Strengthening Zero-Knowledge Protocols using Signatures

Juan Garay	Bell Labs, Lucent Technologies
Philip MacKenzie	Bell Labs, Lucent Technologies
Ke Yang	Carnegie Mellon University

Zero Knowledge Proof Protocols

An extremely successful story in two decades...

Zero Knowledge Proof Protocols

An extremely successful story in two decades...

[Goldwasser Micali Rackoff 85]
introduces the notion

Zero Knowledge Proof Protocols

An extremely successful story in two decades...

[Goldwasser Micali Rackoff 85]
introduces the notion

[Goldreich Micali Wigderson 86]
all NP languages have ZK proofs

Zero Knowledge Proof Protocols

An extremely successful story in two decades...

[Goldwasser Micali Rackoff 85]
introduces the notion

[Goldreich Micali Wigderson 86]
all NP languages have ZK proofs

[hundreds of papers here...]
many many applications in cryptography

- identification protocols
- two-party/multi-party computation
- ...

A Quick Review of ZK Proofs

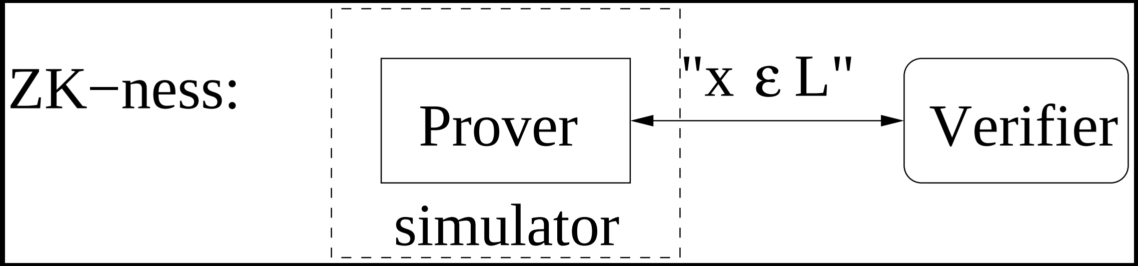
A protocol between Prover and Verifier.

- Completeness — if $x \in L$ then Verifier always accepts
- Soundness — if $x \notin L$ then Verifier accepts with negl. prob.

A Quick Review of ZK Proofs

A protocol between **Prover** and **Verifier**.

- **Completeness** — if $x \in L$ then **Verifier** always accepts
- **Soundness** — if $x \notin L$ then **Verifier** accepts with negl. prob.

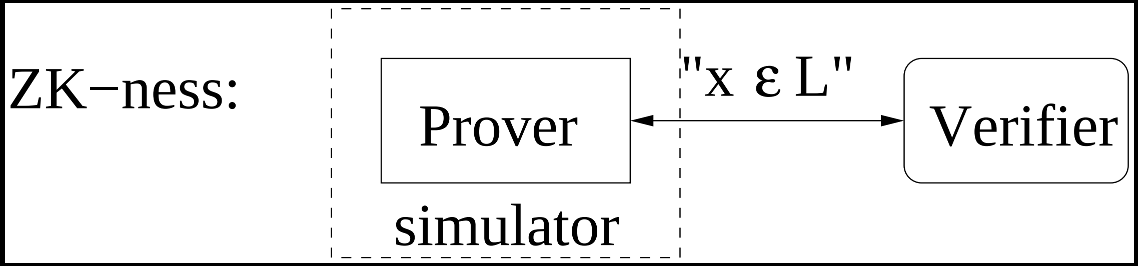


- **ZK-ness** — a **simulator** produces the conversation w/o witness

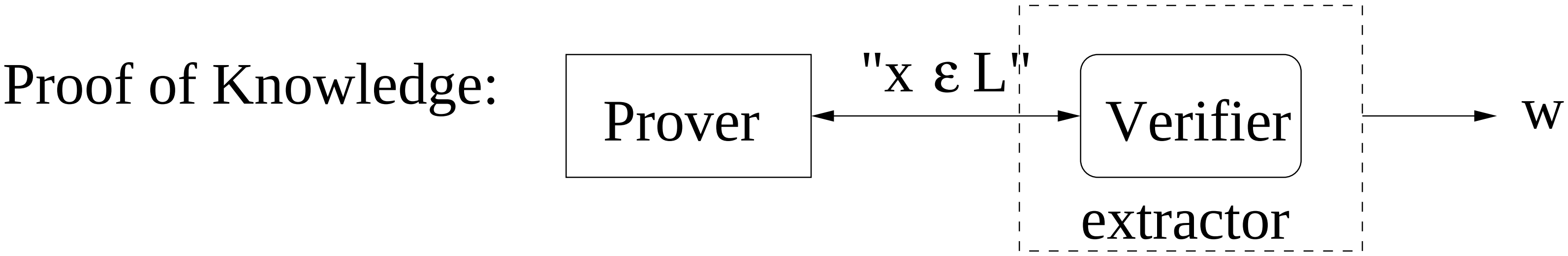
A Quick Review of ZK Proofs

A protocol between Prover and Verifier.

- Completeness — if $x \in L$ then Verifier always accepts
- Soundness — if $x \notin L$ then Verifier accepts with negl. prob.



- ZK-ness — a simulator produces the conversation w/o witness



- POK — an extractor produces a witness w from interaction

Issues of (Strengthening) Zero Knowledge Protocols

Since GMR85, many efforts are made to strengthen the original definition of ZK proofs to fit into the “real world,” a.k.a. the “Internet.”

Issues of (Strengthening) Zero Knowledge Protocols

Since GMR85, many efforts are made to strengthen the original definition of ZK proofs to fit into the “real world,” a.k.a. the “Internet.”

- Concurrency [Dwork Naor Sahai 98]
  remains ZK if many verifiers interact with the prover concurrently
  (your web server is concurrent)

Issues of (Strengthening) Zero Knowledge Protocols

Since GMR85, many efforts are made to strengthen the original definition of ZK proofs to fit into the “real world,” a.k.a. the “Internet.”

- Concurrency [Dwork Naor Sahai 98]
  remains ZK if many verifiers interact with the prover concurrently
  (your web server is concurrent)
- Non-malleability [Dolev Dwork Naor 91]
  secure against the man-in-the-middle attack
  (necessary in a peer-to-peer network/routing protocols)

Issues of (Strengthening) Zero Knowledge Protocols

Since GMR85, many efforts are made to strengthen the original definition of ZK proofs to fit into the “real world,” a.k.a. the “Internet.”

- Concurrency [Dwork Naor Sahai 98]
  remains ZK if many verifiers interact with the prover concurrently
  (your web server is concurrent)
- Non-malleability [Dolev Dwork Naor 91]
  secure against the man-in-the-middle attack
  (necessary in a peer-to-peer network/routing protocols)
- Universal Composability [Canetti 00]
  secure when arbitrarily composed
  (desirable for modularity)

Concurrent ZK

“Protocol remains ZK when concurrently composed.”

- Introduced by [Dwork Naor Sahai 98]

Concurrent ZK

“Protocol remains ZK when concurrently composed.”

- Introduced by [Dwork Naor Sahai 98]
- Difficult in the plain model
  - [Canetti Kilian Petrank Rosen 01]
    blackbox ZK needs $\Omega(\log k)$ rounds
  - [Prabhakaran Rosen Sahai 02]
    $O(\log k)$ rounds suffice
  - [Barak 01]
    constant round non-blackbox ZK (bounded concurrency)

Concurrent ZK

“Protocol remains ZK when concurrently composed.”

- Introduced by [Dwork Naor Sahai 98]
- Difficult in the plain model
  - [Canetti Kilian Petrank Rosen 01]
    blackbox ZK needs $\Omega(\log k)$ rounds
  - [Prabhakaran Rosen Sahai 02]
    $O(\log k)$ rounds suffice
  - [Barak 01]
    constant round non-blackbox ZK (bounded concurrency)
- Easy in the common reference string (CRS) model
  [Damgård 00] constant round ZK (simulator generates CRS)

We work in the CRS model.

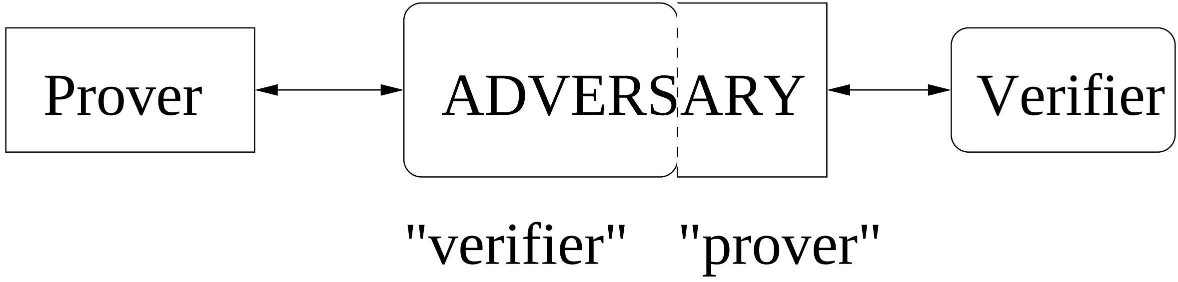
Non-malleable ZK

“Seeing a proof doesn’t help prove something related.”

Non-malleable ZK

“Seeing a proof doesn’t help prove something related.”

