Non-Interactive Publicly-Verifiable Delegation of Committed Programs

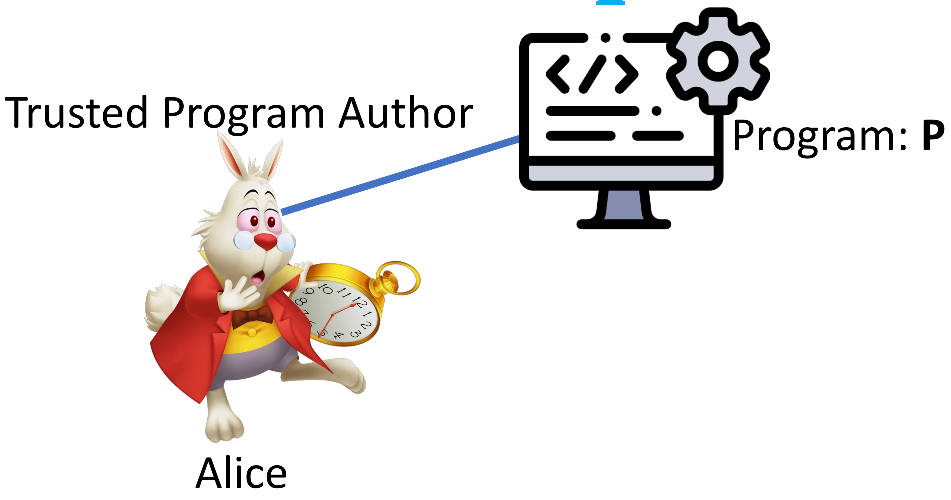
Riddhi Ghosal
UCLA

Amit Sahai
UCLA

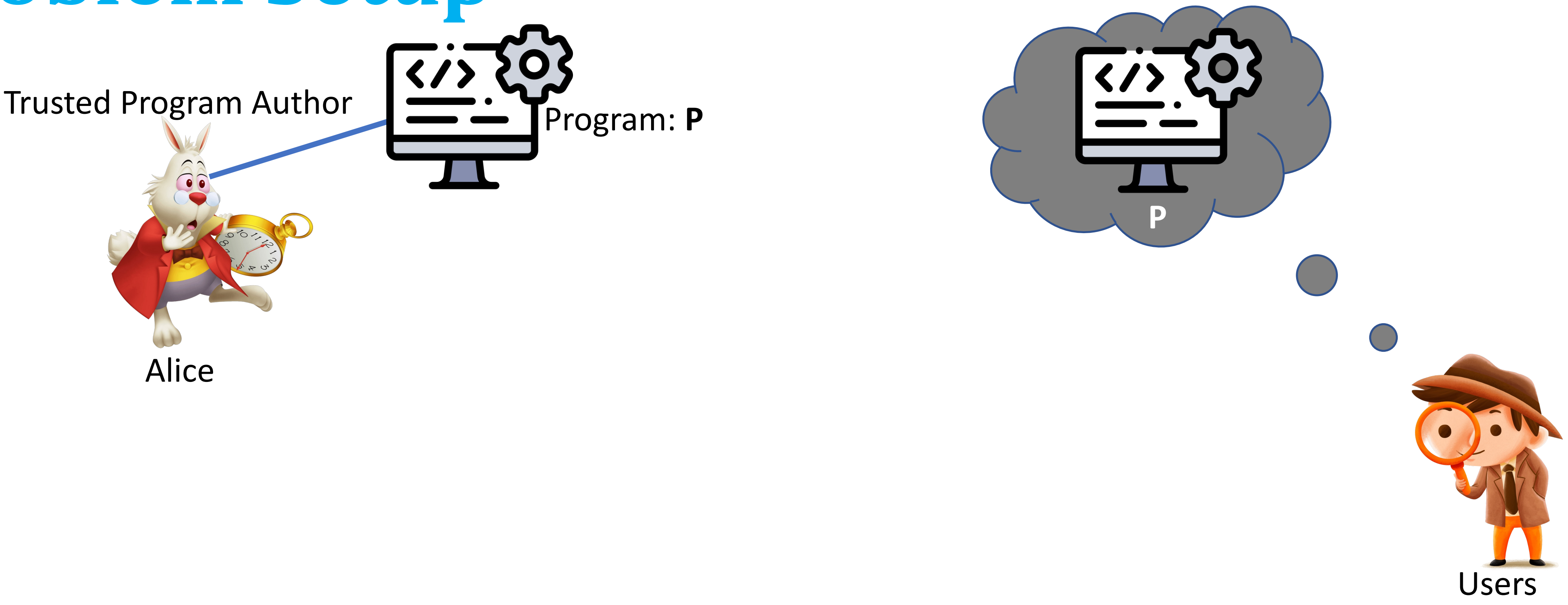
Brent Waters
UT Austin/NTT Research

Problem Setup

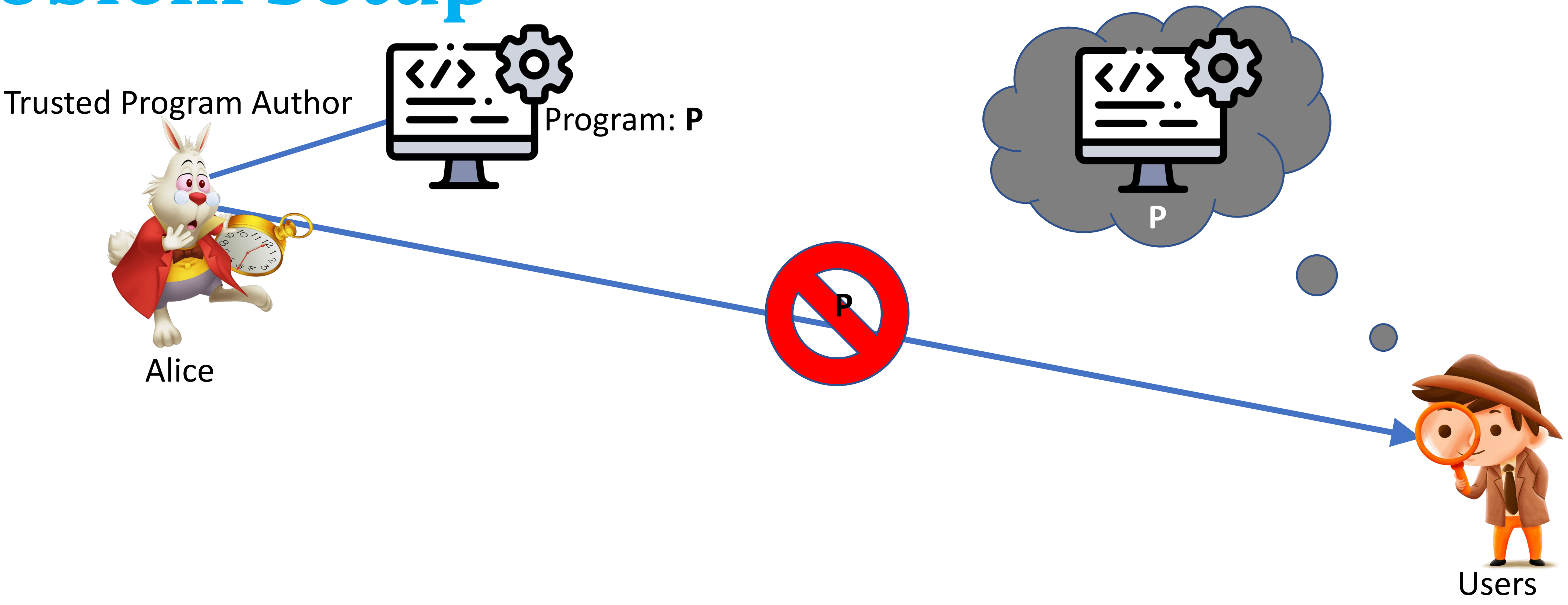
Problem Setup



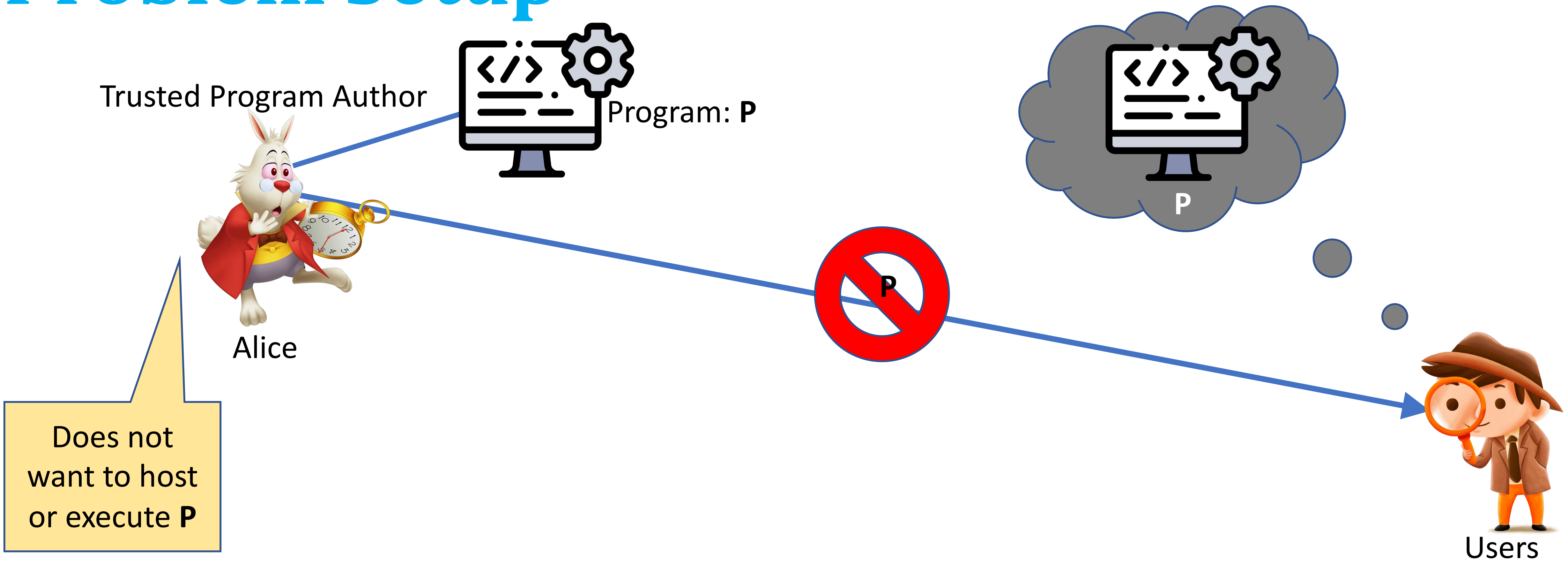
Problem Setup



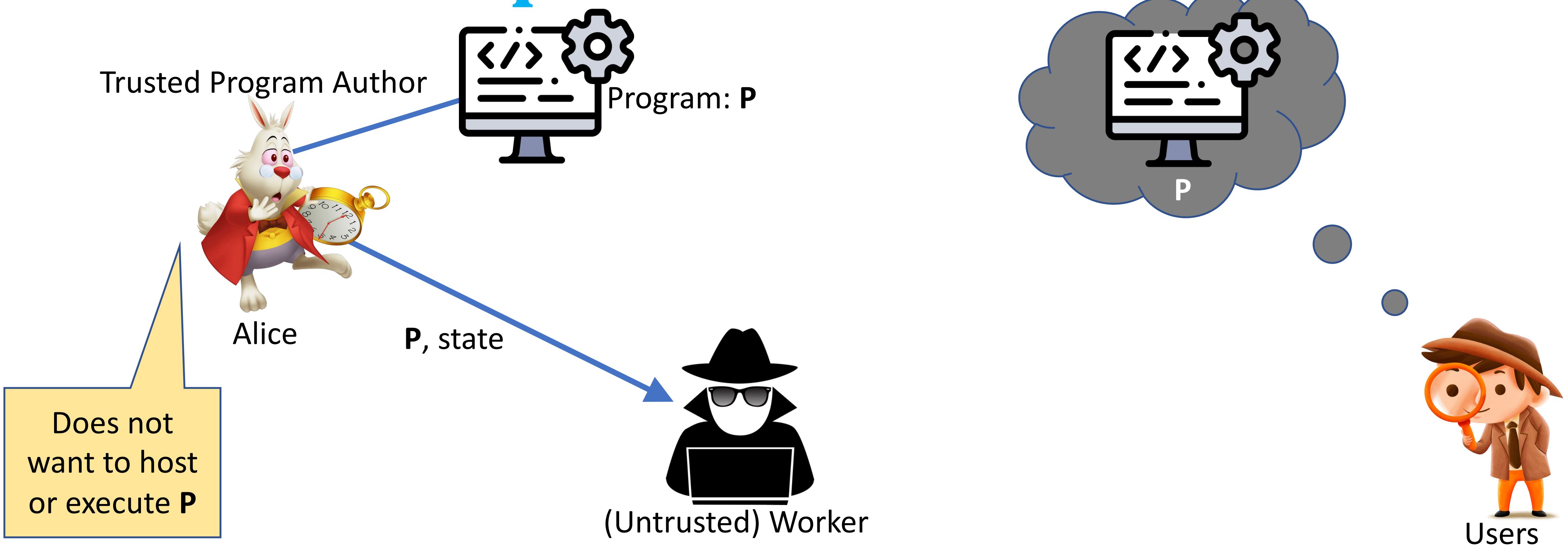
Problem Setup



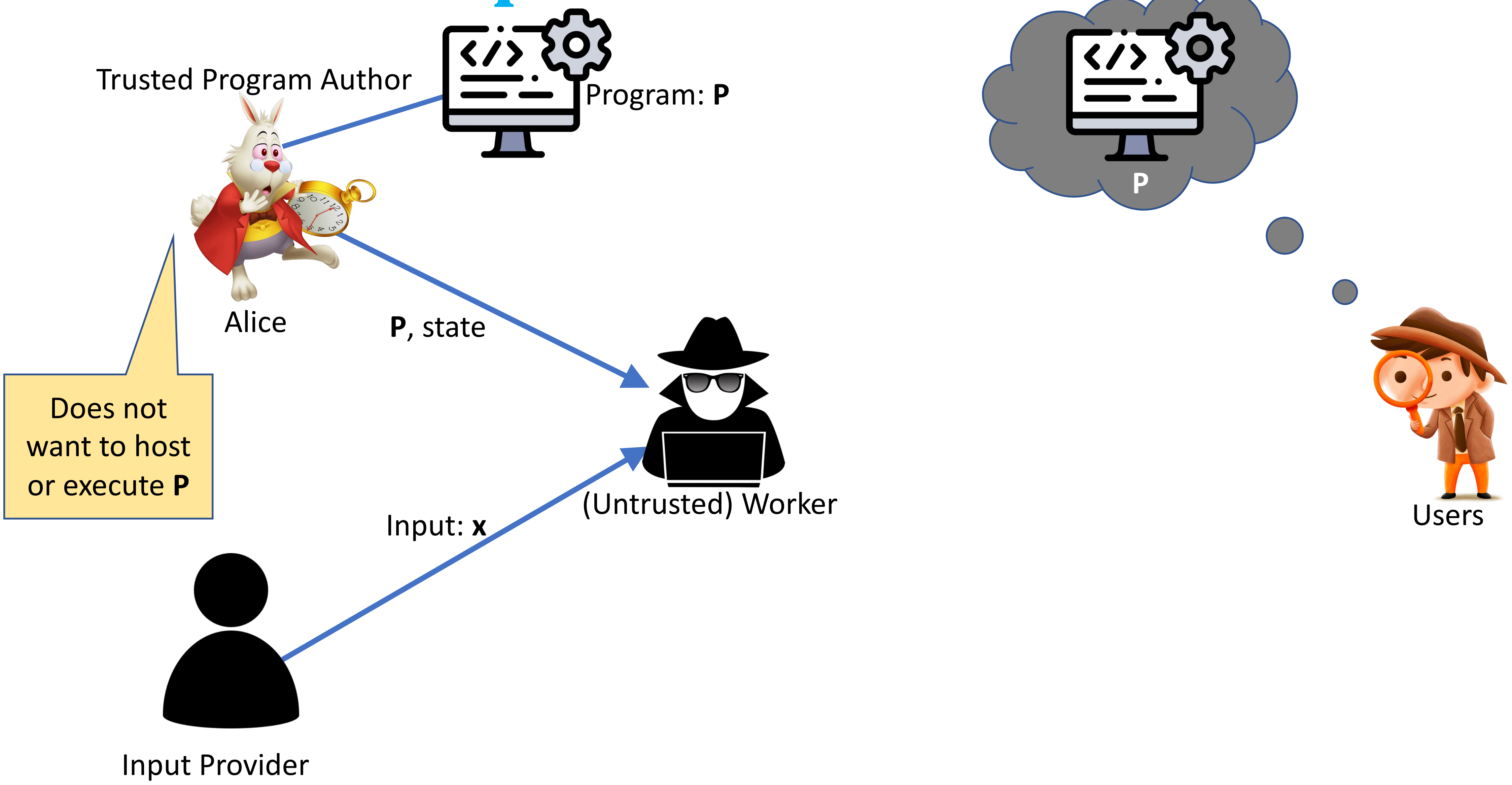
Problem Setup



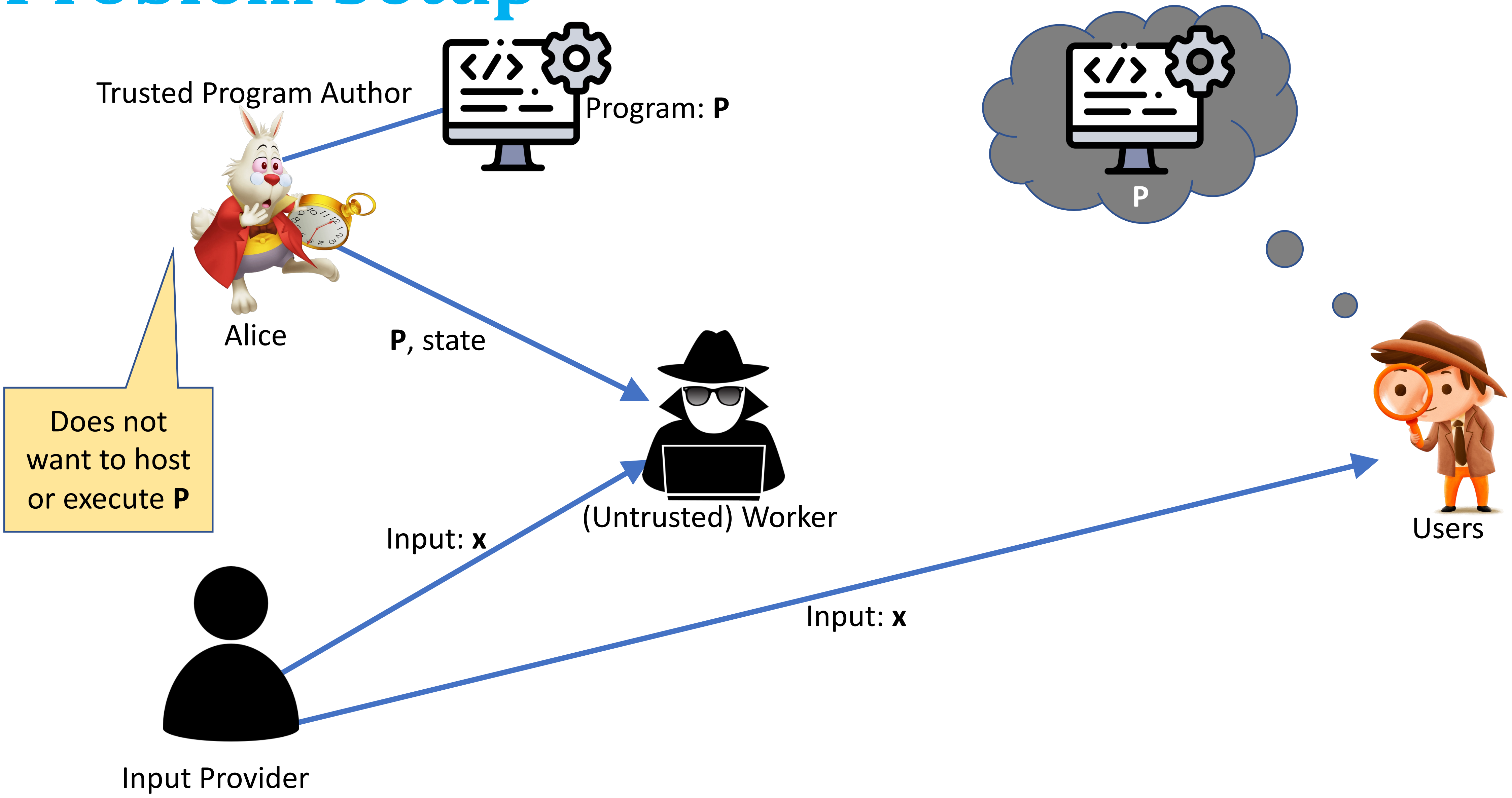
Problem Setup



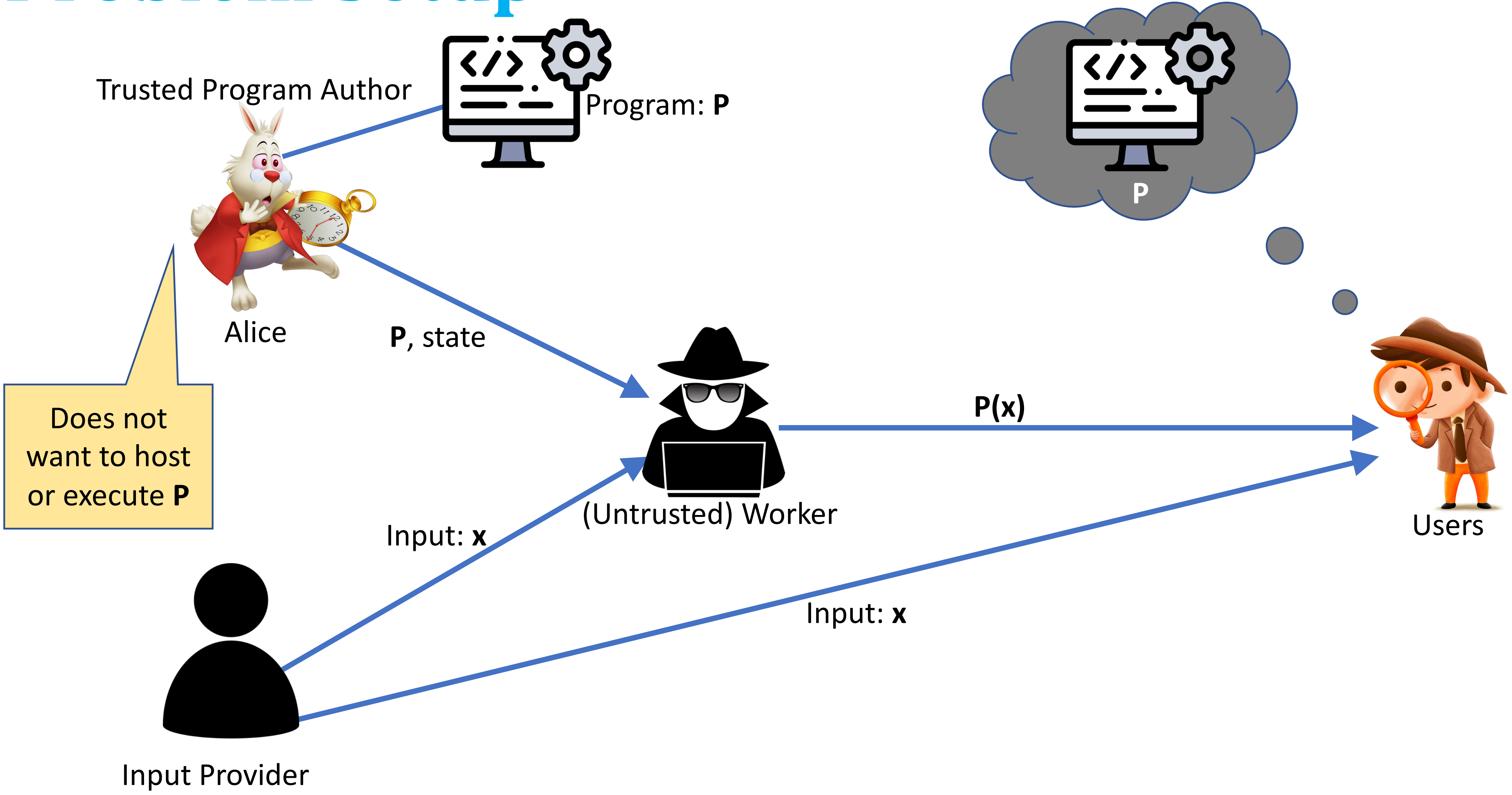
Problem Setup



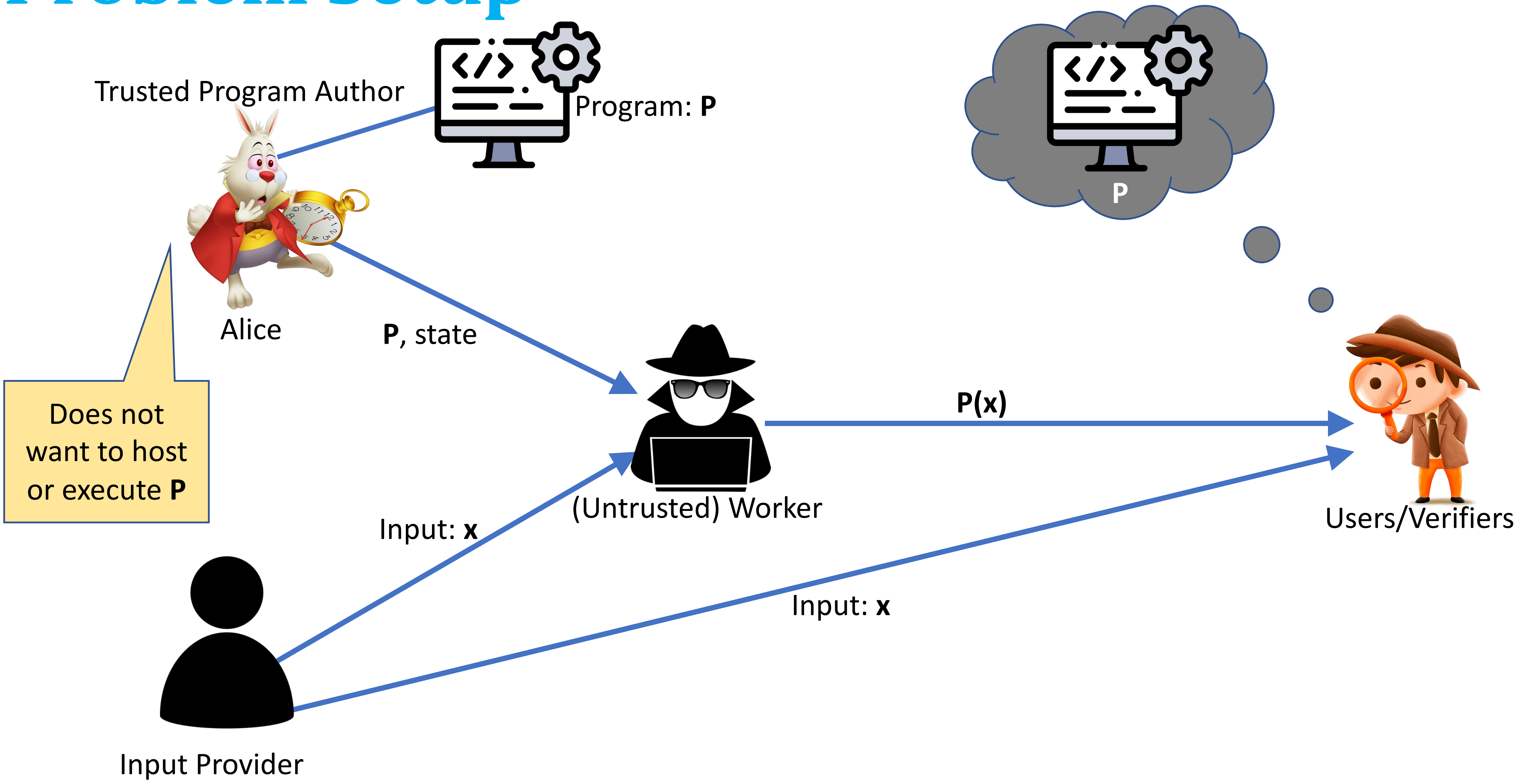
Problem Setup



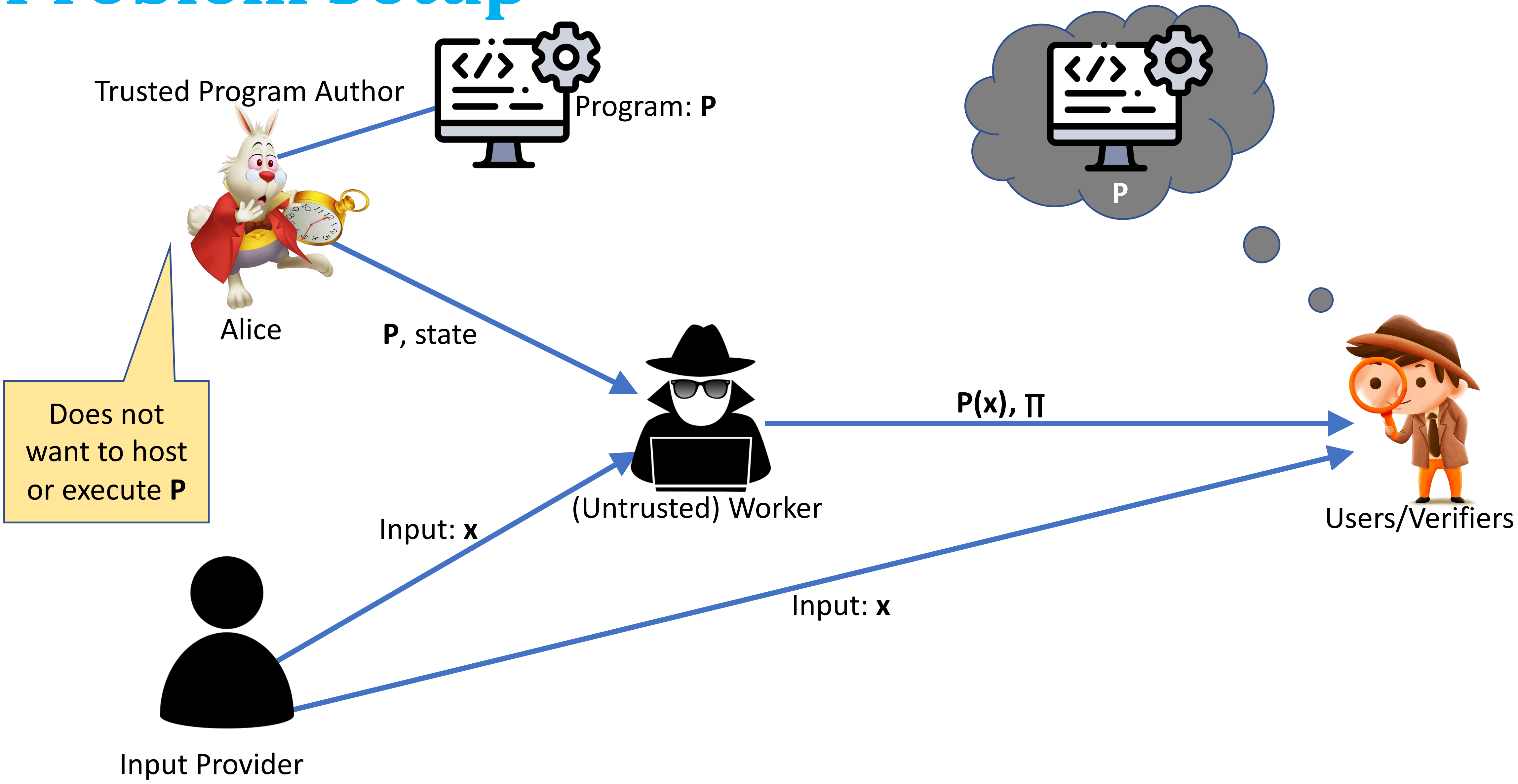
Problem Setup



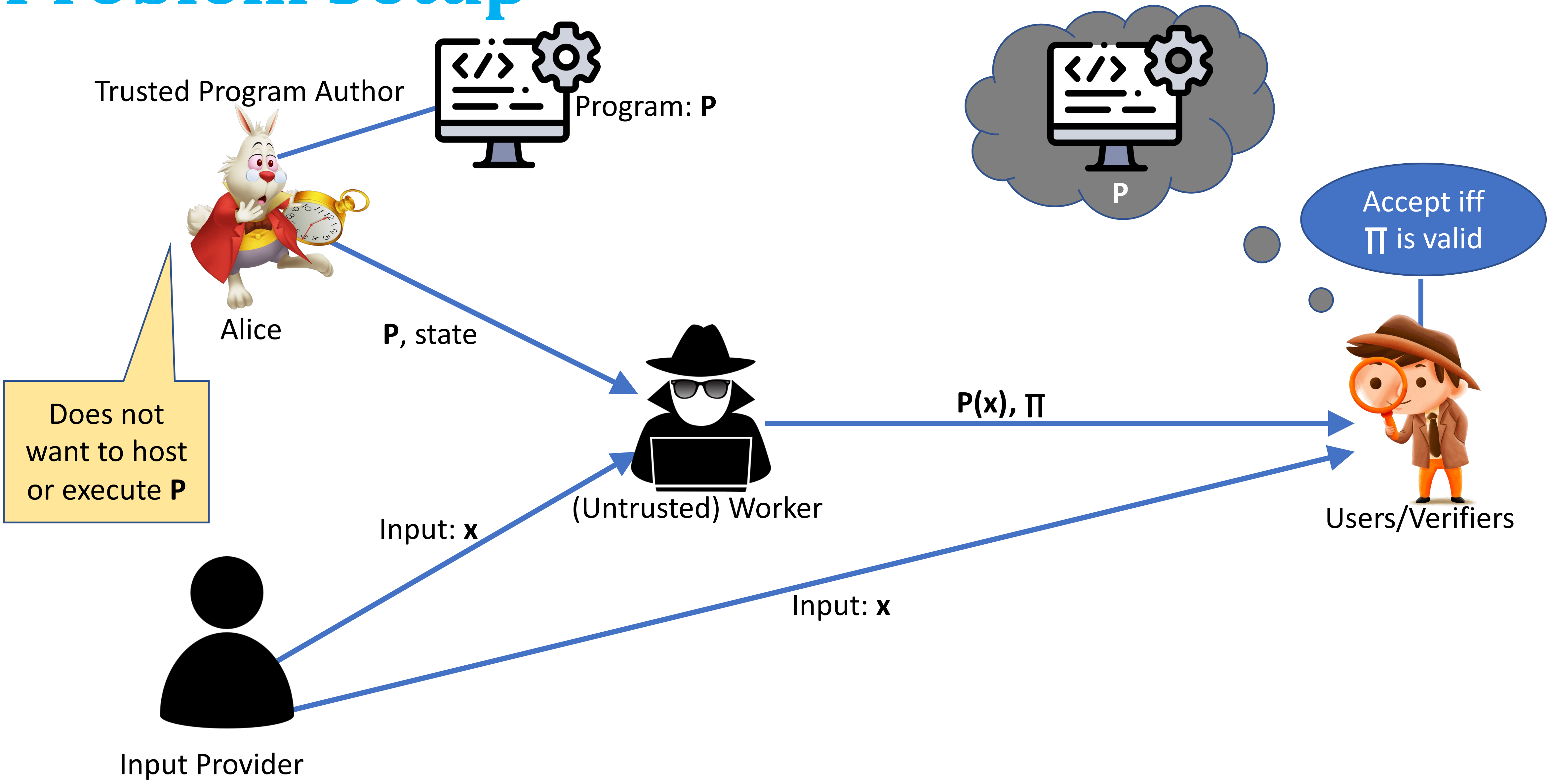
Problem Setup



