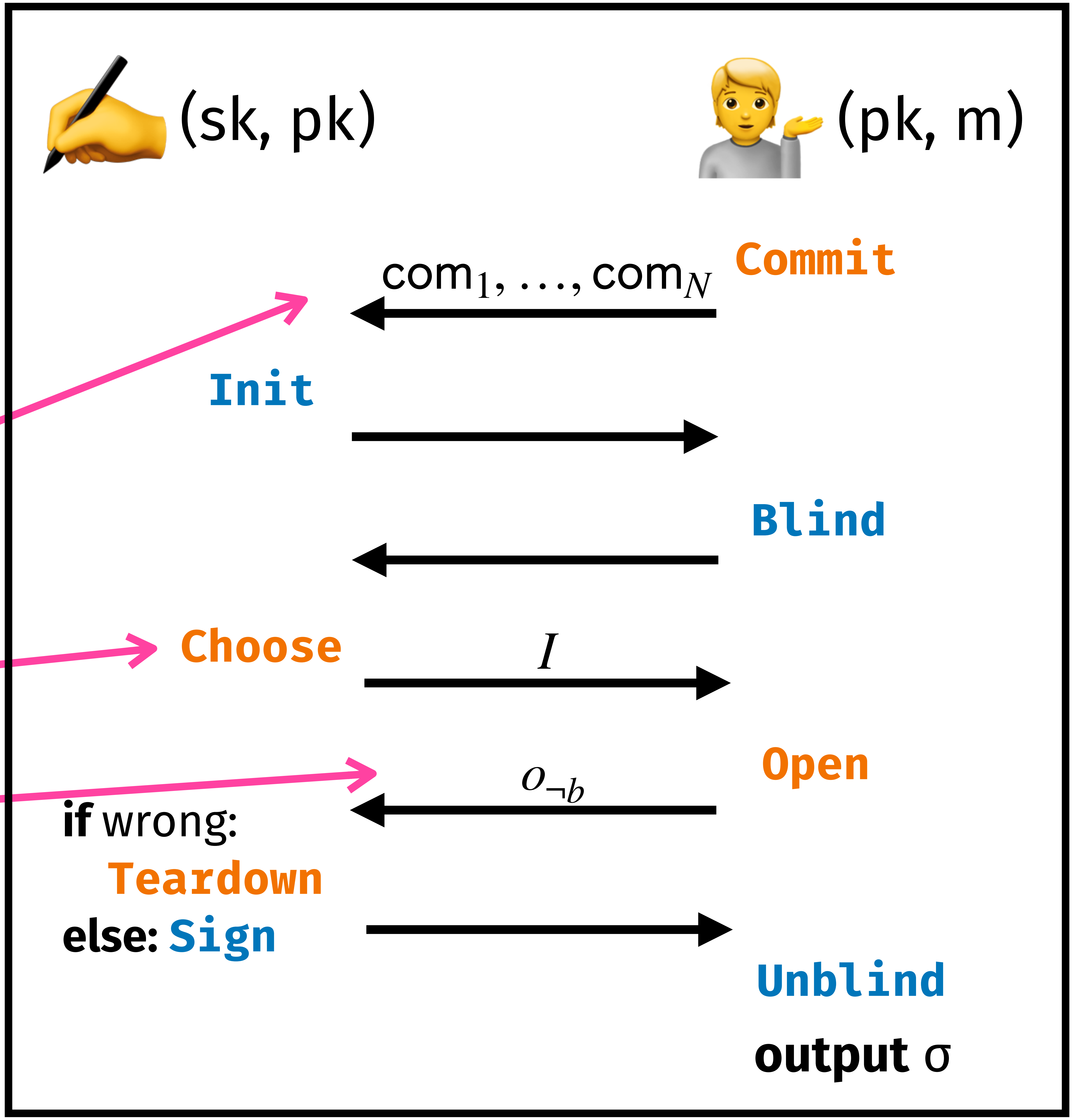
Protocol depends on how many sessions have begun

In session N:

User commits to N sets of random coins

Signer picks an I

User opens all but the I-th com



Our Transform

Construction

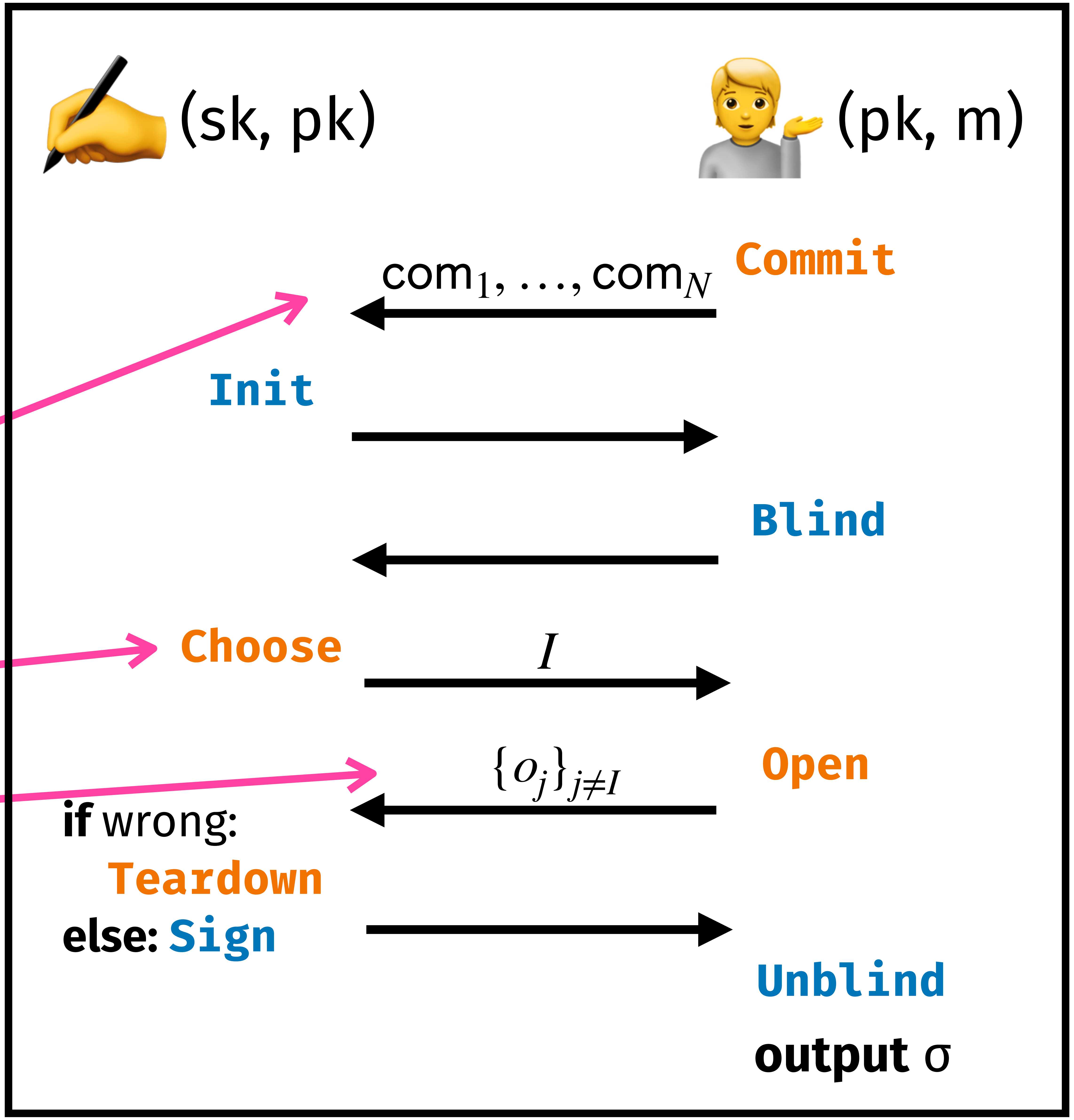
Protocol depends on how many sessions have begun

In session N:

User commits to N sets of random coins

Signer picks an I

User opens all but the I-th com



Our Transform

Construction

Protocol depends on how many sessions have begun

In session N:

User commits to N sets of random coins

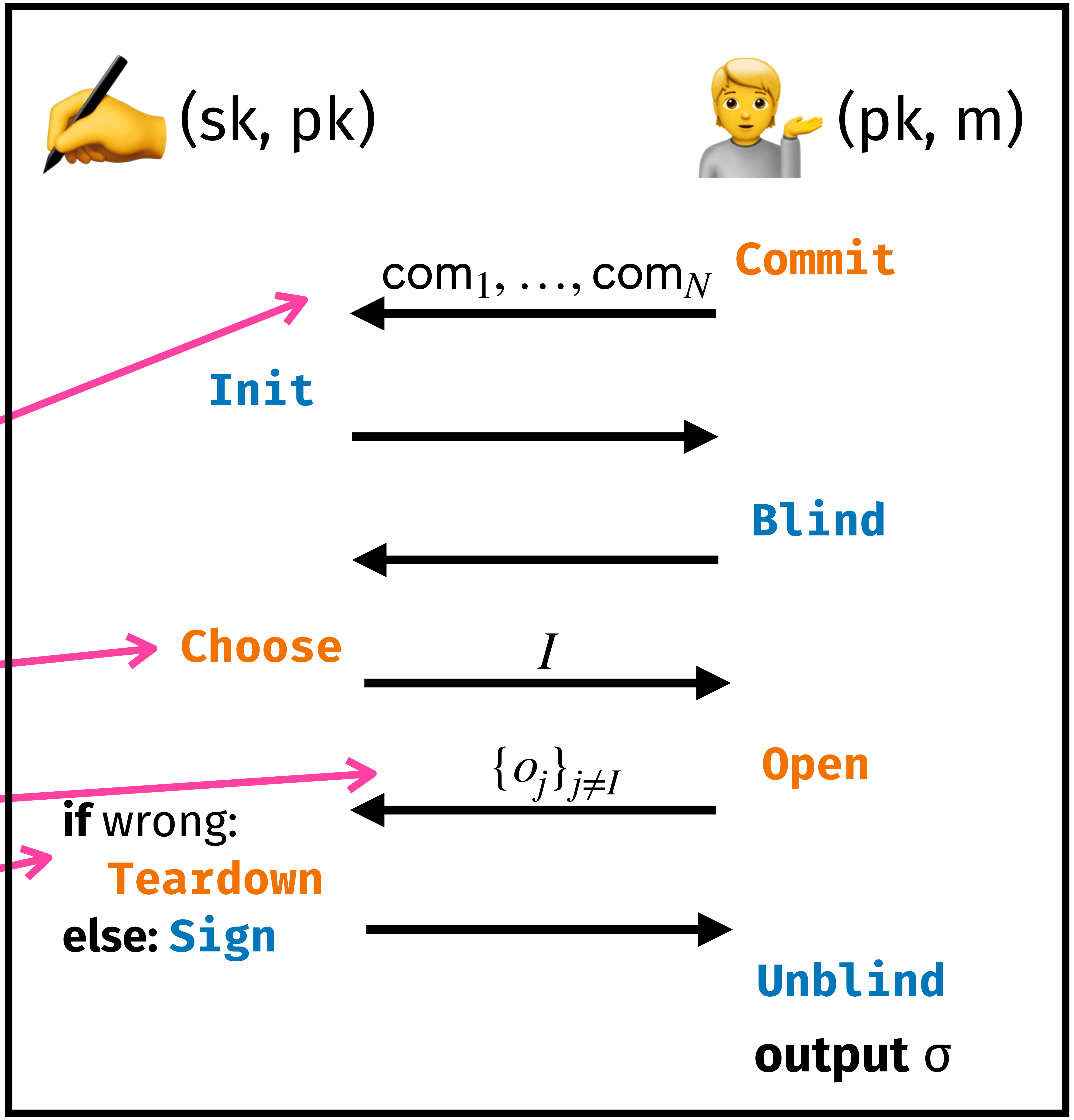
Signer picks an I

User opens all but the I-th com

Signer checks the opening

Success: sign

Failure: abort



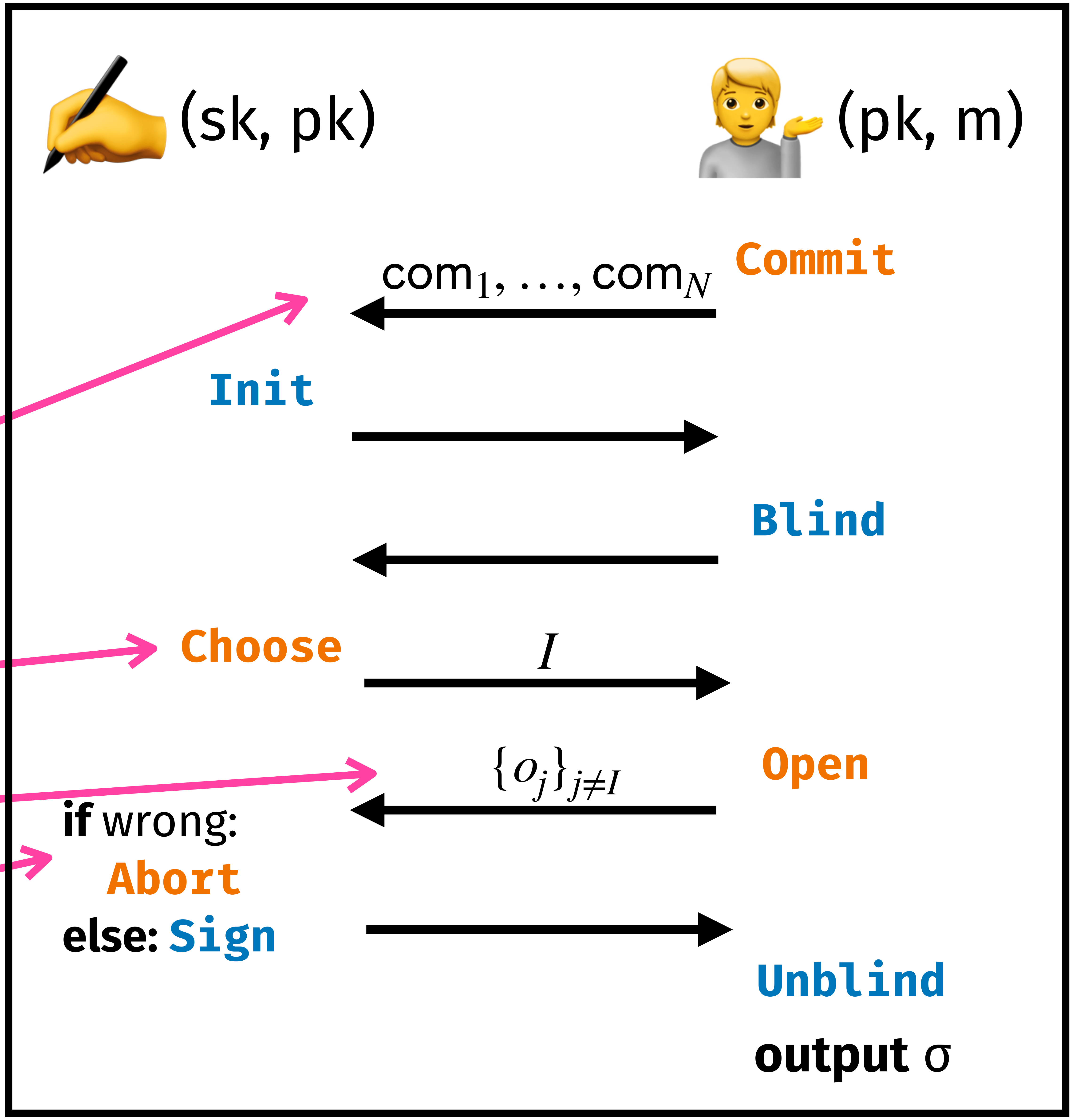
Our Transform

Construction

Protocol depends on how many sessions have begun

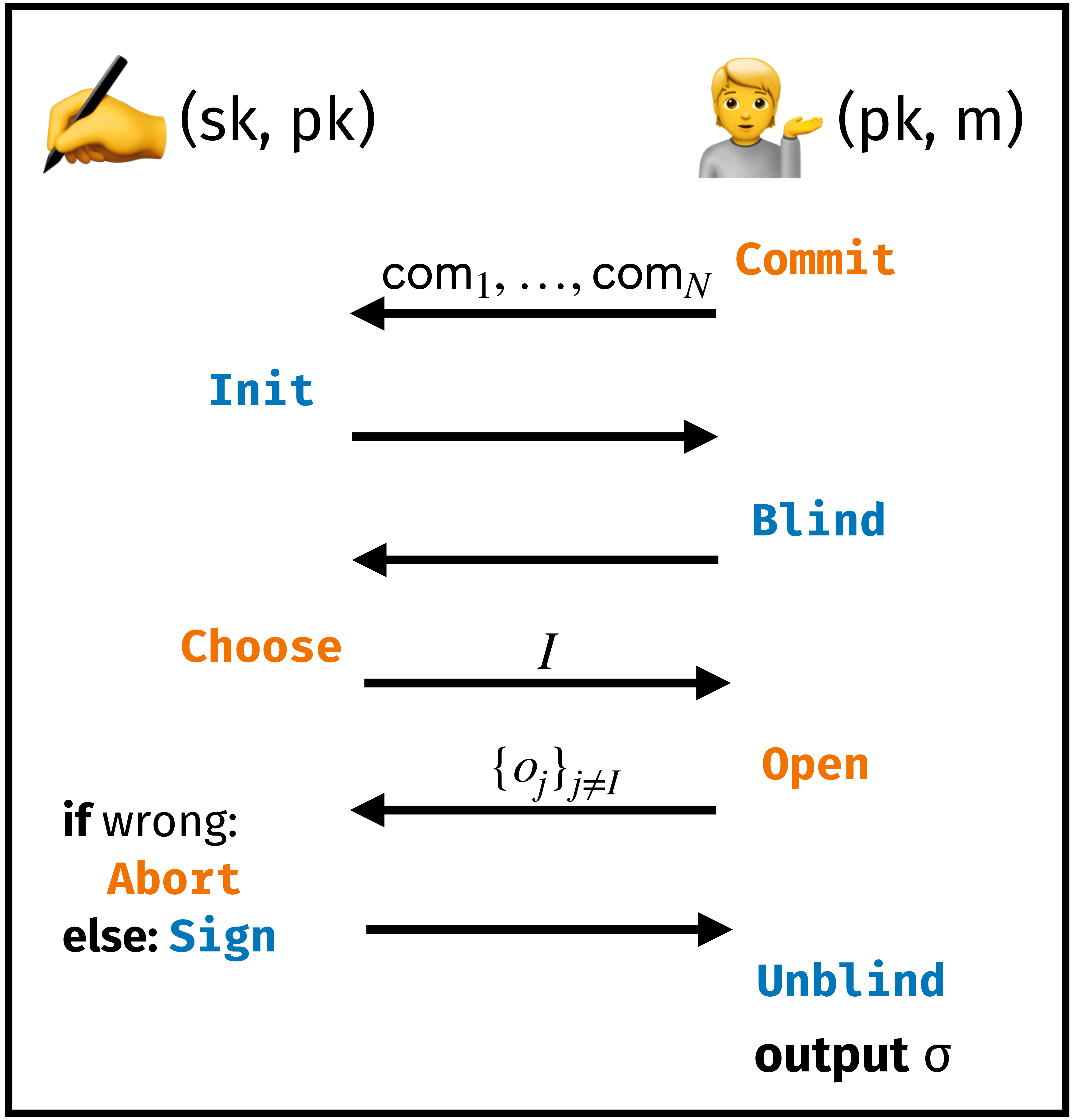
In session N:

- User commits to N sets of random coins
- Signer picks an I
- User opens all but the I-th com
- Signer checks the opening
  - **Success:** sign
  - **Failure: abort**