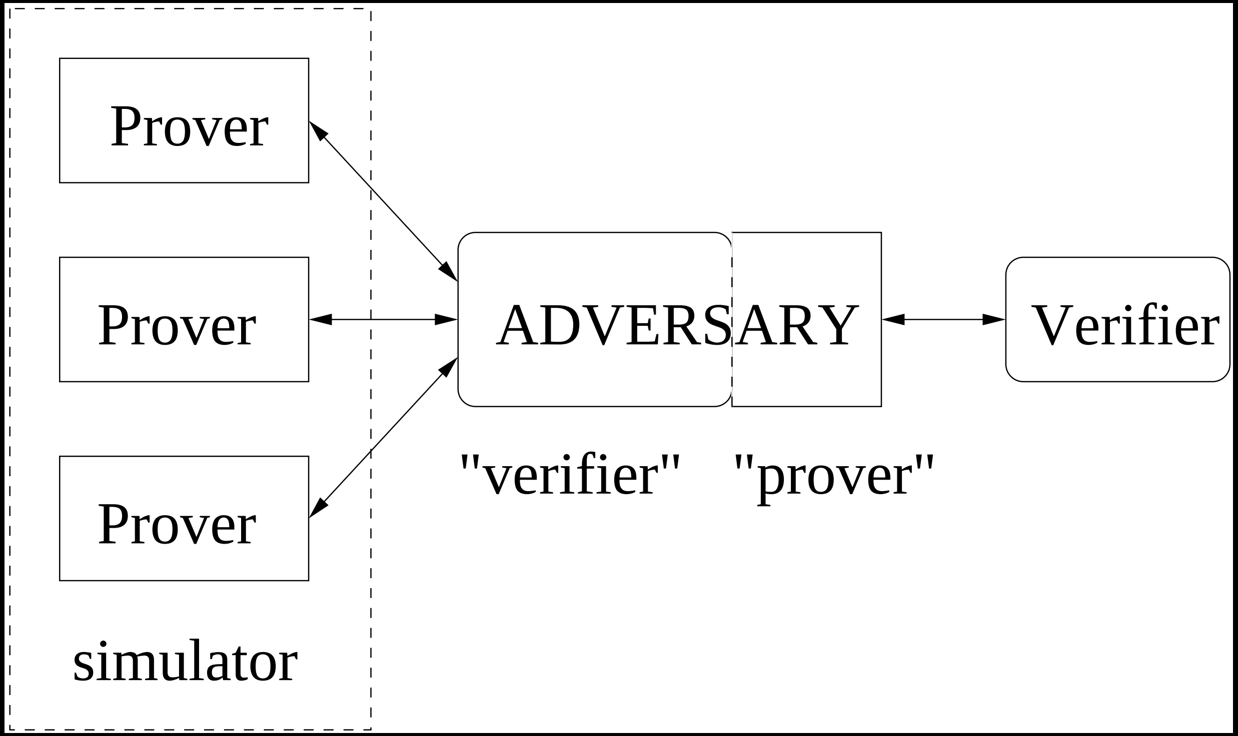
- [Dolev Dwork Naor 91] one-time non-malleable ZK
- [Sahai 99] one-time non-malleable NIZK



Non-malleable ZK

“Seeing a proof doesn’t help prove something related.”

- [Dolev Dwork Naor 91] one-time non-malleable ZK
- [Sahai 99] one-time non-malleable NIZK
- [De Santis, Di Crescenzo, Ostrovsky, Persiano, Sahai 01] unbounded non-malleable NIZK



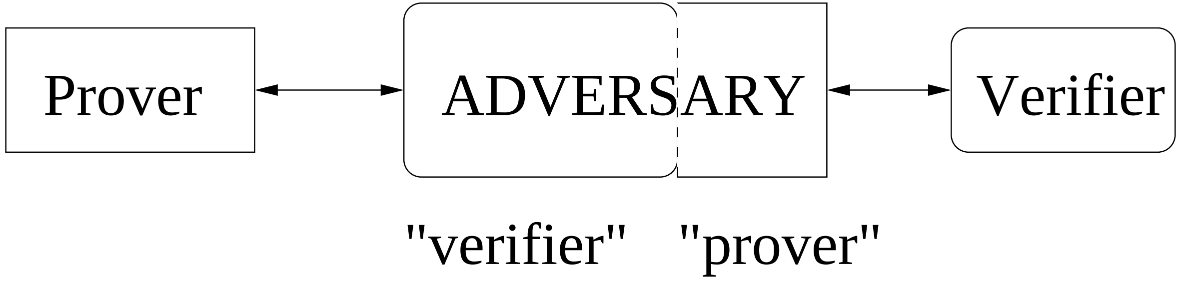
Simulation Sound (NI)ZK

“Seeing a simulated false proof doesn’t help prove something wrong.”

Simulation Sound (NI)ZK

“Seeing a simulated false proof doesn’t help prove something wrong.”

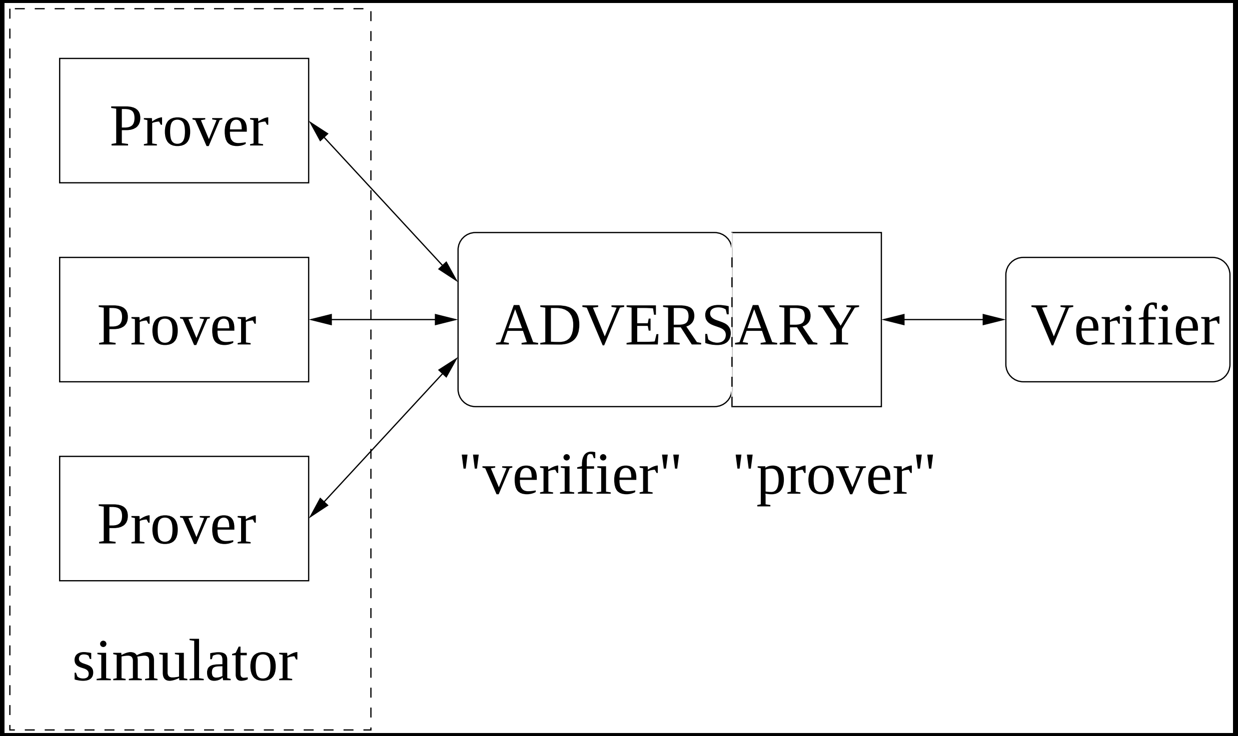
- [Sahai 99] one-time simulation sound NIZK



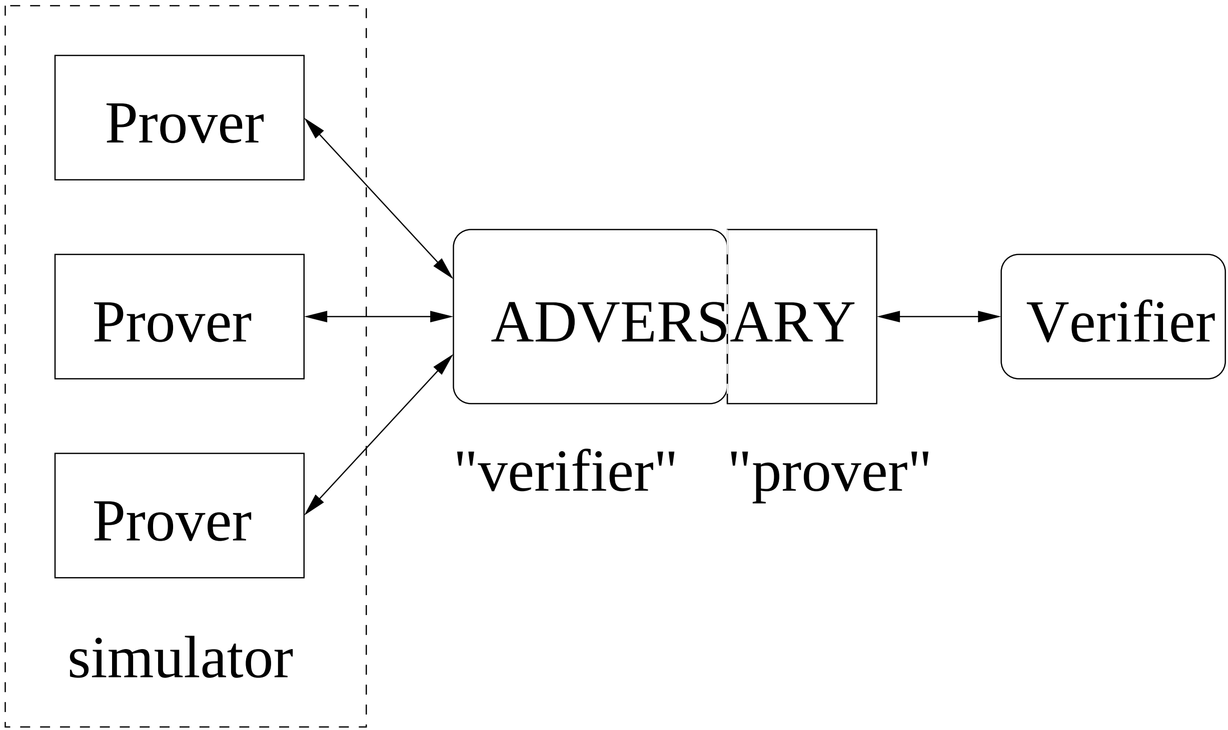
Simulation Sound (NI)ZK

“Seeing a simulated false proof doesn’t help prove something wrong.”