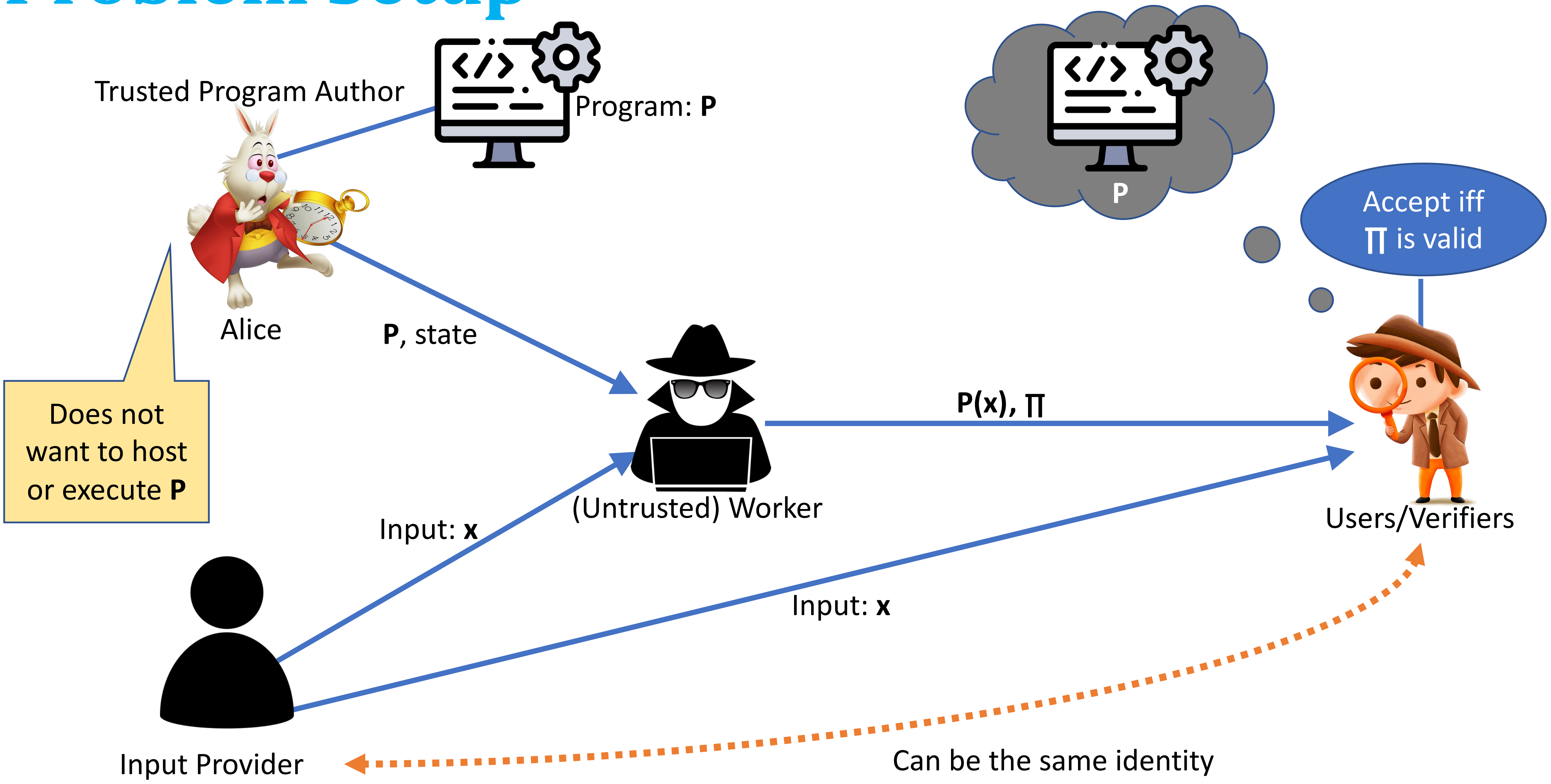
Problem Setup



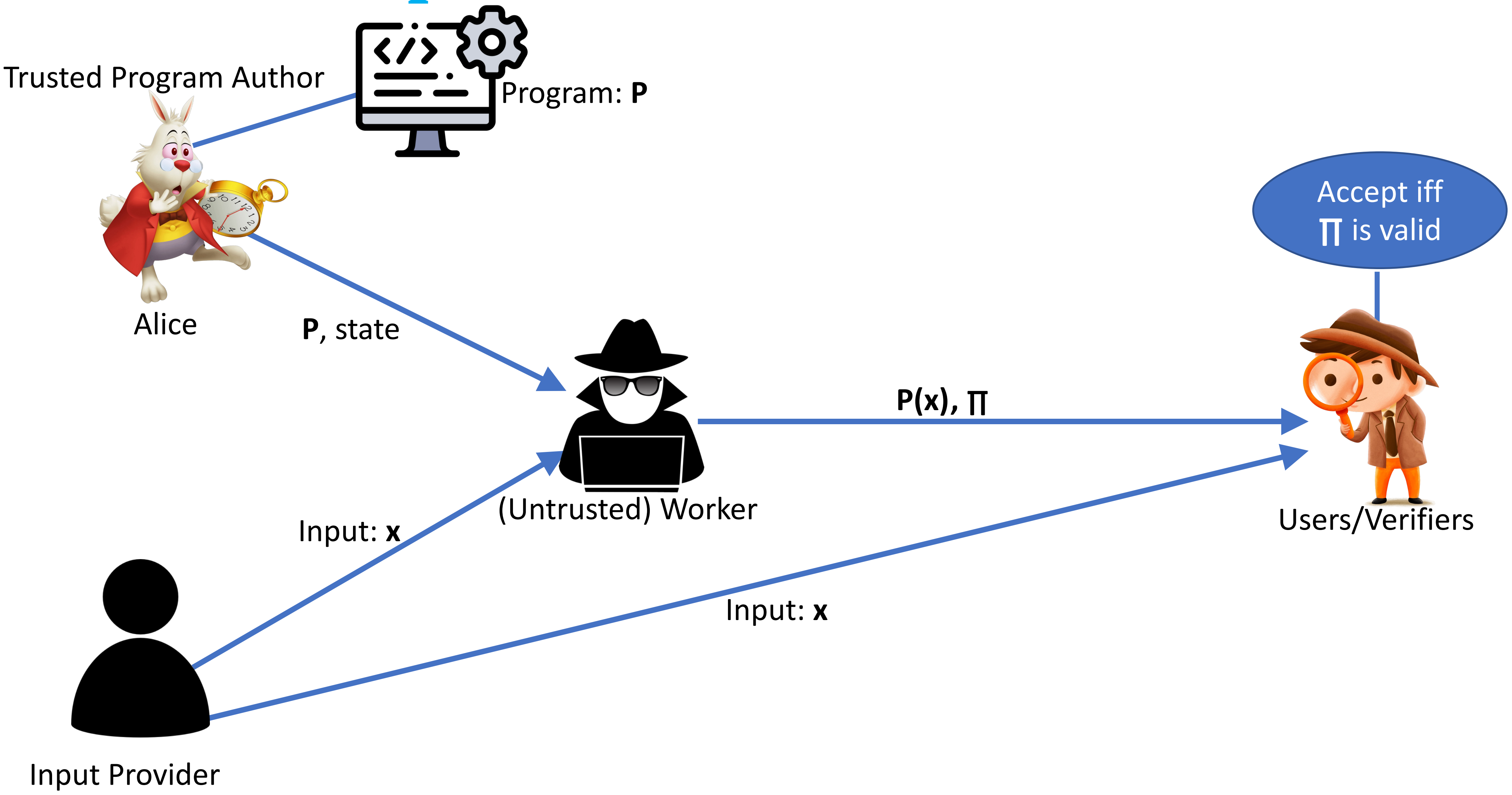
Problem Setup



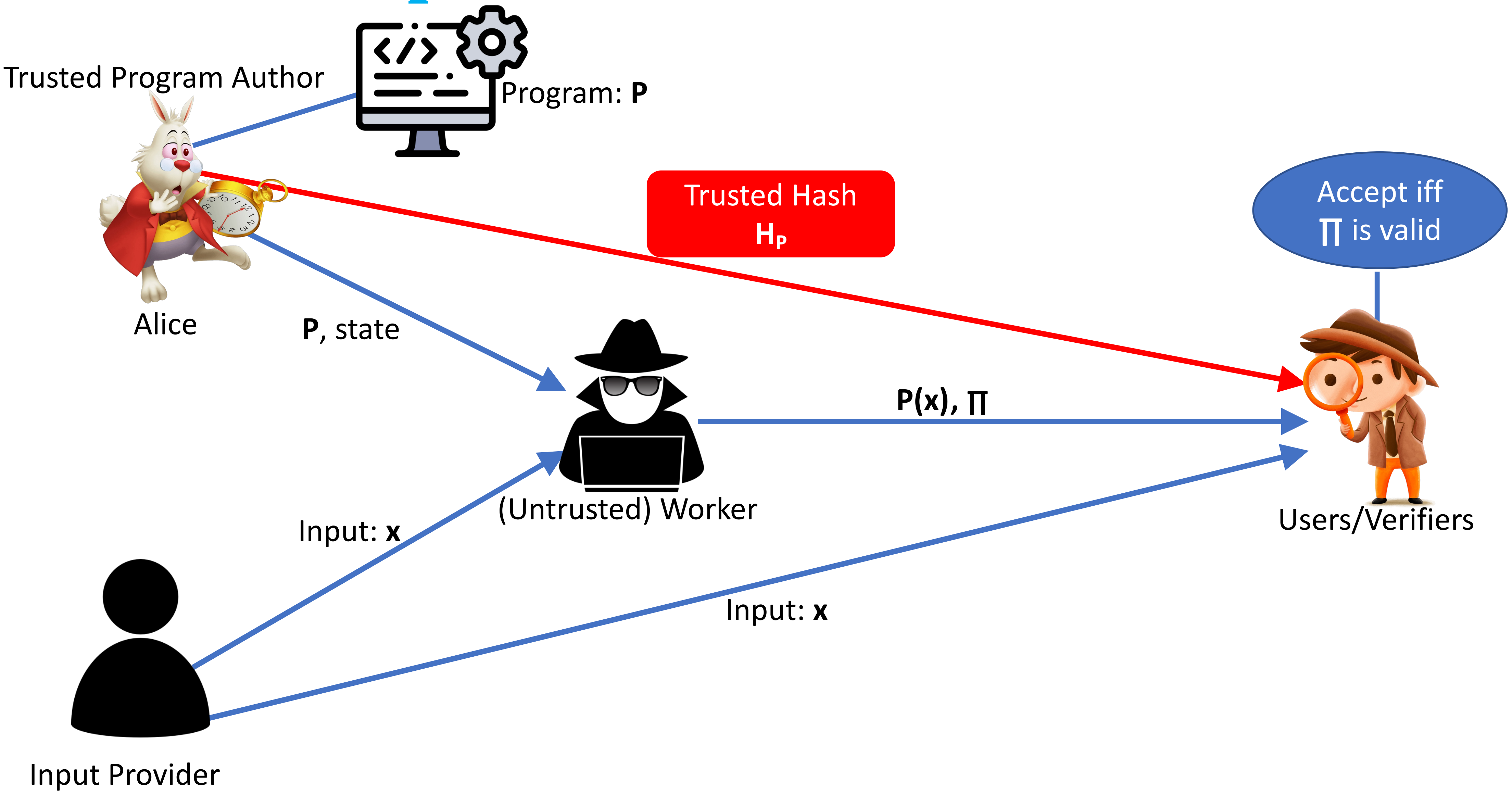
Problem Setup



Problem Setup

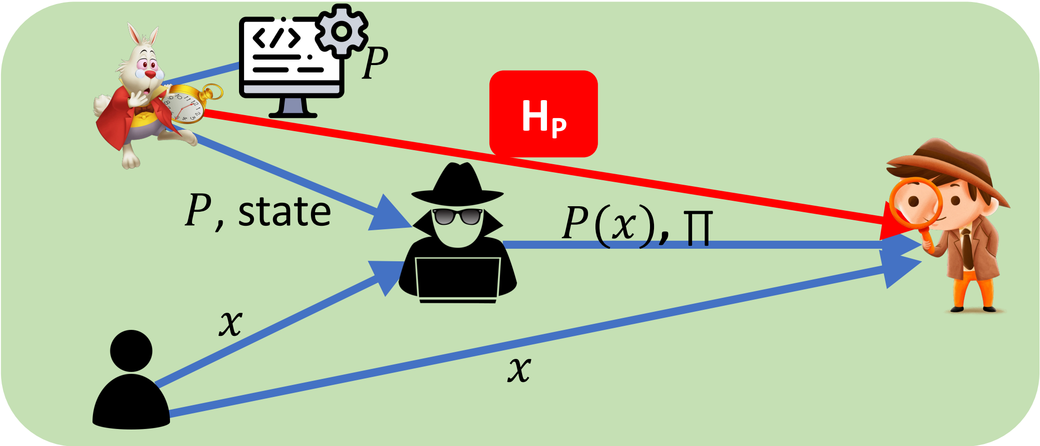


Problem Setup

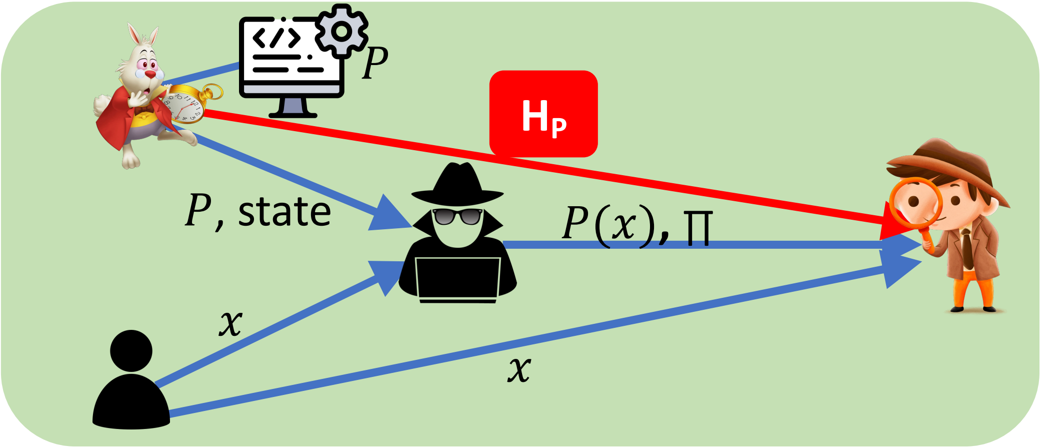


Goals

Goals



Goals



Non-Interactive Delegation

- Uni-directional Arrows
- No Back and Forth Communication

Public Verification

- Same proof ∏ for all verifiers

Public Delegation

- No pre-processing specific to P before delegating to Worker
- No dependence of H_P on input x

Succinctness

- $|\prod|$: poly $(\lambda, \log|P|, |x|)$
- Verifier run-time: poly $(\lambda, \log|P|, |x|)$
- Prover run-time: poly $(\lambda, |P|, |x|)$

Comparison with Prior Work

