Malleable Proof Systems and Applications

Melissa Chase (MSR Redmond)
Markulf Kohlweiss (MSR Cambridge)
Anna Lysyanskaya (Brown University)
Sarah Meiklejohn (UC San Diego)

Non-malleable cryptography

Twenty years ago, saw a strong emphasis on non-malleable cryptography [DDN91,S99,dCIO98,BS99,...]

Non-malleable cryptography

Twenty years ago, saw a strong emphasis on non-malleable cryptography [DDN91,S99,dCIO98,BS99,...]

Non-malleable cryptography

Twenty years ago, saw a strong emphasis on non-malleable cryptography [DDN91,S99,dCIO98,BS99,...]

Non-malleable cryptography

Twenty years ago, saw a strong emphasis on non-malleable cryptography [DDN91,S99,dCIO98,BS99,...]

balance: $100

Non-malleable cryptography

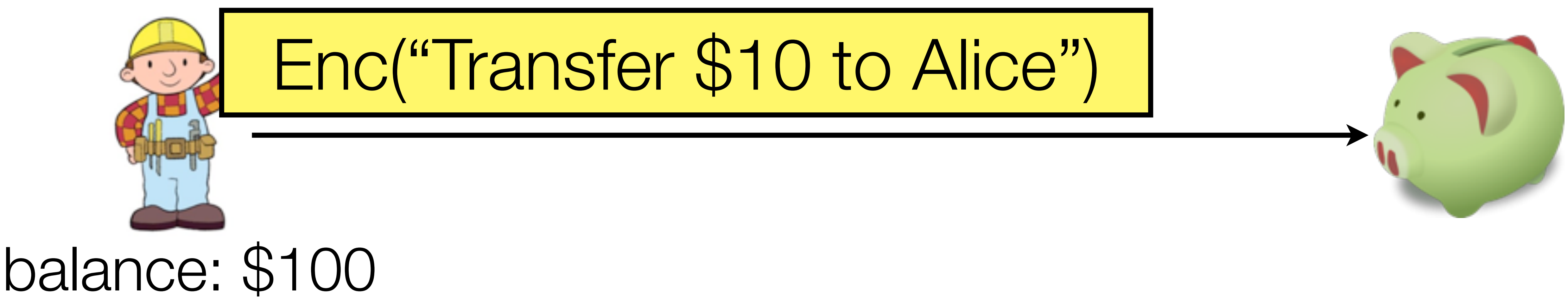
Twenty years ago, saw a strong emphasis on non-malleable cryptography [DDN91,S99,dCIO98,BS99,...]

Enc("Transfer $10 to Alice")

balance: $100

Non-malleable cryptography

Twenty years ago, saw a strong emphasis on non-malleable cryptography [DDN91,S99,dCIO98,BS99,...]



Non-malleable cryptography

Twenty years ago, saw a strong emphasis on non-malleable cryptography [DDN91,S99,dCIO98,BS99,...]

Enc("Transfer $10 to Alice")

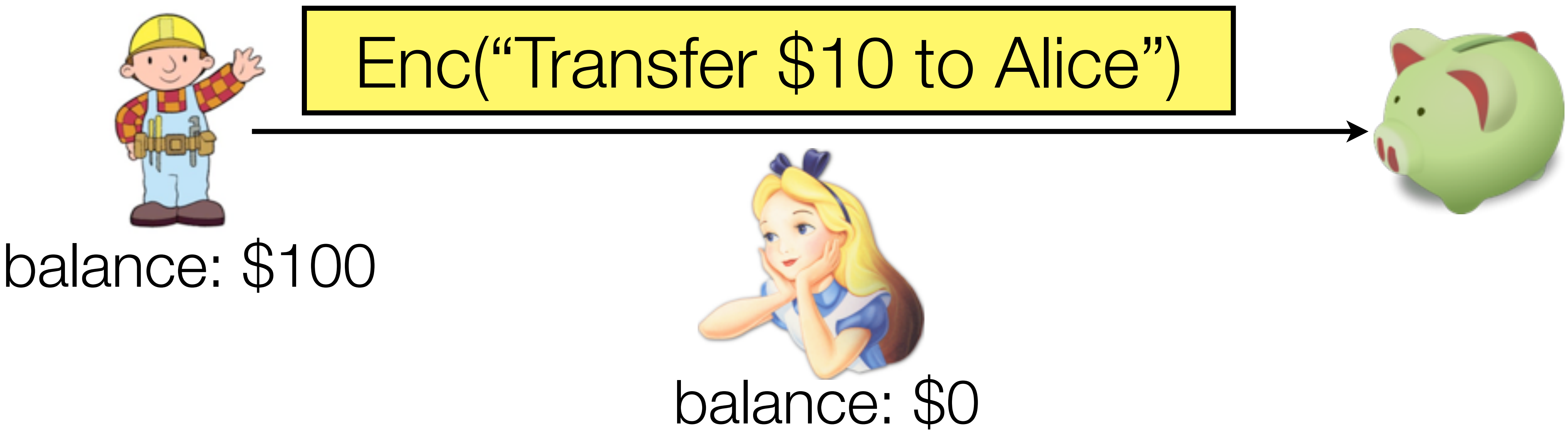
balance: $100

Non-malleable cryptography

Twenty years ago, saw a strong emphasis on non-malleable cryptography [DDN91,S99,dCIO98,BS99,...]

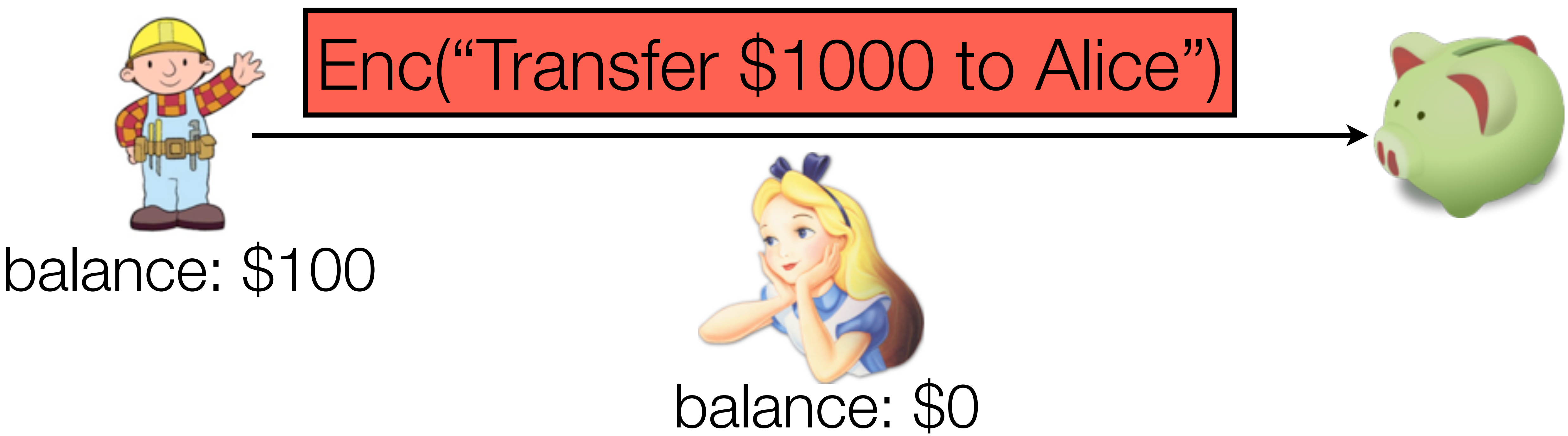
Non-malleable cryptography

Twenty years ago, saw a strong emphasis on non-malleable cryptography [DDN91,S99,dCIO98,BS99,...]



Non-malleable cryptography

Twenty years ago, saw a strong emphasis on non-malleable cryptography [DDN91,S99,dCIO98,BS99,...]

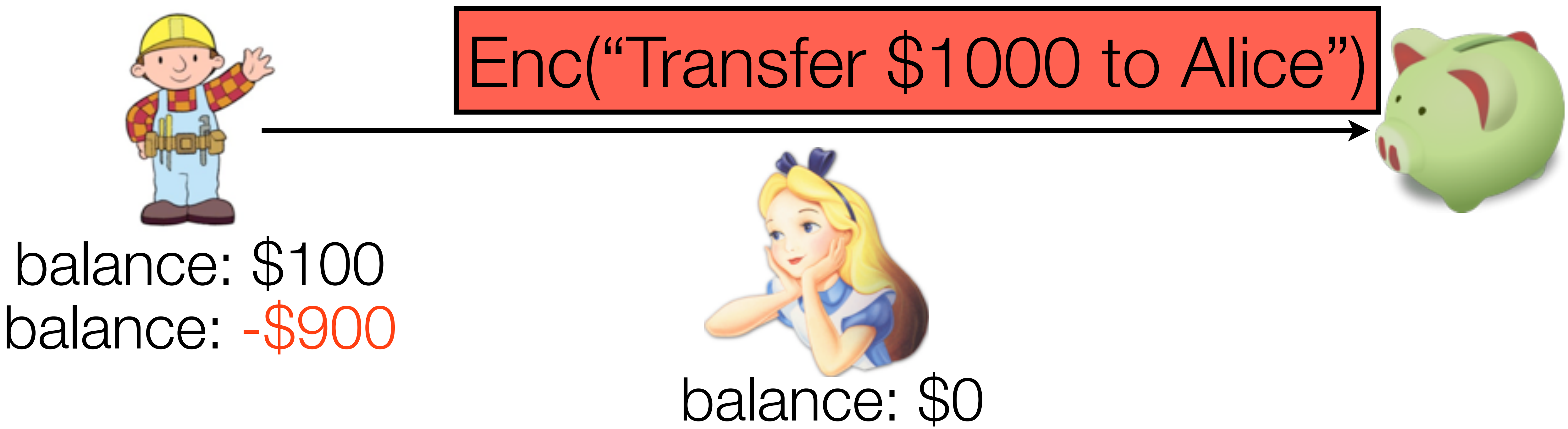


Non-malleable cryptography

Twenty years ago, saw a strong emphasis on non-malleable cryptography [DDN91,S99,dCIO98,BS99,...]

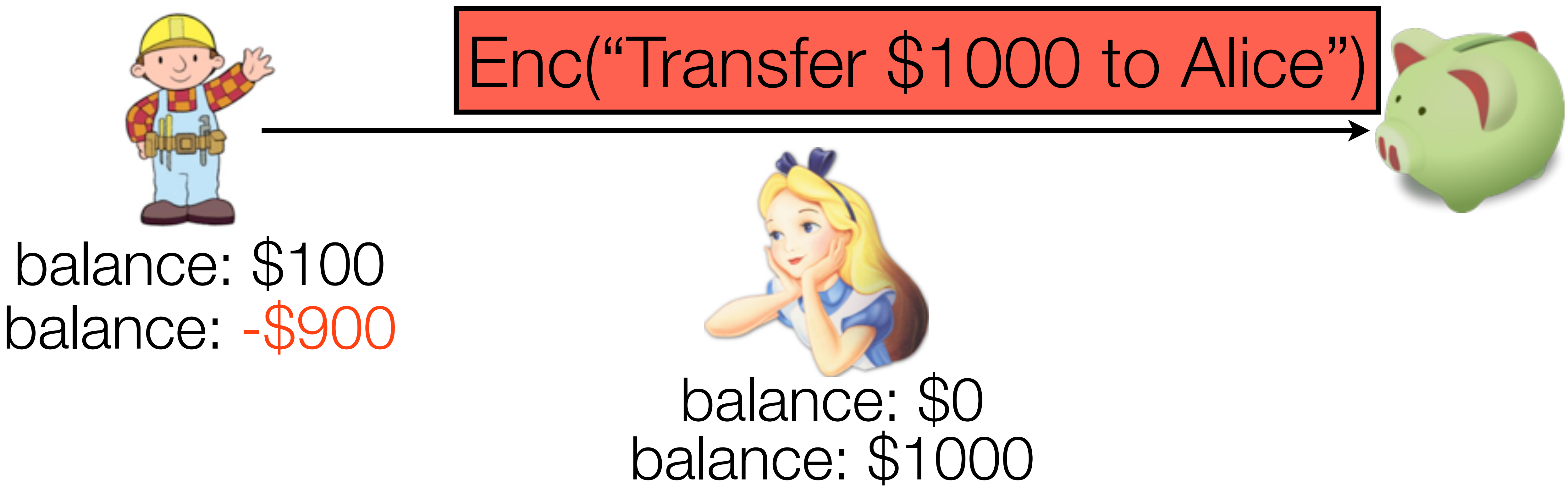
Non-malleable cryptography

Twenty years ago, saw a strong emphasis on non-malleable cryptography [DDN91,S99,dCIO98,BS99,...]



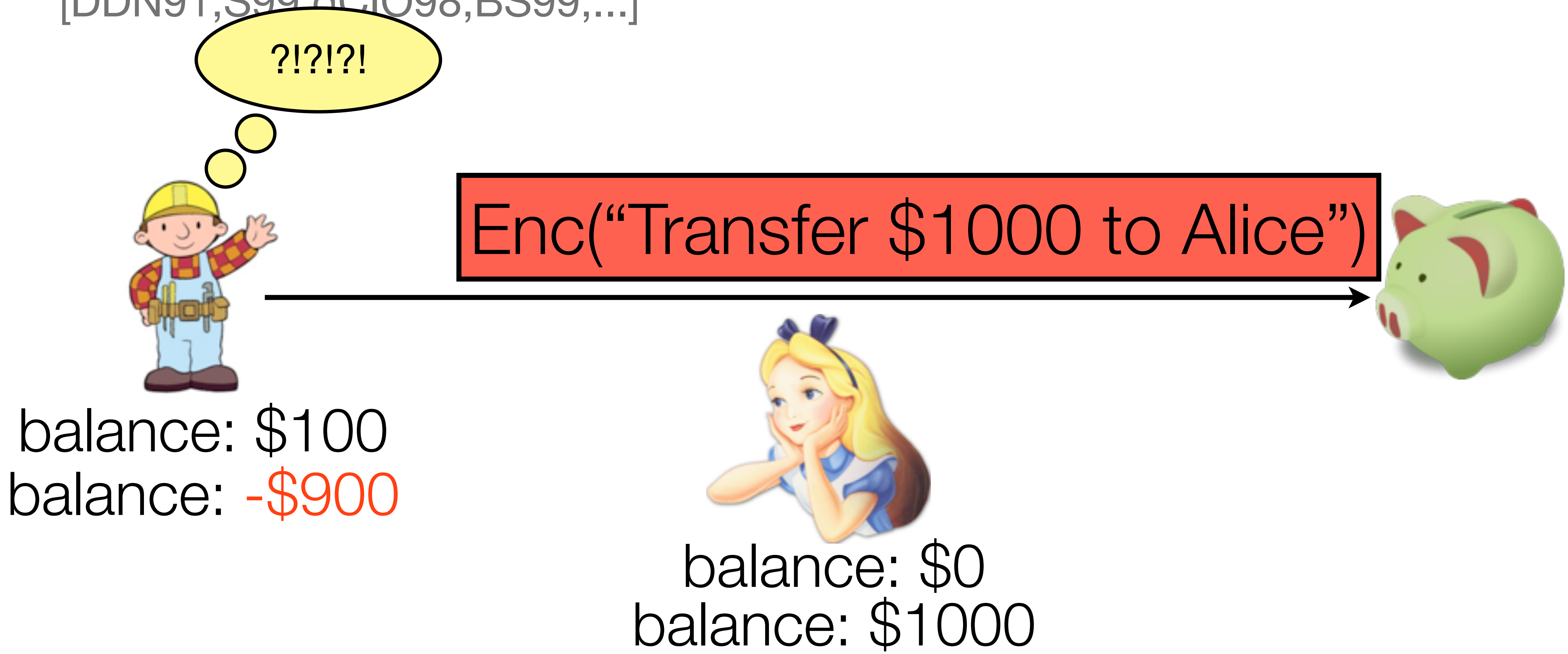
Non-malleable cryptography

Twenty years ago, saw a strong emphasis on non-malleable cryptography [DDN91,S99,dCIO98,BS99,...]



Non-malleable cryptography

Twenty years ago, saw a strong emphasis on non-malleable cryptography [DDN91,S99,dCIO98,BS99,...]