- [Sahai 99] one-time simulation sound NIZK
- [De Santis, Di Crescenzo, Ostrovsky, Persiano, Sahai 01] unbounded simulation sound NIZK

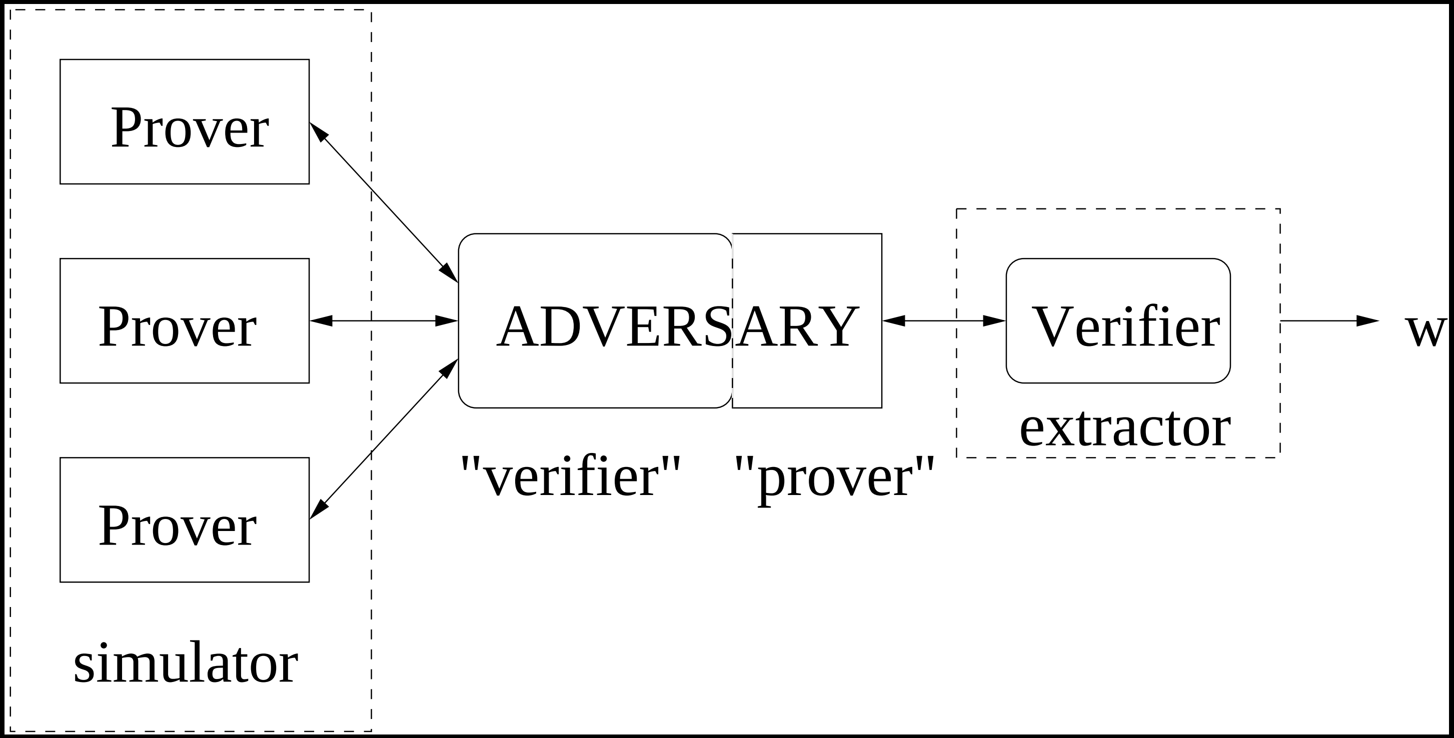


Interactive Simulation Sound ZK



- We allow $\mathcal{A}$ to concurrently interact with many simulated provers.
- Still $\mathcal{A}$ cannot produce a false proof.

Interactive Non-malleable ZK



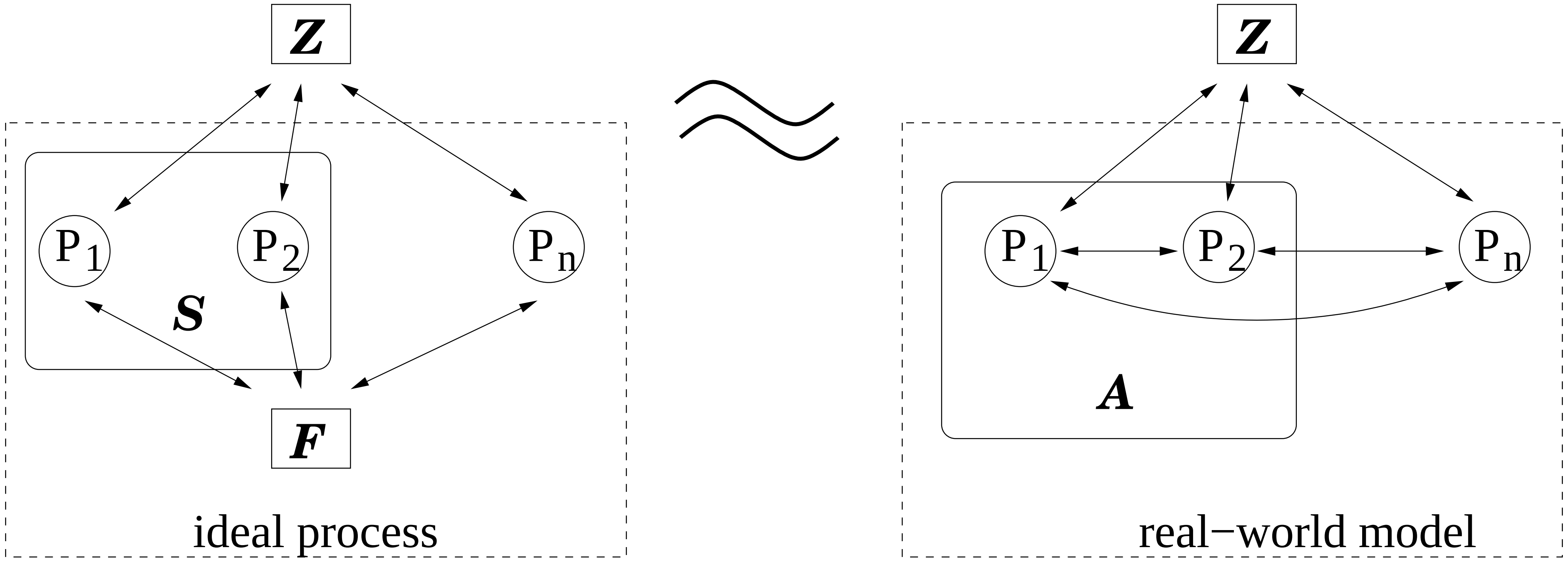
- We allow $\mathcal{A}$ to concurrently interact with many simulated provers.
- Anything $\mathcal{A}$ proves, a witness can be extracted.
- Roughly speaking,
  Non-malleable ZK = Simulation Sound ZK + non-rewinding POK.

Non-malleable/Simulation Sound ZK: Known Constructions

- [Dolev Dwork Naor 91]
  one-time non-malleable ZK, polylogarithmic rounds, plain model
- [Barak 02]
  one-time non-malleable ZK, constant rounds, plain model
- [Katz 03]
  one-time non-malleable ZK, three rounds, CRS model
- [Sahai 00]
  unbounded simulation-sound NIZK
- [De Santis, Di Crescenzo, Ostrovsky, Persiano, Sahai 01]
  unbounded non-malleable NIZK

Universally Composable ZK

- [Canetti 00]
  Universal Composability: a framework for defining (very strong) security that allows arbitrary composition



UCZK: Known Results

- Roughly speaking

  UCZK $\sim$ unbounded non-malleable ZK
- [Canetti 00]

  UCZK impossible in plain model
- [Canetti Fischlin 01]

  three round UCZK in CRS model, adaptive corruption
- [Canetti Lindell Ostrovsky Sahai 02]

  The DDOPS01 construction is NIUCZK, non-adaptive corruption

Efficiency?

Most of the previous constructions are not very efficient.

Efficiency?

Most of the previous constructions are not very efficient.

- Complicated constructions
  e.g. non-blackbox simulation

Efficiency?

Most of the previous constructions are not very efficient.

- Complicated constructions
  e.g. non-blackbox simulation
- NIZK
  Non-interactive ZK is generally inefficient.

Efficiency?

Most of the previous constructions are not very efficient.

- Complicated constructions
  e.g. non-blackbox simulation
- NIZK
  Non-interactive ZK is generally inefficient.
- Cook-Levin theorem
  - pick an NP-complete language L

Efficiency?

Most of the previous constructions are not very efficient.

- Complicated constructions
  e.g. non-blackbox simulation
- NIZK
  Non-interactive ZK is generally inefficient.
- Cook-Levin theorem
  - pick an NP-complete language L
  - construct a (concurrent/simulation sound/non-malleable/UC) ZK proof for L

Efficiency?

Most of the previous constructions are not very efficient.

- Complicated constructions
  e.g. non-blackbox simulation
- NIZK
  Non-interactive ZK is generally inefficient.
- Cook-Levin theorem
  - pick an NP-complete language L
  - construct a (concurrent/simulation sound/non-malleable/UC) ZK proof for L
  - reduce the ZK proof for any NP language to L

Efficiency?

Most of the previous constructions are not very efficient.

- Complicated constructions
  e.g. non-blackbox simulation
- NIZK
  Non-interactive ZK is generally inefficient.
- Cook-Levin theorem
  - pick an NP-complete language L
  - construct a (concurrent/simulation sound/non-malleable/UC) ZK proof for L
  - reduce the ZK proof for any language to L ← Inefficient!

Our Contributions

A novel technique to construct efficient, concurrent, non-malleable, and/or universal composable ZK in the CRS model using signatures

Our Contributions

A novel technique to construct efficient, concurrent, non-malleable, and/or universal composable ZK in the CRS model using signatures

- Σ-protocol (three-round, public-coin, honest-verifier) $\Longrightarrow$ unbounded simulation-sound ZK

Our Contributions

A novel technique to construct efficient, concurrent, non-malleable, and/or universal composable ZK in the CRS model using signatures

- Σ-protocol (three-round, public-coin, honest-verifier)
  $\Longrightarrow$ unbounded simulation-sound ZK
- Ω-protocol (Σ-protocol + non-rewinding POK)
  $\Longrightarrow$ unbounded non-malleable ZK
  $\Longrightarrow$ universally composable ZK

Our Contributions

A novel technique to construct efficient, concurrent, non-malleable, and/or universal composable ZK in the CRS model using signatures

- Σ-protocol (three-round, public-coin, honest-verifier)
  $\Longrightarrow$ unbounded simulation-sound ZK
- Ω-protocol (Σ-protocol + non-rewinding POK)
  $\Longrightarrow$ unbounded non-malleable ZK
  $\Longrightarrow$ universally composable ZK

What's special about our technique?

- conceptually simple

Our Contributions

A novel technique to construct efficient, concurrent, non-malleable, and/or universal composable ZK in the CRS model using signatures

- Σ-protocol (three-round, public-coin, honest-verifier)
  $\Longrightarrow$ unbounded simulation-sound ZK
- Ω-protocol (Σ-protocol + non-rewinding POK)
  $\Longrightarrow$ unbounded non-malleable ZK
  $\Longrightarrow$ universally composable ZK

What's special about our technique?