Comparison with Prior Work

Interactive Schemes:	• [Kil92,KRR13,BKP18]: Standard Assumptions, Interaction with Worker

Comparison with Prior Work

Interactive Schemes:	• [Kil92,KRR13,BKP18]: Standard Assumptions, Interaction with Worker	
Non-Interactive Schemes:	**Private Verification/Delegation**	**Public Delegation + Public Verification**
	• [KP16]: LWE (Private Verification) • [KRR13]: Subexp secure FHE (Private Verification) • [GGP10]: FHE (Private Delegation)	• [Mic00,BCC+17,Gro10,PR17]: Random Oracle/Non Standard Knowledge Assumptions • [KPY19]: Standard Assumptions but verifier knows **P**

Why has constructing a protocol that caters to the fully non-interactive setting which we have defined been so elusive?

Why has constructing a protocol that caters to the fully non-interactive setting which we have defined been so elusive?

Verifier does not know P. From Verifier's perspective, P is like an NP Witness

The SNARGS for NP barrier

Why has constructing a protocol that caters to the fully non-interactive setting which we have defined been so elusive?

Verifier does not know P. From Verifier's perspective, P is like an NP Witness

Seems intricately related to finding a solution to the "SNARG for NP" problem

The SNARGS for NP barrier

Why has constructing a protocol that caters to the fully non-interactive setting which we have defined been so elusive?

Verifier does not know P. From Verifier's perspective, P is like an NP Witness

Seems intricately related to finding a solution to the "SNARG for NP" problem

Only solutions known in the Random Oracle Model or using non-standard knowledge assumptions

The SNARGS for NP barrier

Why has constructing a protocol that caters to the fully non-interactive setting which we have defined been so elusive?

Verifier does not know P. From Verifier's perspective, P is like an NP Witness

Seems intricately related to finding a solution to the "SNARG for NP" problem

Only solutions known in the Random Oracle Model or using non-standard knowledge assumptions

Solutions from standard assumptions has been open for over a decade

The SNARGS for NP barrier

Why has constructing a protocol that caters to the fully non-interactive setting which we have defined been so elusive?