Malleable cryptography

Malleable cryptography

Recently, see more emphasis on malleable cryptography [G09,BCCKLS09,DHLW10,F11,BF11,ABCHSW12]

Malleable cryptography

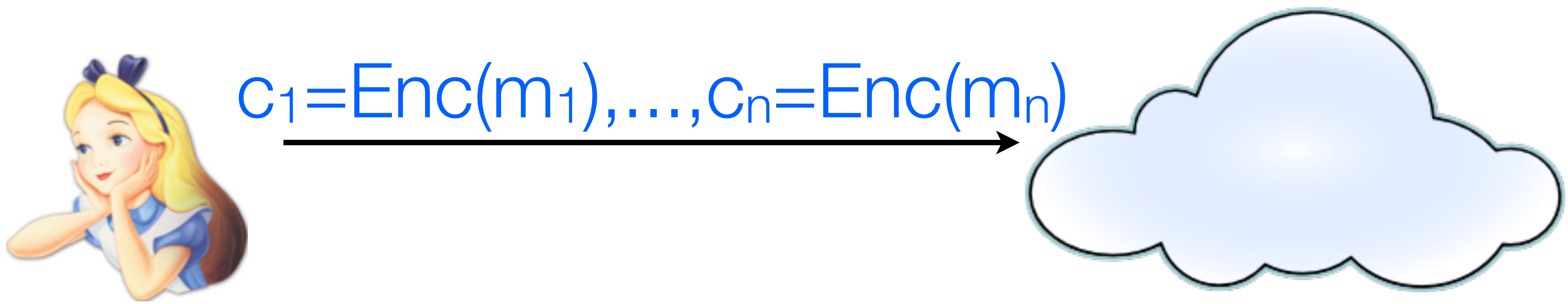
Recently, see more emphasis on malleable cryptography [G09,BCCKLS09,DHLW10,F11,BF11,ABCHSW12]

Malleable cryptography

Recently, see more emphasis on malleable cryptography [G09,BCCKLS09,DHLW10,F11,BF11,ABCHSW12]

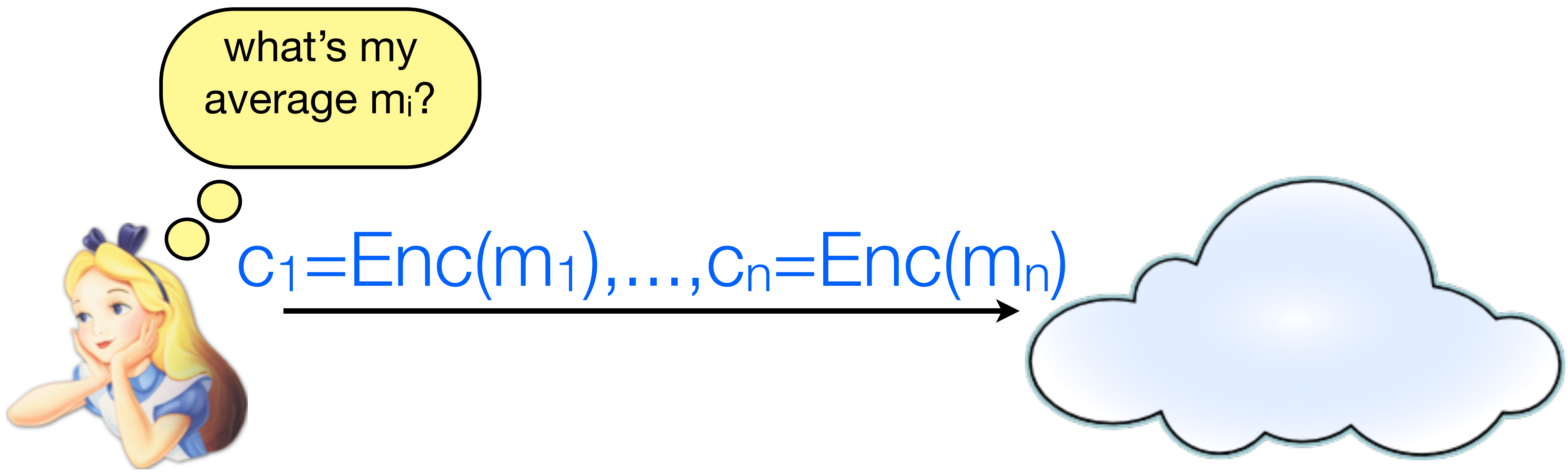
Malleable cryptography

Recently, see more emphasis on malleable cryptography [G09,BCCKLS09,DHLW10,F11,BF11,ABCHSW12]



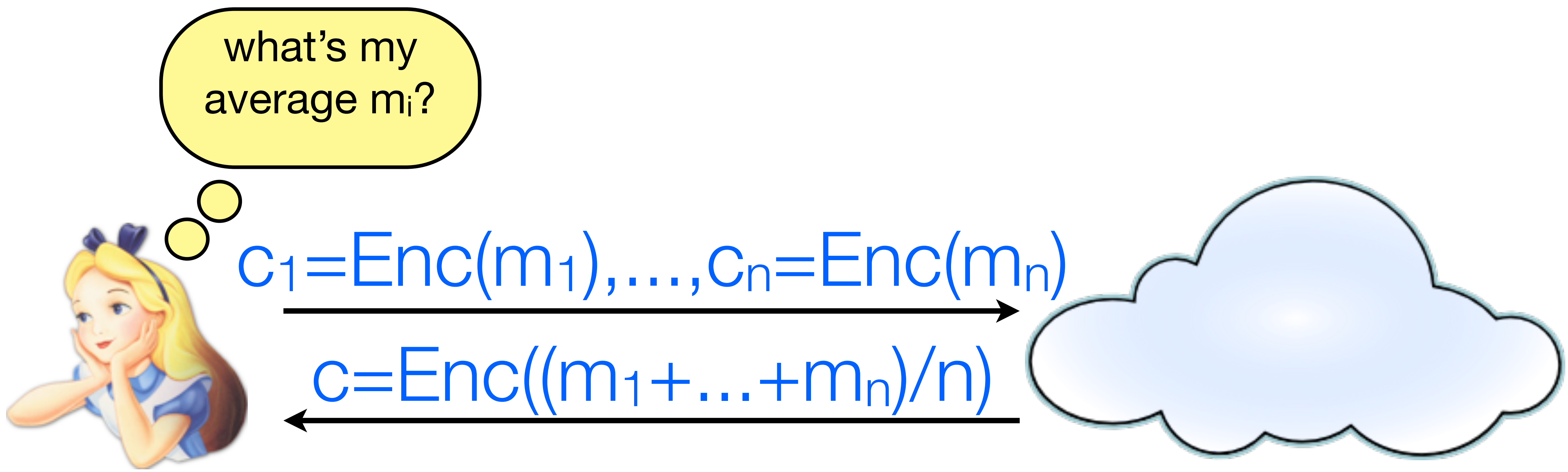
Malleable cryptography

Recently, see more emphasis on malleable cryptography [G09,BCCKLS09,DHLW10,F11,BF11,ABCHSW12]



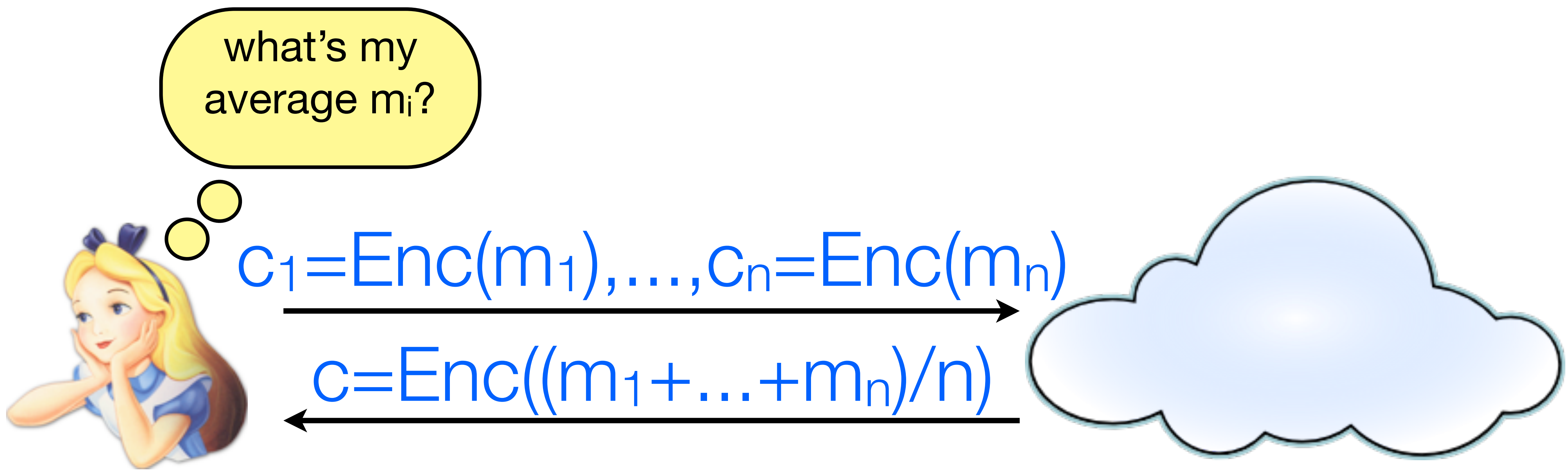
Malleable cryptography

Recently, see more emphasis on malleable cryptography [G09,BCCKLS09,DHLW10,F11,BF11,ABCHSW12]



Malleable cryptography

Recently, see more emphasis on malleable cryptography [G09,BCCKLS09,DHLW10,F11,BF11,ABCHSW12]



Has applications in cloud storage, outsourcing computation, search on encrypted data, etc.

Our contribution: controlled malleable cryptography

Our contribution: controlled malleable cryptography

Methods for controlling malleability can provide a compromise between functionality and security [PR08,BSW12]

Our contribution: controlled malleable cryptography

Methods for controlling malleability can provide a compromise between functionality and security [PR08,BSW12]

- E.g., in cloud storage, only allowable transformation is the average

Our contribution: controlled malleable cryptography

Methods for controlling malleability can provide a compromise between functionality and security [PR08,BSW12]

- E.g., in cloud storage, only allowable transformation is the average
- E.g., with bank account, mauling can only decrease amount

Our contribution: controlled malleable cryptography

Methods for controlling malleability can provide a compromise between functionality and security [PR08,BSW12]

- E.g., in cloud storage, only allowable transformation is the average
- E.g., with bank account, mauling can only decrease amount

In this work:

- Introduce notions of uncontrolled and controlled malleability for proofs
- Give two applications: CM-CCA security and compact verifiable shuffles
- Examine malleability within existing proof systems

Outline

Outline

Definitions

Outline

Definitions

cm-NIZK construction

Outline

Definitions

cm-NIZK construction

Applications

Outline

Definitions

cm-NIZK construction

Applications

Conclusions

Outline

Definitions

Zero knowledge

Malleability

Controlled malleability

Derivation privacy

cm-NIZK construction

Applications

Conclusions

Notions of malleability for proofs

Notions of malleability for proofs

Example: take a proof π_1 that b_1 is a bit and a proof π_2 that b_2 is a bit, and "maul" them somehow to get a proof that $b_1 * b_2$ is a bit

Notions of malleability for proofs

Example: take a proof π_1 that b_1 is a bit and a proof π_2 that b_2 is a bit, and "maul" them somehow to get a proof that $b_1 * b_2$ is a bit

More generally, a proof is malleable with respect to T if there exists an algorithm Eval that on input $(T,\{x_i,\pi_i\})$, outputs a proof π for $T(\{x_i\})$

Notions of malleability for proofs

Example: take a proof π_1 that b_1 is a bit and a proof π_2 that b_2 is a bit, and "maul" them somehow to get a proof that $b_1 * b_2$ is a bit

More generally, a proof is malleable with respect to T if there exists an algorithm Eval that on input $(T,\{x_i,\pi_i\})$, outputs a proof π for $T(\{x_i\})$

- E.g., $T = \times$, x_i = "b_i is a bit"

Notions of malleability for proofs

Example: take a proof π_1 that b_1 is a bit and a proof π_2 that b_2 is a bit, and "maul" them somehow to get a proof that $b_1 * b_2$ is a bit

More generally, a proof is malleable with respect to T if there exists an algorithm Eval that on input $(T,\{x_i,\pi_i\})$, outputs a proof π for $T(\{x_i\})$

- E.g., $T = \times$, x_i = "b_i is a bit"

If we want zero knowledge, need to make sure proofs are malleable only with respect to operations under which the language is closed

- E.g., with bits, we run into trouble if we try to use $T = +$

Reconciling (controlled) malleability with soundness

Reconciling (controlled) malleability with soundness

What if we want to be able to maul proofs of knowledge only in certain ways?

Reconciling (controlled) malleability with soundness

What if we want to be able to maul proofs of knowledge only in certain ways?

- Define an allowable set of transformations $\mathcal{T}$

Reconciling (controlled) malleability with soundness

What if we want to be able to maul proofs of knowledge only in certain ways?

- Define an allowable set of transformations $\mathcal{T}$

- Next we look at simulation soundness [S99,dSdCOPS01]: adversary can't provide proofs of false statements, even with access to a simulation oracle that can

Reconciling (controlled) malleability with soundness

What if we want to be able to maul proofs of knowledge only in certain ways?

- Define an allowable set of transformations $\mathcal{T}$
- Next we look at simulation soundness [S99,dSdCOPS01]: adversary can't provide proofs of false statements, even with access to a simulation oracle that can
- Even more, simulation-sound extractability [G06] says that in fact we can always pull out a witness from any proof output by the adversary

Reconciling (controlled) malleability with soundness

What if we want to be able to maul proofs of knowledge only in certain ways?

- Define an allowable set of transformations $\mathcal{T}$
- Next we look at simulation soundness [S99,dSdCOPS01]: adversary can't provide proofs of false statements, even with access to a simulation oracle that can
- Even more, simulation-sound extractability [G06] says that in fact we can always pull out a witness from any proof output by the adversary
- Our definition goes one step further: either we can pull out a witness, or it was derived from a simulated proof under a transformation in $\mathcal{T}$

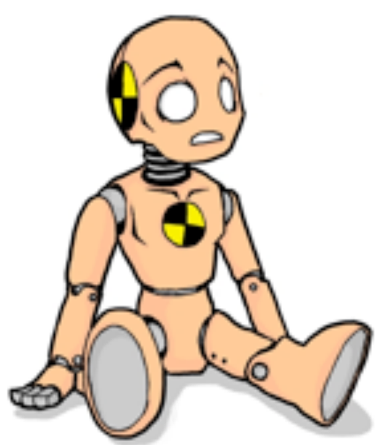
Controlled-malleable SSE zero-knowledge proofs

Controlled-malleable SSE zero-knowledge proofs

High-level idea: extractor can pull out either a witness, or a previously queried statement and a transformation from that statement to the new one

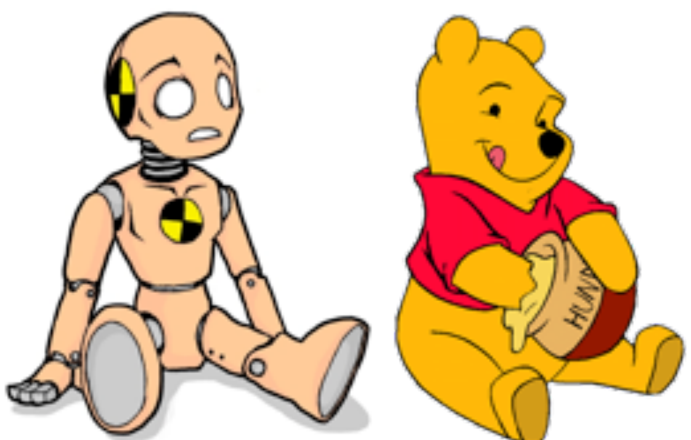
Controlled-malleable SSE zero-knowledge proofs

High-level idea: extractor can pull out either a witness, or a previously queried statement and a transformation from that statement to the new one



Controlled-malleable SSE zero-knowledge proofs

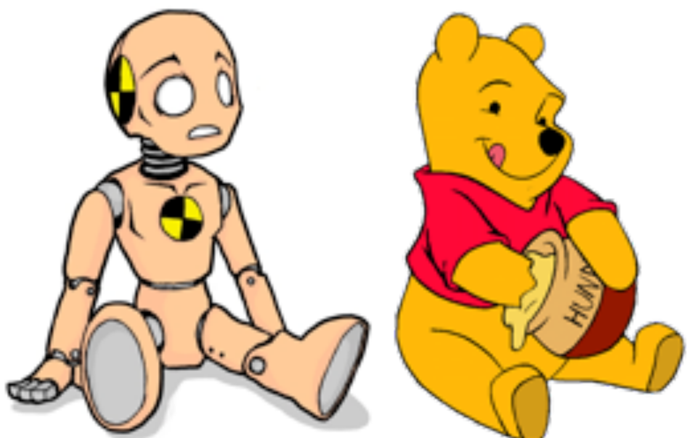
High-level idea: extractor can pull out either a witness, or a previously queried statement and a transformation from that statement to the new one



σ, τ_s, τ_e

Controlled-malleable SSE zero-knowledge proofs

High-level idea: extractor can pull out either a witness, or a previously queried statement and a transformation from that statement to the new one

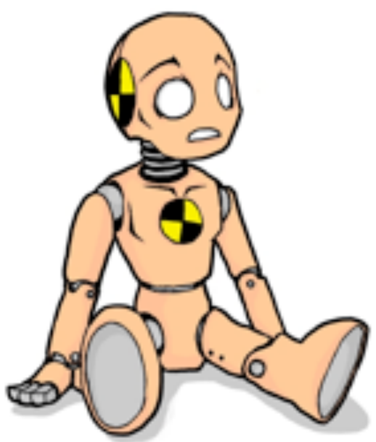


σ, τ_s, τ_e

σ, τ_e

Controlled-malleable SSE zero-knowledge proofs

High-level idea: extractor can pull out either a witness, or a previously queried statement and a transformation from that statement to the new one