Our Transform

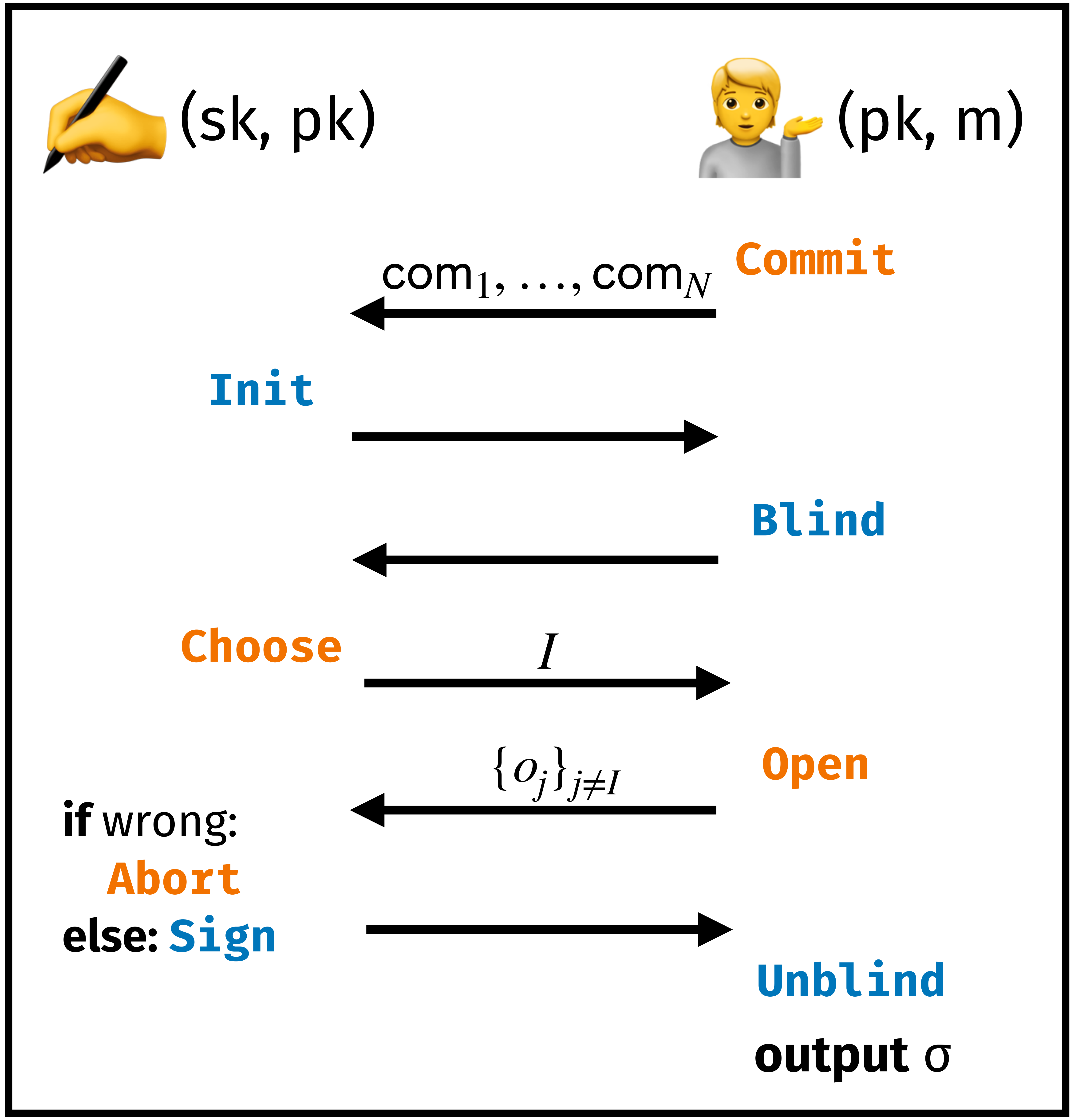
Analysis - OMUF



Our Transform

Analysis - OMUF

Recall: Underlying scheme is only secure when #cheats = $O(\log(\kappa))$

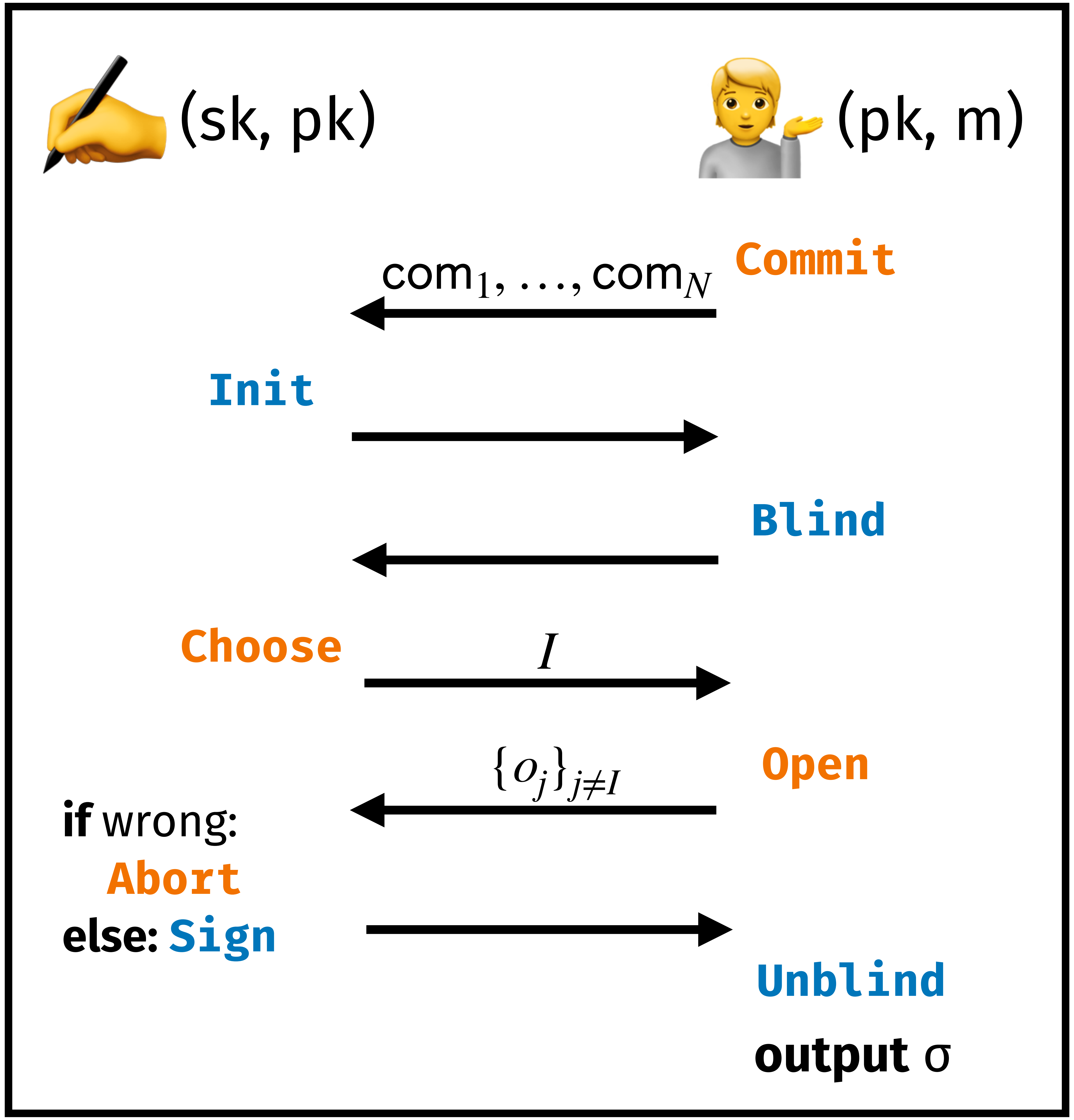


Our Transform

Analysis - OMUF

Recall: Underlying scheme is only secure when #cheats = $O(\log(\kappa))$

Previously



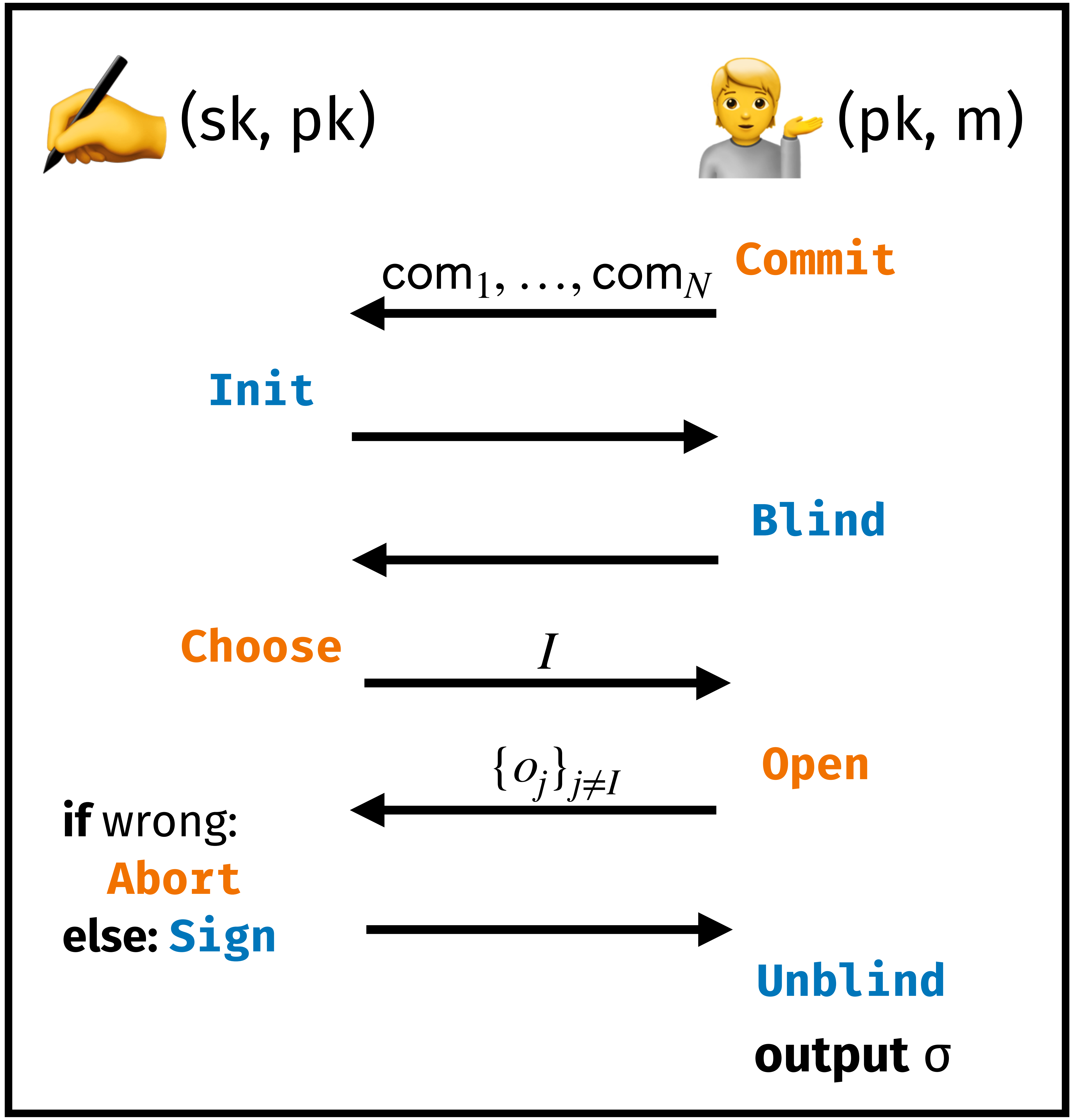
Our Transform

Analysis - OMUF

Recall: Underlying scheme is only secure when #cheats = $O(\log(\kappa))$

Previously

$\Pr[\text{cheat}] = 1/2$ each session



Our Transform

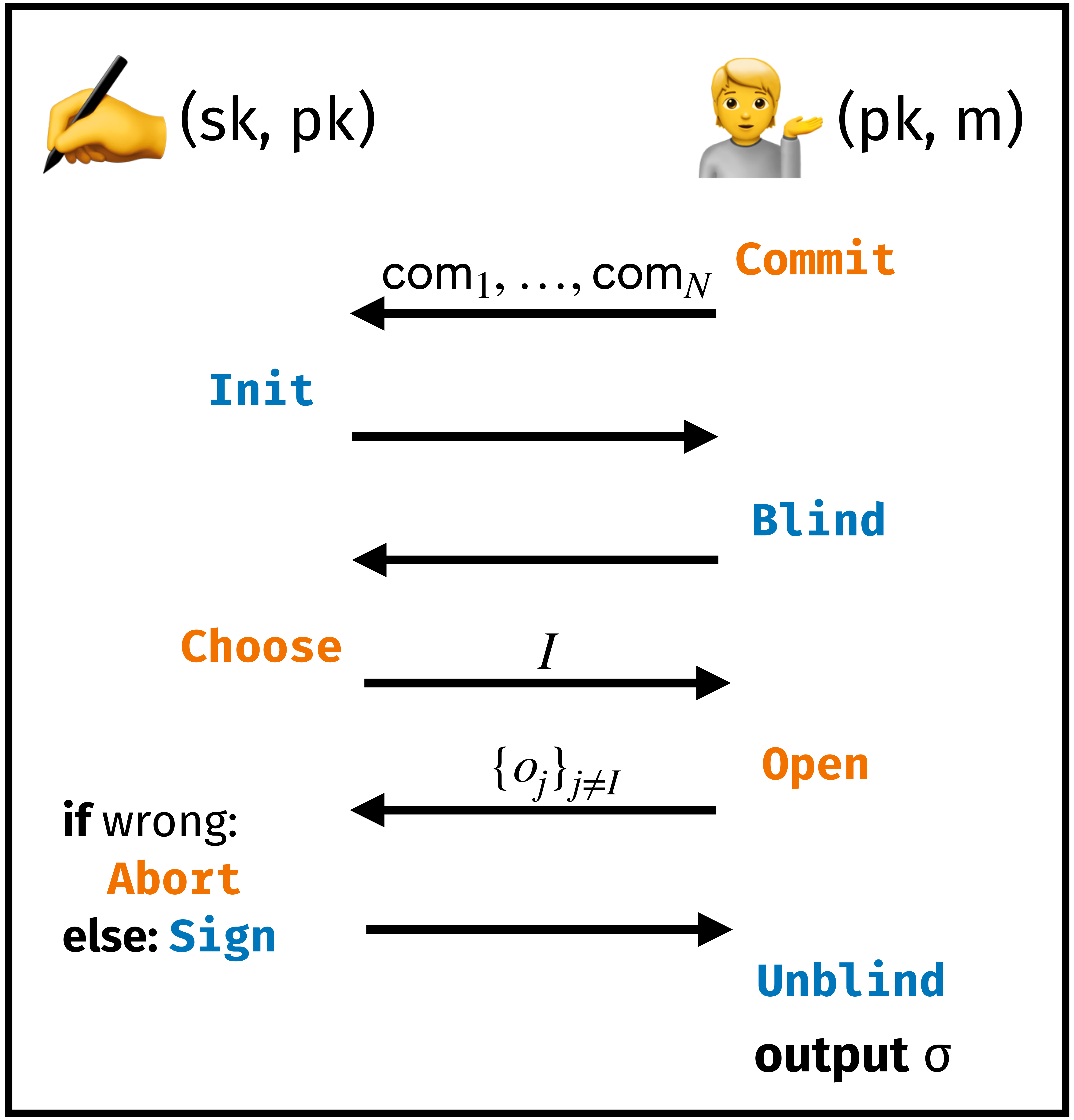
Analysis - OMUF

Recall: Underlying scheme is only secure when #cheats = $O(\log(\kappa))$

Previously

$\Pr[\text{cheat}] = 1/2$ each session

Without teardown, after s sessions $\mathbb{E}[\#\text{cheats}] = s/2$



Our Transform

Analysis - OMUF

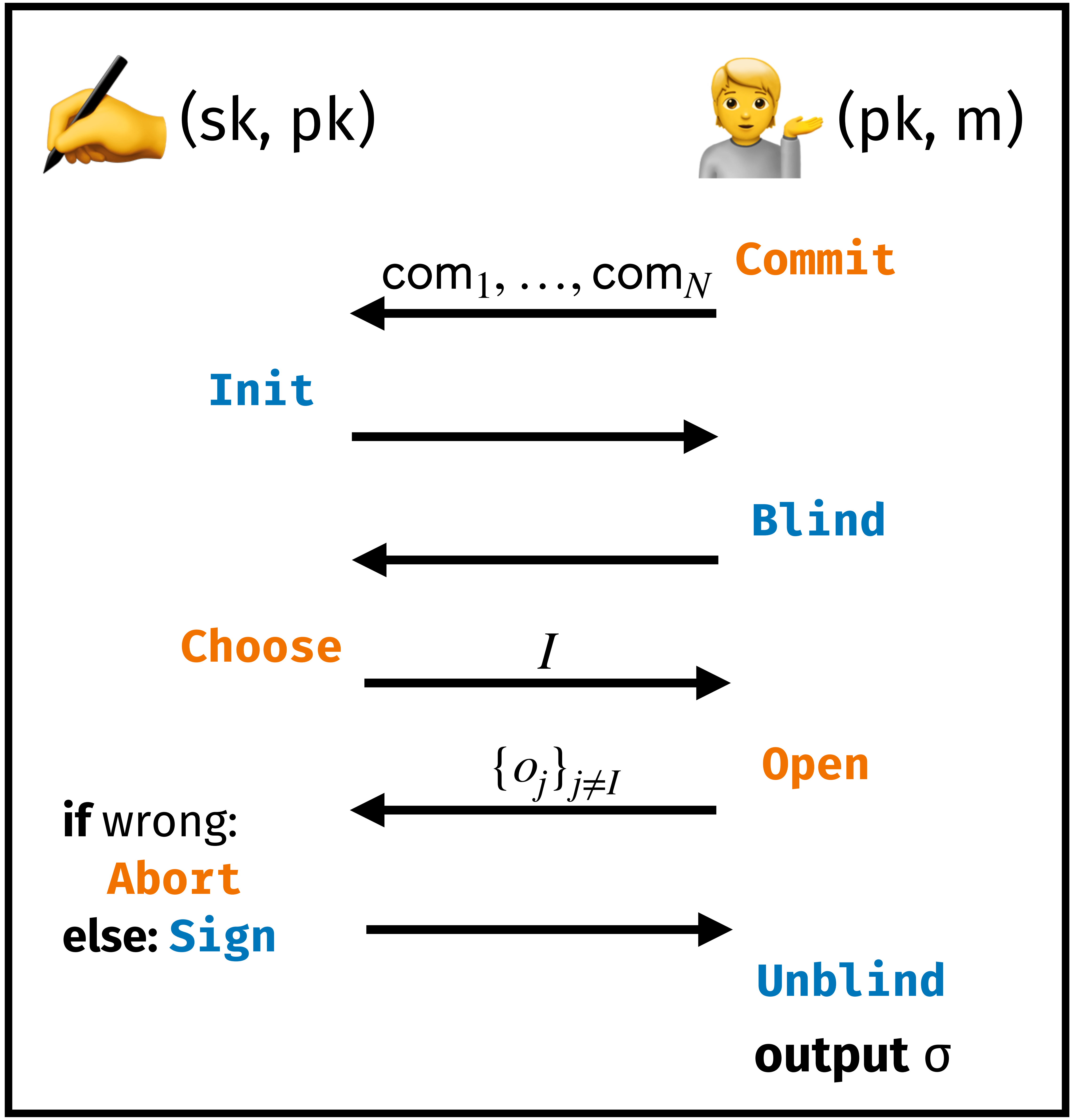
Recall: Underlying scheme is only secure when #cheats = $O(\log(\kappa))$