Verifier does not know P. From Verifier's perspective, P is like an NP Witness

Seems intricately related to finding a solution to the "SNARG for NP" problem

Only solutions known in the Random Oracle Model or using non-standard knowledge assumptions

Solutions from standard assumptions has been open for over a decade

Very recent work [CJJ21] comes the closest by achieving "SNARG for P from LWE"

Comparison with Prior Work

Interactive Schemes:	• [Kil92,KRR13,BKP18]: Standard Assumptions, Interaction with Worker	
	Private Verification/Delegation	**Public Delegation + Public Verification**
Non-Interactive Schemes:	• [KP16]: LWE (Private Verification) • [KRR13]: Subexp secure FHE (Private Verification) • [GGP10]: FHE (Private Delegation)	• [Mic00,BCC+17,Gro10,PR17]: Random Oracle/Non Standard Knowledge Assumptions • [KPY19]: Standard Assumptions but verifier knows **P**

Comparison with Prior Work

Interactive Schemes:	• [Kil92,KRR13,BKP18]: Standard Assumptions, Interaction with Worker	
Non-Interactive Schemes:	**Private Verification/Delegation**	**Public Delegation + Public Verification**
	• [KP16]: LW • [KRR13]: S (Private Ve...) • [GGP10]: FHE (Private Delegation)	...C+17,Gro10,PR17]: Random ...n Standard Knowledge ...ons ...tandard Assumptions but verifier knows **P**

Our Work: Fully Non-Interactive Public Delegation + Public Verification from Standard Assumptions and Verifier does not know **P**

Comparison with Prior Work

Interactive Schemes:	• [Kil92,KRR13,BKP18]: Standard Assumptions, Interaction with Worker	
Non-Interactive Schemes:	**Private Verification/Delegation**	**Public Delegation + Public Verification**
	• [KP16]: LWE (Private Verification) • [KRR13]: Subexp secure FHE (Private Verification) • [GGP10]: FHE (Private Delegation)	• [Mic00,BCC+17,Gro10,PR17]: Random Oracle/Non Standard Knowledge Assumptions • [KPY19]: Standard Assumptions but verifier knows **P** Our Work: Standard Assumptions and Verifier does not know **P**

Our Contribution

Our Contribution

Can bypass the need of "SNARG for NP"

Our Contribution

Can bypass the need of "SNARG for NP"

- Ideas from the "SNARG for P" can indeed be used
- Suffices for Alice to communicate a tiny amount of information about P to the Verifier
- Size of trusted $\mathrm{H_P} = \mathrm{poly}\log|\mathrm{P}|$

Our Contribution

Can bypass the need of "SNARG for NP"

- Ideas from the "SNARG for P" can indeed be used
- Suffices for Alice to communicate a tiny amount of information about P to the Verifier
- Size of trusted $\mathrm{H_P} = \mathrm{poly}\log|\mathrm{P}|$

Main Theorem

Assuming the hardness of the LWE problem, there exists a construction for publicly verifiable non-interactive succinct delegation for committed programs with CRS size, proof size and verifier time poly(λ, log |P|, |x|) and prover run time being poly(λ, |P|, |x|).

Publicly Verifiable Non-Interactive Succinct Delegation

Such a delegation scheme in the CRS model involves the following PPT algorithms:

- Setup(1^λ): Randomized setup algorithm that outputs crs
- ProgAuth($1^\lambda, crs$): Randomized algorithm that outputs $P, state, H_P$
- Prover(crs, $P, state, H_P, x$): Deterministic algorithm that outputs a value y and proof $\prod$
- Verifier(crs, $H_P, x, y,\prod$): Deterministic verifier which either accepts or rejects

Publicly Verifiable Non-Interactive Succinct Delegation

Completeness

For all PPT ProgAuth, if crs is appropriately generated and Prover runs honestly then,

$$\Pr[V(crs, H_P, x, y, \prod) = 1 \wedge P(x) = y] = 1$$

Publicly Verifiable Non-Interactive Succinct Delegation

Completeness

For all PPT ProgAuth, if crs is appropriately generated and Prover runs honestly then,

$$\Pr[V(crs, H_P, x, y, \prod) = 1 \wedge P(x) = y] = 1$$

Efficiency

Setup runs in time $poly(\lambda)$

Proof Size: $poly(\lambda, log\ |P|, |x|)$

W runs in time $poly(\lambda, |P|, |x|)$

V runs in time $poly(\lambda, log\ |P|, |x|)$

Publicly Verifiable Non-Interactive Succinct Delegation

Completeness

For all PPT ProgAuth, if crs is appropriately generated and Prover runs honestly then,

$$\Pr[V(crs, H_P, x, y, \prod) = 1 \wedge P(x) = y] = 1$$

Efficiency

Setup runs in time $poly(\lambda)$

Proof Size: $poly(\lambda, log\ |P|, |x|)$

W runs in time $poly(\lambda, |P|, |x|)$

V runs in time $poly(\lambda, log\ |P|, |x|)$

Soundness

For all PPT adversaries (A_1, A_2) and all ProgAuth, if crs is appropriately generated and $(\mathrm{x}, \mathrm{aux}) \leftarrow A_1(crs)$, $(\mathrm{y}, \prod) \leftarrow A_2(crs, P, H_P, x, aux)$, then

$$\Pr[V(crs, H_P, x, y, \prod) = 1 \wedge P(x) \neq y] \leq negl(\lambda)$$

Informal Technical Overview

Informal Technical Overview

- Consider the execution of P as a deterministic Turing Machine computation on input x.

Informal Technical Overview

- Consider the execution of P as a deterministic Turing Machine computation on input x.

- P accepts x if $P(x) = y$ within T steps .

Informal Technical Overview

- Consider the execution of P as a deterministic Turing Machine computation on input x.

- P accepts x if $P(x) = y$ within T steps .

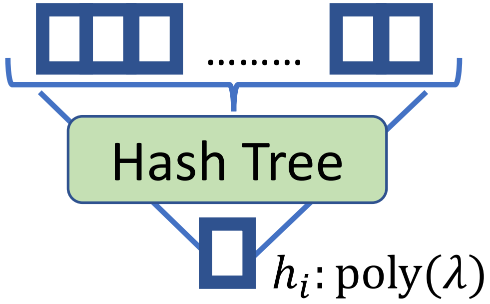
- Let $h_0, h_1, h_2 \ldots h_T$ be succinct encodings of the Turing Machine state and tape content during these T intermediate steps.

Informal Technical Overview

- Consider the execution of P as a deterministic Turing Machine computation on input x.

- P accepts x if $P(x) = y$ within T steps .

- Let $h_0, h_1, h_2 \ldots h_T$ be succinct encodings of the Turing Machine state and tape content during these T intermediate steps.

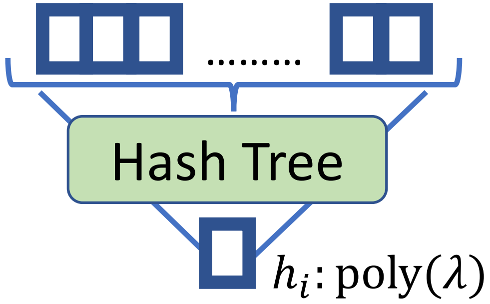


Informal Technical Overview

- Consider the execution of P as a deterministic Turing Machine computation on input x.

- P accepts x if $P(x) = y$ within T steps .

- Let $h_0, h_1,\ h_2 \ldots h_T$ be succinct encodings of the Turing Machine state and tape content during these T intermediate steps.



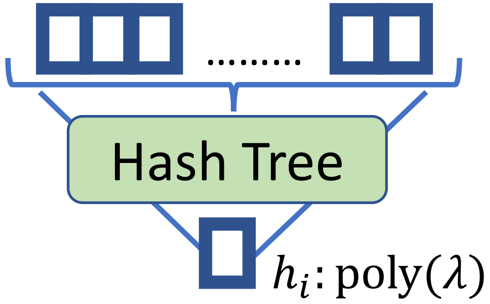
- If h_0 encodes the correct input x and h_T encodes an accepting state, then by CJJ21, we can construct a SNARG to prove that the computation was performed honestly.

Informal Technical Overview

- Consider the execution of P as a deterministic Turing Machine computation on input x.

- P accepts x if $P(x) = y$ within T steps .

- Let $h_0, h_1, h_2 \ldots h_T$ be succinct encodings of the Turing Machine state and tape content during these T intermediate steps.



- If h_0 encodes the correct input x and h_T encodes an accepting state, then by CJJ21, we can construct a SNARG to prove that the computation was performed honestly.

! We do not know P

Informal Technical Overview

An alternate way to interpret computation of P on x:

- Consider a Universal Turing Machine TM which takes (P, x, y) as input
- TM accepts in $T' = O(|P| \log |P|)$ steps if $P(x) = y$

Informal Technical Overview

An alternate way to interpret computation of P on x:

- Consider a Universal Turing Machine TM which takes (P, x, y) as input
- TM accepts in $T' = O(|P| \log |P|)$ steps if $P(x) = y$

- Consider TM has three tapes: 3 read only input tapes for P, x and y, and one work tape.

Informal Technical Overview

An alternate way to interpret computation of P on x:

- Consider a Universal Turing Machine TM which takes (P, x, y) as input
- TM accepts in $T' = O(|P| \log |P|)$ steps if $P(x) = y$

- Consider TM has three tapes: 3 read only input tapes for P, x and y, and one work tape.

- Let $h_0, h_1, h_2 \dots h_{T'}$ be the turing machine state and tape content during the T' intermediate steps.

Informal Technical Overview

An alternate way to interpret computation of P on x:

- Consider a Universal Turing Machine TM which takes (P, x, y) as input
- TM accepts in $T' = O(|P| \log |P|)$ steps if $P(x) = y$

- Consider TM has three tapes: 3 read only input tapes for P, x and y, and one work tape.

- Let $h_0, h_1,\ h_2 \dots h_{T'}$ be the turing machine state and tape content during the T' intermediate steps.

- If h_0 encodes the correct inputs (P, x, y) and $h_{T'}$ encodes an accepting state, then by CJJ21, we can construct SNARG to prove that the computation was performed honestly.

Informal Technical Overview

An alternate way to interpret computation of P on x:

- Consider a Universal Turing Machine TM which takes (P, x, y) as input
- TM accepts in $T' = O(|P| \log |P|)$ steps if $P(x) = y$

- Consider TM has three tapes: 3 read only input tapes for P, x and y, and one work tape.

- Let $h_0, h_1,\ h_2 \ldots h_{T'}$ be the turing machine state and tape content during the T' intermediate steps.

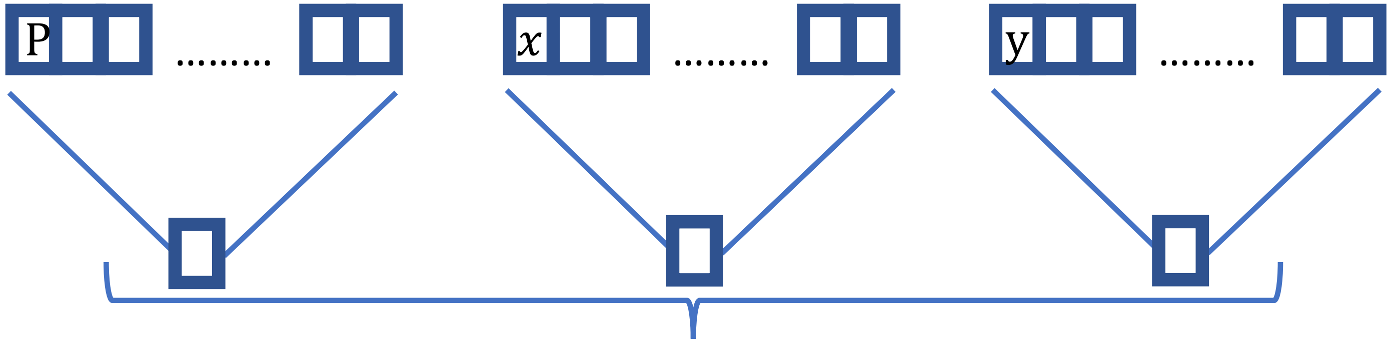
- If h_0 encodes the correct inputs (P, x, y) and $h_{T'}$ encodes an accepting state, then by CJJ21, we can construct SNARG to prove that the computation was performed honestly.