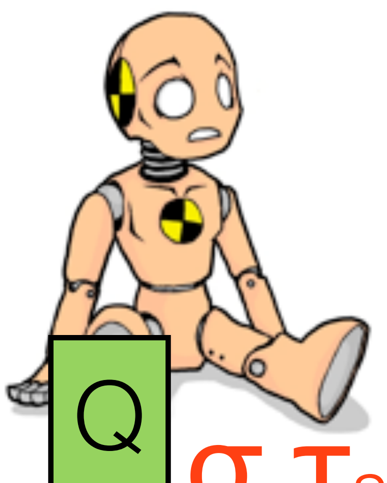
σ, τ_s,

σ, τ_e

τ_e

Controlled-malleable SSE zero-knowledge proofs

High-level idea: extractor can pull out either a witness, or a previously queried statement and a transformation from that statement to the new one



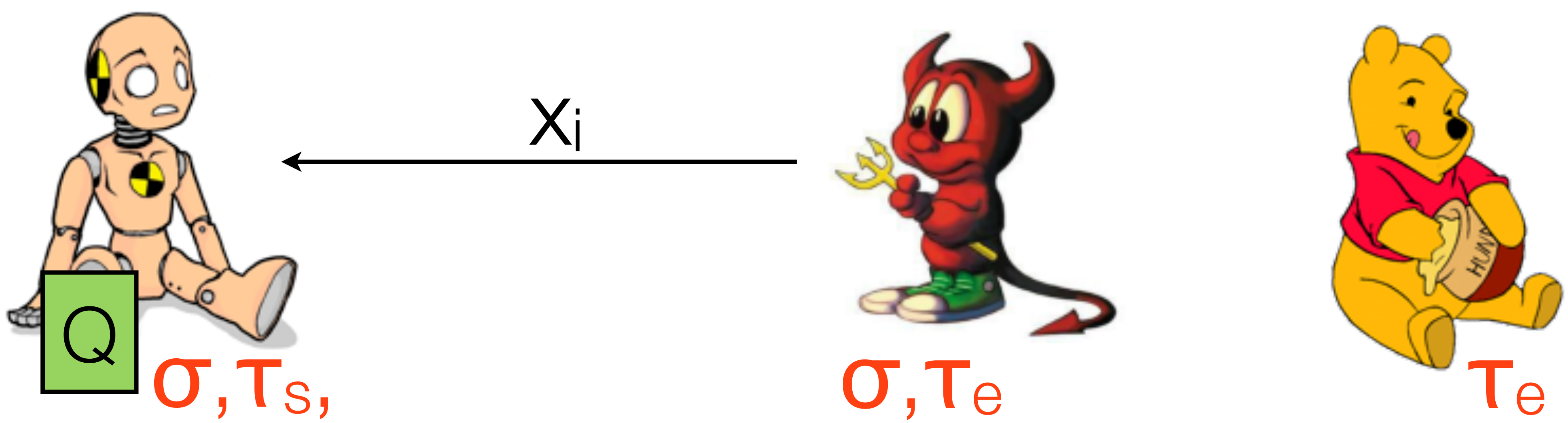
$\sigma, \tau_s,$

σ, τ_e

τ_e

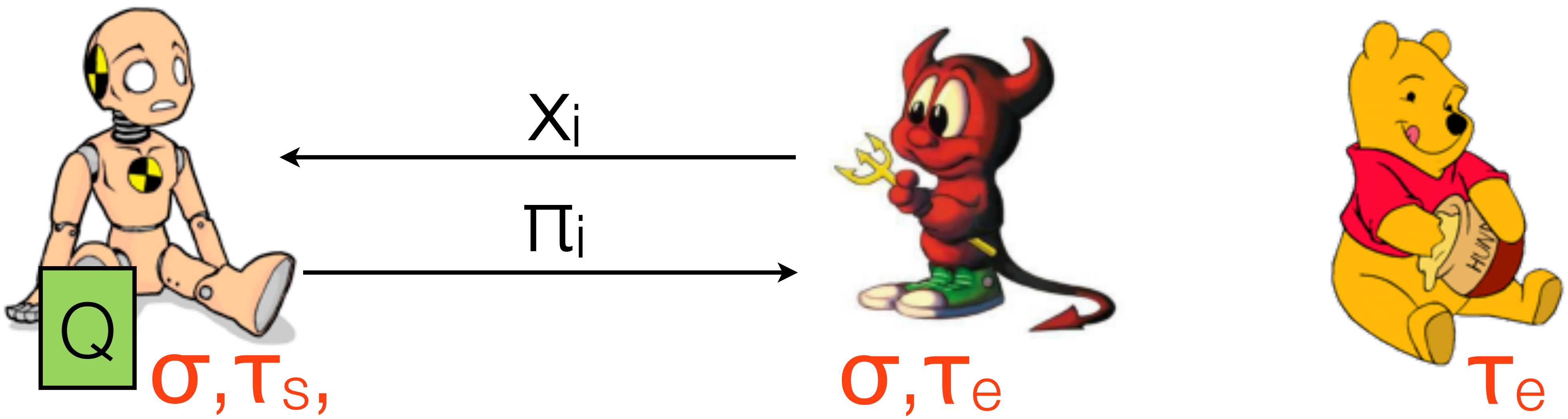
Controlled-malleable SSE zero-knowledge proofs

High-level idea: extractor can pull out either a witness, or a previously queried statement and a transformation from that statement to the new one



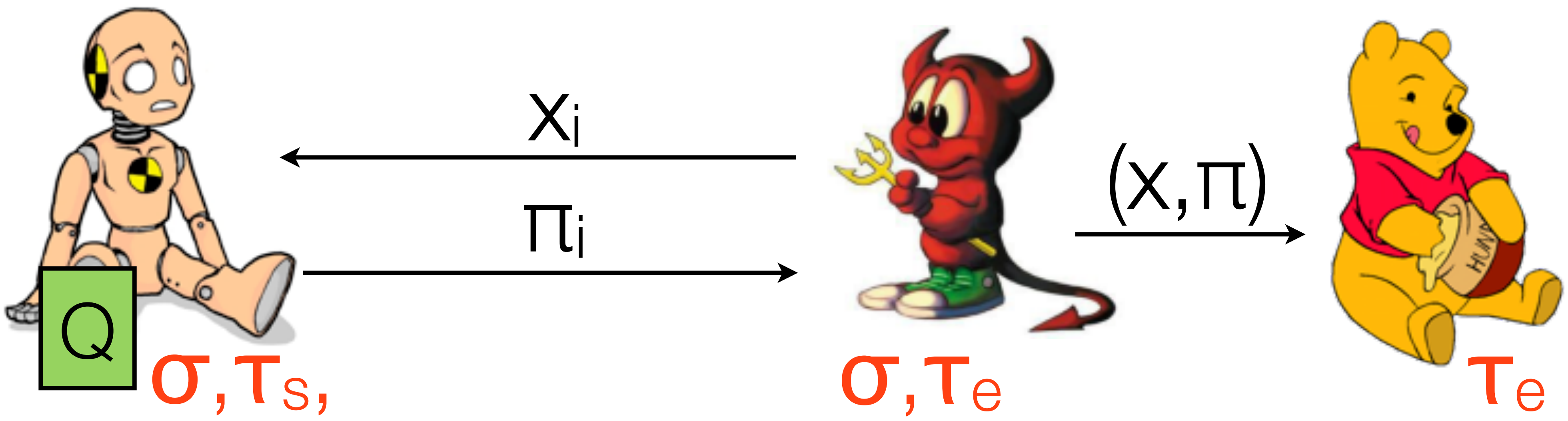
Controlled-malleable SSE zero-knowledge proofs

High-level idea: extractor can pull out either a witness, or a previously queried statement and a transformation from that statement to the new one



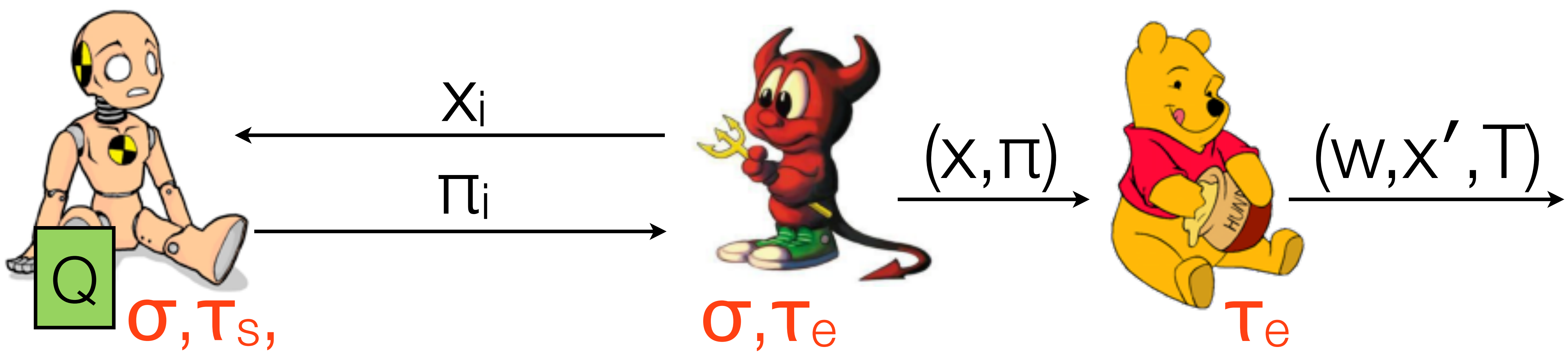
Controlled-malleable SSE zero-knowledge proofs

High-level idea: extractor can pull out either a witness, or a previously queried statement and a transformation from that statement to the new one



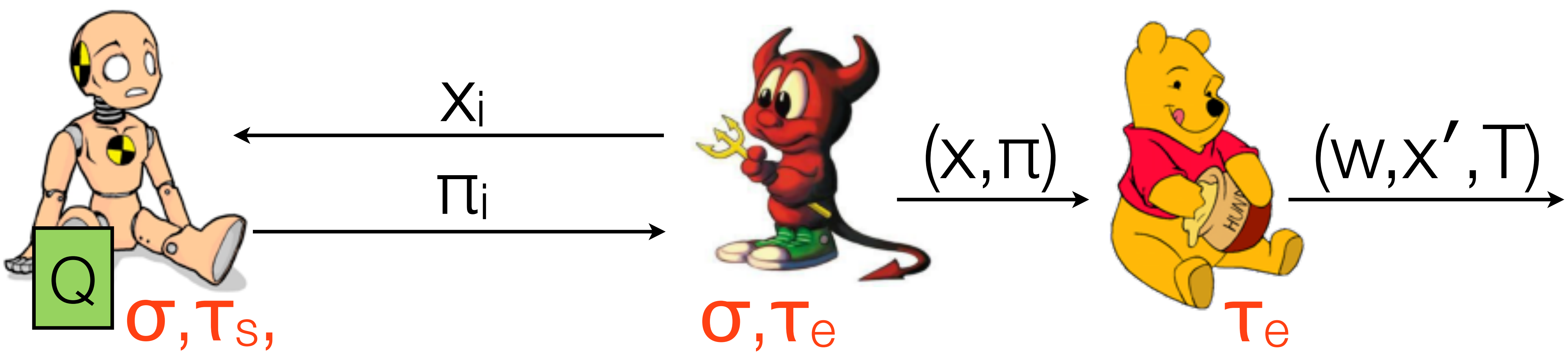
Controlled-malleable SSE zero-knowledge proofs

High-level idea: extractor can pull out either a witness, or a previously queried statement and a transformation from that statement to the new one



Controlled-malleable SSE zero-knowledge proofs

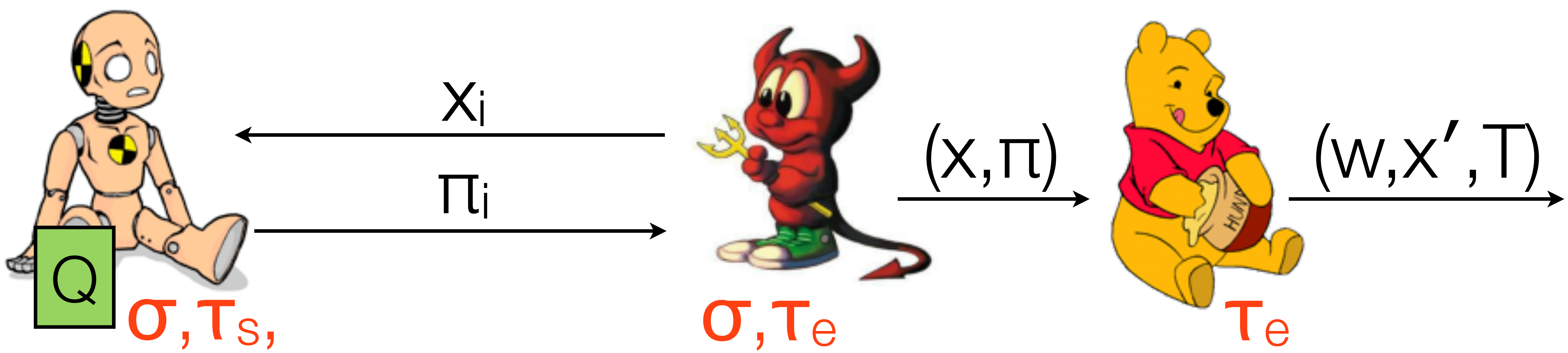
High-level idea: extractor can pull out either a witness, or a previously queried statement and a transformation from that statement to the new one



A wins if the proof verifies and $x \notin Q$ but (1) $w \neq \perp$ but isn't a valid witness, (2) $(x', T) \neq (\perp, \perp)$ but $x' \notin Q$, $x \neq T(x')$, or T is not in $\mathcal{T}$, or (3) $(w, x', T) = (\perp, \perp, \perp)$

Controlled-malleable SSE zero-knowledge proofs

High-level idea: extractor can pull out either a witness, or a previously queried statement and a transformation from that statement to the new one

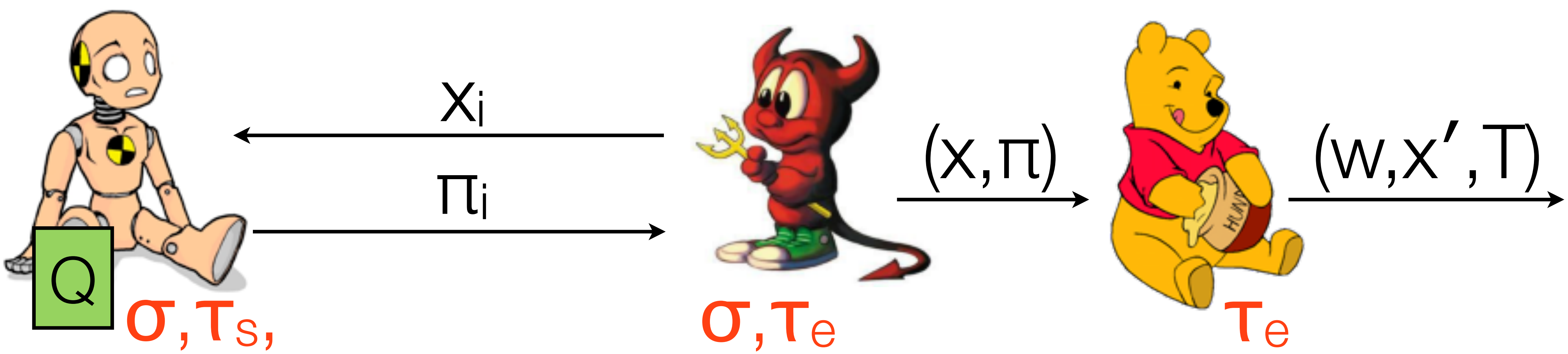


A wins if the proof verifies and $x \notin Q$ but (1) $w \neq \perp$ but isn't a valid witness, (2) $(x', T) \neq (\perp, \perp)$ but $x' \notin Q$, $x \neq T(x')$, or T is not in $\mathcal{T}$, or (3) $(w, x', T) = (\perp, \perp, \perp)$

We call the proof CM-SSE (controlled malleable simulation sound extractable) if any PPT adversary A has at most negligible probability in winning this game

Controlled-malleable SSE zero-knowledge proofs

High-level idea: extractor can pull out either a witness, or a previously queried statement and a transformation from that statement to the new one



A wins if the proof verifies and $x \notin Q$ but (1) $w \neq \perp$ but isn't a valid witness, (2) $(x', T) \neq (\perp, \perp)$ but $x' \notin Q$, $x \neq T(x')$, or T is not in $\mathcal{T}$, or (3) $(w, x', T) = (\perp, \perp, \perp)$

We call the proof CM-SSE (controlled malleable simulation sound extractable) if any PPT adversary A has at most negligible probability in winning this game

(like function privacy for encryption)

If a proof is zero knowledge, CM-SSE, and strongly derivation private, then we call it a cm-NIZK

Outline

Definitions

cm-NIZK construction

Generic construction

Efficient instantiation

Applications

Conclusions

How to construct cm-NIZKs

How to construct cm-NIZKs

We will combine malleable NIWIPoKs with unforgeable signatures

How to construct cm-NIZKs

We will combine malleable NIWIPoKs with unforgeable signatures

cm-NIZK(x,w) = NIWIPoK{(x,(w,x′,T,σ)) s.t. either (x,w)∈R or Verify(vk,x′,σ)=1, x=T(x′), and T is in $\mathscr{T}$}