Previously

$\Pr[\text{cheat}] = 1/2$ each session

Without teardown, after s sessions $\mathbb{E}[\#\text{cheats}] = s/2$

For $s = \text{poly}(\kappa)$, this is not secure. Hence, teardowns



Our Transform

Analysis - OMUF

Recall: Underlying scheme is only secure when #cheats = $O(\log(\kappa))$

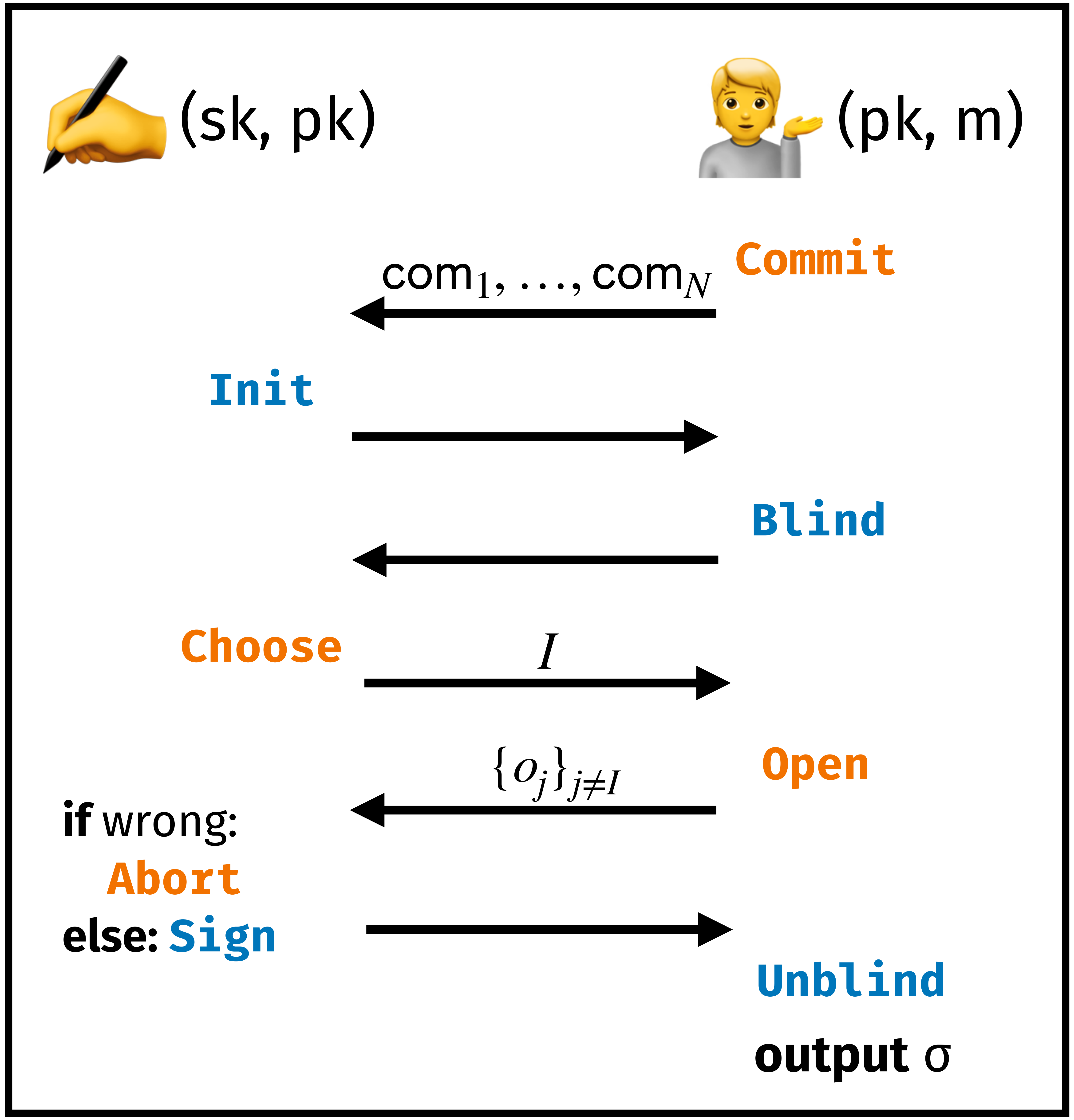
Previously

$\Pr[\text{cheat}] = 1/2$ each session

Without teardown, after s sessions $\mathbb{E}[\#\text{cheats}] = s/2$

For $s = \text{poly}(\kappa)$, this is not secure. Hence, teardowns

Now



Our Transform

Analysis - OMUF

Recall: Underlying scheme is only secure when #cheats = $O(\log(\kappa))$

Previously

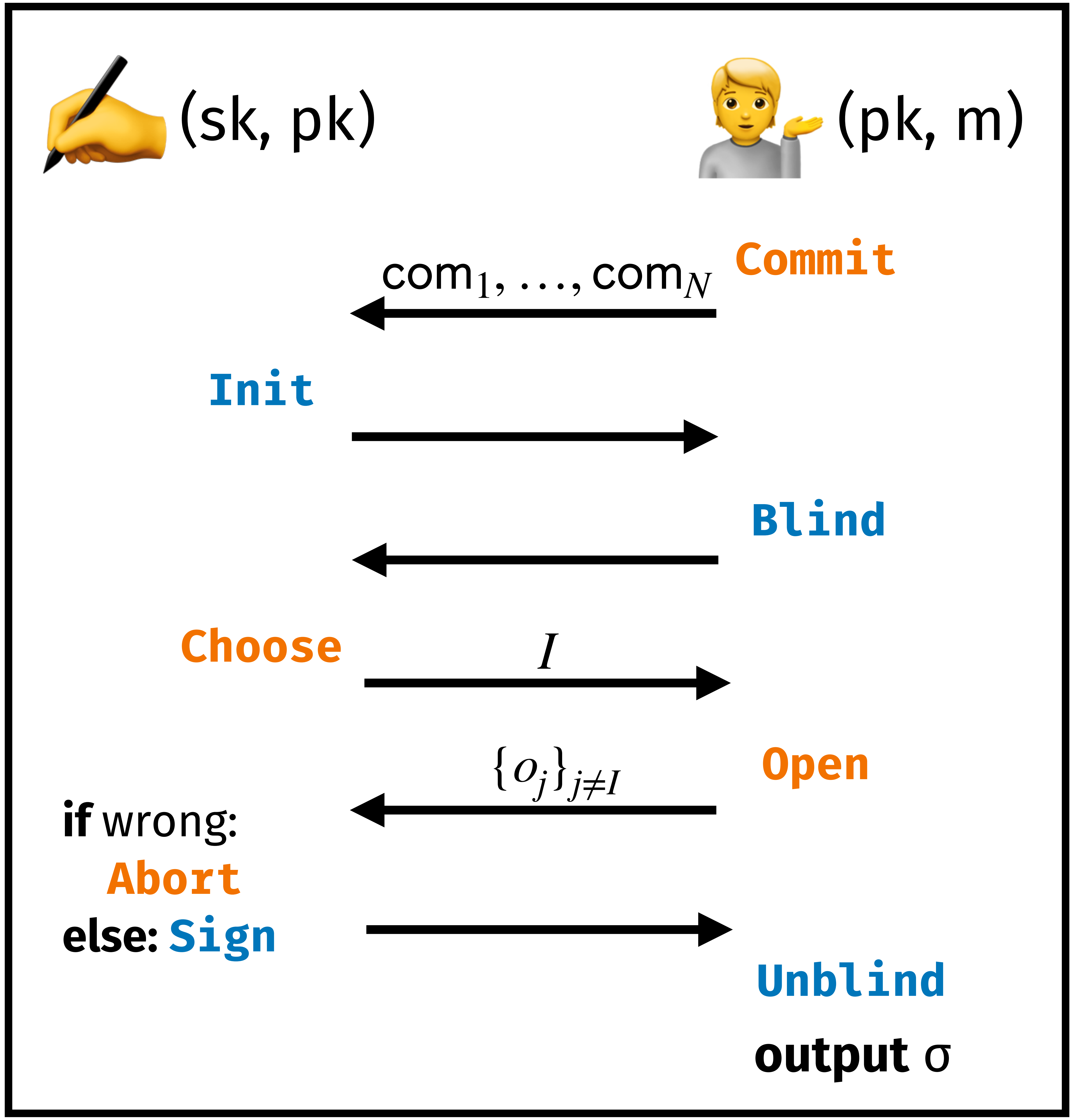
$\Pr[\text{cheat}] = 1/2$ each session

Without teardown, after s sessions $\mathbb{E}[\#\text{cheats}] = s/2$

For $s = \text{poly}(\kappa)$, this is not secure. Hence, teardowns

Now

$\Pr[\text{cheat}] = 1/N$ in session N



Our Transform

Analysis - OMUF

Recall: Underlying scheme is only secure when #cheats = $O(\log(\kappa))$

Previously

$\Pr[\text{cheat}] = 1/2$ each session

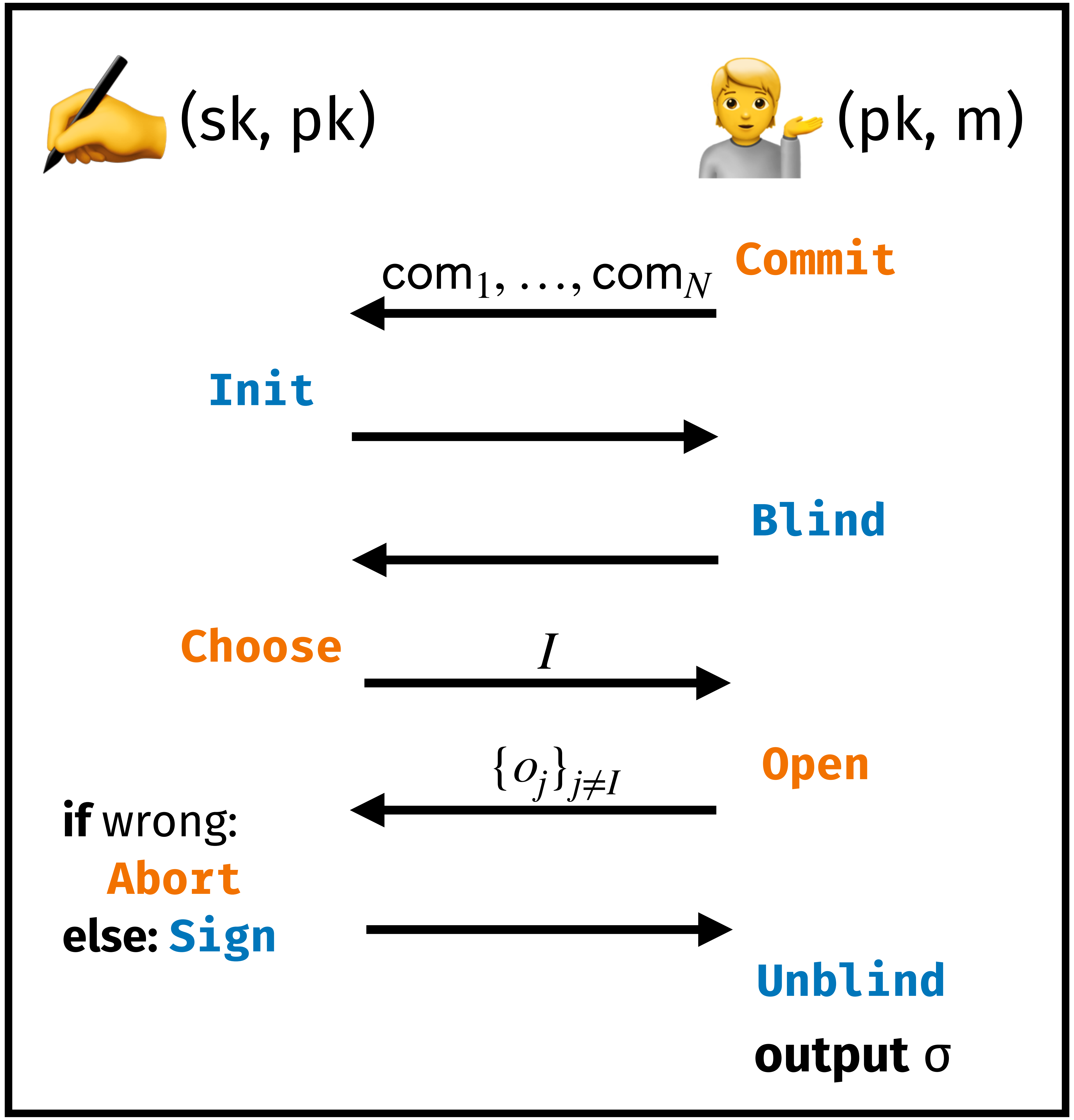
Without teardown, after s sessions $\mathbb{E}[\#\text{cheats}] = s/2$

For $s = \text{poly}(\kappa)$, this is not secure. Hence, teardowns

Now

$\Pr[\text{cheat}] = 1/N$ in session N

Without teardown, after s sessions,



Our Transform

Analysis - OMUF

Recall: Underlying scheme is only secure when #cheats = $O(\log(\kappa))$

Previously

$\Pr[\text{cheat}] = 1/2$ each session

Without teardown, after s sessions $\mathbb{E}[\#\text{cheats}] = s/2$

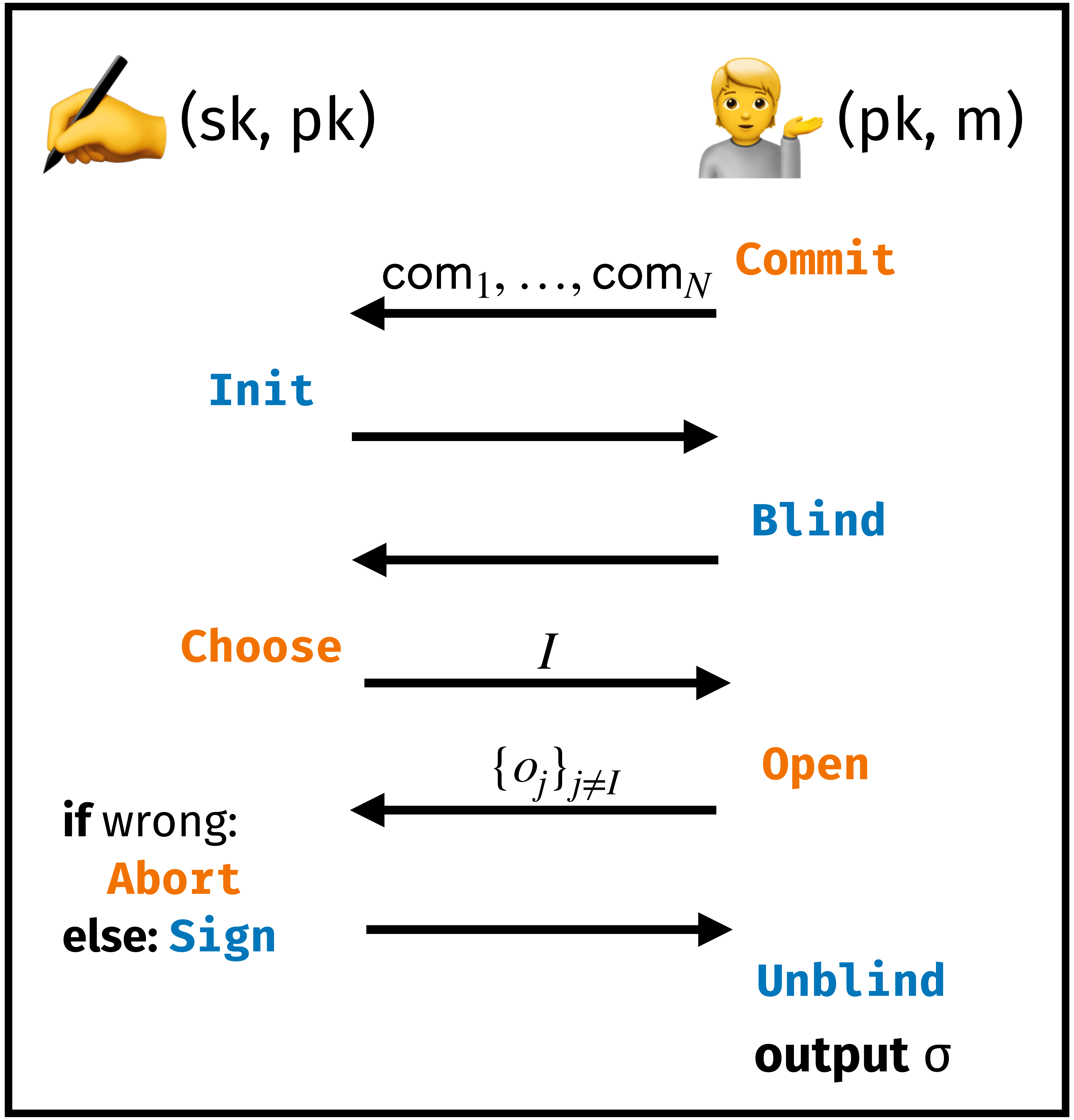
For $s = \text{poly}(\kappa)$, this is not secure. Hence, teardowns