Informal Technical Overview

- Note h_0 contains succinct encodings of the inputs (P, x, y). Since (x, y) are publicly known, it can be easily verified if they have been correctly input to the TM.

Informal Technical Overview

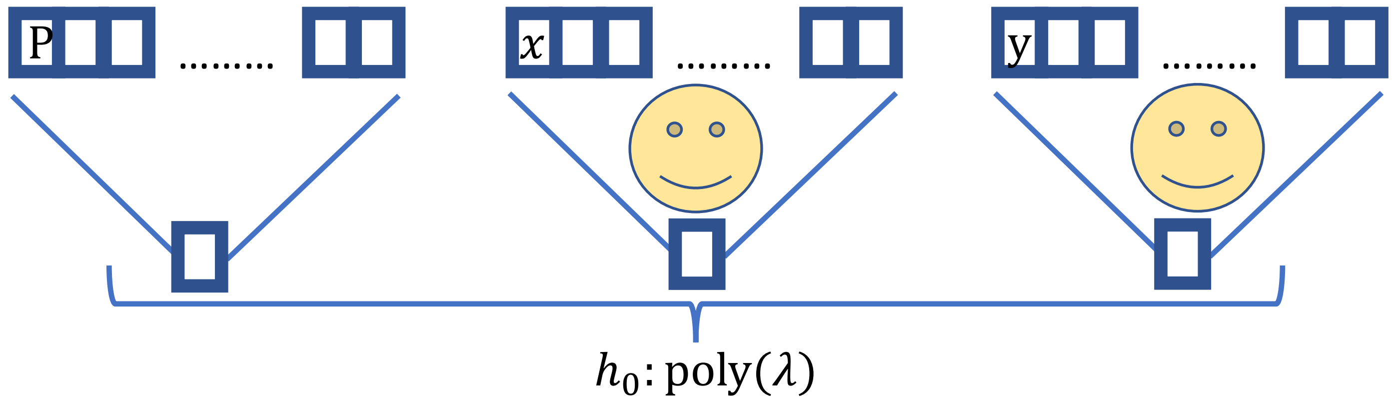
- Note h_0 contains succinct encodings of the inputs (P, x, y). Since (x, y) are publicly known, it can be easily verified if they have been correctly input to the TM.



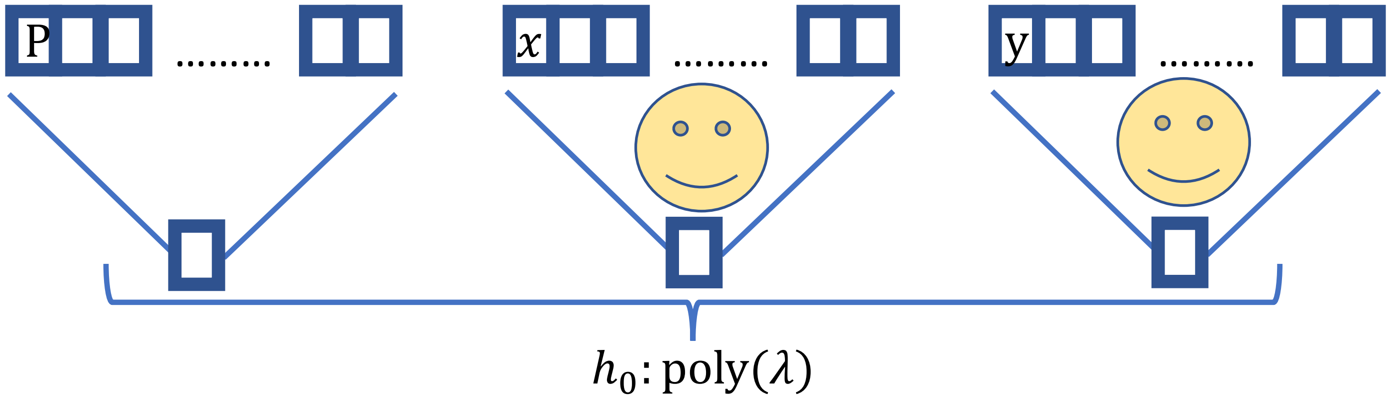
h_0: poly(λ)

Informal Technical Overview

- Note h_0 contains succinct encodings of the inputs (P, x, y). Since (x, y) are publicly known, it can be easily verified if they have been correctly input to the TM.

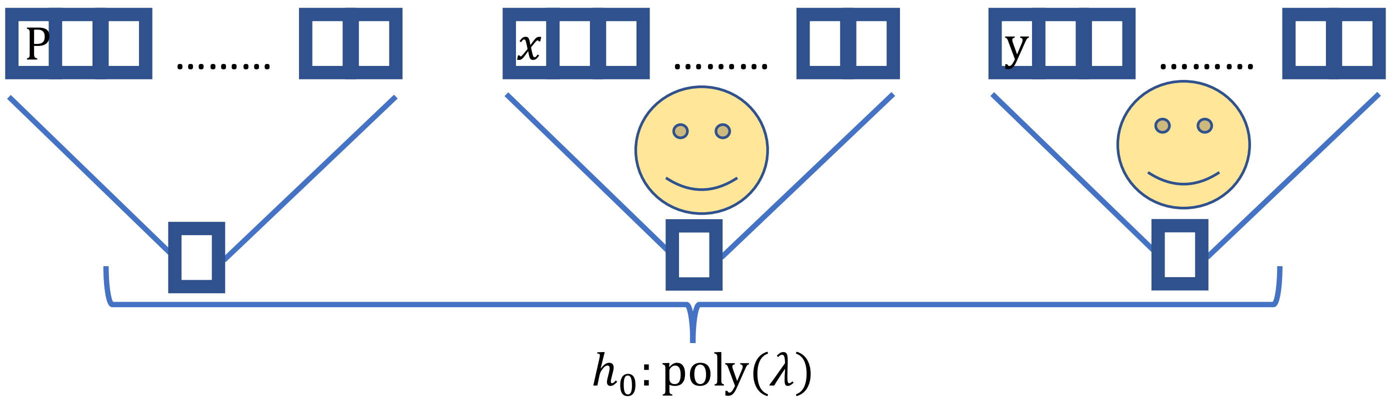


Construction: Brief Technical Details



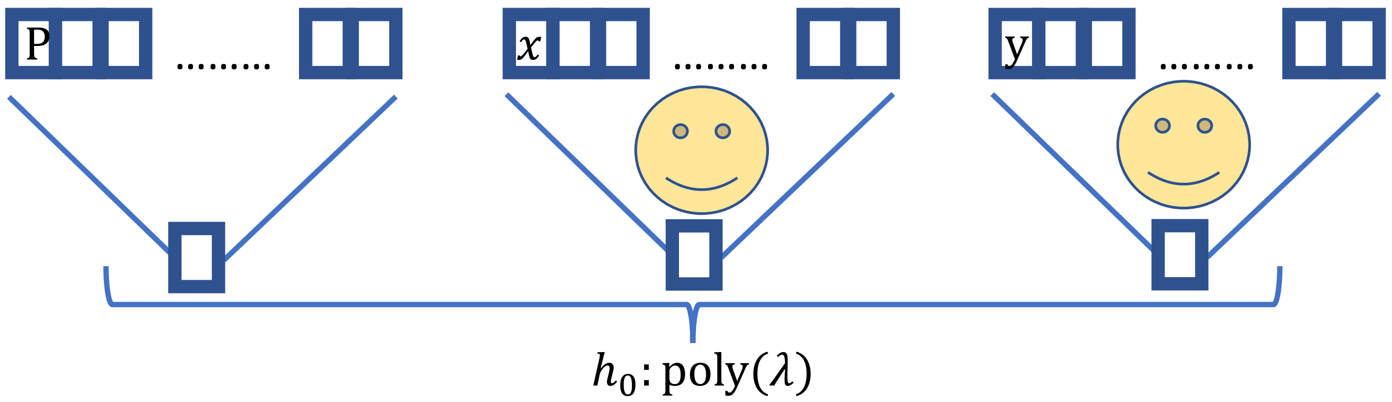
Construction: Brief Technical Details

Remember **H_P**?



Construction: Brief Technical Details

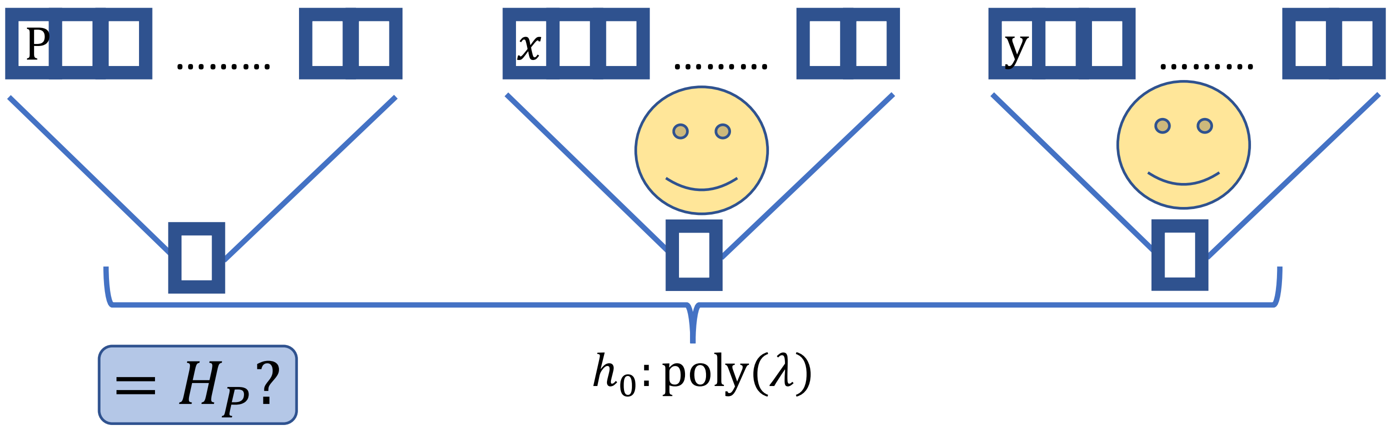
Remember H_P?



- Hardcode succinct H_P to the verification circuit.

Construction: Brief Technical Details

Remember
H_P?