How to construct cm-NIZKs

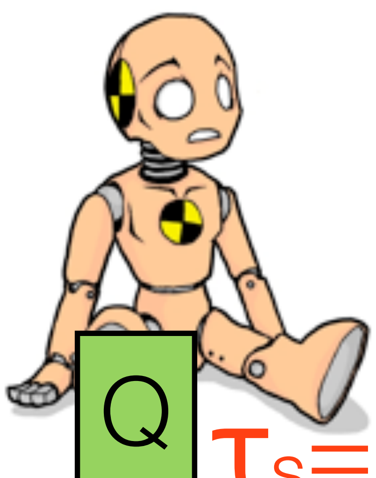
We will combine malleable NIWIPoKs with unforgeable signatures

cm-NIZK(x,w) = NIWIPoK{(x,(w,x′,T,σ)) s.t. either (x,w)$\in$R or Verify(vk,x′,σ)=1, x=T(x′), and T is in $\mathscr{T}$}

How to construct cm-NIZKs

We will combine malleable NIWIPoKs with unforgeable signatures

cm-NIZK(x,w) = NIWIPoK{(x,(w,x′,T,σ)) s.t. either (x,w)∈R or Verify(vk,x′,σ)=1, x=T(x′), and T is in $\mathcal{T}$}

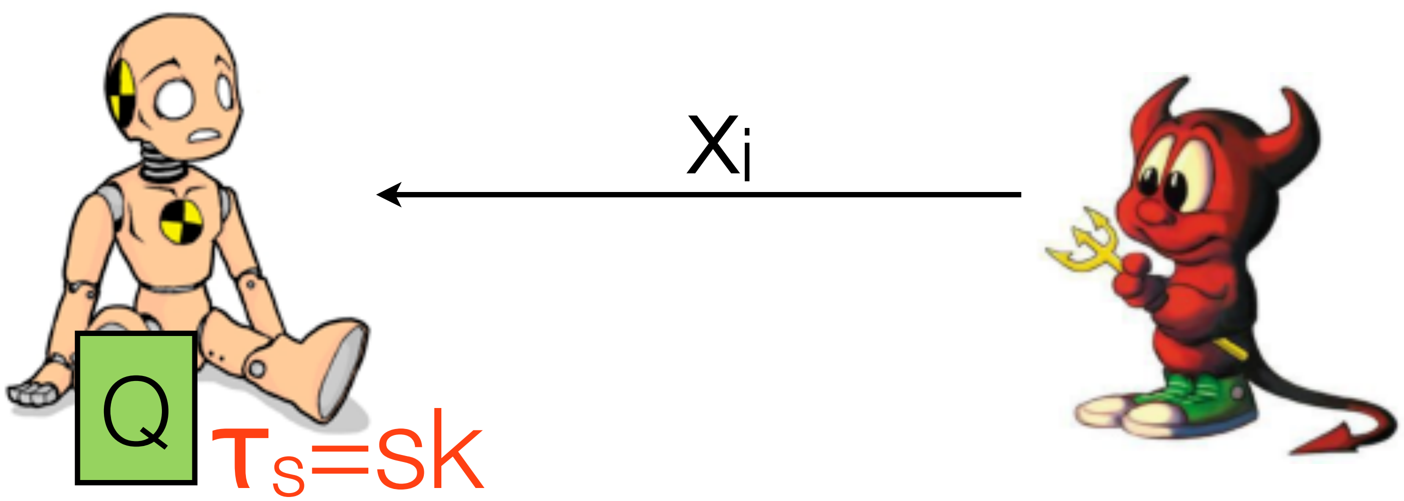


τ_S=sk

How to construct cm-NIZKs

We will combine malleable NIWIPoKs with unforgeable signatures

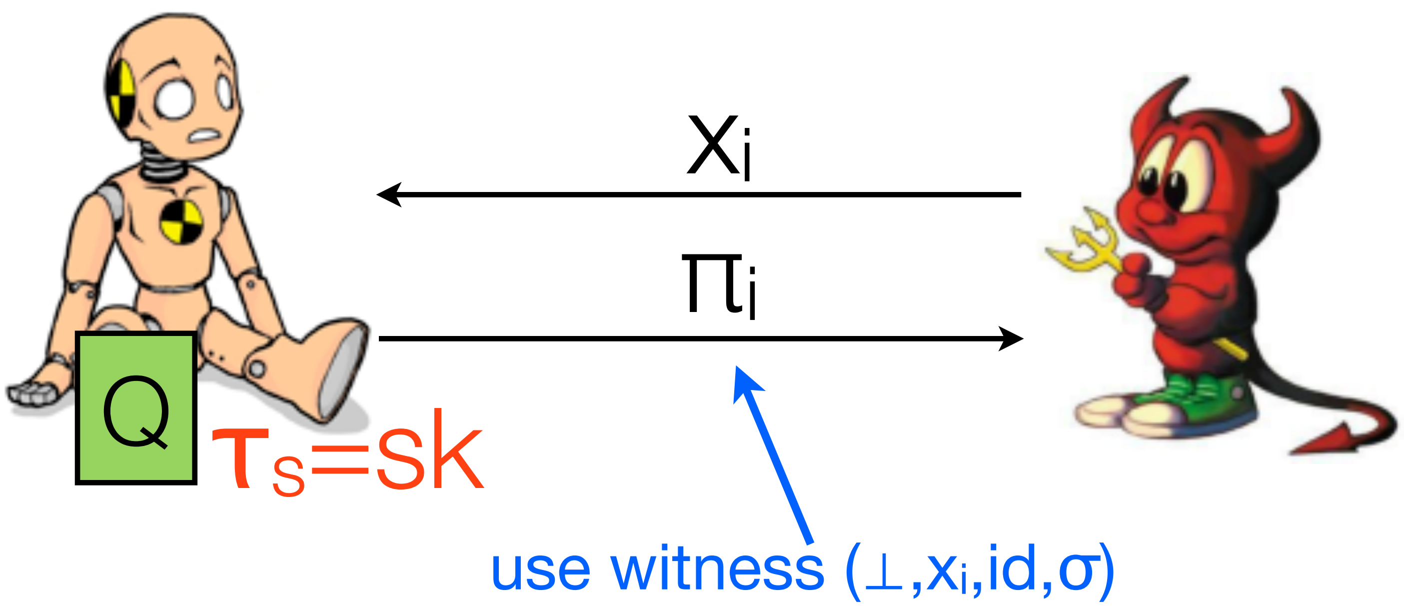
cm-NIZK(x,w) = NIWIPoK{(x,(w,x′,T,σ)) s.t. either (x,w)∈R or Verify(vk,x′,σ)=1, x=T(x′), and T is in $\mathcal{T}$}



How to construct cm-NIZKs

We will combine malleable NIWIPoKs with unforgeable signatures

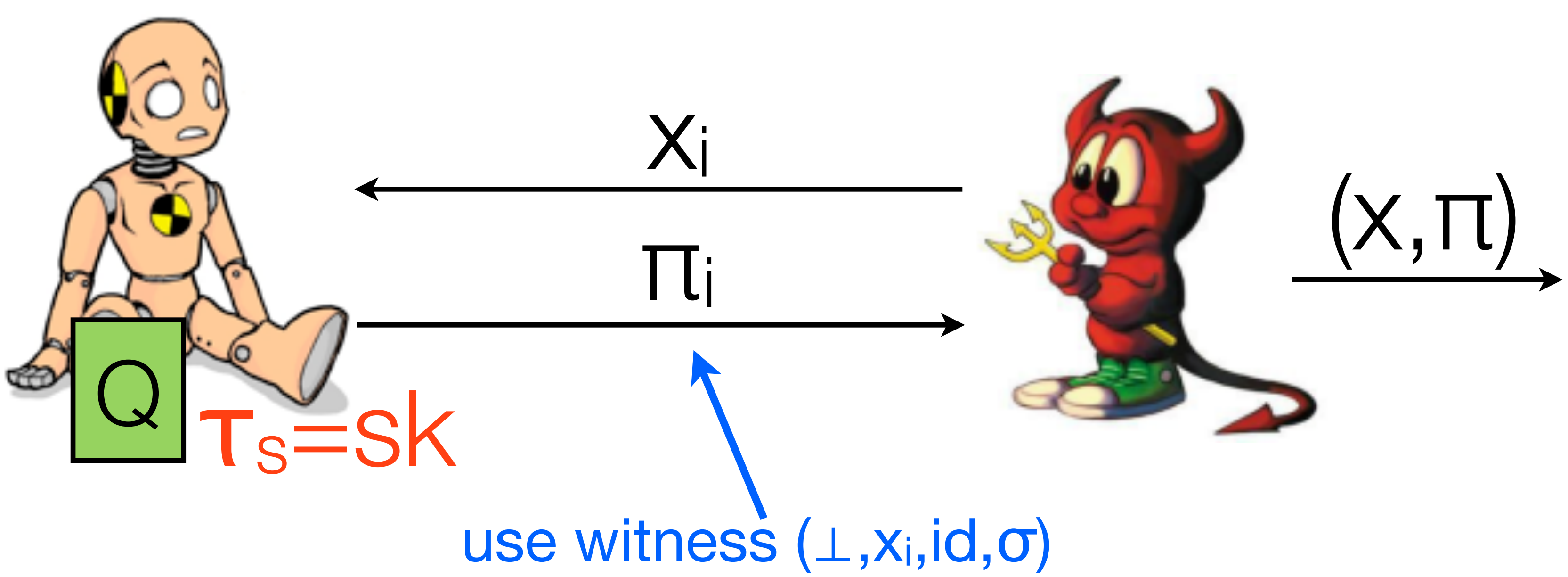
cm-NIZK(x,w) = NIWIPoK{(x,(w,x′,T,σ)) s.t. either (x,w)∈R or Verify(vk,x′,σ)=1, x=T(x′), and T is in $\mathcal{J}$}



How to construct cm-NIZKs

We will combine malleable NIWIPoKs with unforgeable signatures

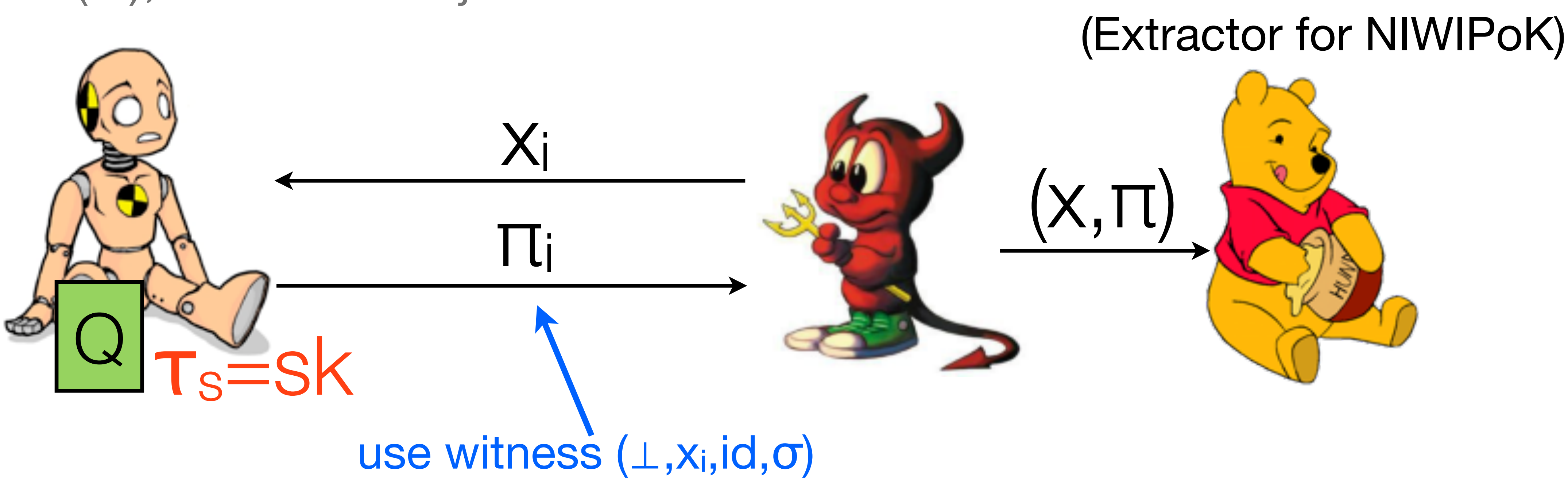
cm-NIZK(x,w) = NIWIPoK{(x,(w,x′,T,σ)) s.t. either (x,w)∈R or Verify(vk,x′,σ)=1, x=T(x′), and T is in $\mathscr{T}$}



How to construct cm-NIZKs

We will combine malleable NIWIPoKs with unforgeable signatures

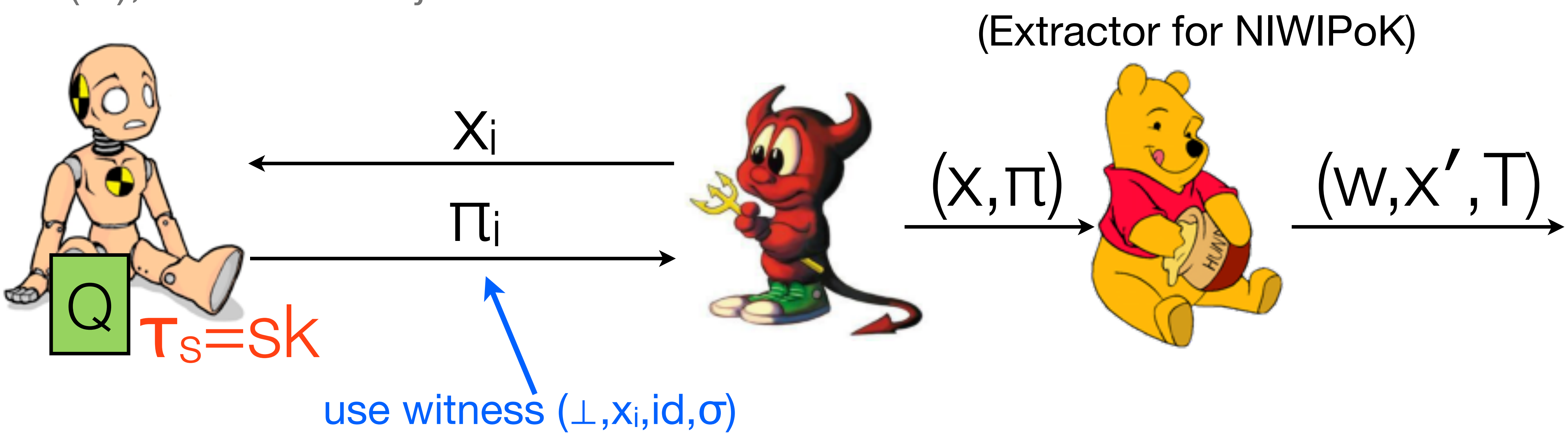
cm-NIZK(x,w) = NIWIPoK{(x,(w,x′,T,σ)) s.t. either (x,w)∈R or Verify(vk,x′,σ)=1, x=T(x′), and T is in $\mathcal{T}$}



How to construct cm-NIZKs

We will combine malleable NIWIPoKs with unforgeable signatures

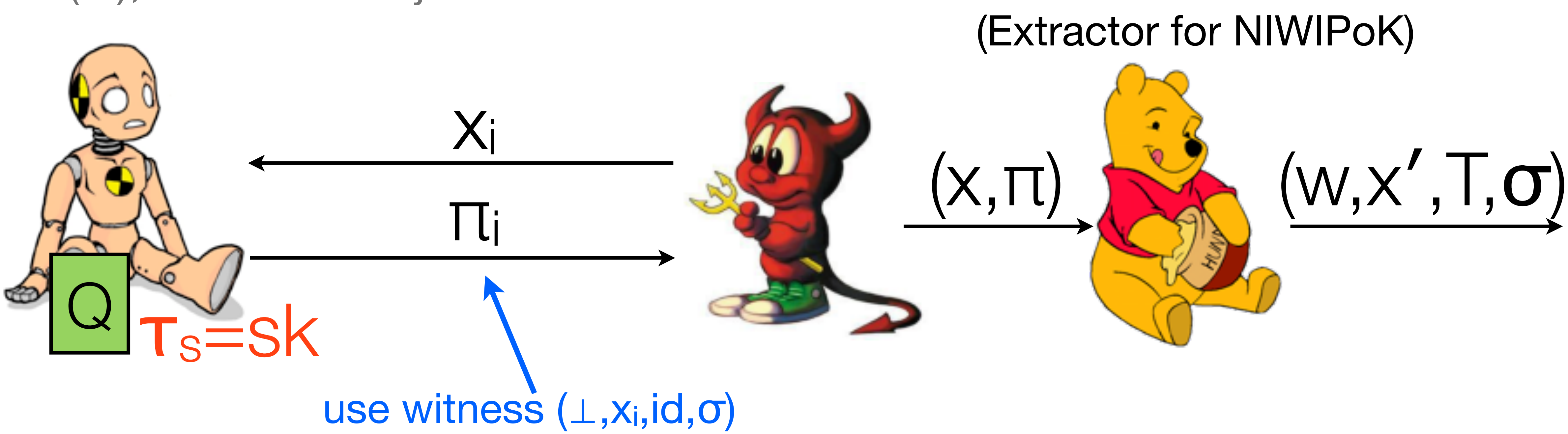
cm-NIZK(x,w) = NIWIPoK{(x,(w,x′,T,σ)) s.t. either (x,w)∈R or Verify(vk,x′,σ)=1, x=T(x′), and T is in $\mathcal{T}$}