Now

$\Pr[\text{cheat}] = 1/N$ in session N

Without teardown, after s sessions,

$$\mathbb{E}[\#\text{cheats}] = \sum_{N=2}^{s+1} \frac{1}{N} \leq \ln(s+1)$$



Our Transform

Analysis - OMUF

Recall: Underlying scheme is only secure when #cheats = $O(\log(\kappa))$

Previously

$\Pr[\text{cheat}] = 1/2$ each session

Without teardown, after s sessions $\mathbb{E}[\#\text{cheats}] = s/2$

For $s = \text{poly}(\kappa)$, this is not secure. Hence, teardowns

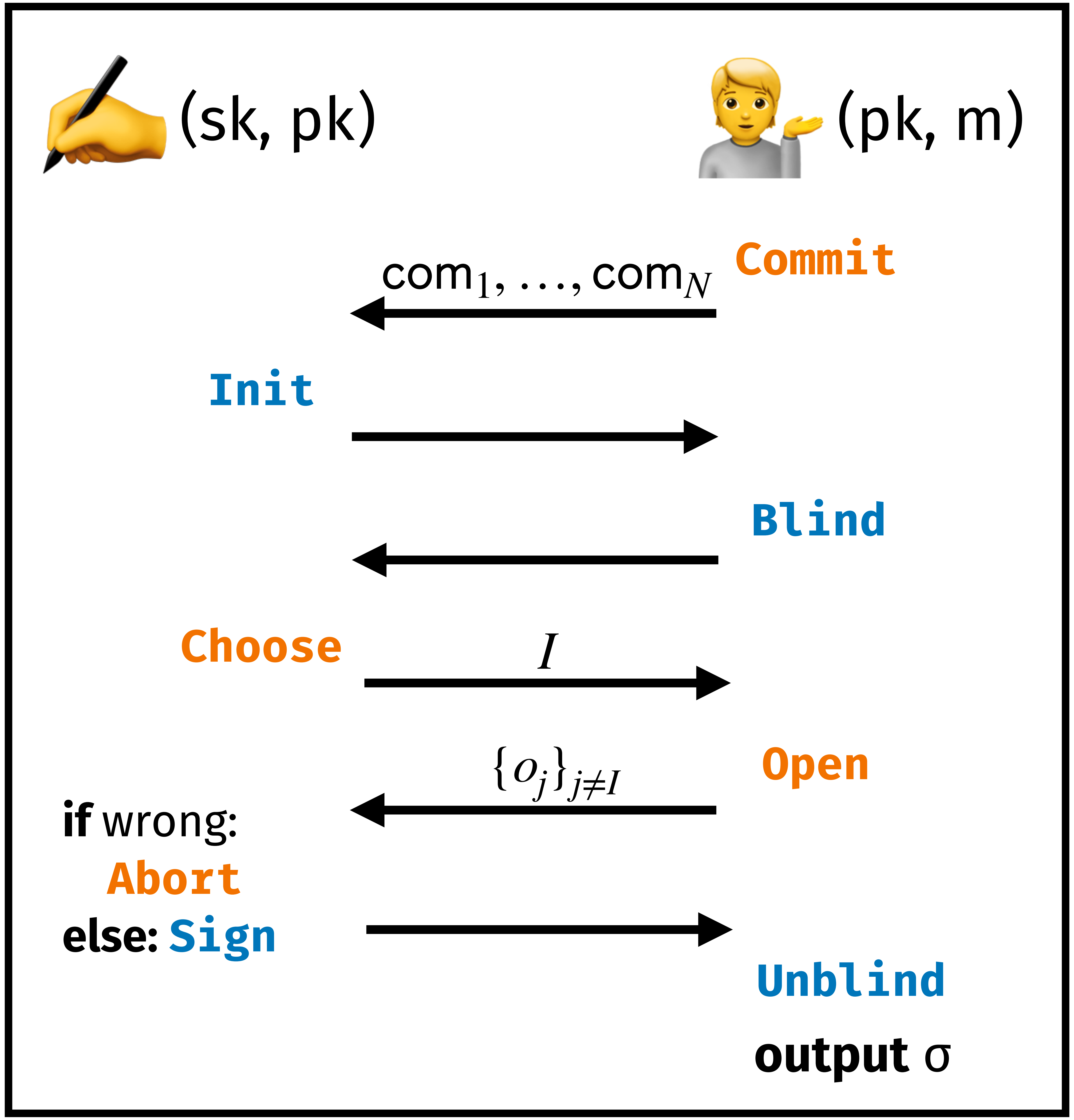
Now

$\Pr[\text{cheat}] = 1/N$ in session N

Without teardown, after s sessions,

$$\mathbb{E}[\#\text{cheats}] = \sum_{N=2}^{s+1} \frac{1}{N} \leq \ln(s+1)$$

Tail bound: within $O(\log(\kappa))$ w.h.p



Our Transform

Analysis - OMUF

Recall: Underlying scheme is only secure when #cheats = $O(\log(\kappa))$

Previously

$\Pr[\text{cheat}] = 1/2$ each session

Without teardown, after s sessions $\mathbb{E}[\#\text{cheats}] = s/2$

For $s = \text{poly}(\kappa)$, this is not secure. Hence, teardowns

Now

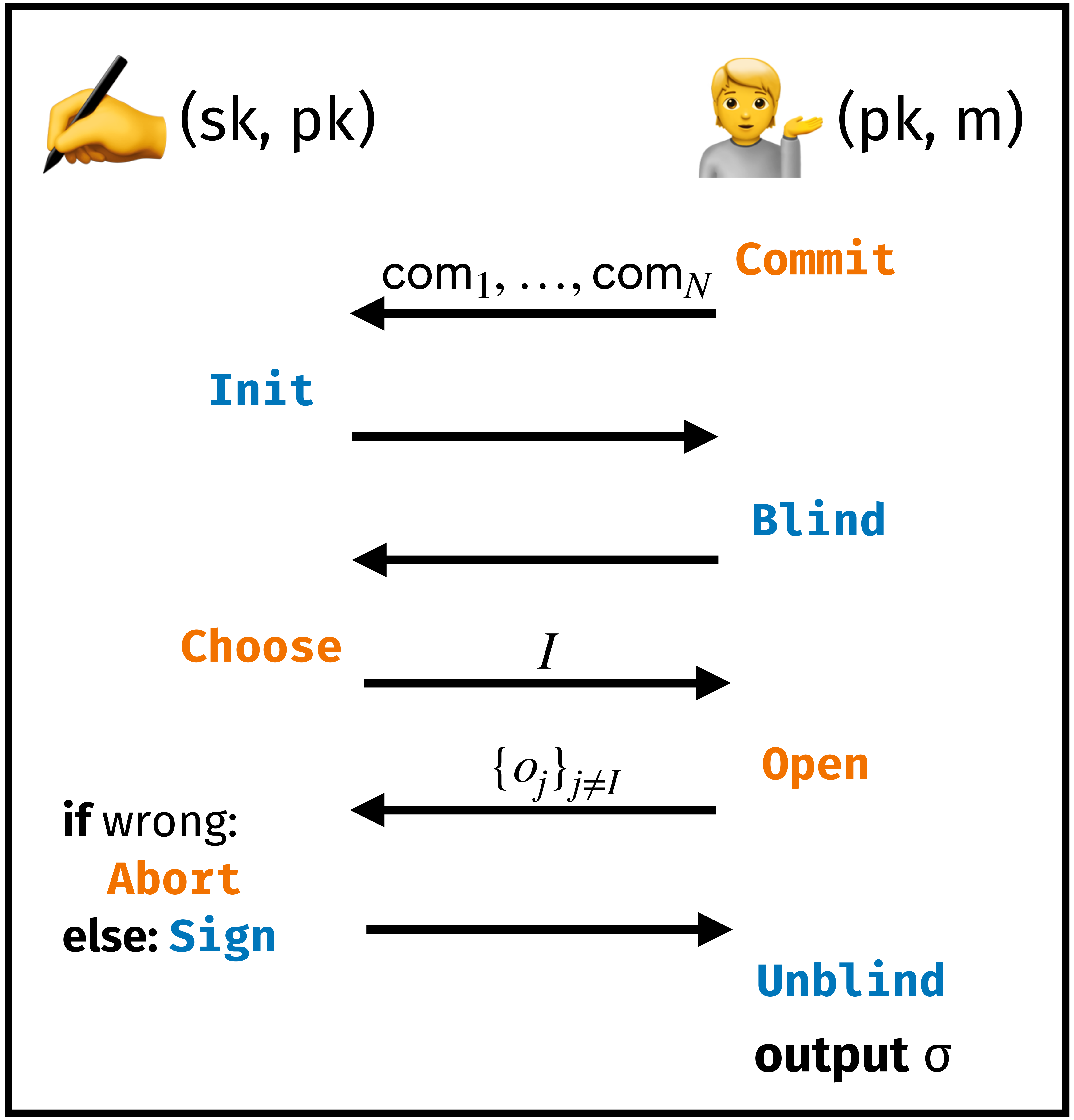
$\Pr[\text{cheat}] = 1/N$ in session N

Without teardown, after s sessions,

$$\mathbb{E}[\#\text{cheats}] = \sum_{N=2}^{s+1} \frac{1}{N} \leq \ln(s+1)$$

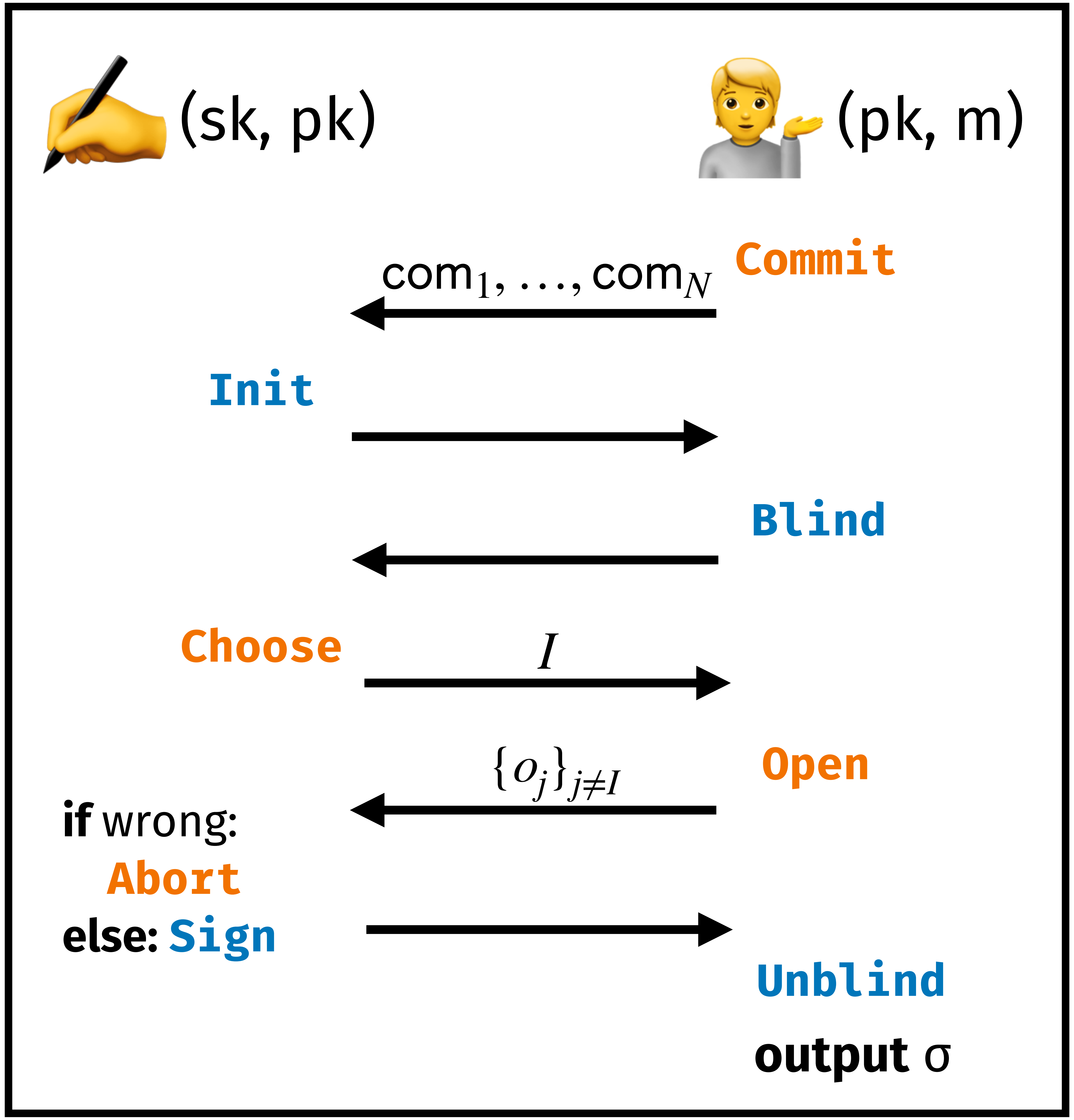
Tail bound: within $O(\log(\kappa))$ w.h.p

Thus, our scheme is $\text{poly}(\kappa)$**-OMUF without teardowns**



Our Transform

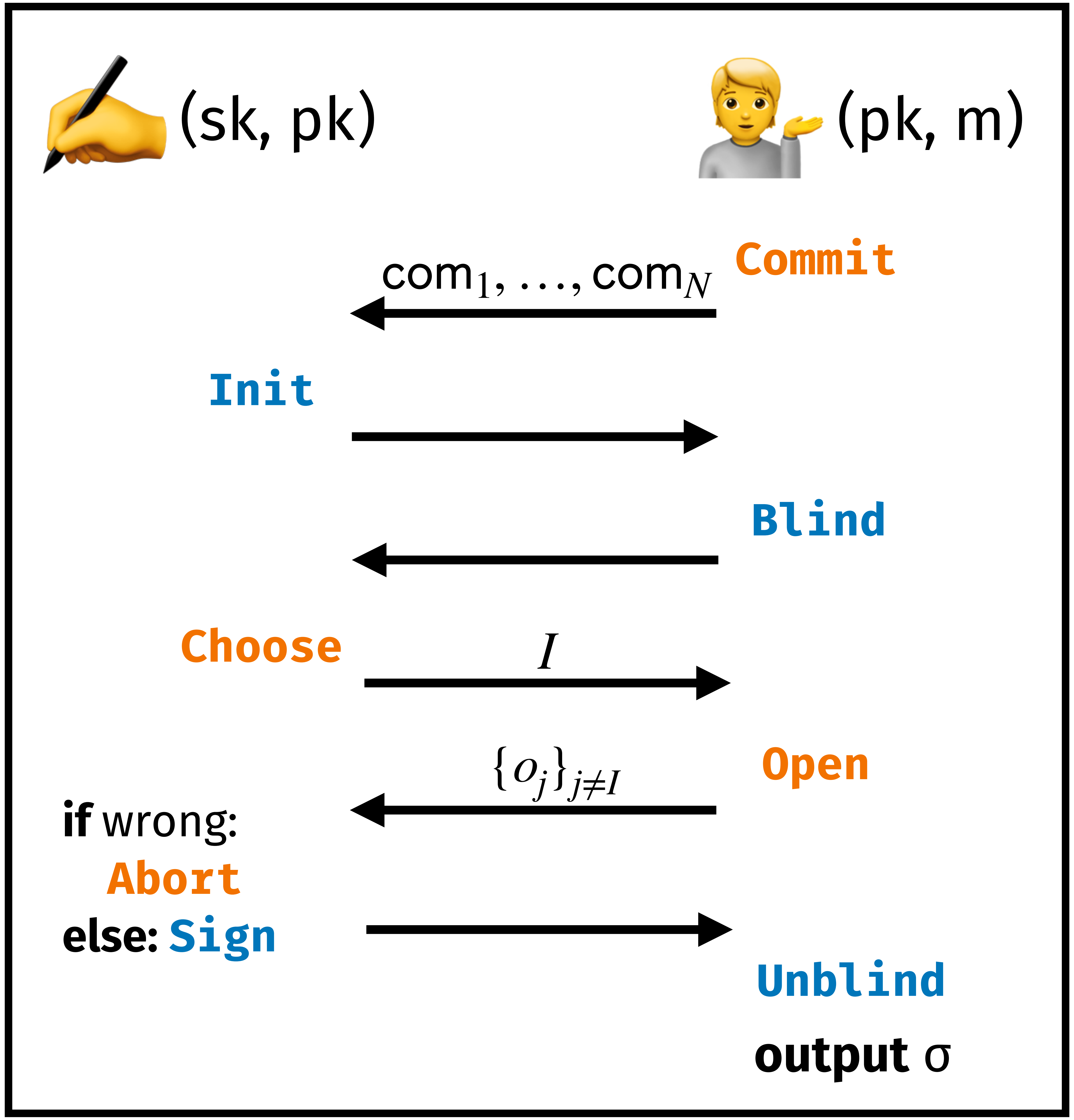
Analysis - Concurrency



Our Transform

Analysis - Concurrency

Previously

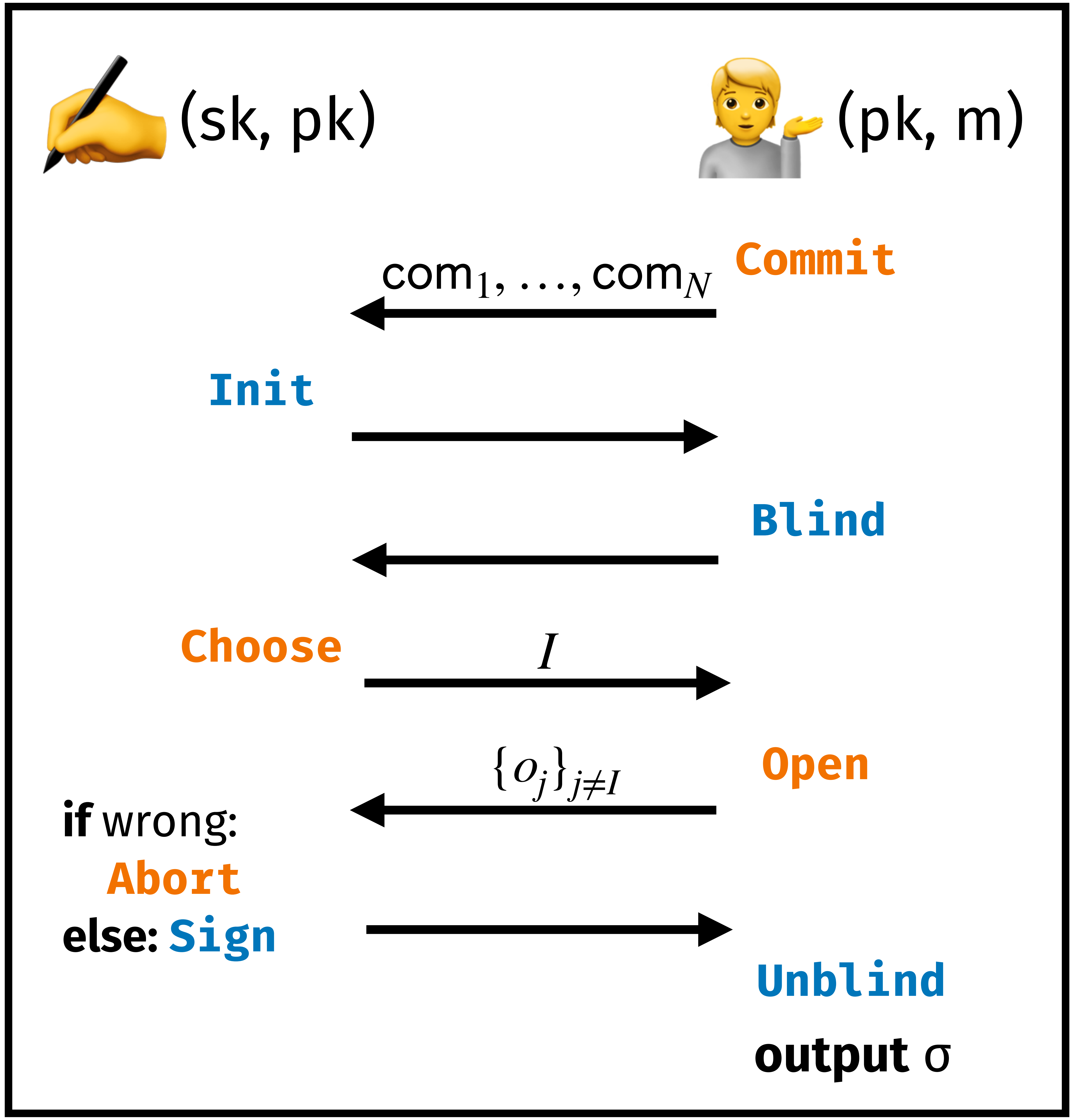


Our Transform

Analysis - Concurrency

Previously

Supported **parallel sessions:** a session cannot advance faster than its predecessors