- conceptually simple
- efficient

Our Contributions

A novel technique to construct efficient, concurrent, non-malleable, and/or universal composable ZK in the CRS model using signatures

- Σ-protocol (three-round, public-coin, honest-verifier)
  $\Longrightarrow$ unbounded simulation-sound ZK
- Ω-protocol (Σ-protocol + non-rewinding POK)
  $\Longrightarrow$ unbounded non-malleable ZK
  $\Longrightarrow$ universally composable ZK

What's special about our technique?

- conceptually simple
- efficient
  - three rounds, small additive overhead (const. pub. key op's)

Our Contributions

A novel technique to construct efficient, concurrent, non-malleable, and/or universal composable ZK in the CRS model using signatures

- Σ-protocol (three-round, public-coin, honest-verifier)
  $\Longrightarrow$ unbounded simulation-sound ZK
- Ω-protocol (Σ-protocol + non-rewinding POK)
  $\Longrightarrow$ unbounded non-malleable ZK
  $\Longrightarrow$ universally composable ZK

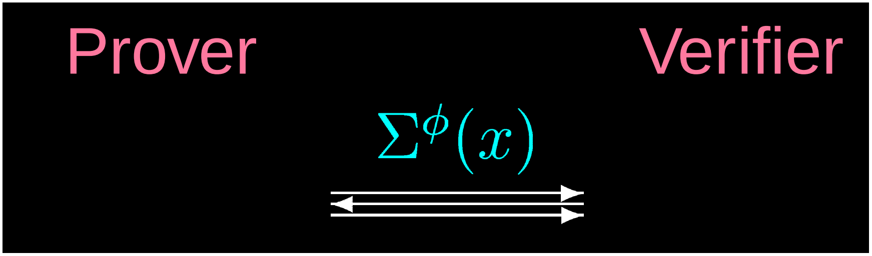
What's special about our technique?

- conceptually simple
- efficient
  - three rounds, small additive overhead (const. pub. key op's)
  - completely avoid the Cook-Levin Theorem
    (c.f. Micciancio and Petrank,
    "Simulatable Commitments and Efficient Concurrent Zero-Knowledge,"
    an hour ago.)

Ideas of the Conversion

Start with a Σ-protocol

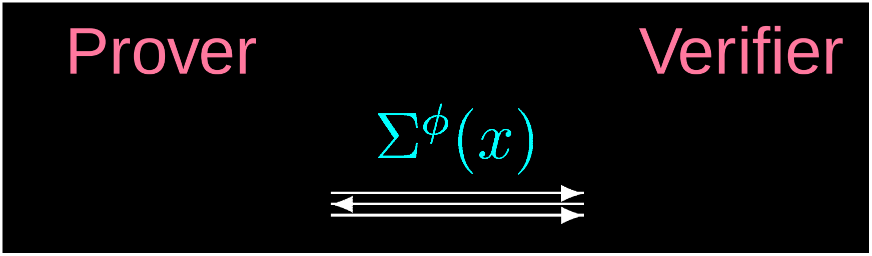
Π= “ϕ is true.”



Ideas of the Conversion

Start with a Σ-protocol

Π= “ϕ is true.”



Prover

Verifier

$\Sigma^{\phi}(x)$

Convert to

Π' = “Either ϕ is true, or I know a signature for message m w.r.t. vk.”

Protocol in More Details

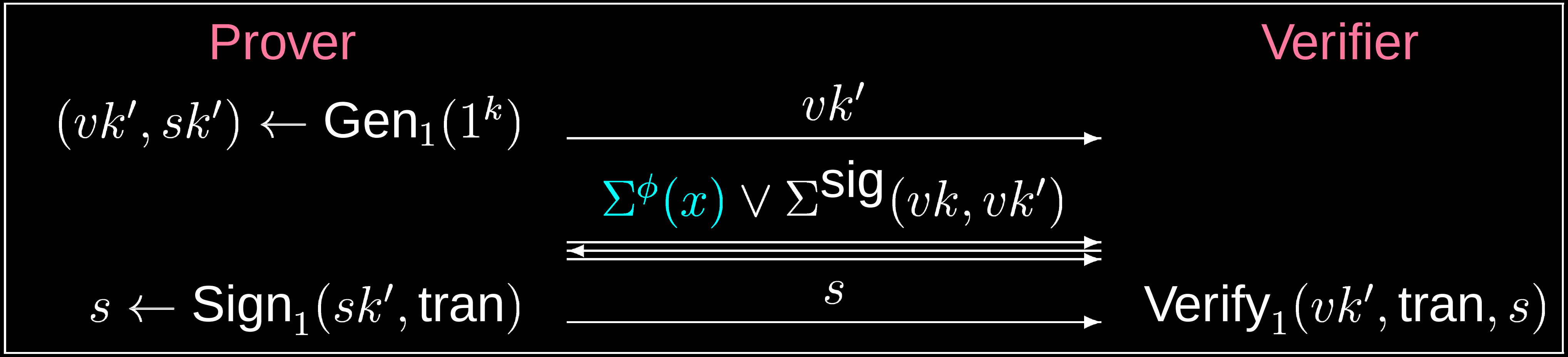
Π' = "Either ϕ is true, or I know a signature for message m w.r.t. vk."

- vk is from a digital signature scheme SIG $=$ (Gen, Sign, Verify) existential unforgeable against chosen message attack.
- vk is in the common reference string (sk unknown).
- $m = vk'$ is a fresh verification key of a one-time signature scheme SIG$_1$ $=$ (Gen$_1$, Sign$_1$, Verify$_1$).

Protocol in More Details

Π' = “Either ϕ is true, or I know a signature for message m w.r.t. vk.”

- vk is from a digital signature scheme $\text{SIG} = (\text{Gen}, \text{Sign}, \text{Verify})$ existential unforgeable against chosen message attack.
- vk is in the common reference string (sk unknown).
- $m = vk'$ is a fresh verification key of a one-time signature scheme $\text{SIG}_1 = (\text{Gen}_1, \text{Sign}_1, \text{Verify}_1)$.



Prover

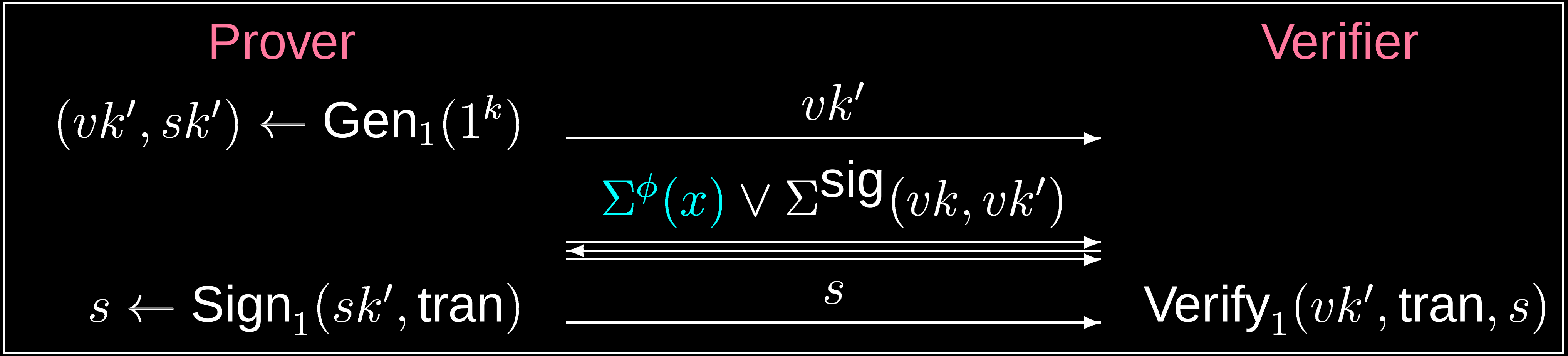
Verifier

$(vk', sk') \leftarrow \text{Gen}_1(1^k)$ $\xrightarrow{vk'}$

$\Sigma^{\phi}(x) \vee \Sigma^{\text{sig}}(vk, vk')$

$s \leftarrow \text{Sign}_1(sk', \text{tran})$ $\xrightarrow{s}$ $\text{Verify}_1(vk', \text{tran}, s)$

How does it Work?



Prover

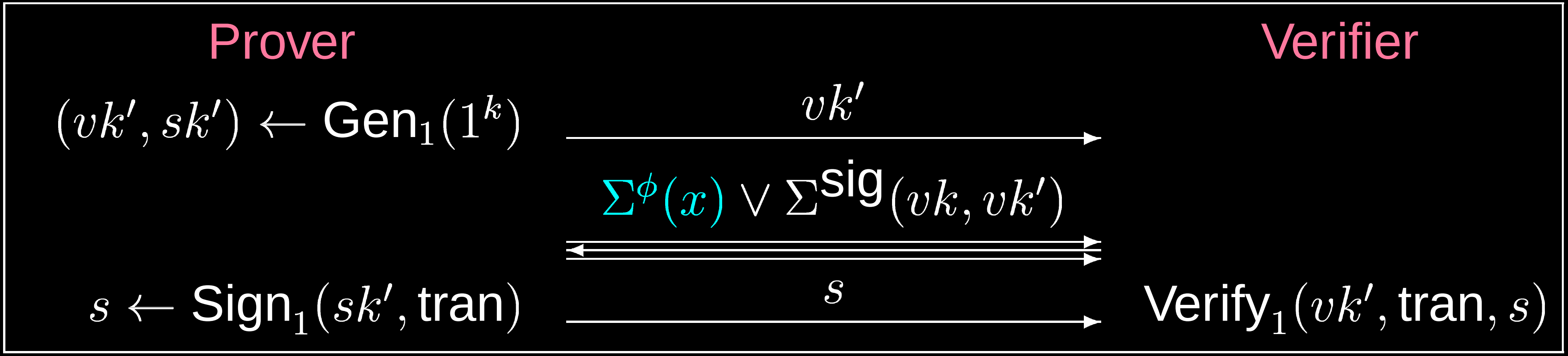
Verifier

$(vk', sk') \leftarrow \mathsf{Gen}_1(1^k) \xrightarrow{vk'}$

$\Sigma^{\phi}(x) \vee \Sigma^{\mathsf{sig}}(vk, vk')$

$s \leftarrow \mathsf{Sign}_1(sk', \mathsf{tran}) \xrightarrow{s} \mathsf{Verify}_1(vk', \mathsf{tran}, s)$

How does it Work?