How to construct cm-NIZKs

We will combine malleable NIWIPoKs with unforgeable signatures

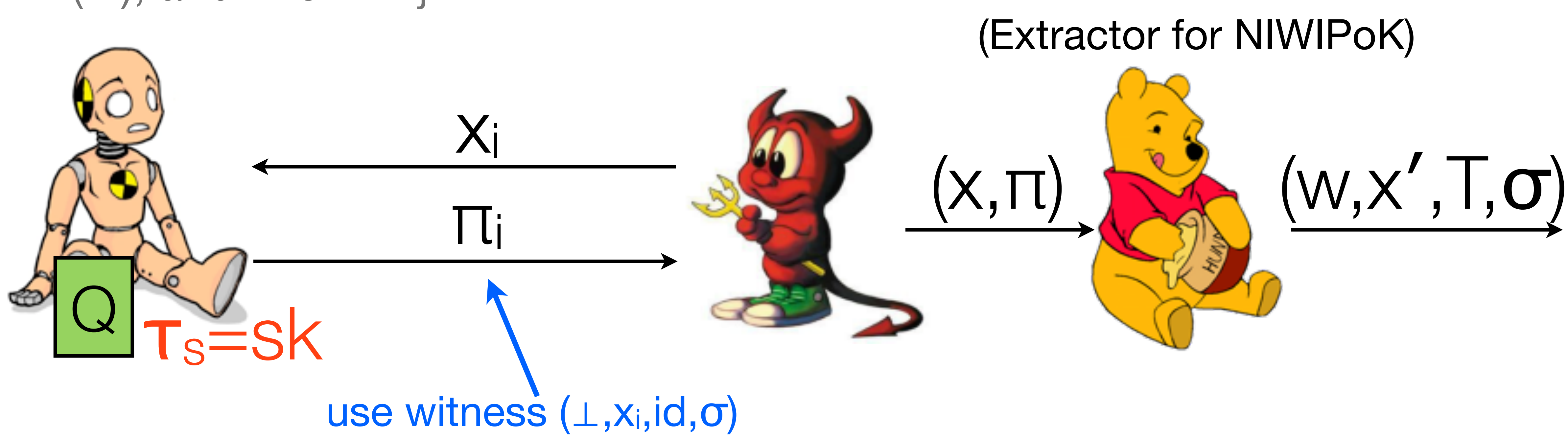
cm-NIZK(x,w) = NIWIPoK{(x,(w,x′,T,σ)) s.t. either (x,w)∈R or Verify(vk,x′,σ)=1, x=T(x′), and T is in $\mathcal{J}$}



How to construct cm-NIZKs

We will combine malleable NIWIPoKs with unforgeable signatures

cm-NIZK(x,w) = NIWIPoK{(x,(w,x′,T,σ)) s.t. either $(x,w)\in R$ or Verify(vk,x′,σ)=1, x=T(x′), and T is in $\mathcal{T}$}

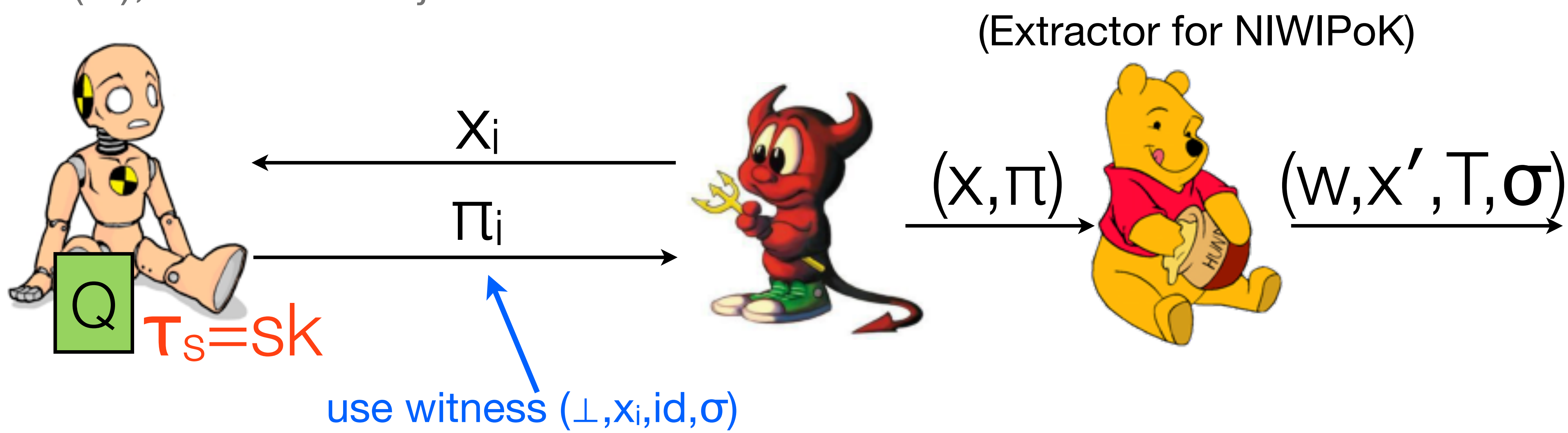


A wins if (1) $w\neq\perp$ but isn't a valid witness, (2) $(x',T)\neq(\perp,\perp)$ but $x'\notin Q$, $x\neq T(x')$, or T is not in $\mathcal{T}$, or (3) $(w,x',T)=(\perp,\perp,\perp)$

How to construct cm-NIZKs

We will combine malleable NIWIPoKs with unforgeable signatures

cm-NIZK(x,w) = NIWIPoK{(x,(w,x′,T,σ)) s.t. either (x,w)∈R or Verify(vk,x′,σ)=1, x=T(x′), and T is in $\mathscr{T}$}

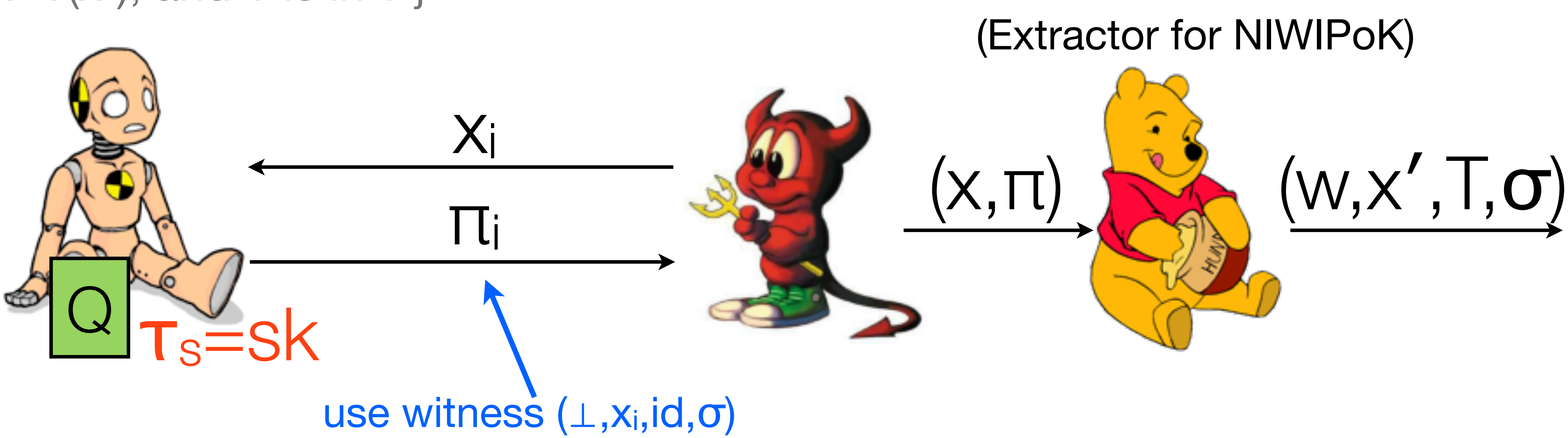


A wins if (1) w≠⊥ but isn't a valid witness, (2) (x′,T)≠(⊥,⊥) but x′∉Q, x≠T(x′), or T is not in $\mathscr{T}$, or (3) (w,x′,T)=(⊥,⊥,⊥)

How to construct cm-NIZKs

We will combine malleable NIWIPoKs with unforgeable signatures

cm-NIZK(x,w) = NIWIPoK{(x,(w,x′,T,σ)) s.t. either (x,w)∈R or Verify(vk,x′,σ)=1, x=T(x′), and T is in $\mathcal{T}$}



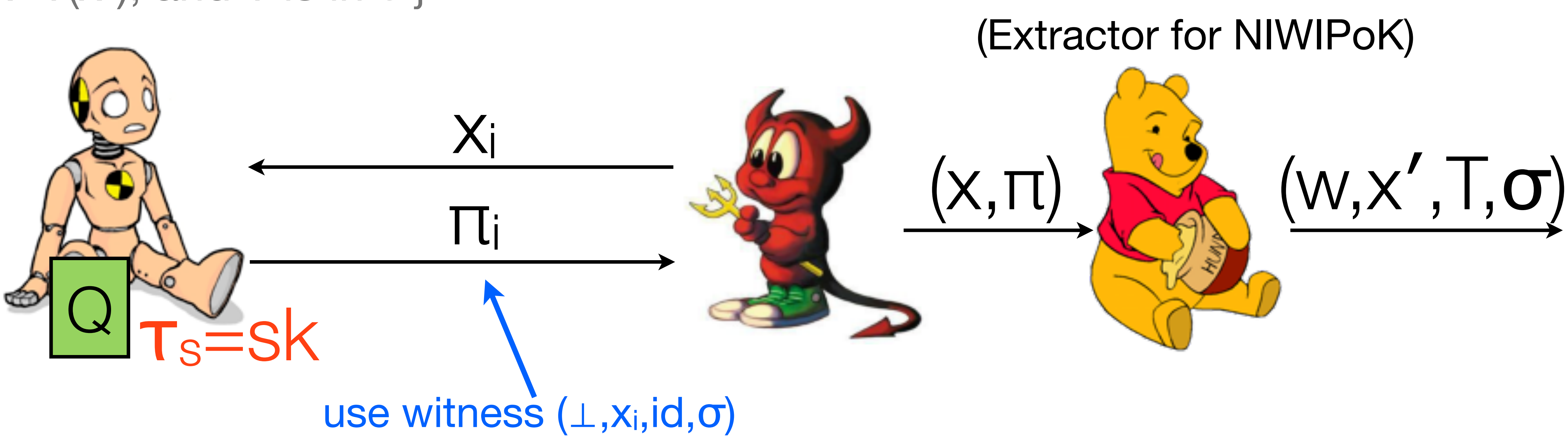
violates extractability

A wins if (1) w≠⊥ but isn't a valid witness, (2) (x′,T)≠(⊥,⊥) but x′∉Q, x≠T(x′), or T is not in $\mathcal{T}$, or (3) (w,x′,T)=(⊥,⊥,⊥)

How to construct cm-NIZKs

We will combine malleable NIWIPoKs with unforgeable signatures

cm-NIZK(x,w) = NIWIPoK{(x,(w,x′,T,σ)) s.t. either (x,w)∈R or Verify(vk,x′,σ)=1, x=T(x′), and T is in $\mathscr{T}$}

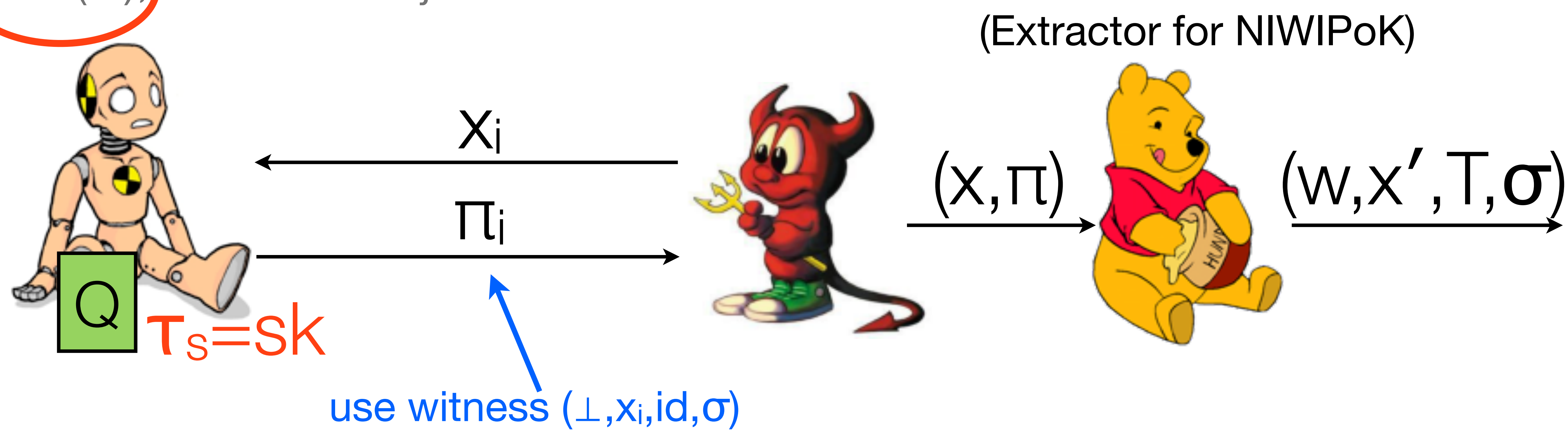


A wins if ~~(1)~~ w≠⊥ but isn't a valid witness, (2) (x′,T)≠(⊥,⊥) but x′∉Q, x≠T(x′), or T is not in $\mathscr{T}$, or (3) (w,x′,T)=(⊥,⊥,⊥)

How to construct cm-NIZKs

We will combine malleable NIWIPoKs with unforgeable signatures

cm-NIZK(x,w) = NIWIPoK{(x,(w,x′,T,σ)) s.t. either (x,w)∈R or Verify(vk,x′,σ)=1, x=T(x′), and T is in $\mathcal{J}$}



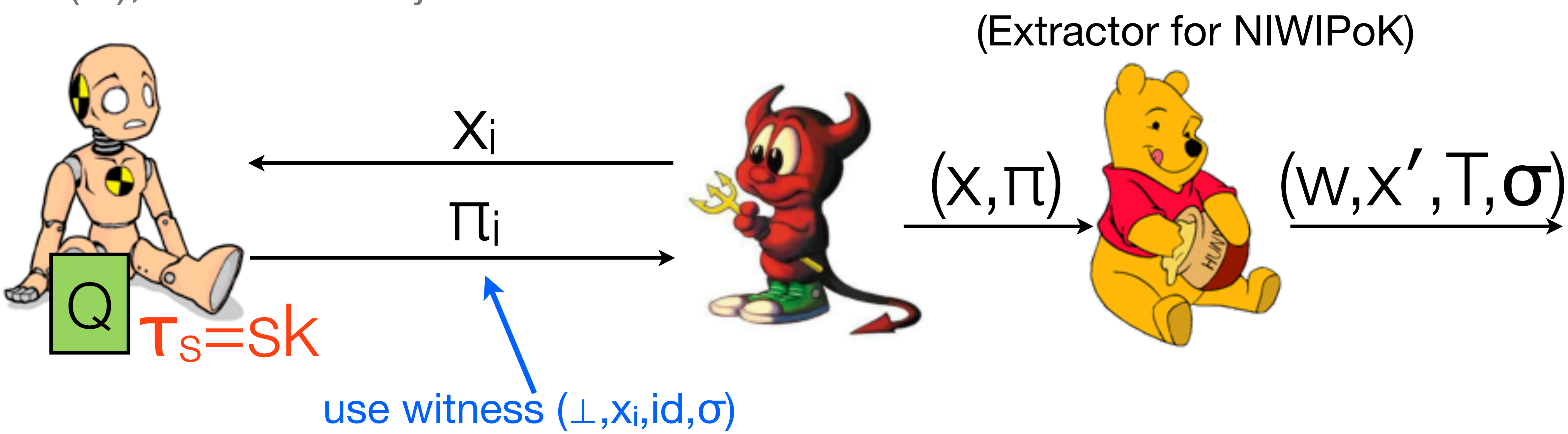
violates extractability

A wins if ~~(1)~~ w≠⊥ but isn't a valid witness, (2) (x′,T)≠(⊥,⊥) but x′∉Q, ~~x≠T(x′)~~, or T is not in $\mathcal{J}$, or (3) (w,x′,T)=(⊥,⊥,⊥)

How to construct cm-NIZKs

We will combine malleable NIWIPoKs with unforgeable signatures

cm-NIZK(x,w) = NIWIPoK{(x,(w,x′,T,σ)) s.t. either (x,w)∈R or Verify(vk,x′,σ)=1, x=T(x′), and T is in $\mathcal{T}$}