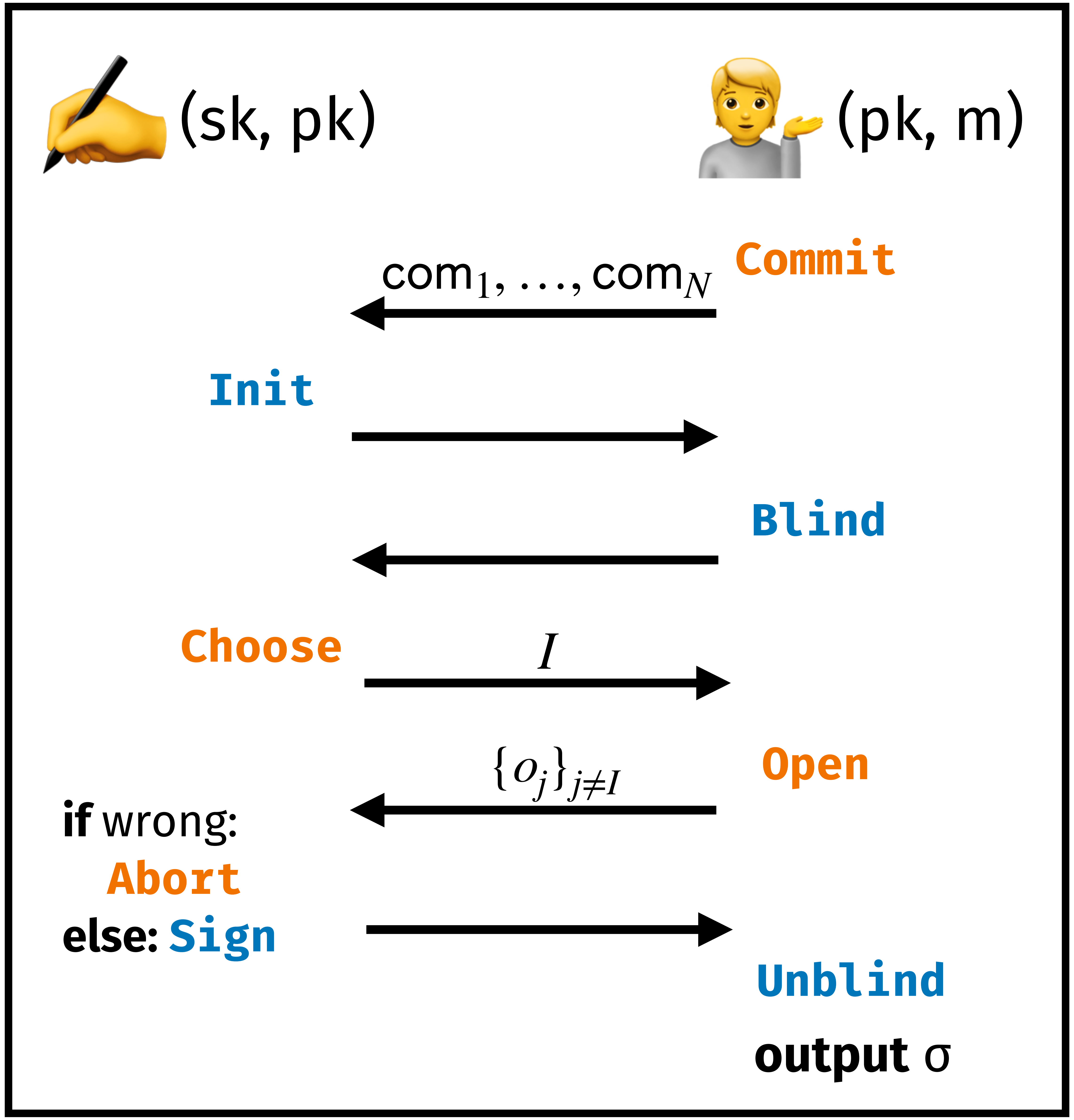
Our Transform

Analysis - Concurrency

Previously

Supported **parallel sessions:** a session cannot advance faster than its predecessors

Now



Our Transform

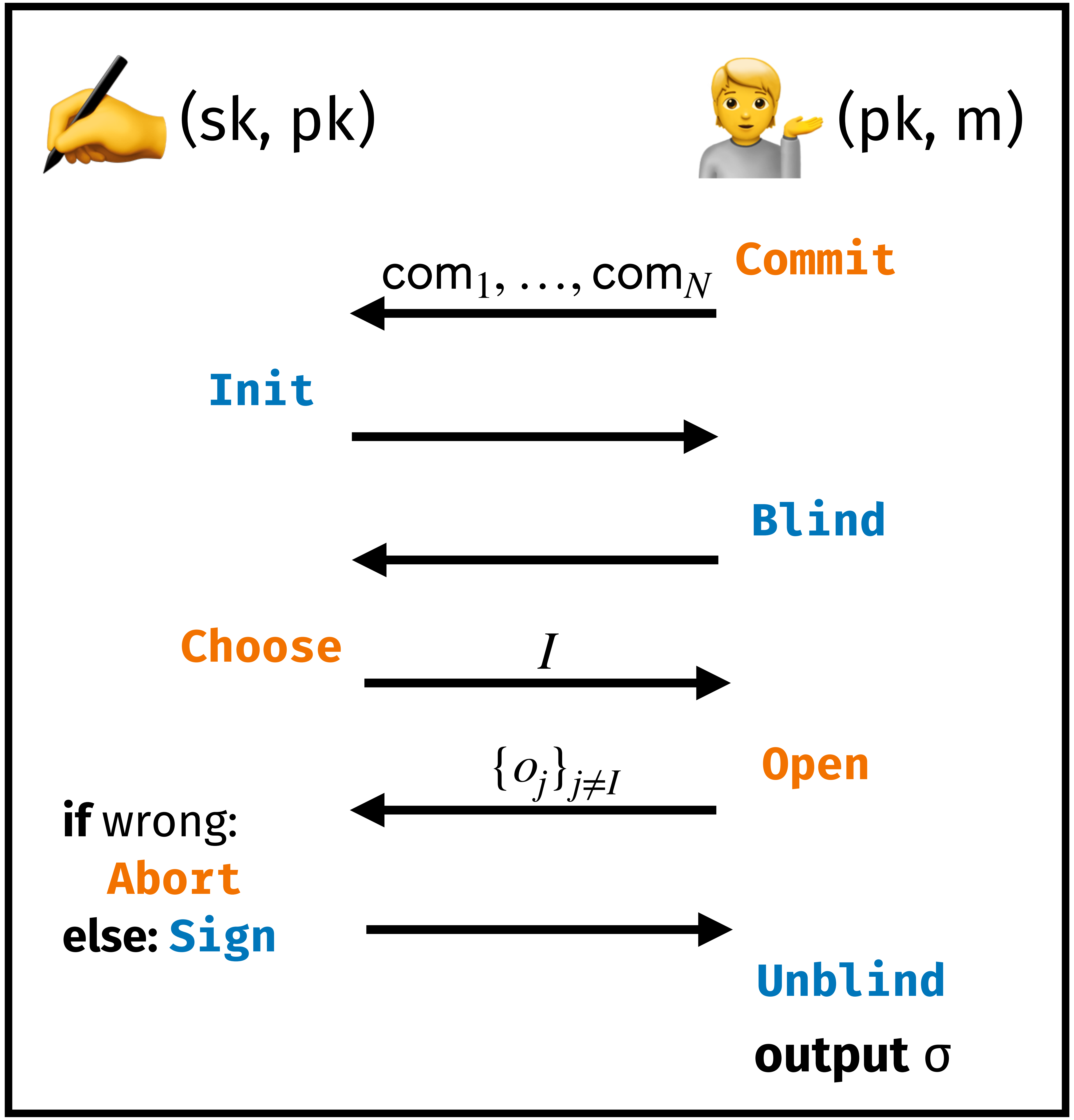
Analysis - Concurrency

Previously

Supported **parallel sessions:** a session cannot advance faster than its predecessors

Now

If N is incremented atomically every session, **we get arbitrary concurrency**



Our Transform

Analysis - Concurrency

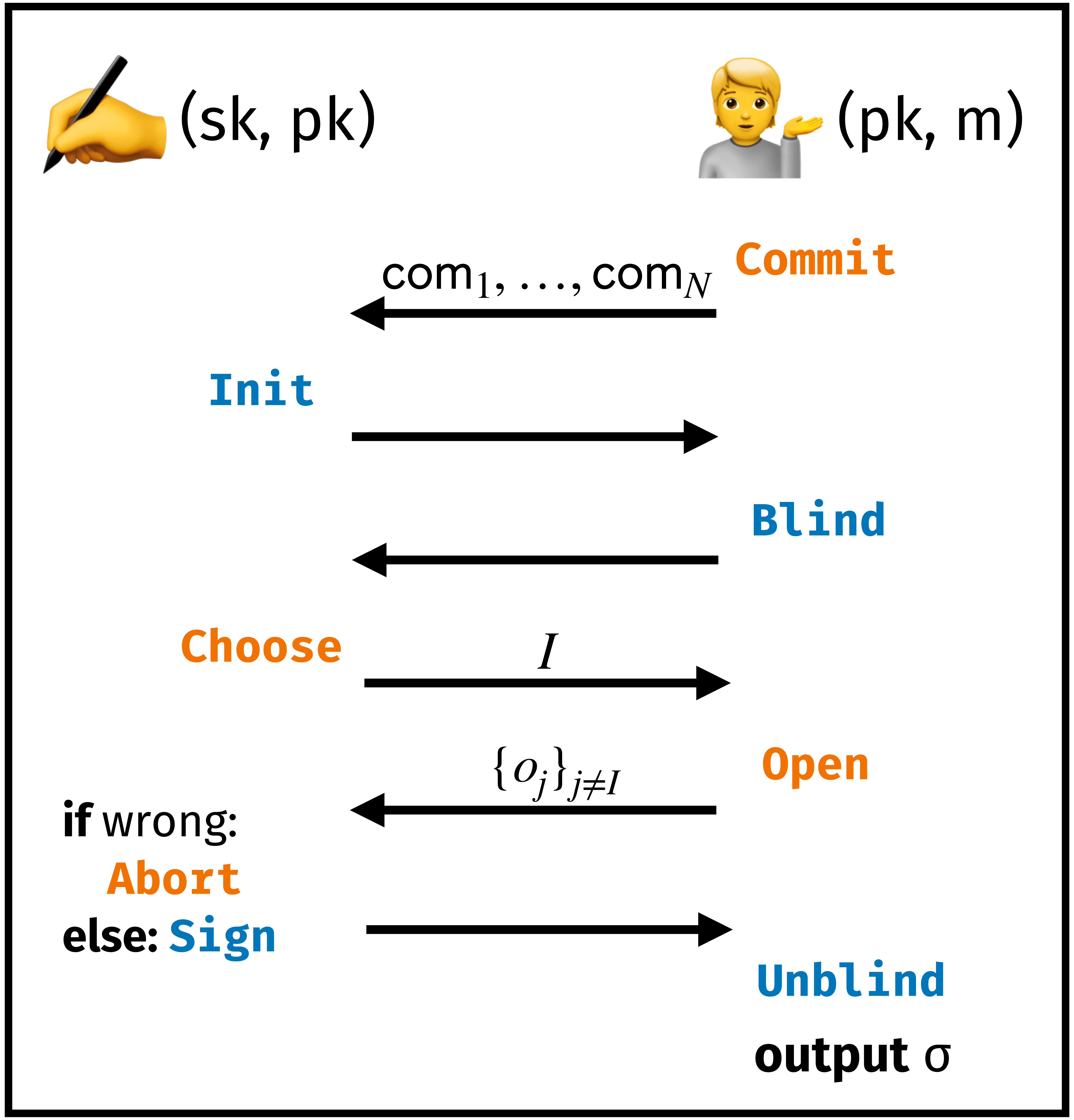
Previously

Supported **parallel sessions:** a session cannot advance faster than its predecessors

Now

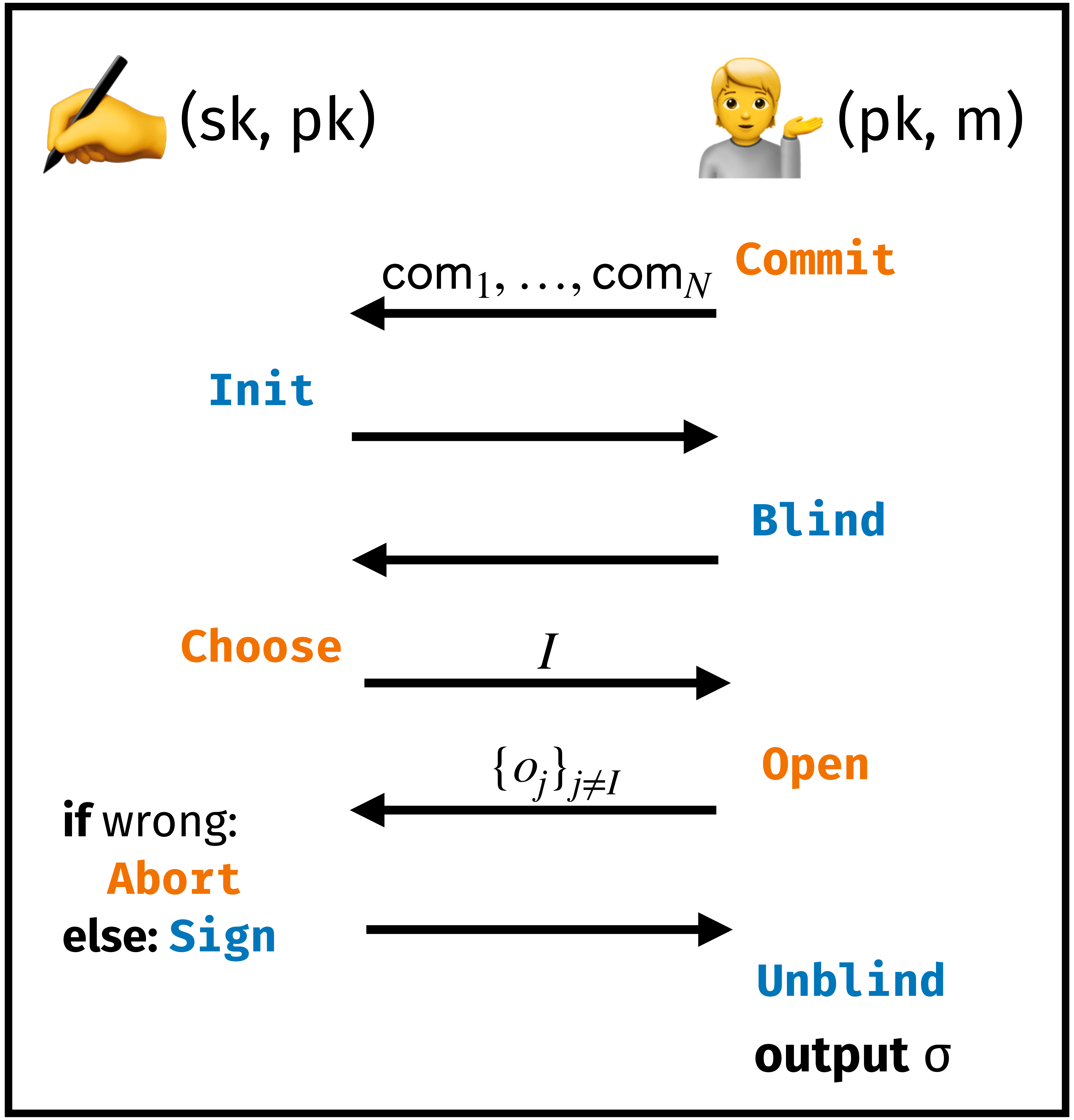
If N is incremented atomically every session, **we get arbitrary concurrency**