Prover

Verifier

$(vk', sk') \leftarrow \text{Gen}_1(1^k)$

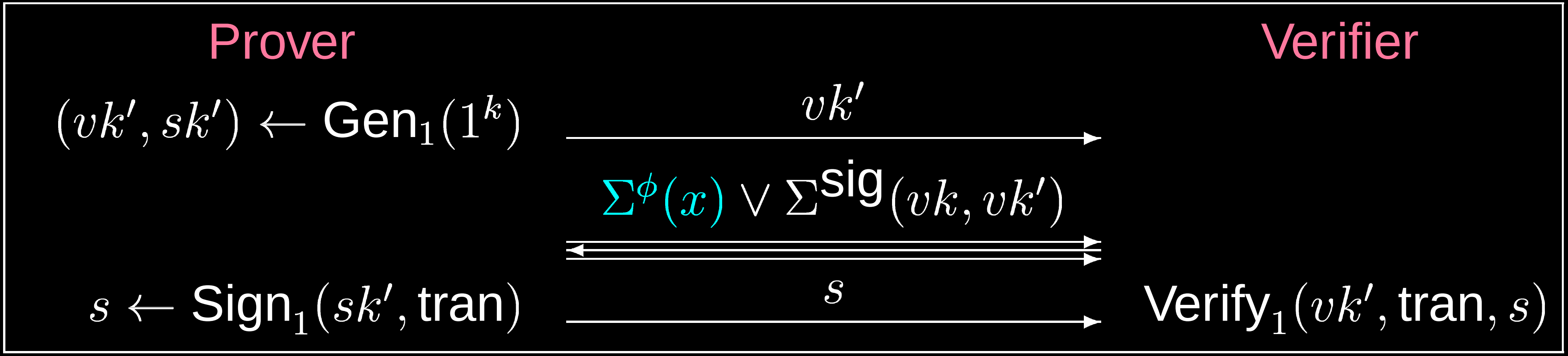
$$(vk', sk') \leftarrow \text{Gen}_1(1^k) \xrightarrow{vk'}$$

$$\Sigma^{\phi}(x) \vee \Sigma^{\text{sig}}(vk, vk')$$

$$s \leftarrow \text{Sign}_1(sk', \text{tran}) \xrightarrow{s} \text{Verify}_1(vk', \text{tran}, s)$$

- Completeness — straightforward

How does it Work?



Prover **Verifier**

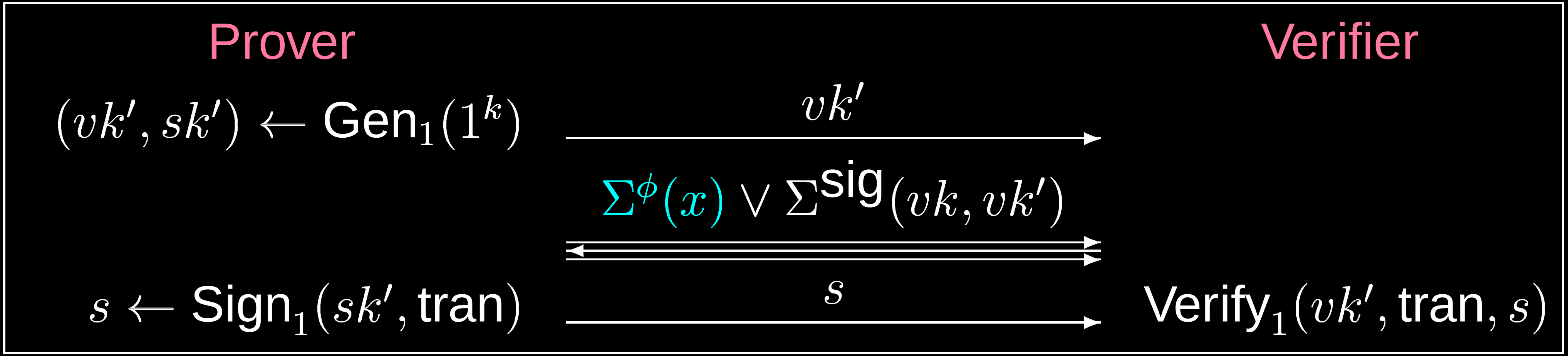
$(vk', sk') \leftarrow \text{Gen}_1(1^k) \xrightarrow{vk'}$

$\Sigma^{\phi}(x) \vee \Sigma^{\text{sig}}(vk, vk')$

$s \leftarrow \text{Sign}_1(sk', \text{tran}) \xrightarrow{s} \text{Verify}_1(vk', \text{tran}, s)$

- Completeness — straightforward
- Soundness — since sk unknown, infeasible to fake a signature

How does it Work?



Prover

Verifier

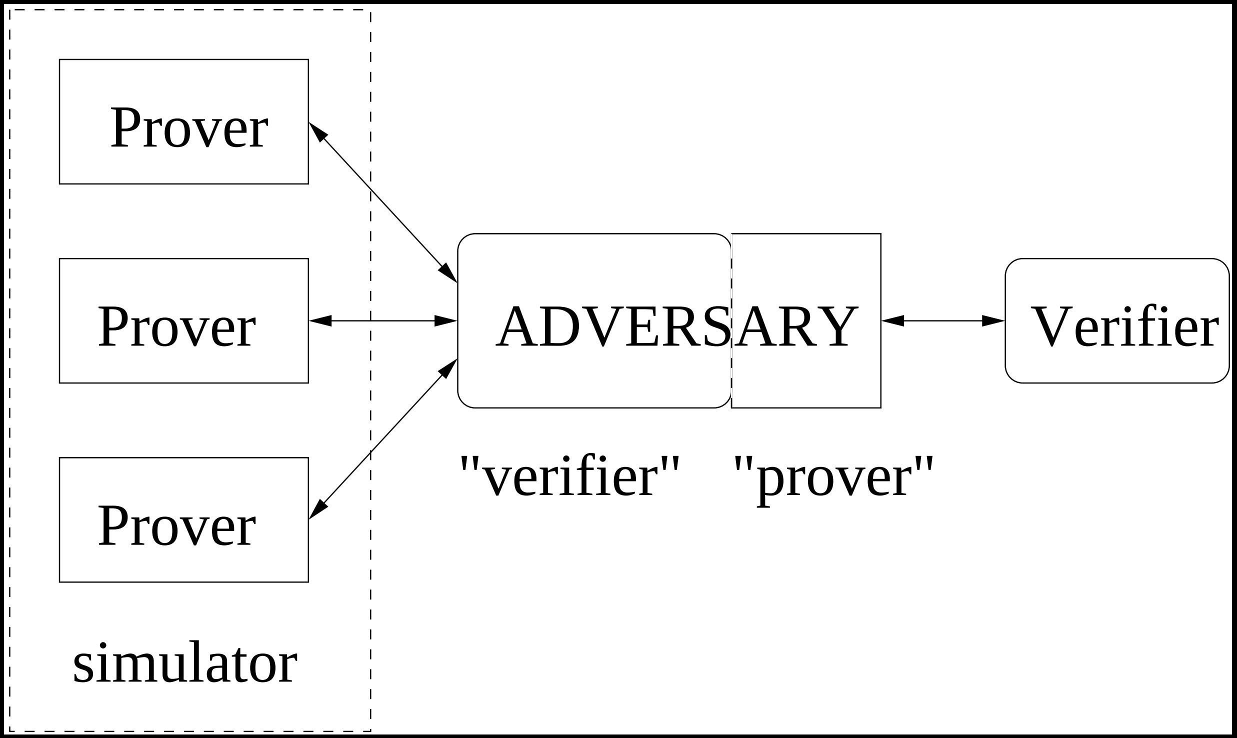
$(vk', sk') \leftarrow \text{Gen}_1(1^k) \xrightarrow{\quad vk' \quad}$

$\Sigma^{\phi}(x) \vee \Sigma^{\text{sig}}(vk, vk')$

$s \leftarrow \text{Sign}_1(sk', \text{tran}) \xrightarrow{\quad s \quad} \text{Verify}_1(vk', \text{tran}, s)$

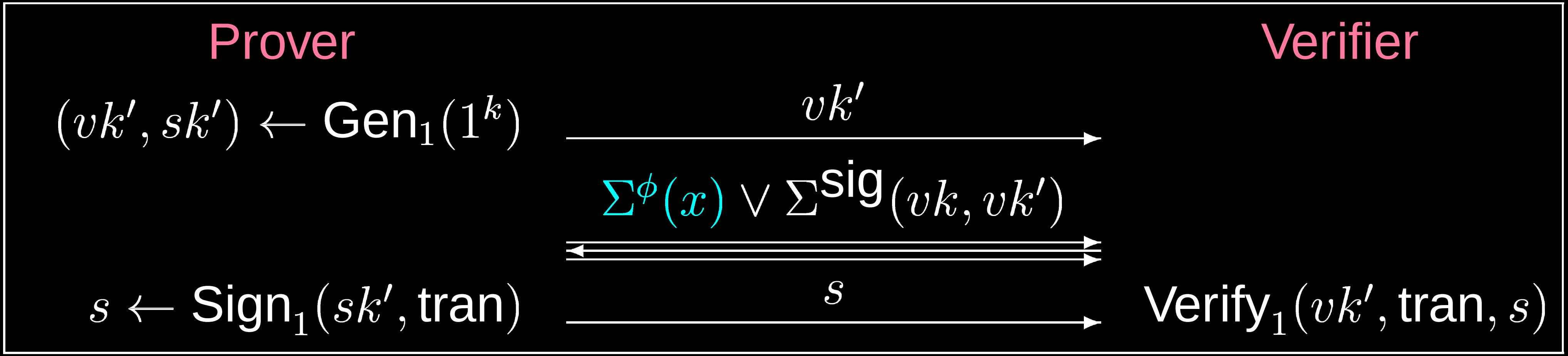
- Completeness — straightforward
- Soundness — since sk unknown, infeasible to fake a signature
- ZK-ness — S generates (vk, sk) and can produce signatures (non-rewinding simulation means concurrency)

How does it Work — Unbounded Simulation Soundness



- We allow $\mathcal{A}$ to (arbitrarily) interact with many (simulated) provers.
- Still $\mathcal{A}$ cannot produce a false proof.