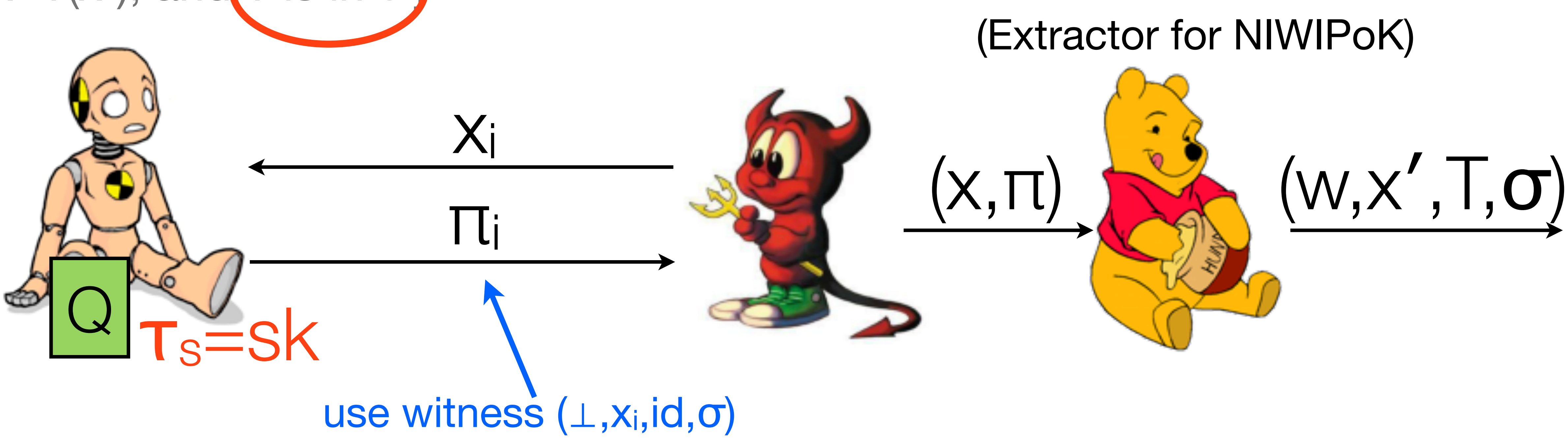


A wins if (1) w≠⊥ but isn't a valid witness, (2) (x′,T)≠(⊥,⊥) but x′∉Q, x≠T(x′), or T is not in $\mathcal{T}$, or (3) (w,x′,T)=(⊥,⊥,⊥)

How to construct cm-NIZKs

We will combine malleable NIWIPoKs with unforgeable signatures

cm-NIZK(x,w) = NIWIPoK{(x,(w,x′,T,σ)) s.t. either (x,w)∈R or Verify(vk,x′,σ)=1, x=T(x′), and T is in $\mathcal{T}$}



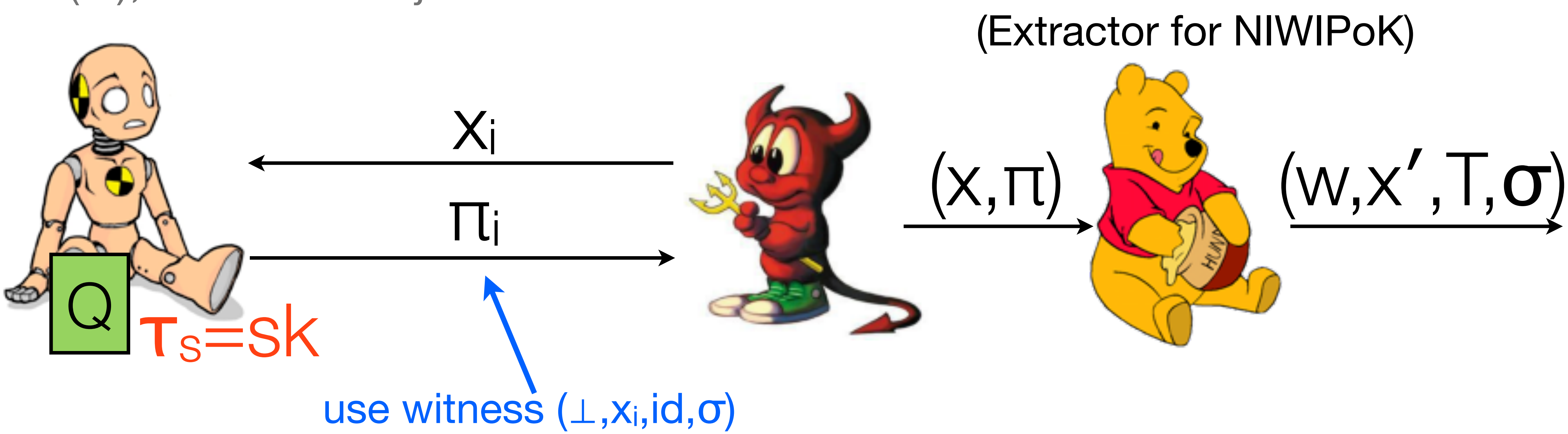
A wins if (1) w≠⊥ but isn't a valid witness, (2) (x′,T)≠(⊥,⊥) but x′∉Q, x≠T(x′), or T is not in $\mathcal{T}$, or (3) (w,x′,T)=(⊥,⊥,⊥)

violates extractability

How to construct cm-NIZKs

We will combine malleable NIWIPoKs with unforgeable signatures

cm-NIZK(x,w) = NIWIPoK{(x,(w,x′,T,σ)) s.t. either (x,w)∈R or Verify(vk,x′,σ)=1, x=T(x′), and T is in $\mathcal{T}$}

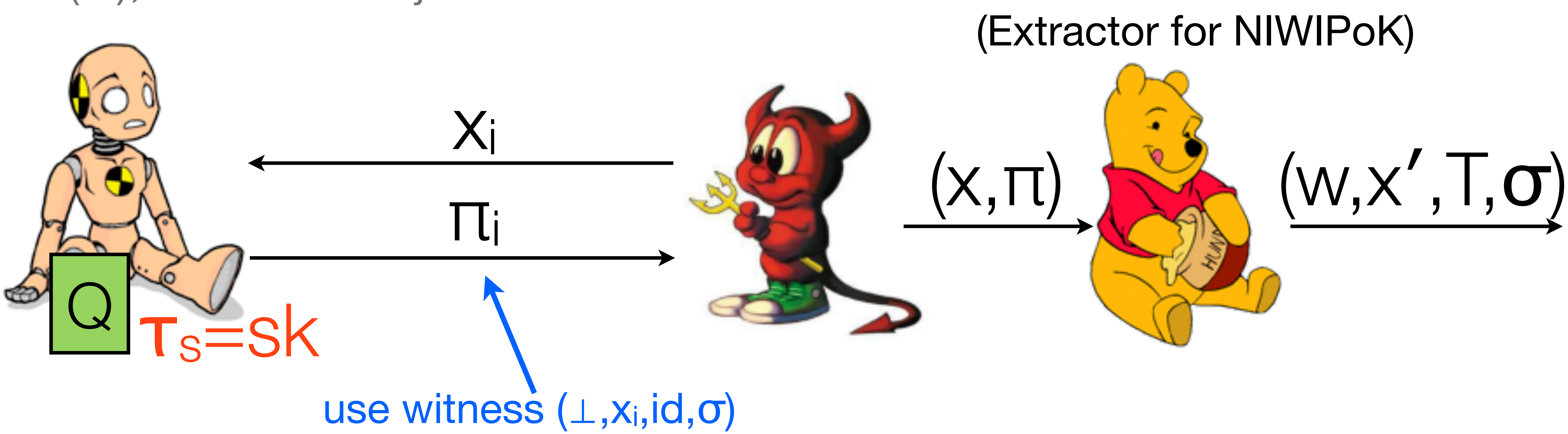


A wins if (1) w≠⊥ but isn't a valid witness, (2) (x′,T)≠(⊥,⊥) but x′∉Q, x≠T(x′), or T is not in $\mathcal{T}$, or (3) (w,x′,T)=(⊥,⊥,⊥)

How to construct cm-NIZKs

We will combine malleable NIWIPoKs with unforgeable signatures

cm-NIZK(x,w) = NIWIPoK{(x,(w,x′,T,σ)) s.t. either (x,w)∈R or Verify(vk,x′,σ)=1, x=T(x′), and T is in $\mathcal{T}$}



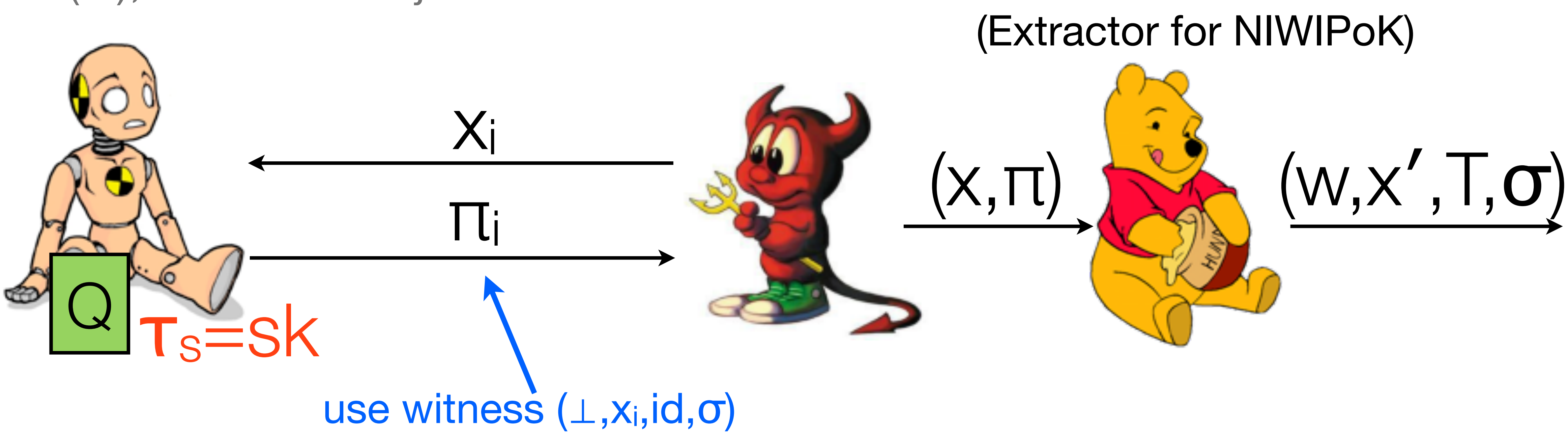
A wins if (1) w≠⊥ but isn't a valid witness, (2) (x′,T)≠(⊥,⊥) but x′∉Q, x≠T(x′), or T is not in $\mathcal{T}$, or (3) (w,x′,T)=(⊥,⊥,⊥)

violates extractability

How to construct cm-NIZKs

We will combine malleable NIWIPoKs with unforgeable signatures

cm-NIZK(x,w) = NIWIPoK{(x,(w,x′,T,σ)) s.t. either (x,w)∈R or Verify(vk,x′,σ)=1, x=T(x′), and T is in $\mathcal{J}$}

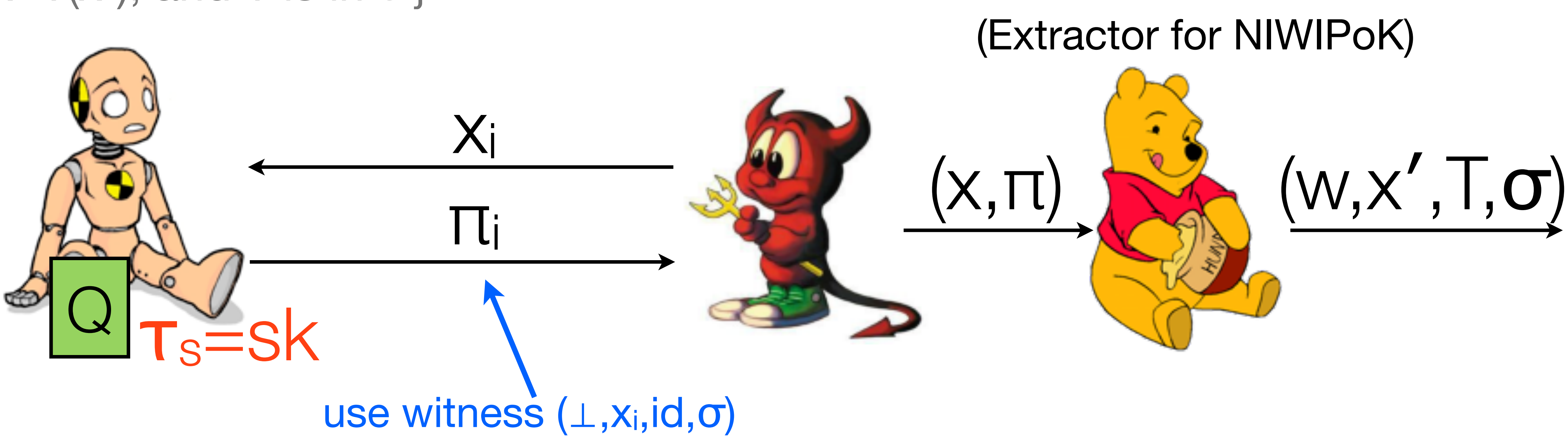


A wins if (1) w≠⊥ but isn't a valid witness, (2) (x′,T)≠(⊥,⊥) but x′∉Q, x≠T(x′), or T is not in $\mathcal{J}$, or (3) (w,x′,T)=(⊥,⊥,⊥)

How to construct cm-NIZKs

We will combine malleable NIWIPoKs with unforgeable signatures

cm-NIZK(x,w) = NIWIPoK{(x,(w,x′,T,σ)) s.t. either (x,w)∈R or (Verify(vk,x′,σ)=1, x=T(x′), and T is in $\mathscr{T}$)}



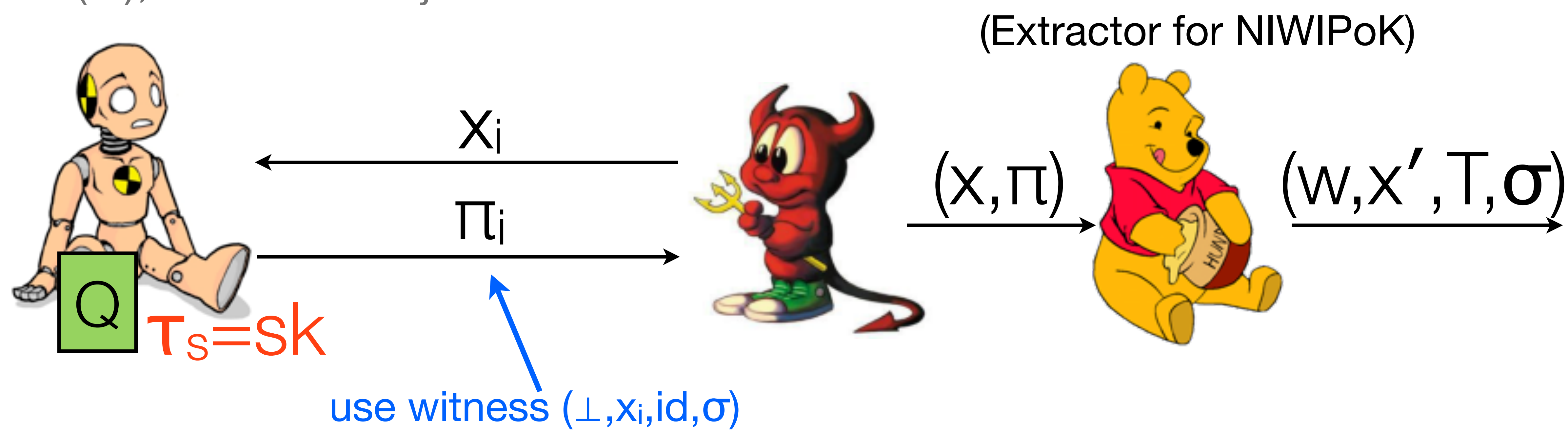
A wins if (1) w≠⊥ but isn't a valid witness, (2) (x′,T)≠(⊥,⊥) but x′∉Q, x≠T(x′), or T is not in $\mathscr{T}$, or (3) (w,x′,T)=(⊥,⊥,⊥)

violates extractability

How to construct cm-NIZKs

We will combine malleable NIWIPoKs with unforgeable signatures

cm-NIZK(x,w) = NIWIPoK{(x,(w,x′,T,σ)) s.t. either (x,w)∈R or Verify(vk,x′,σ)=1, x=T(x′), and T is in $\mathcal{T}$}



A wins if (1) w≠⊥ but isn't a valid witness, (2) (x′,T)≠(⊥,⊥) but x′∉Q, x≠T(x′), or T is not in $\mathcal{T}$, or (3) (w,x′,T)=(⊥,⊥,⊥)

violates extractability
violates unforgeability

Instantiating this (relatively) efficiently

Instantiating this (relatively) efficiently

For the NIWIPoK, we use Groth-Sahai proofs [GS08]