Can be improved: only increment N every time a user is caught cheating



Our Transform

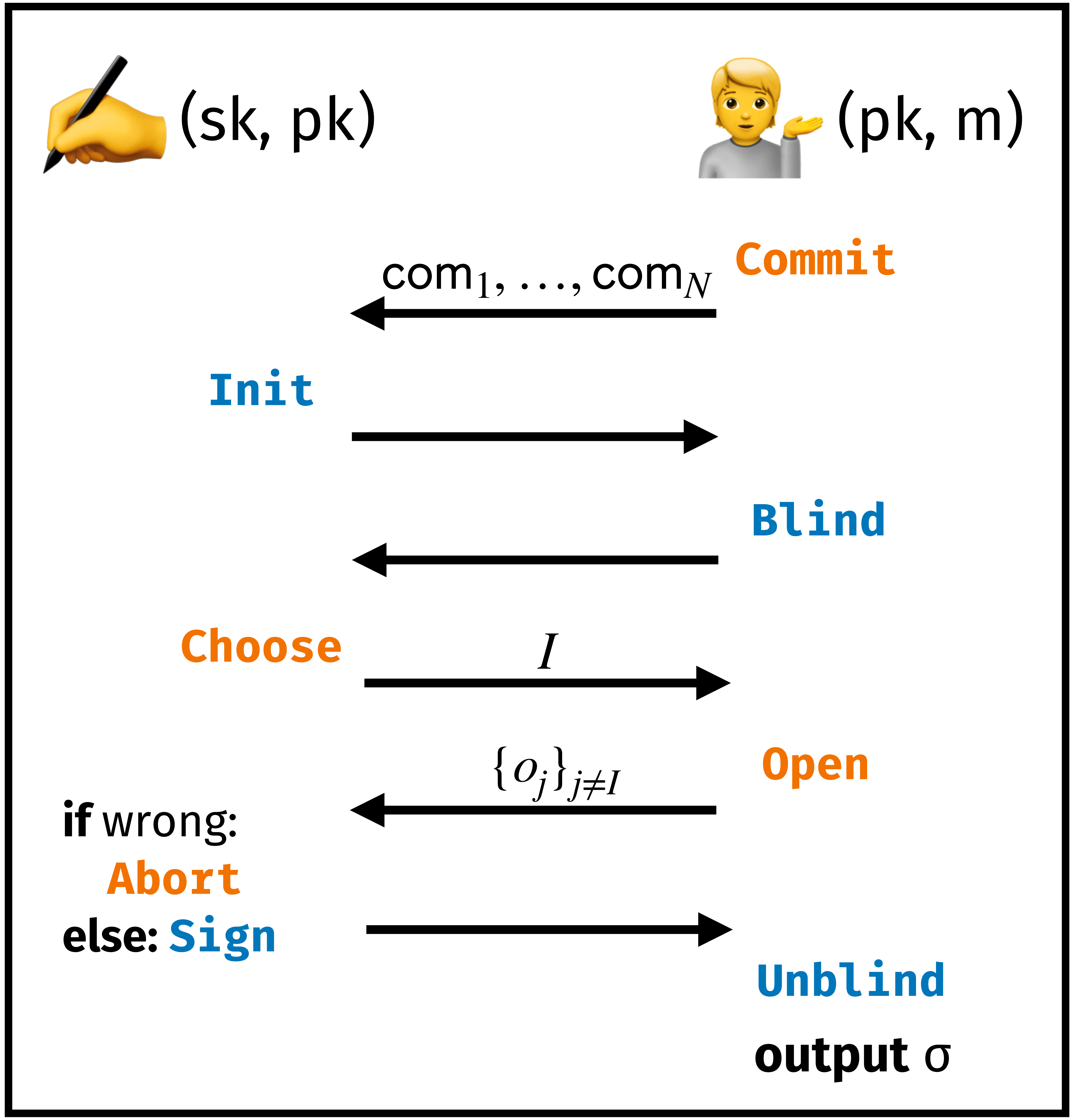
Analysis - Genericity



Our Transform

Analysis - Genericity

Previously

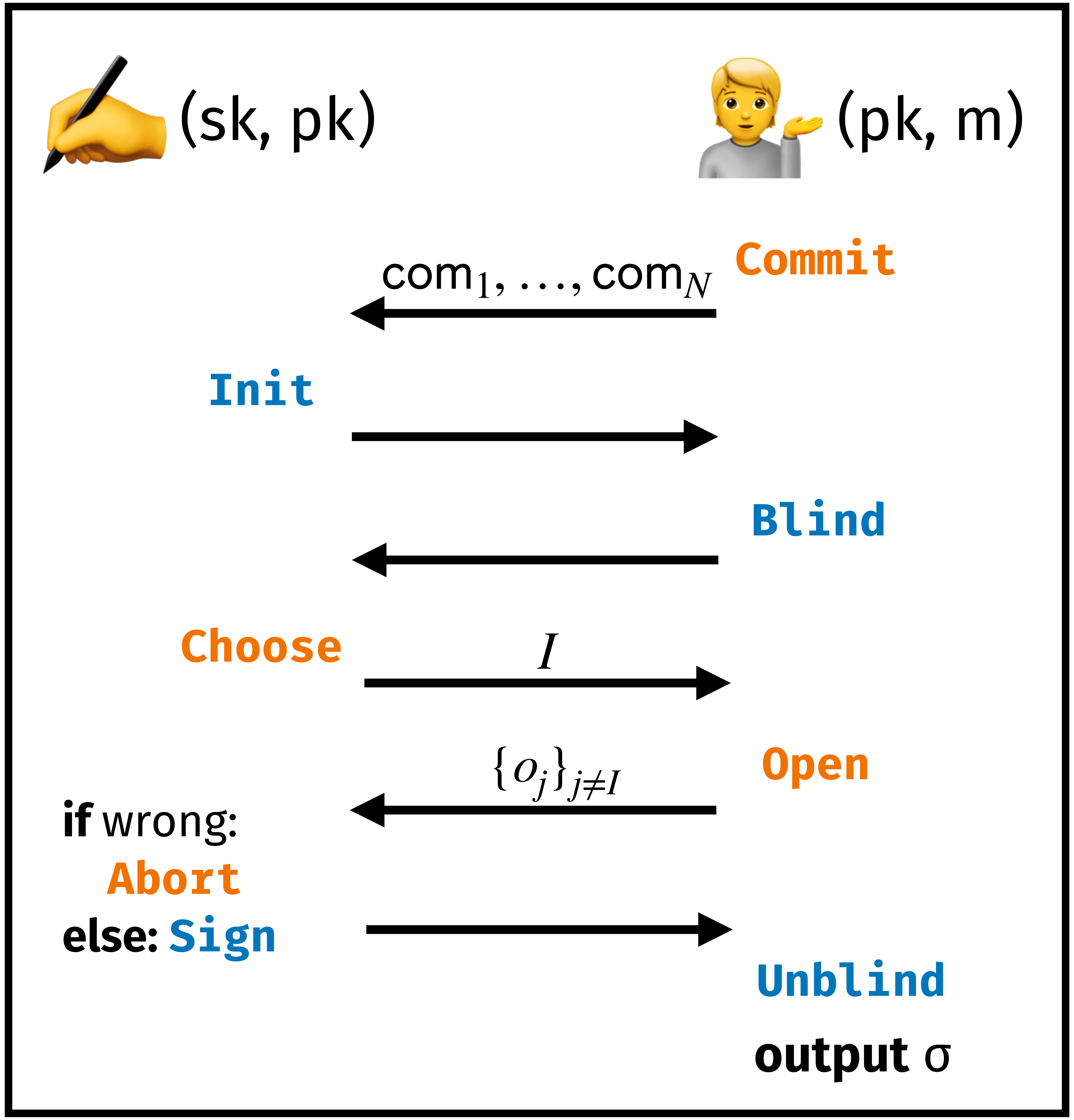


Our Transform

Analysis - Genericity

Previously

Not generic. Only defined for Okamoto-Schnorr



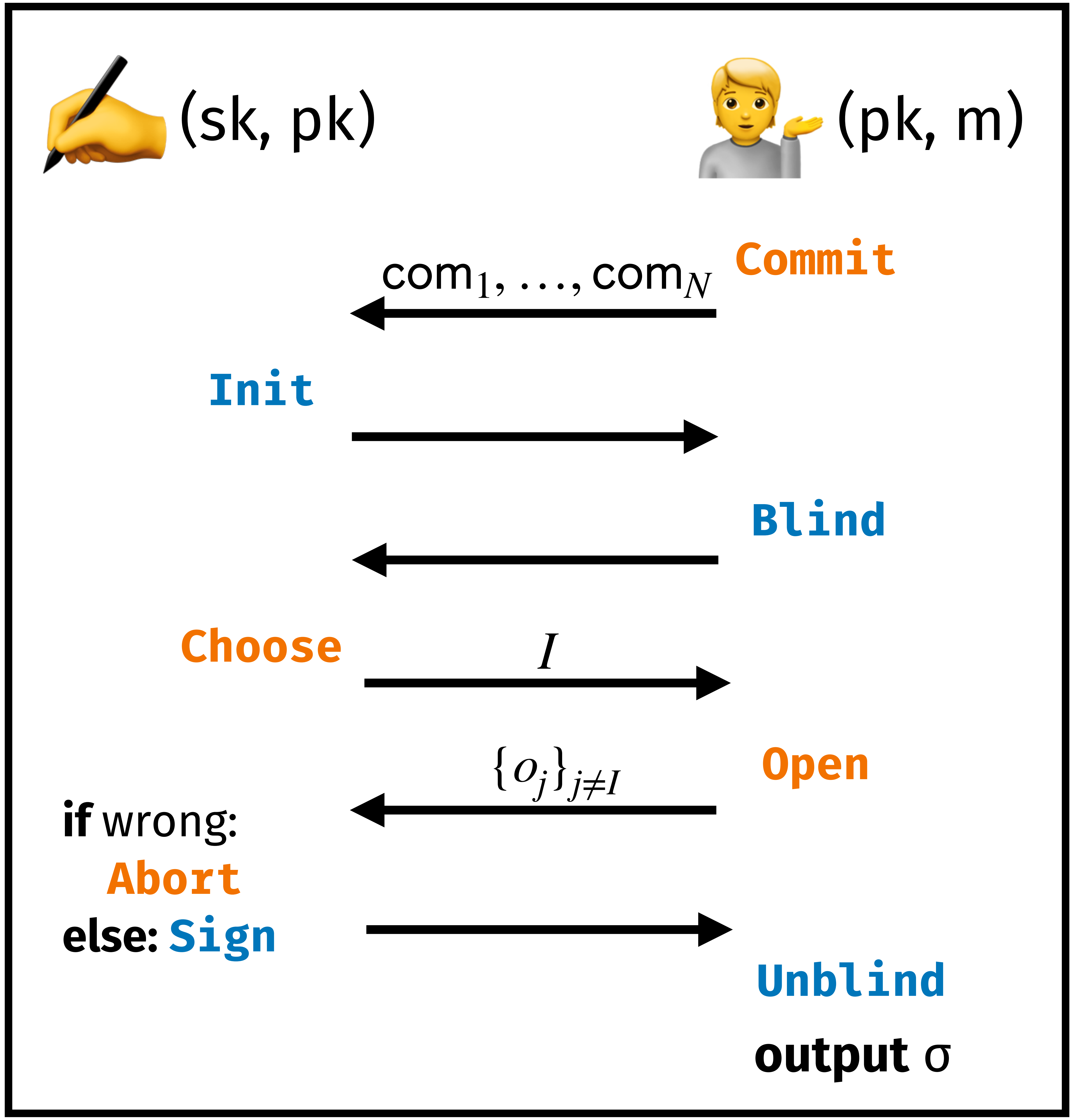
Our Transform

Analysis - Genericity

Previously

Not generic. Only defined for Okamoto-Schnorr

Now



Our Transform

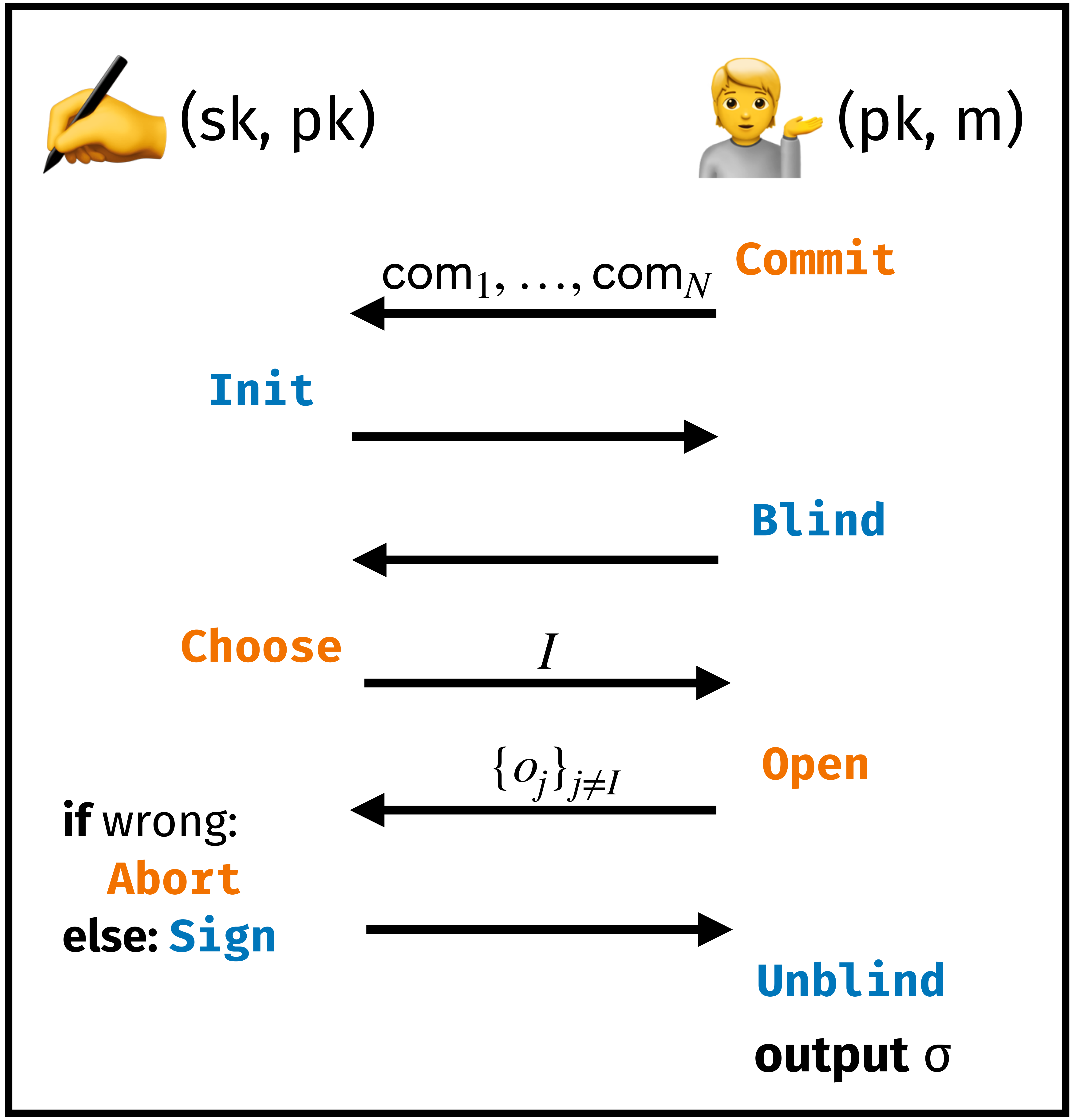
Analysis - Genericity

Previously

Not generic. Only defined for Okamoto-Schnorr

Now

Generic for any algebraic hash function with homomorphic properties



Our Transform

Analysis - Genericity

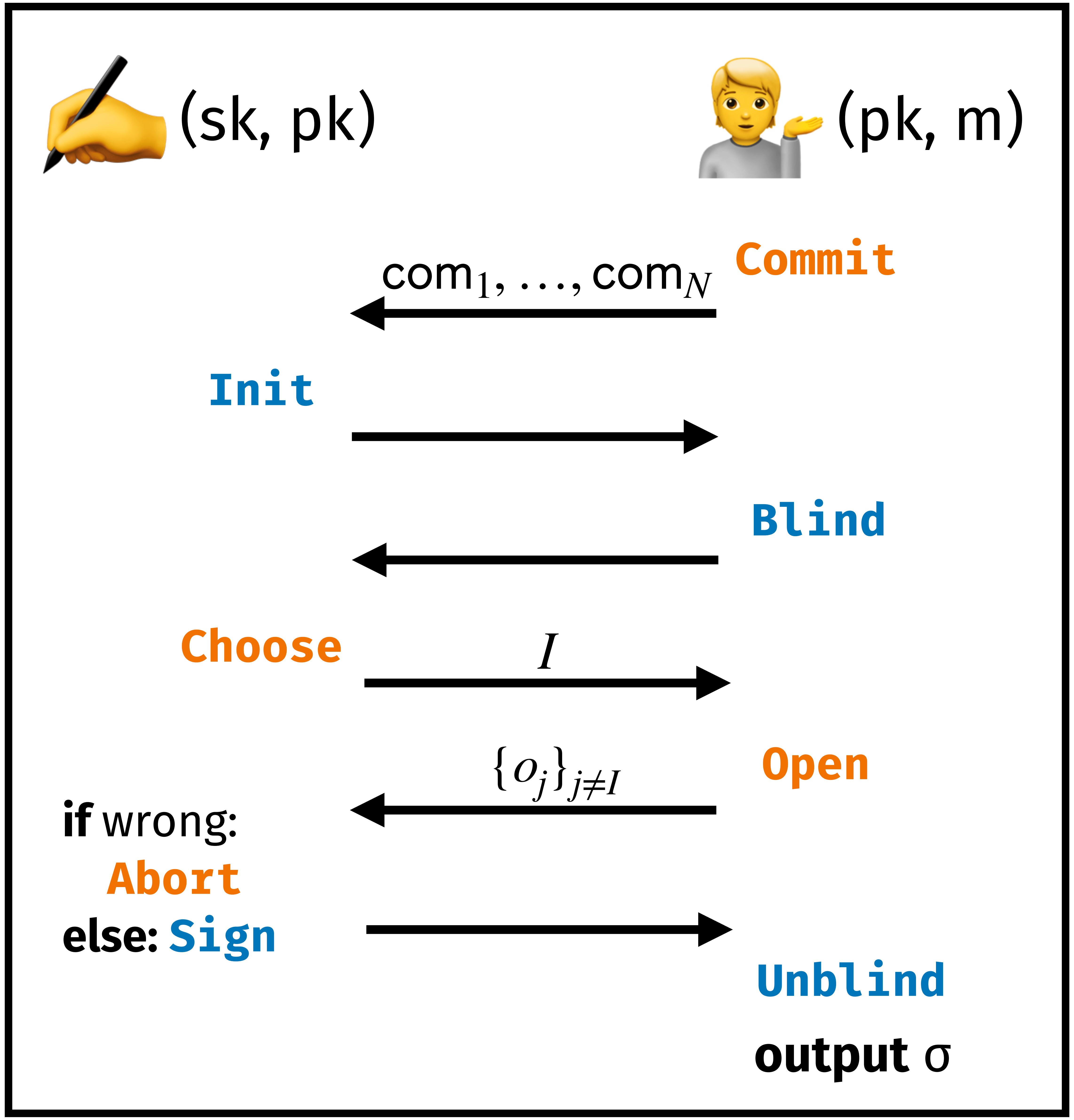
Previously

Not generic. Only defined for Okamoto-Schnorr

Now

Generic for any algebraic hash function with homomorphic properties

- Schnorr



Our Transform

Analysis - Genericity

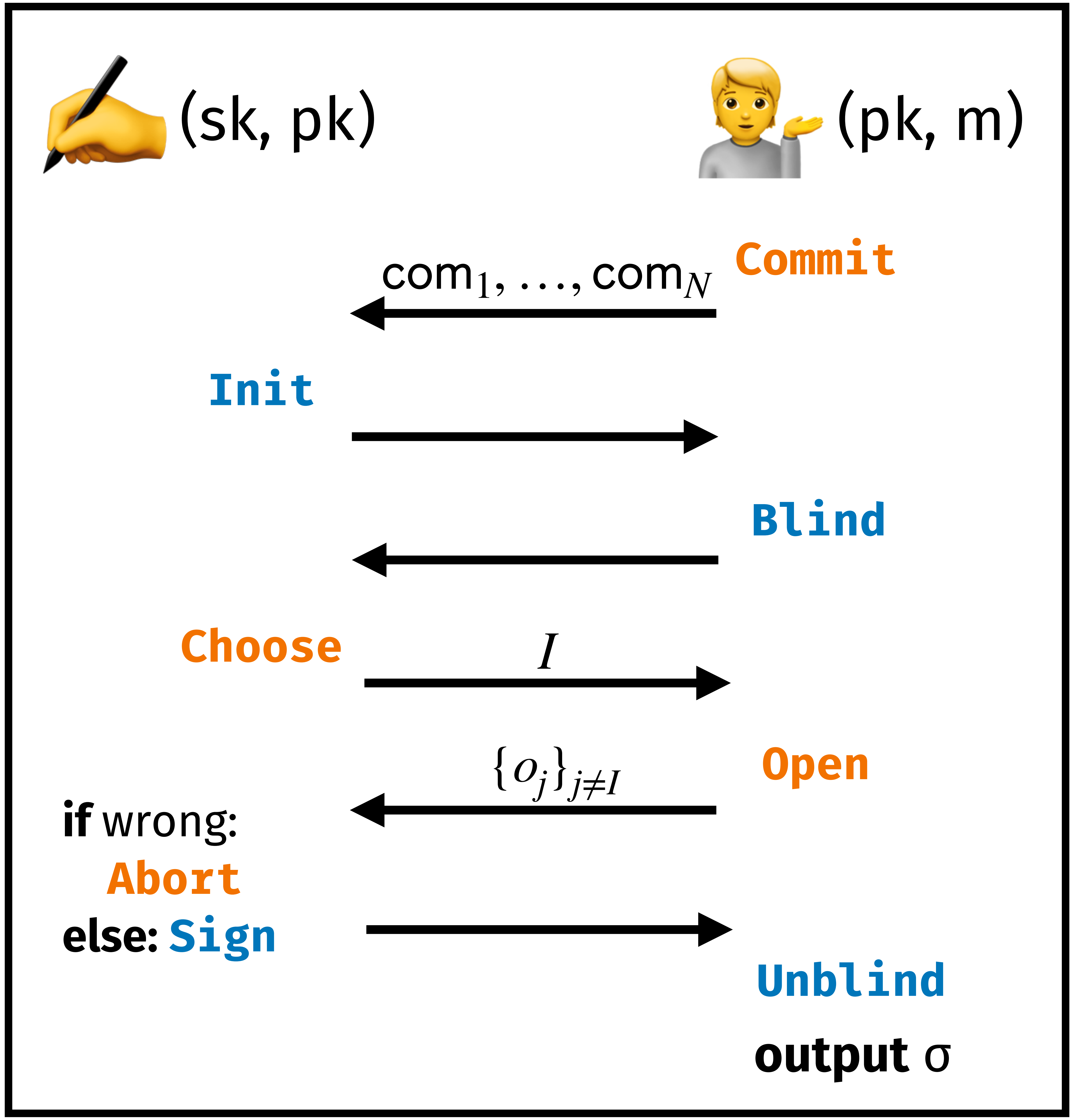
Previously

Not generic. Only defined for Okamoto-Schnorr

Now

Generic for any algebraic hash function with homomorphic properties

- Schnorr
- Okamoto-Schnorr



Our Transform

Analysis - Genericity

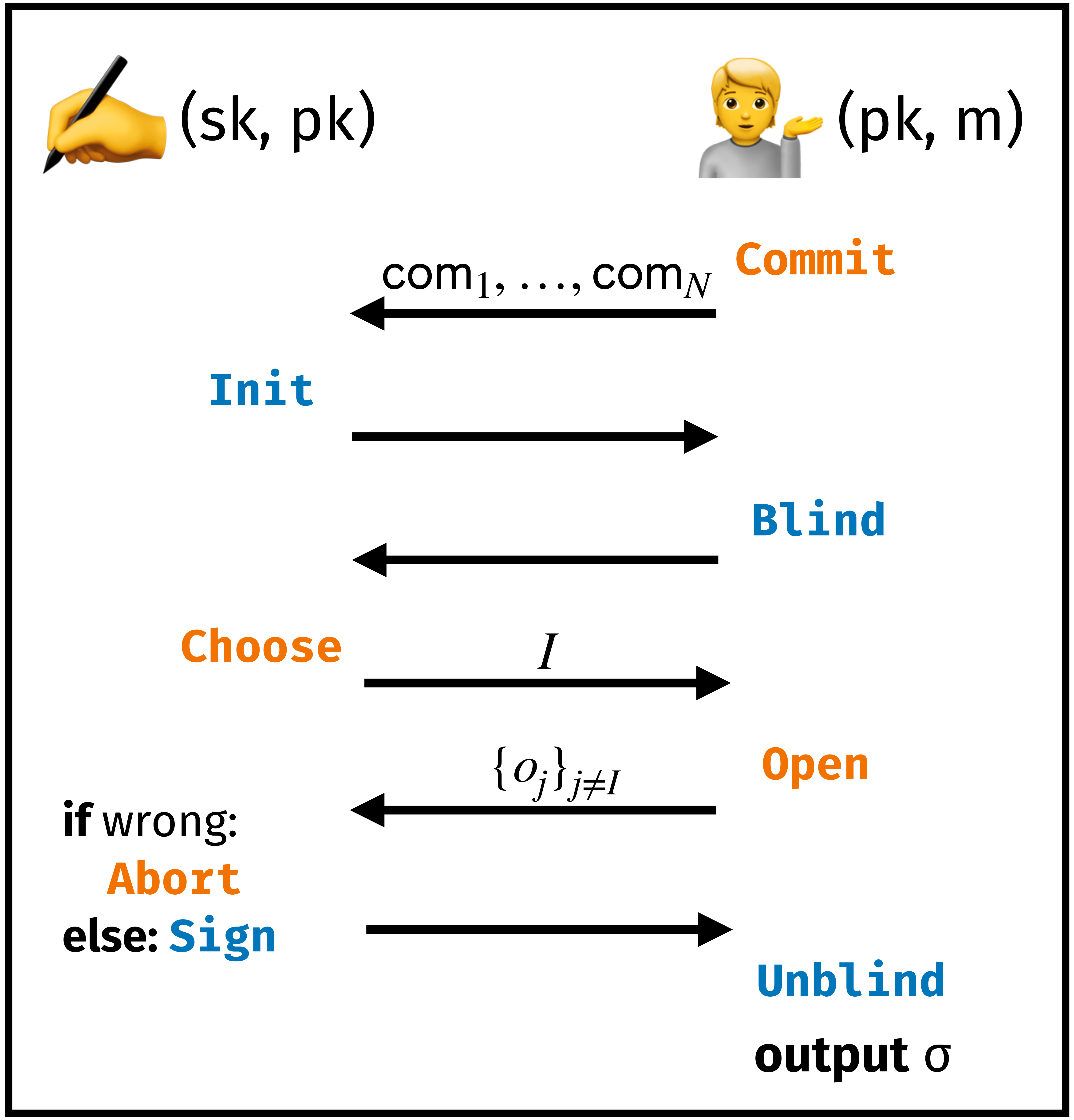
Previously

Not generic. Only defined for Okamoto-Schnorr

Now

Generic for any algebraic hash function with homomorphic properties

- Schnorr
- Okamoto-Schnorr
- Fiat-Shamir



Our Transform

Analysis - Genericity

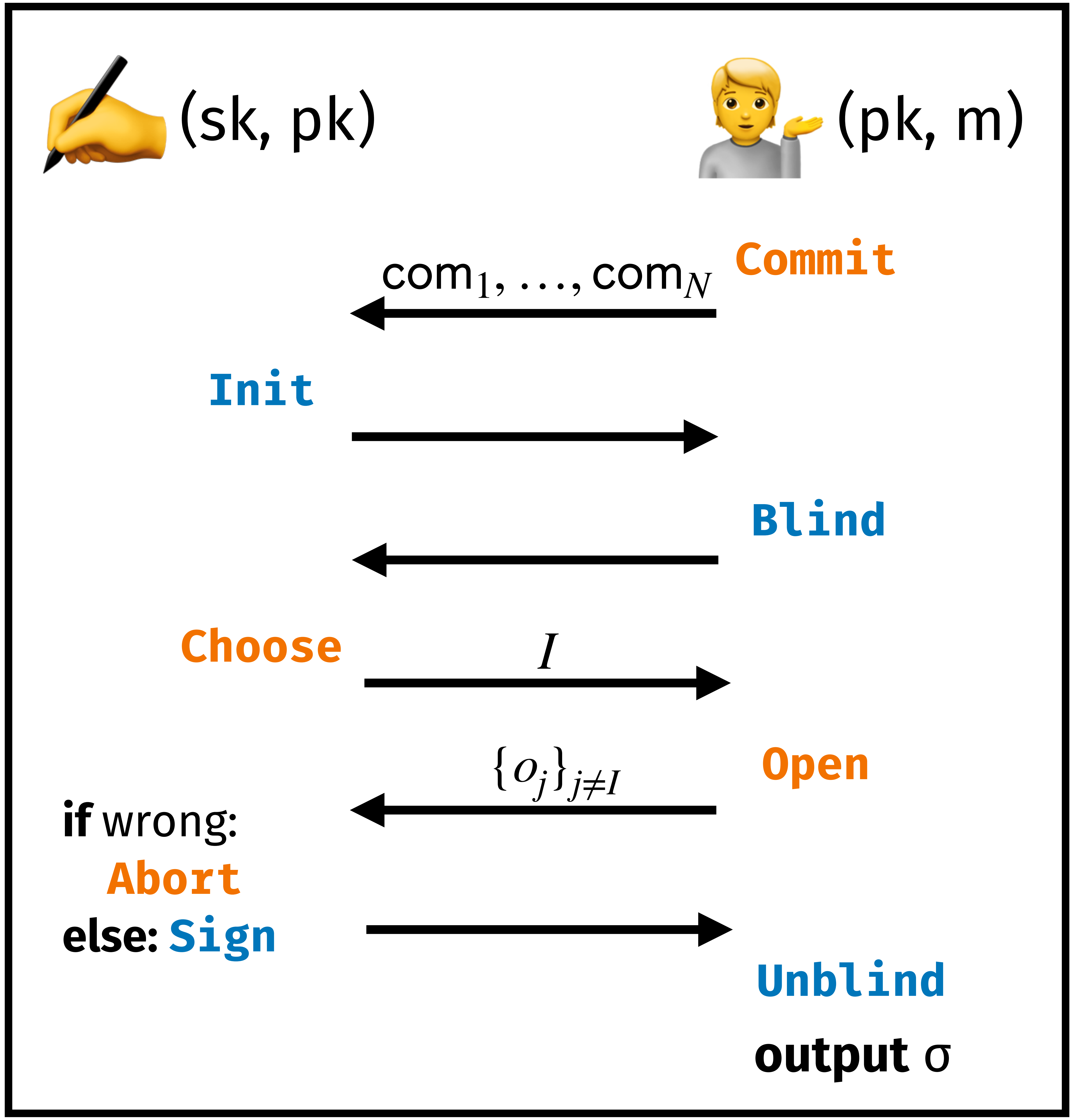