How does it Work — Unbounded Simulation Soundness



Prover **Verifier**

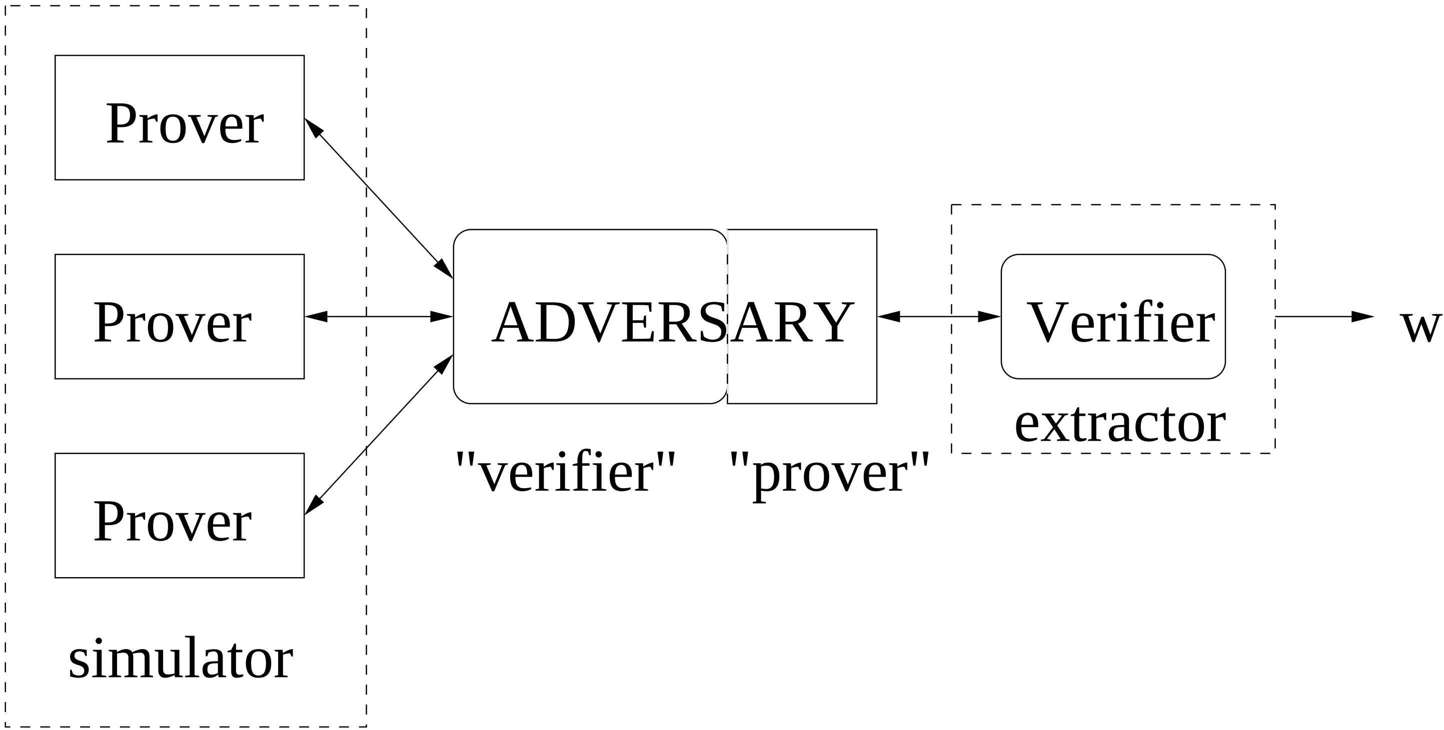
$(vk', sk') \leftarrow \mathsf{Gen}_1(1^k) \xrightarrow{vk'}$

$\Sigma^{\phi}(x) \vee \Sigma^{\mathsf{sig}}(vk, vk')$

$s \leftarrow \mathsf{Sign}_1(sk', \mathsf{tran}) \xrightarrow{s} \mathsf{Verify}_1(vk', \mathsf{tran}, s)$

- “producing a false proof” = “faking a signature for vk'”
- $\mathcal{A}$ does not know sk' $\Rightarrow$ cannot reuse vk'
- $\mathcal{A}$ fakes a signature for a fresh vk' $\Rightarrow$ $\mathcal{A}$ breaks SIG

How about Unbounded Non-malleability?

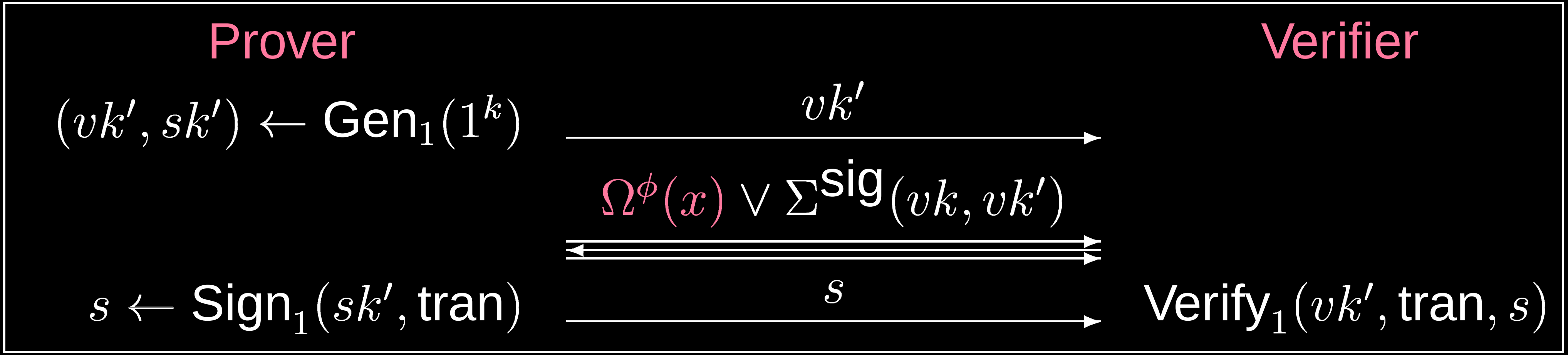


Non-malleable ZK = Simulation Sound ZK + non-rewinding POK

- We allow $\mathcal{A}$ to interact with many (simulated) provers.
- Anything $\mathcal{A}$ proves, a witness can be extracted.

From Ω-protocols to Unbounded Non-malleability

same construction, let Π be an Ω-protocol



Prover **Verifier**

$(vk', sk') \leftarrow \mathsf{Gen}_1(1^k) \xrightarrow{vk'}$

$\Omega^{\phi}(x) \vee \Sigma^{\mathsf{sig}}(vk, vk')$

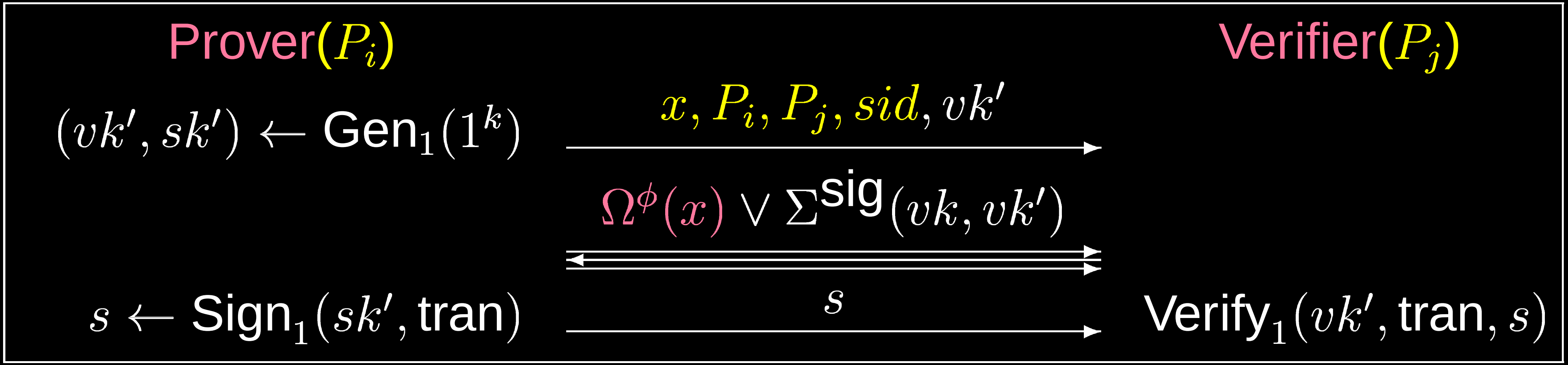
$$\longrightarrow \quad \longleftarrow \quad \longrightarrow$$

$s \leftarrow \mathsf{Sign}_1(sk', \mathsf{tran}) \xrightarrow{s} \mathsf{Verify}_1(vk', \mathsf{tran}, s)$

- Ω-protocol = Σ-protocol + non-rewinding POK

  "failing to extract" = "faking a signature for vk'"
- $\mathcal{A}$ does not know sk' $\Rightarrow$ cannot reuse vk'
- $\mathcal{A}$ fakes a signature for a fresh vk' $\Rightarrow$ $\mathcal{A}$ breaks SIG

From Unbounded Non-malleability to Universal Composability



Prover(P_i) Verifier(P_j)

$(vk', sk') \leftarrow$ Gen$_1(1^k)$ $\xrightarrow{x, P_i, P_j, sid, vk'}$

$\Omega^{\phi}(x) \vee \Sigma^{\text{sig}}(vk, vk')$

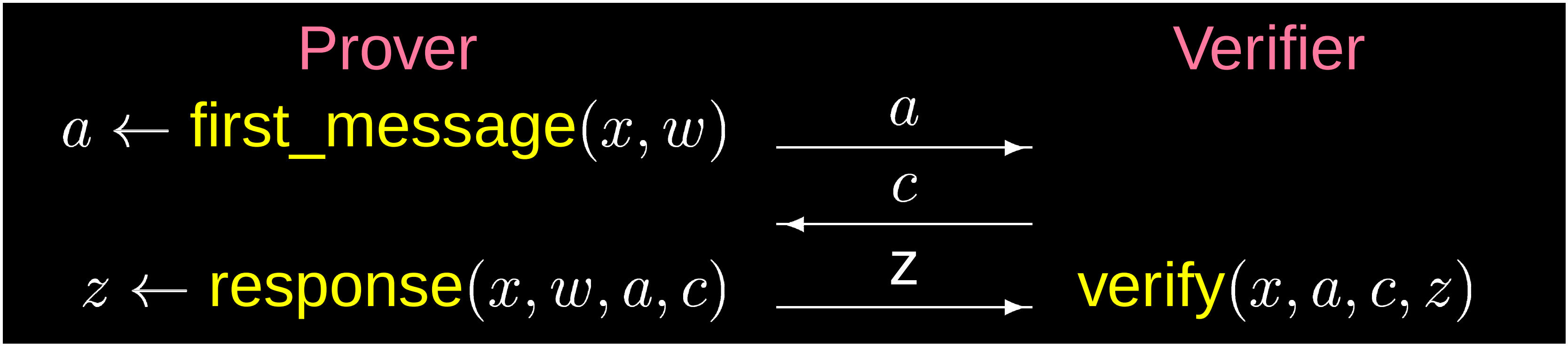
$s \leftarrow$ Sign$_1(sk', \text{tran})$ $\xrightarrow{s}$ Verify$_1(vk', \text{tran}, s)$