Instantiating this (relatively) efficiently

For the NIWIPoK, we use Groth-Sahai proofs [GS08]

For the signature, we need a structure-preserving signature [AFGHO10,CK11] to integrate with GS proofs (verifying signature = verifying set of pairing product equations), this means we can instantiate based solely on Decision Linear

Instantiating this (relatively) efficiently

For the NIWIPoK, we use Groth-Sahai proofs [GS08]

For the signature, we need a structure-preserving signature [AFGHO10,CK11] to integrate with GS proofs (verifying signature = verifying set of pairing product equations), this means we can instantiate based solely on Decision Linear

The efficiency of our scheme hinges on the efficiency of the signature and the representation of the transformation (depends on the transformation)

Instantiating this (relatively) efficiently

For the NIWIPoK, we use Groth-Sahai proofs [GS08]

For the signature, we need a structure-preserving signature [AFGHO10,CK11] to integrate with GS proofs (verifying signature = verifying set of pairing product equations), this means we can instantiate based solely on Decision Linear

The efficiency of our scheme hinges on the efficiency of the signature and the representation of the transformation (depends on the transformation)

For the class of transformations, need it to contain the identity (for simulation) and be closed under composition (for compactness): given proof for $x = T_1(x')$, size won't increase for $T_2(x) = T_2 \circ T_1(x')$

Instantiating this (relatively) efficiently

For the NIWIPoK, we use Groth-Sahai proofs [GS08]

For the signature, we need a structure-preserving signature [AFGHO10,CK11] to integrate with GS proofs (verifying signature = verifying set of pairing product equations), this means we can instantiate based solely on Decision Linear

The efficiency of our scheme hinges on the efficiency of the signature and the representation of the transformation (depends on the transformation)

For the class of transformations, need it to contain the identity (for simulation) and be closed under composition (for compactness): given proof for $x = T_1(x')$, size won't increase for $T_2(x) = T_2 \circ T_1(x')$

In the paper, we examine the many ways in which GS proofs are malleable

Outline

Definitions

cm-NIZK construction

Applications
Boosting encryption security
Compactly verifiable shuffles

Conclusions

CM-CCA security

CM-CCA security

Expand our notion of controlled malleability from proofs to encryption to get CM-CCA security (inspired by HCCA [PR08] and related to targeted malleability [BSW12])

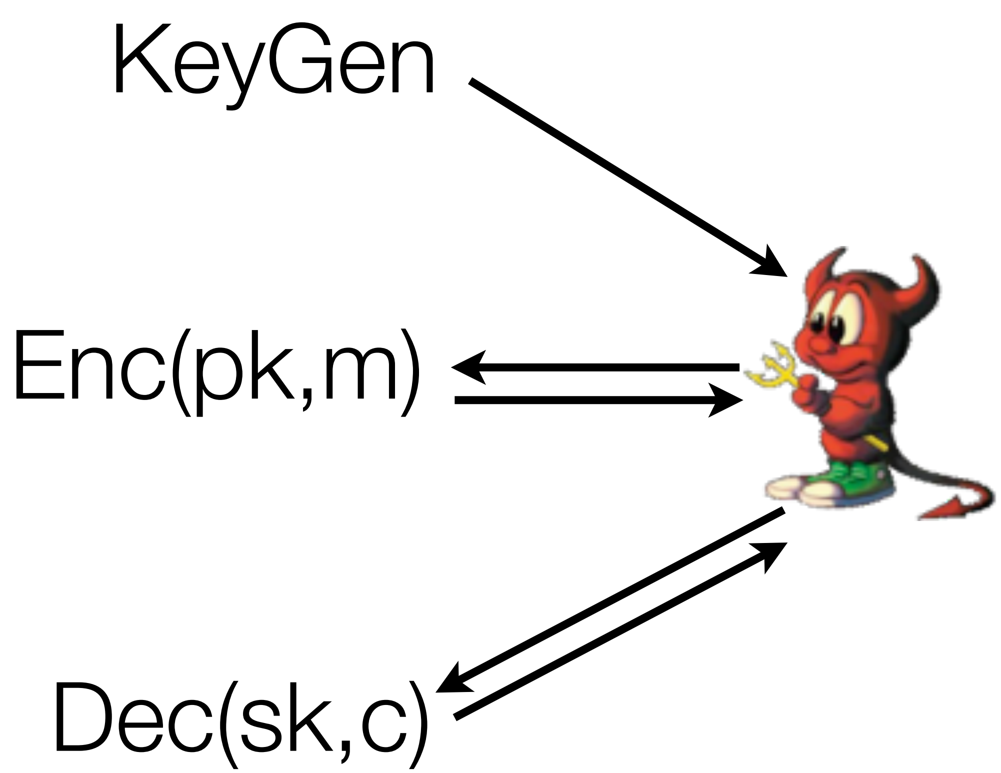
CM-CCA security

Expand our notion of controlled malleability from proofs to encryption to get CM-CCA security (inspired by HCCA [PR08] and related to targeted malleability [BSW12])

CM-CCA security

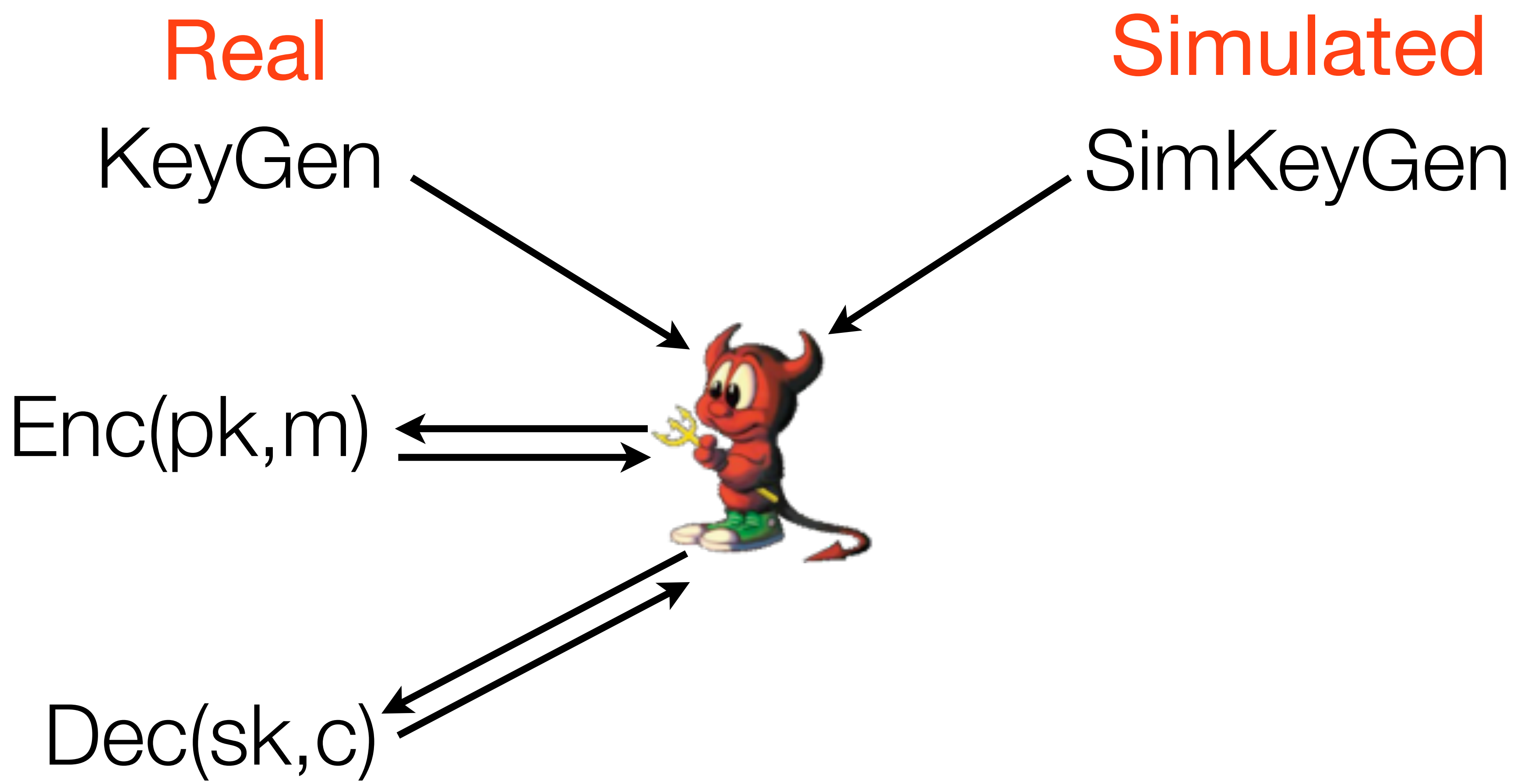
Expand our notion of controlled malleability from proofs to encryption to get CM-CCA security (inspired by HCCA [PR08] and related to targeted malleability [BSW12])

Real



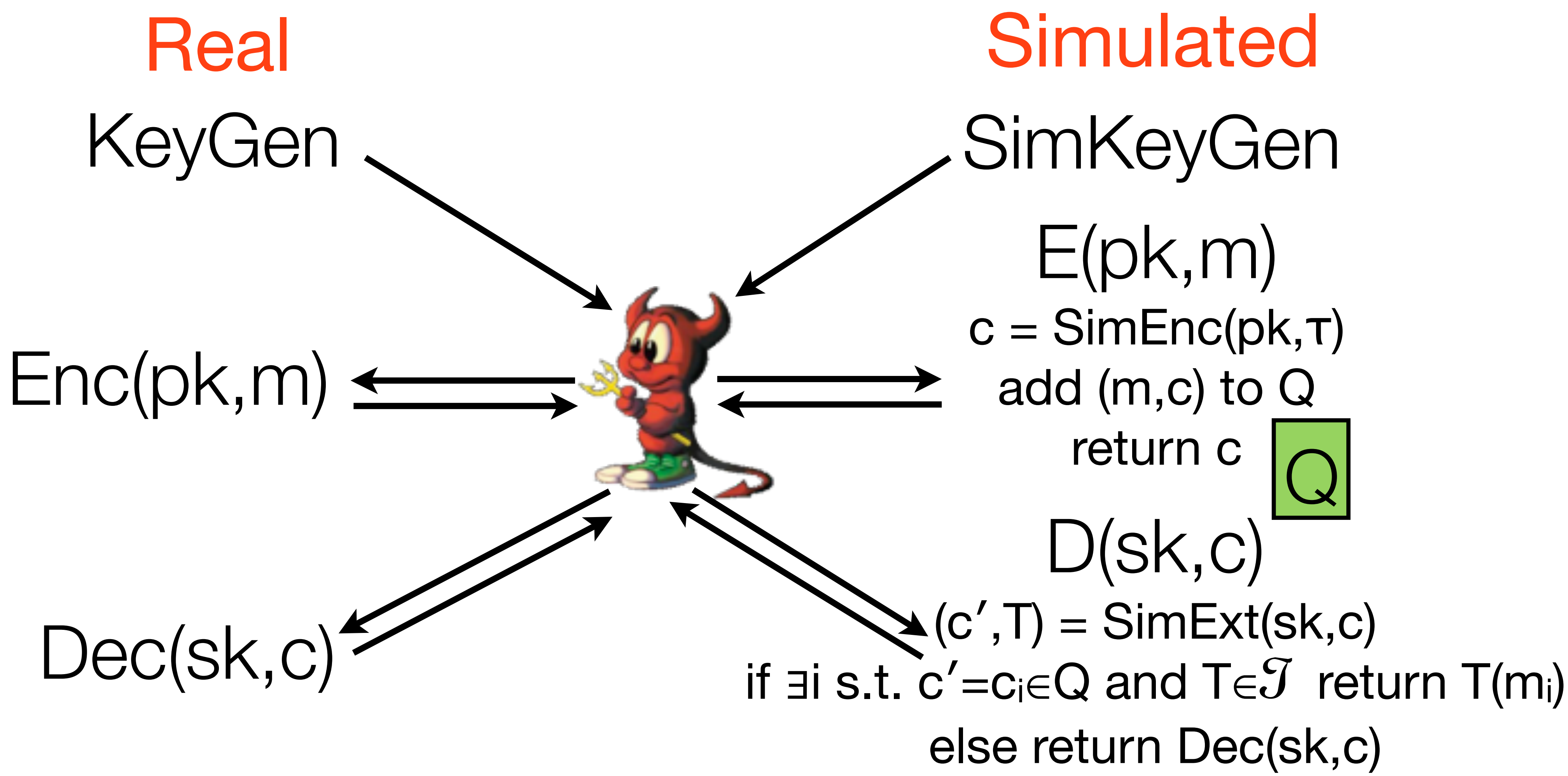
CM-CCA security

Expand our notion of controlled malleability from proofs to encryption to get CM-CCA security (inspired by HCCA [PR08] and related to targeted malleability [BSW12])



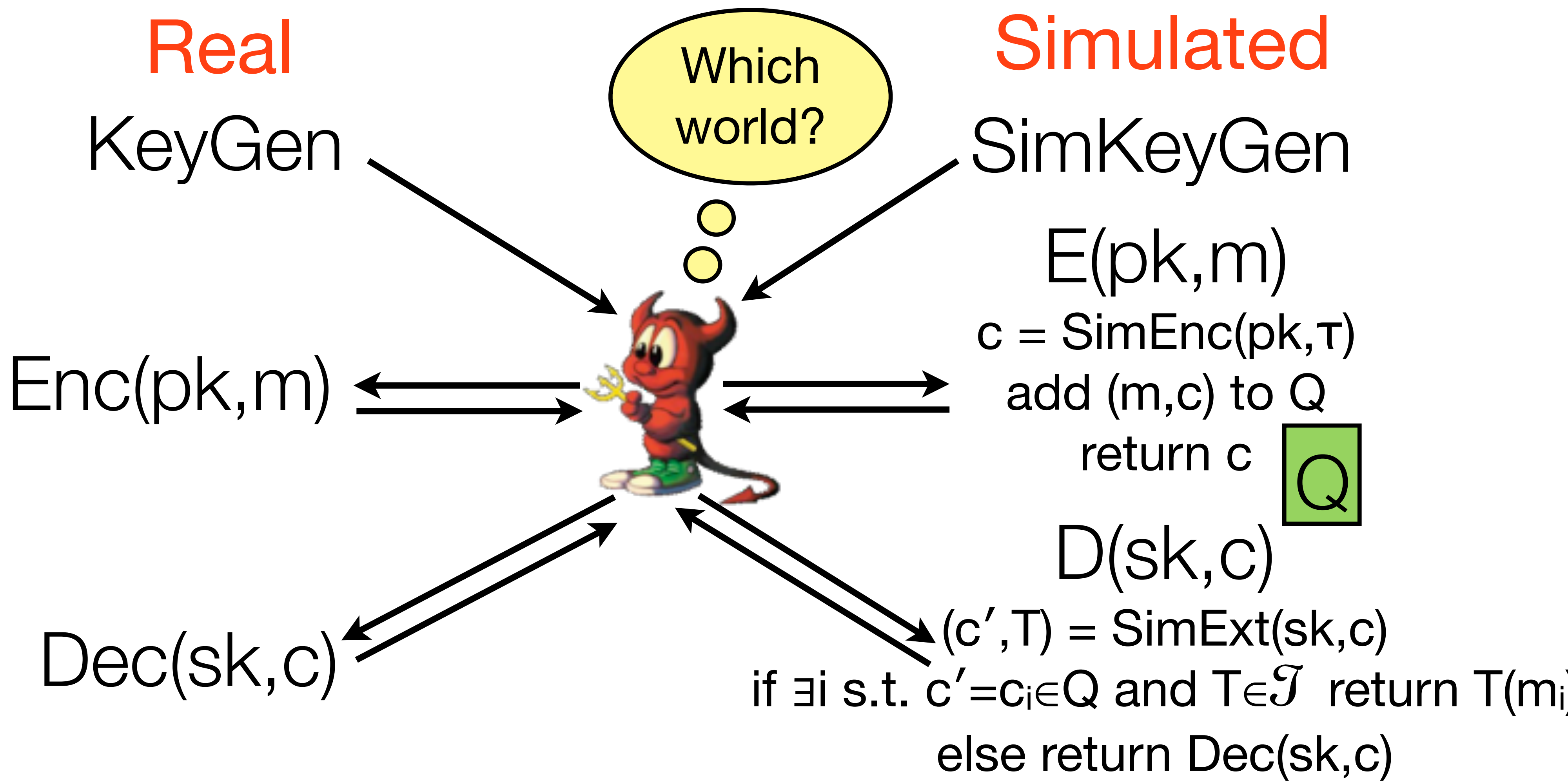
CM-CCA security

Expand our notion of controlled malleability from proofs to encryption to get CM-CCA security (inspired by HCCA [PR08] and related to targeted malleability [BSW12])