Previously

Not generic. Only defined for Okamoto-Schnorr

Now

Generic for any algebraic hash function with homomorphic properties

- Schnorr
- Okamoto-Schnorr
- Fiat-Shamir
- Okamoto-Guillou-Quisquater



References

Thank you!

[BL12] F. Baldimtsi, A. Lysyanskaya, *Anonymous Credentials Light*, https://eprint.iacr.org/2012/298

[BLLOR] F. Benhamouda, T. Lepoint, J. Loss, M. Orrù, M. Raykova, *On the (in)security of ROS*, https://eprint.iacr.org/2020/945

[BNPS01] M. Bellare, C. Namprempre, D. Pointcheval, M. Semanko, *The One-More-RSA-Inversion Problems and the Security of Chaum's Blind Signature Scheme*, https://eprint.iacr.org/2001/002

[Chaum83] D. Chaum, *Blind Signatures for Untraceable Payments*

[FHS15] G. Fuchsbauer, C. Hanser, D. Slamanig, *Practical Round-Optimal Blind Signatures in the Standard Model* https://eprint.iacr.org/2015/626

[HBG19] E. Heilman, F. Baldimtsi, S. Goldberg, *Blindly Signed Contracts: Anonymous On-Blockchain and Off-Blockchain Bitcoin Transactions*, https://eprint.iacr.org/2016/056

[HKL19] E. Hauck, E. Kiltz, J. Loss., *A modular treatment of blind signatures from identification schemes*, https://eprint.iacr.org/2019/260

[JLO97] A. Juels, M. Luby, R. Ostrovsky, *Security of blind digital signatures*

[KLX20] J. Kastner, J. Loss, J. Xu, *On Pairing-Free Blind Signature Schemes in the Algebraic Group Model*, https://eprint.iacr.org/2020/1071

[Pointcheval98] D. Pointcheval, *Strengthened security for blind signatures*