- roughly speaking

  UCZK $\sim$ unbounded non-malleable ZK
- easily augmentable to UCZK for non-adaptive corruption (add common input, ProverID, VerifierID, SessionID)

UCZK: Adaptive Corruption (With Erasure)

- start with the UCZK non-adaptive construction
- technique from [Damgård 00, Jarecki Lysyanskaya 00]



Prover | Verifier

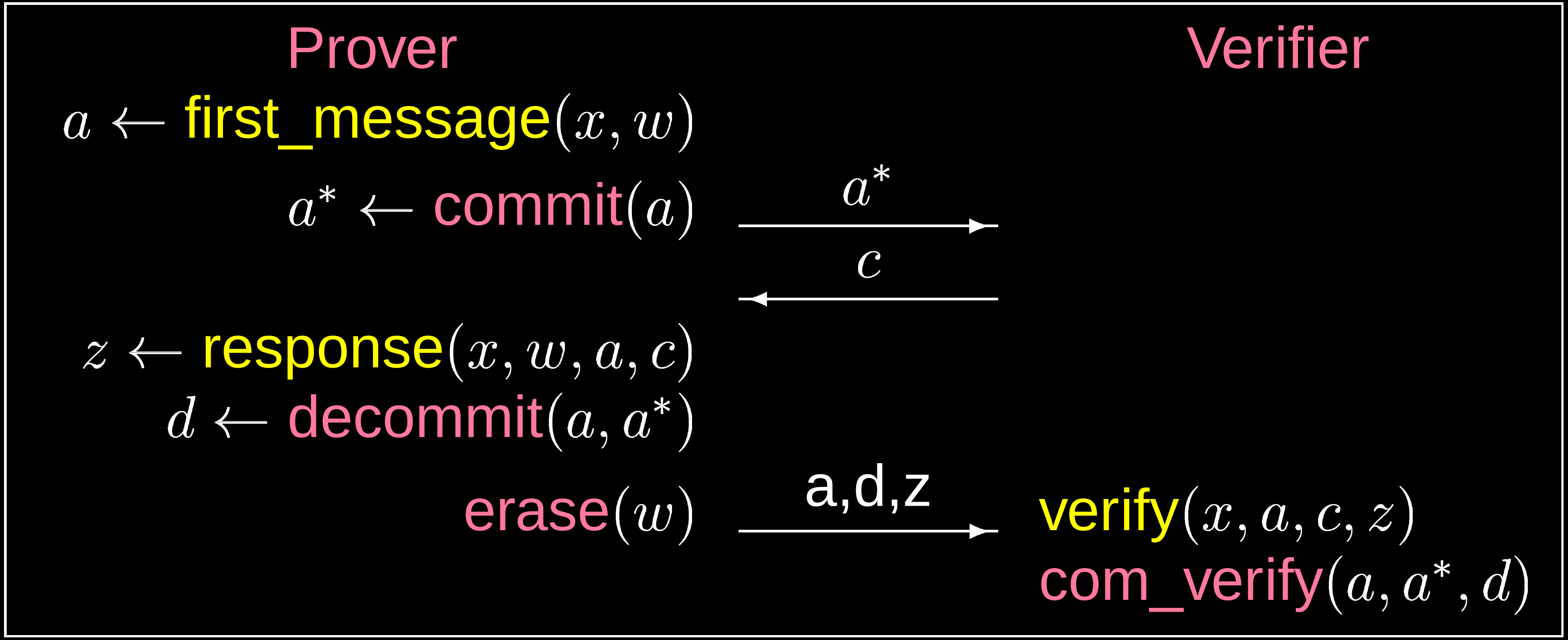
$a \leftarrow$ first_message(x, w) $\xrightarrow{a}$

$\xleftarrow{c}$

$z \leftarrow$ response(x, w, a, c) $\xrightarrow{\text{z}}$ verify(x, a, c, z)

UCZK: Adaptive Corruption (With Erasure)

- start with the UCZK non-adaptive construction
- technique from [Damgård 00, Jarecki Lysyanskaya 00]



Prover

Verifier

$a \leftarrow$ first_message(x, w)

$a^* \leftarrow$ commit(a) $\xrightarrow{a^*}$

$\xleftarrow{c}$

$z \leftarrow$ response(x, w, a, c)

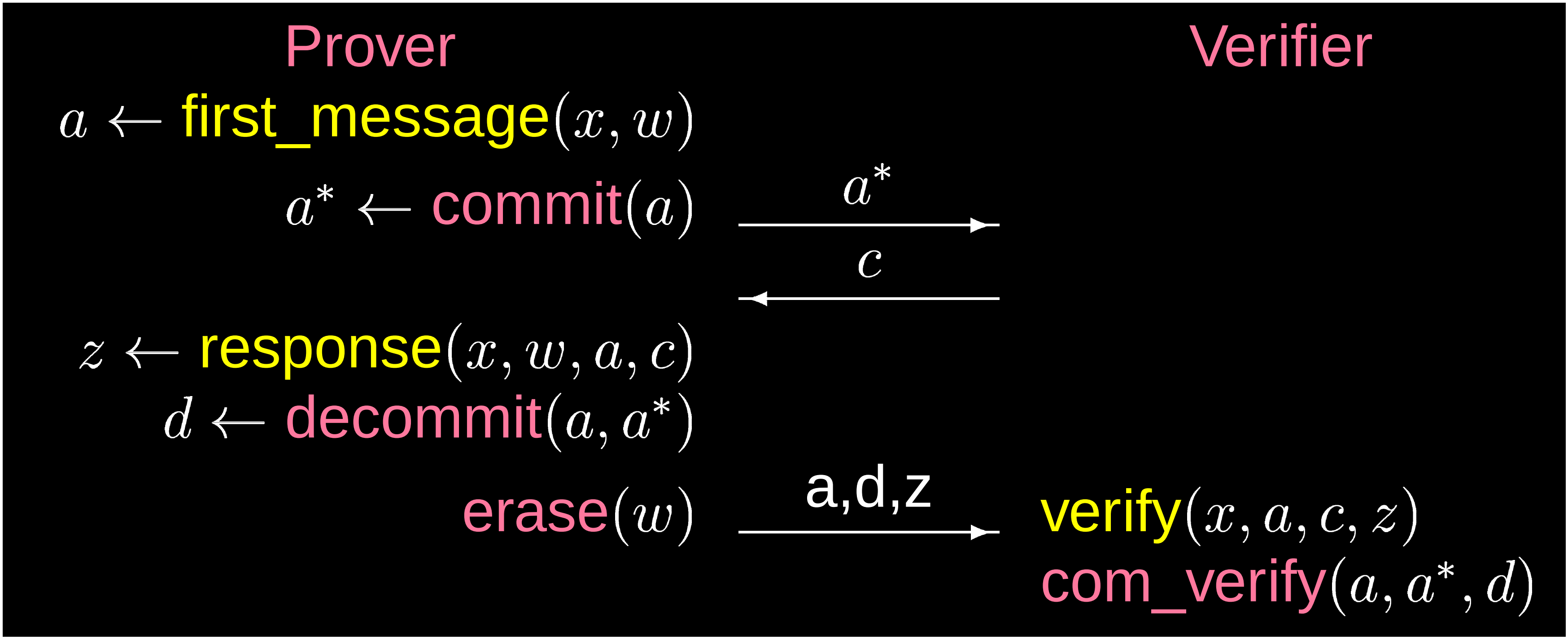
$d \leftarrow$ decommit(a, a^*)

erase(w) $\xrightarrow{\text{a,d,z}}$ verify(x, a, c, z)

com_verify(a, a^*, d)

UCZK: Adaptive Corruption (With Erasure)

- start with the UCZK non-adaptive construction
- technique from [Damgård 00, Jarecki Lysyanskaya 00]



Prover

Verifier

$a \leftarrow$ first_message(x, w)

$a^* \leftarrow$ commit(a) $\xrightarrow{a^*}$

$\xleftarrow{c}$

$z \leftarrow$ response(x, w, a, c)

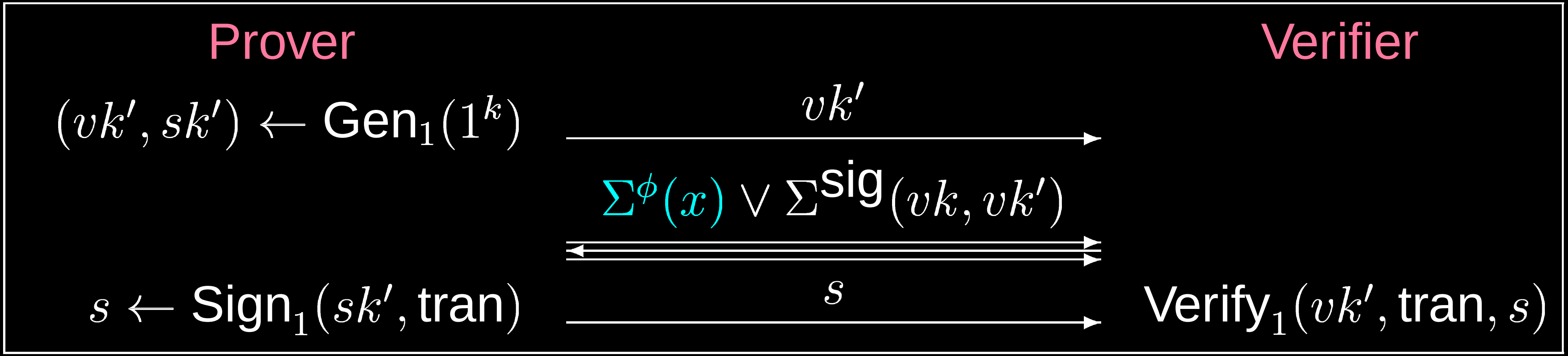
$d \leftarrow$ decommit(a, a^*)

erase(w) $\xrightarrow{\text{a,d,z}}$ verify(x, a, c, z)

com_verify(a, a^*, d)

- "Simulation Sound Trapdoor Commitment": $\mathcal{A}$ cannot fake a decommitment even after seeing a simulator faking

What About Efficiency?