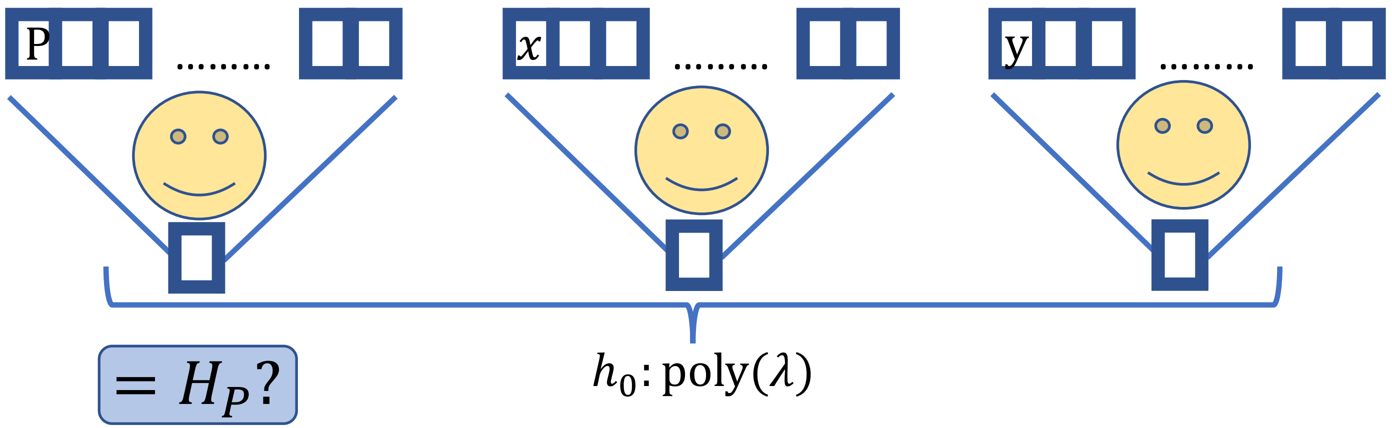


- Hardcode succinct H_P to the verification circuit.

- The verifier can now test if the encoding corresponding to P send by the Prover indeed matches the honestly generated H_P.

Construction: Brief Technical Details

Remember H_P?

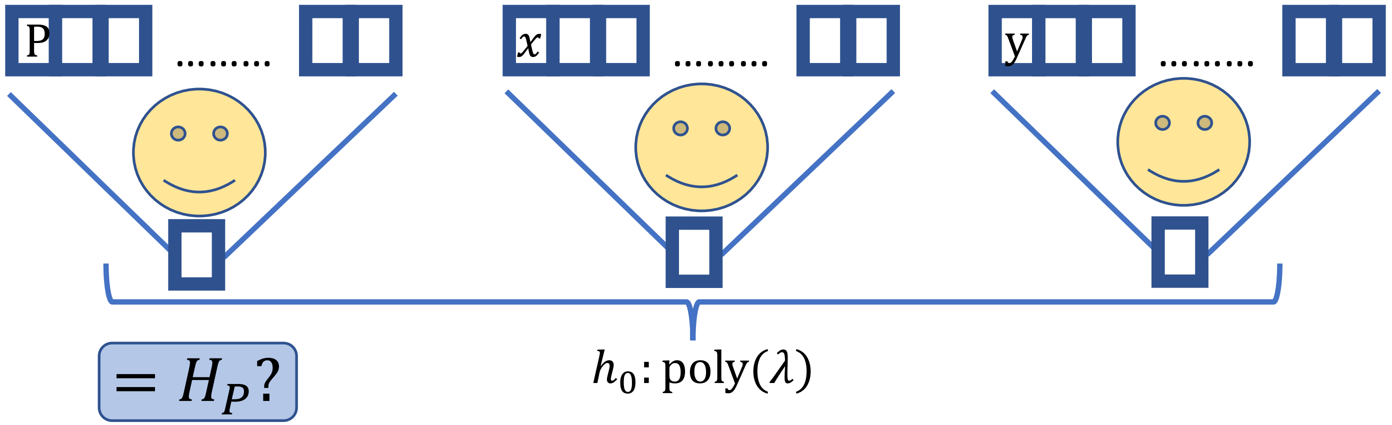


- Hardcode succinct H_P to the verification circuit.

- The verifier can now test if the encoding corresponding to P send by the Prover indeed matches the honestly generated H_P.

Construction: Brief Technical Details

Remember
H_P?

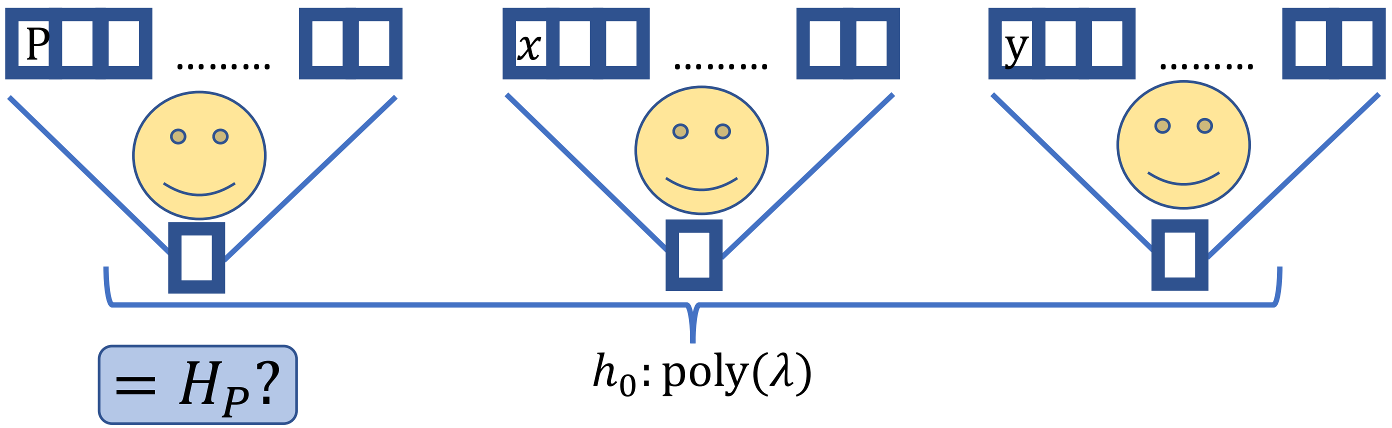


- Hardcode succinct H_P to the verification circuit.

- The verifier can now test if the encoding corresponding to P send by the Prover indeed matches the honestly generated H_P.

- If it does, then TM must have received the correct input P, otherwise the prover would break Collision Resistance of the hash function which generated H_P.

Construction: Brief Technical Details



- Hardcode succinct H_P to the verification circuit.

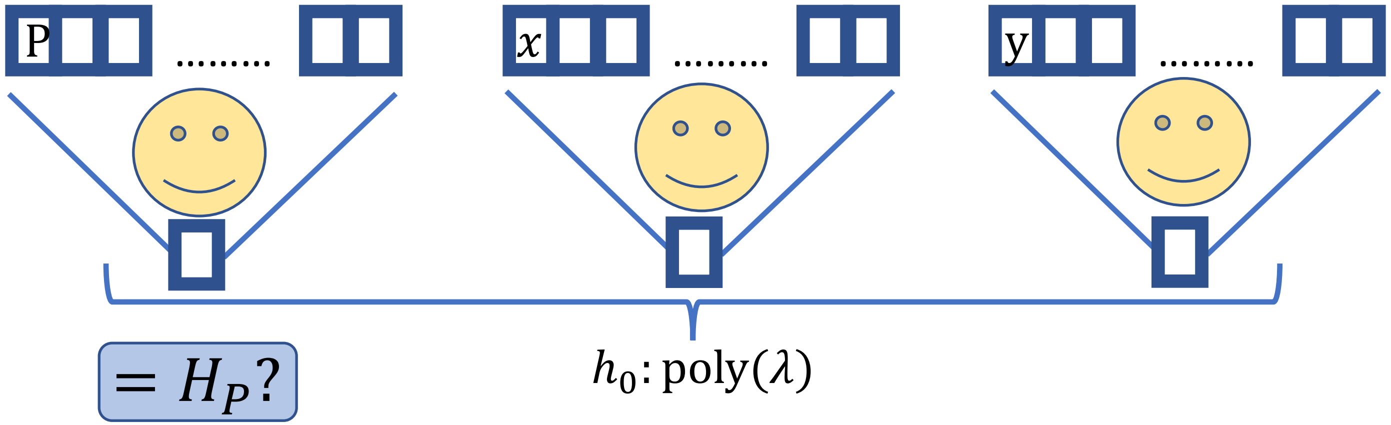
- The verifier can now test if the encoding corresponding to P send by the Prover indeed matches the honestly generated H_P.

- If it does, then TM must have received the correct input P, otherwise the prover would break Collision Resistance of the hash function which generated H_P.

- Now, we can use the techniques from CJJ21 to construct a SNARG

Construction: Brief Technical Details

Remember H_P?



- Hardcode succinct H_P to the verification circuit.

- The verifier can now test if the encoding corresponding to P send by the Prover indeed matches the honestly generated H_P.

- If it does, then TM must have received the correct input P, otherwise the prover would break Collision Resistance of the hash function which generated H_P.

- Now, we can use the techniques from CJJ21 to construct a **SEMI-TRUSTED** SNARG **For NP.**

Bonus Contribution: Zero Knowledge

Bonus Contribution: Zero Knowledge

- We give a generic transformation to convert any delegation scheme of this type to achieve zero-knowledge.

Bonus Contribution: Zero Knowledge

- We give a generic transformation to convert any delegation scheme of this type to achieve zero-knowledge.

- The program author can send an **Extractable Statistically Binding Commitment** to H_P instead of sending it out in the open.

Bonus Contribution: Zero Knowledge

- We give a generic transformation to convert any delegation scheme of this type to achieve zero-knowledge.

- The program author can send an **Extractable Statistically Binding Commitment** to H_P instead of sending it out in the open.

Get ZK by a standard CRS appending trick and a NIZK/NIWI proof in a modular fashion

Bonus Contribution: Zero Knowledge

- We give a generic transformation to convert any delegation scheme of this type to achieve zero-knowledge.

- The program author can send an **Extractable Statistically Binding Commitment** to H_P instead of sending it out in the open.

Get ZK by a standard CRS appending trick and a NIZK/NIWI proof in a modular fashion

- Add a commitment of 0 to the CRS.

Bonus Contribution: Zero Knowledge

- We give a generic transformation to convert any delegation scheme of this type to achieve zero-knowledge.

- The program author can send an **Extractable Statistically Binding Commitment** to H_P instead of sending it out in the open.

Get ZK by a standard CRS appending trick and a NIZK/NIWI proof in a modular fashion

- Add a commitment of 0 to the CRS.

- Instead of sending the proof $\prod$ which we discussed before, the prover sends a NIZK/NIWI argument for the following NP statement:

  $\prod$ is a valid proof for the verification circuit **OR** the CRS contains a commitment to 1.

Bonus Contribution: Zero Knowledge

- We give a generic transformation to convert any delegation scheme of this type to achieve zero-knowledge.

- The program author can send an **Extractable Statistically Binding Commitment** to H_P instead of sending it out in the open.

Get ZK by a standard CRS appending trick and a NIZK/NIWI proof in a modular fashion

- Add a commitment of 0 to the CRS.

- Instead of sending the proof $\prod$ which we discussed before, the prover sends a NIZK/NIWI argument for the following NP statement:

  $\prod$ is a valid proof for the verification circuit **OR** the CRS contains a commitment to 1.

- Nothing changes in the real world. In the simulated world, we can switch the CRS to have a commitment of 1 and the NIZK/NIWI proof will not use $\prod$ at all.

THANK YOU!