Prover **Verifier**

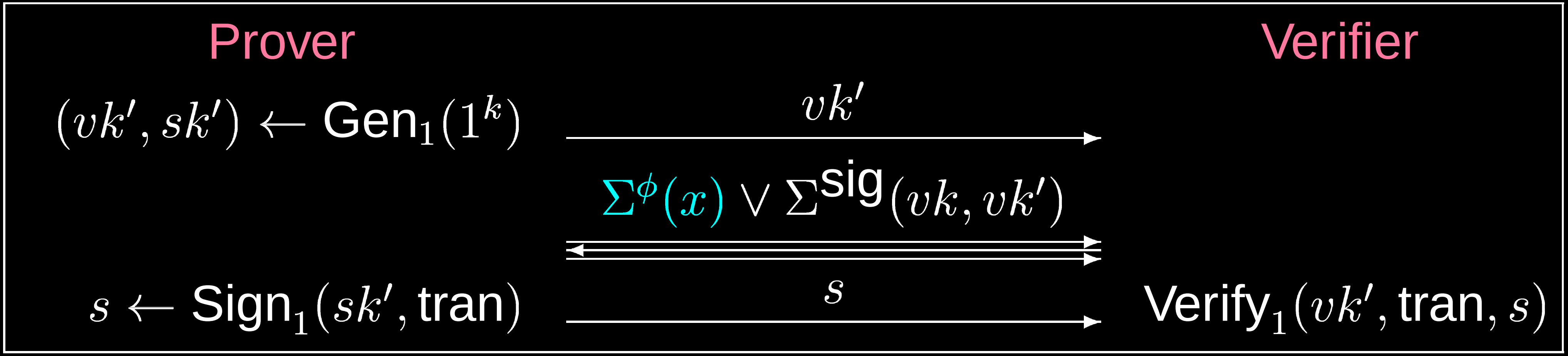
$(vk', sk') \leftarrow \mathrm{Gen}_1(1^k) \quad \xrightarrow{\quad vk' \quad}$

$\Sigma^{\phi}(x) \vee \Sigma^{\mathrm{sig}}(vk, vk')$

$\xrightarrow{\qquad\qquad} \quad \xleftarrow{\qquad\qquad} \quad \xrightarrow{\qquad\qquad}$

$s \leftarrow \mathrm{Sign}_1(sk', \mathrm{tran}) \quad \xrightarrow{\quad s \quad} \quad \mathrm{Verify}_1(vk', \mathrm{tran}, s)$

What About Efficiency?



Prover | **Verifier**

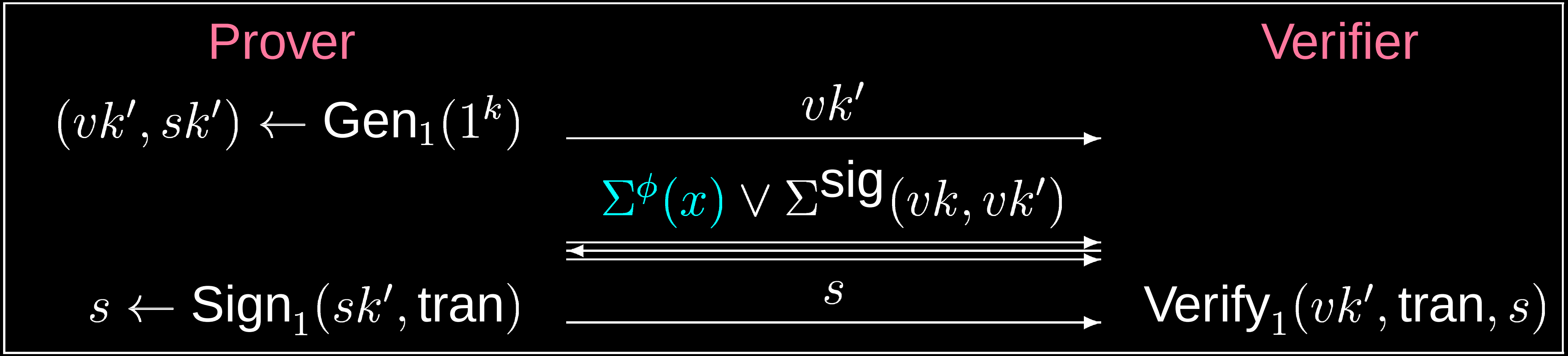
$(vk', sk') \leftarrow \mathsf{Gen}_1(1^k) \xrightarrow{vk'}$

$\Sigma^{\phi}(x) \vee \Sigma^{\mathsf{sig}}(vk, vk')$

$s \leftarrow \mathsf{Sign}_1(sk', \mathsf{tran}) \xrightarrow{s} \mathsf{Verify}_1(vk', \mathsf{tran}, s)$

- Building Π' by adding POK of signature to Π

What About Efficiency?



Prover | Verifier

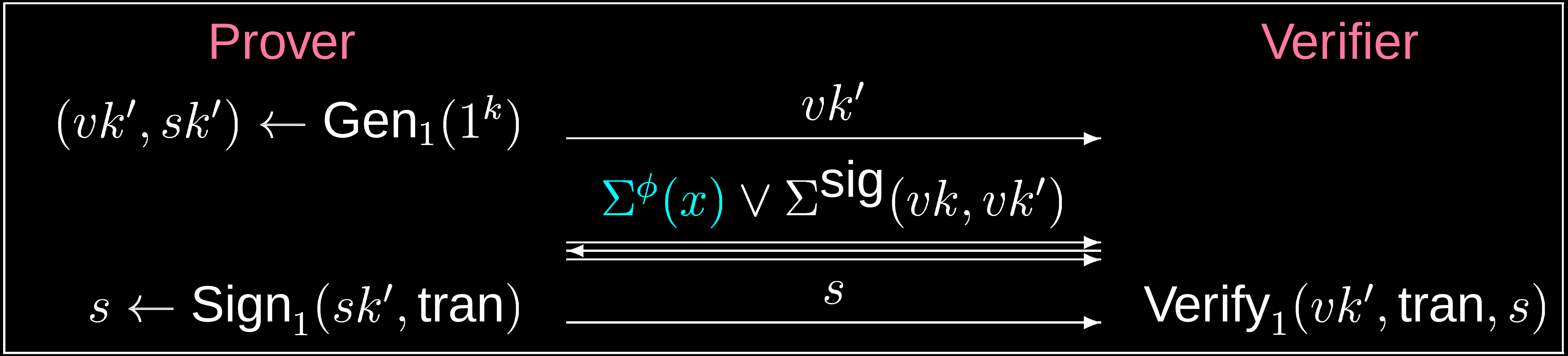
$(vk', sk') \leftarrow \text{Gen}_1(1^k)$ $\xrightarrow{vk'}$

$\Sigma^{\phi}(x) \vee \Sigma^{\text{sig}}(vk, vk')$

$s \leftarrow \text{Sign}_1(sk', \text{tran})$ $\xrightarrow{s}$ $\text{Verify}_1(vk', \text{tran}, s)$

- Building Π' by adding POK of signature to Π
  - avoids the Cook-Levin Theorem
  - efficient POK of signatures exists (Cramer-Shoup, DSA)
  - Σ-protocols $\Rightarrow$ efficient composition of “OR”

What About Efficiency?



Prover **Verifier**

$(vk', sk') \leftarrow \text{Gen}_1(1^k) \xrightarrow{\quad vk' \quad}$

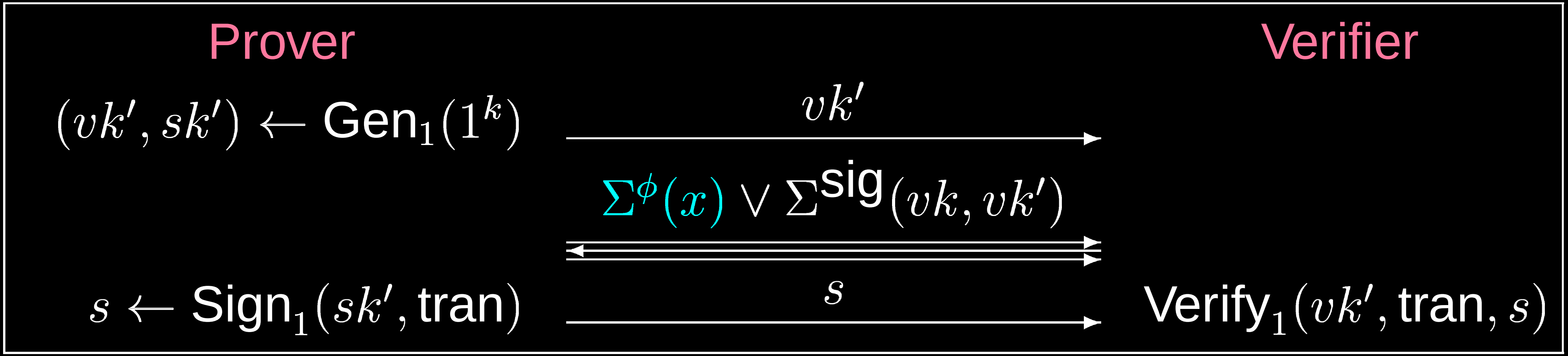
$\Sigma^{\phi}(x) \vee \Sigma^{\text{sig}}(vk, vk')$

$\longrightarrow$
$\longleftarrow$
$\longrightarrow$

$s \leftarrow \text{Sign}_1(sk', \text{tran}) \xrightarrow{\quad s \quad} \text{Verify}_1(vk', \text{tran}, s)$

- Building Π' by adding POK of signature to Π
  - avoids the Cook-Levin Theorem
  - efficient POK of signatures exists (Cramer-Shoup, DSA)
  - Σ-protocols $\Rightarrow$ efficient composition of “OR”
- Efficient one-time signatures and SSTCs

What About Efficiency?



Prover | **Verifier**

$(vk', sk') \leftarrow \text{Gen}_1(1^k) \xrightarrow{vk'}$

$\Sigma^{\phi}(x) \vee \Sigma^{\text{sig}}(vk, vk')$

$s \leftarrow \text{Sign}_1(sk', \text{tran}) \xrightarrow{s} \text{Verify}_1(vk', \text{tran}, s)$

- Building Π' by adding POK of signature to Π
    - avoids the Cook-Levin Theorem
    - efficient POK of signatures exists (Cramer-Shoup, DSA)
    - Σ-protocols $\Rightarrow$ efficient composition of "OR"
- Efficient one-time signatures and SSTCs

(honest-verifier ZK) + (additive const. pub. key operations) $\Rightarrow$

(concurrent, non-malleable, and/or universally composable ZK)