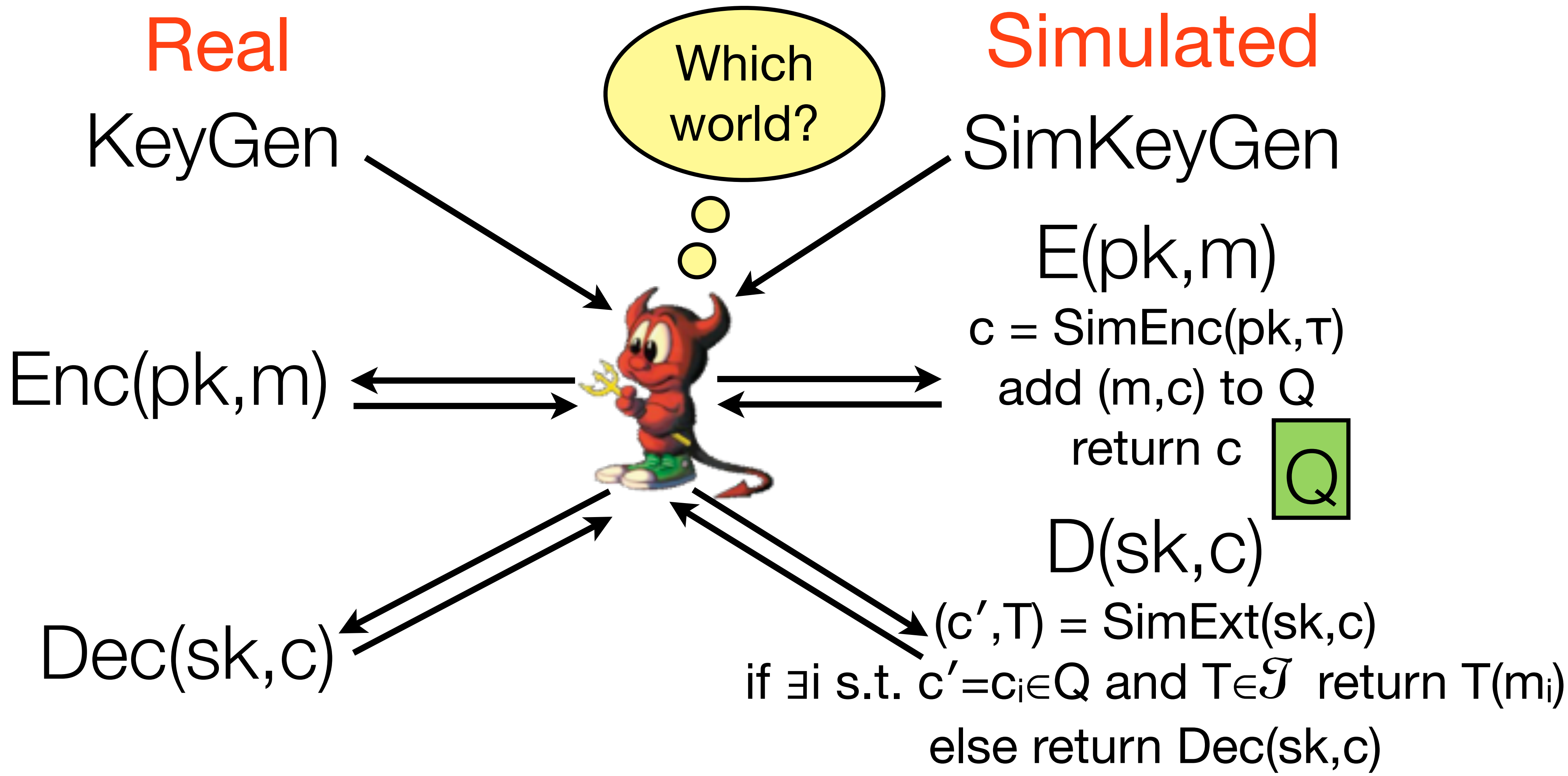
CM-CCA security

Expand our notion of controlled malleability from proofs to encryption to get CM-CCA security (inspired by HCCA [PR08] and related to targeted malleability [BSW12])



CM-CCA security

Expand our notion of controlled malleability from proofs to encryption to get CM-CCA security (inspired by HCCA [PR08] and related to targeted malleability [BSW12])



Give a generic construction for achieving CM-CCA-secure encryption: just define $Enc(pk,m) = (c,\pi)$, where c is IND-CPA-secure and π is a cm-NIZK

A shuffle

A shuffle

c_1
c_2
c_3
c_4
c_5

Users encrypt their individual values to yield a public set of ciphertexts $\{c_i\}$

A shuffle

c_1
c_2
c_3
c_4
c_5

Users encrypt their individual values to yield a public set of ciphertexts $\{c_i\}$

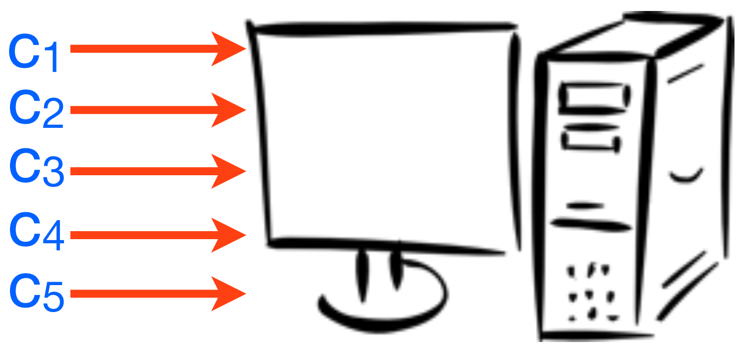
A shuffle

c_1
c_2
c_3
c_4
c_5

Users encrypt their individual values to yield a public set of ciphertexts $\{c_i\}$

Individual mix servers permute and re-randomize ciphertexts

A shuffle



Users encrypt their individual values to yield a public set of ciphertexts $\{c_i\}$

Individual mix servers permute and re-randomize ciphertexts

A shuffle

Users encrypt their individual values to yield a public set of ciphertexts $\{c_i\}$

Individual mix servers permute and re-randomize ciphertexts

A shuffle

Users encrypt their individual values to yield a public set of ciphertexts $\{c_i\}$

Individual mix servers permute and re-randomize ciphertexts

A shuffle

Users encrypt their individual values to yield a public set of ciphertexts $\{c_i\}$

Individual mix servers permute and re-randomize ciphertexts

A shuffle

Users encrypt their individual values to yield a public set of ciphertexts $\{c_i\}$

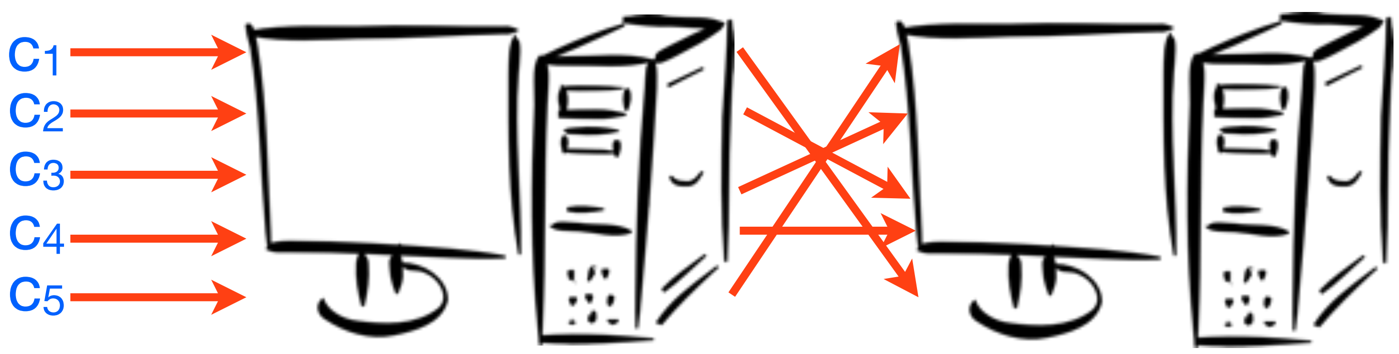
Individual mix servers permute and re-randomize ciphertexts

A shuffle

Users encrypt their individual values to yield a public set of ciphertexts $\{c_i\}$

Individual mix servers permute and re-randomize ciphertexts

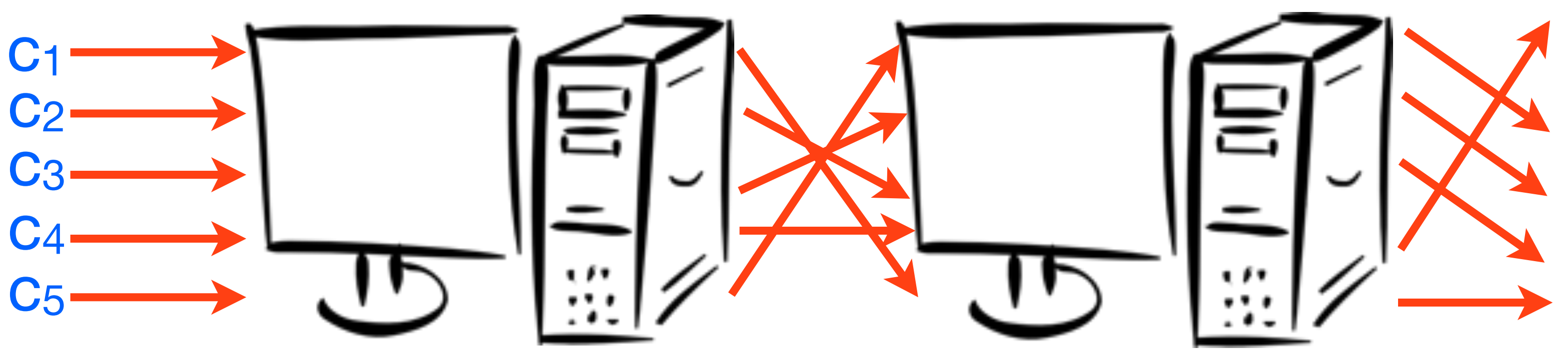
A shuffle



Users encrypt their individual values to yield a public set of ciphertexts $\{c_i\}$

Individual mix servers permute and re-randomize ciphertexts

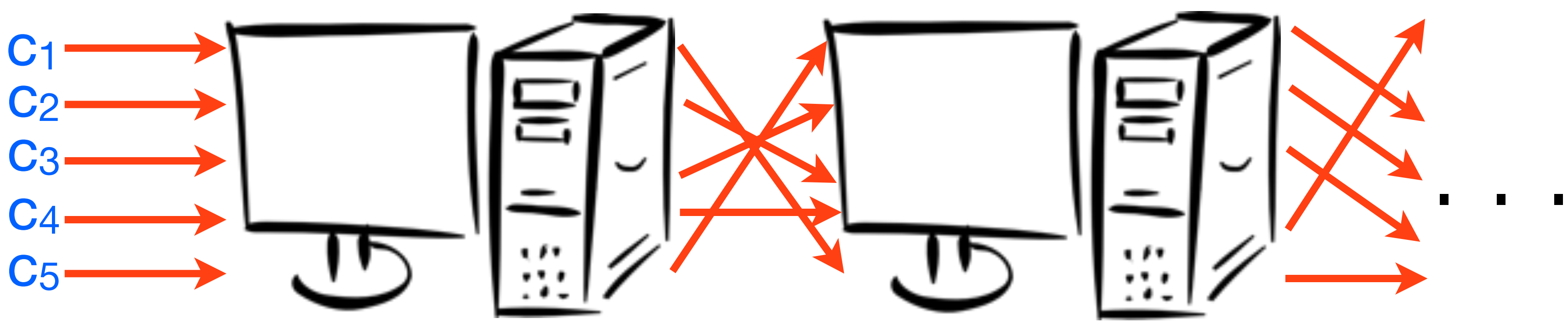
A shuffle



Users encrypt their individual values to yield a public set of ciphertexts $\{c_i\}$

Individual mix servers permute and re-randomize ciphertexts

A shuffle



Users encrypt their individual values to yield a public set of ciphertexts $\{c_i\}$

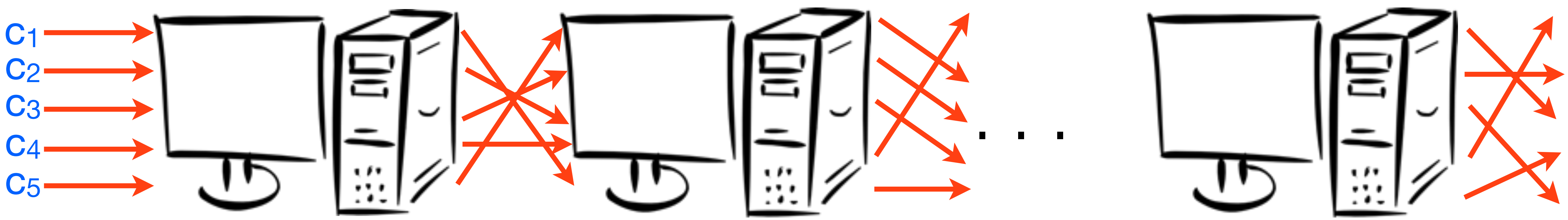
Individual mix servers permute and re-randomize ciphertexts

A shuffle

Users encrypt their individual values to yield a public set of ciphertexts $\{c_i\}$

Individual mix servers permute and re-randomize ciphertexts

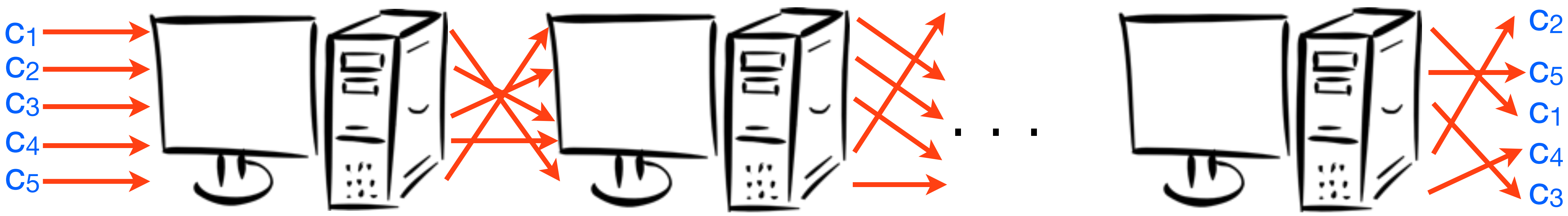
A shuffle



Users encrypt their individual values to yield a public set of ciphertexts $\{c_i\}$

Individual mix servers permute and re-randomize ciphertexts

A shuffle

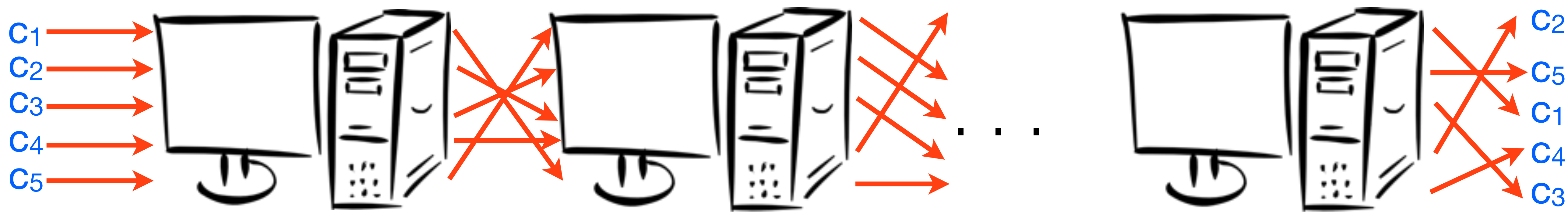


Users encrypt their individual values to yield a public set of ciphertexts $\{c_i\}$

Individual mix servers permute and re-randomize ciphertexts

Final outcome is a set of ciphertexts

A shuffle



Users encrypt their individual values to yield a public set of ciphertexts $\{c_i\}$

Individual mix servers permute and re-randomize ciphertexts

Final outcome is a set of ciphertexts

Because values are shuffled, decryption won't reveal whose vote is whose

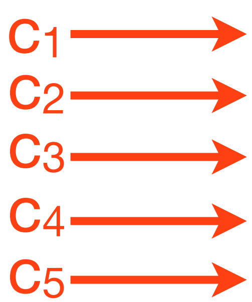
A verifiable shuffle [SK95,...,GL07]

A verifiable shuffle [SK95,...,GL07]

Problem: How do we know these mix servers are behaving honestly?

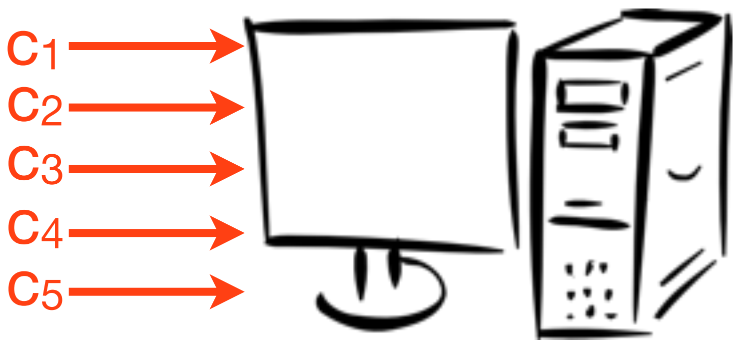
A verifiable shuffle [SK95,...,GL07]

Problem: How do we know these mix servers are behaving honestly?



A verifiable shuffle [SK95,...,GL07]

Problem: How do we know these mix servers are behaving honestly?

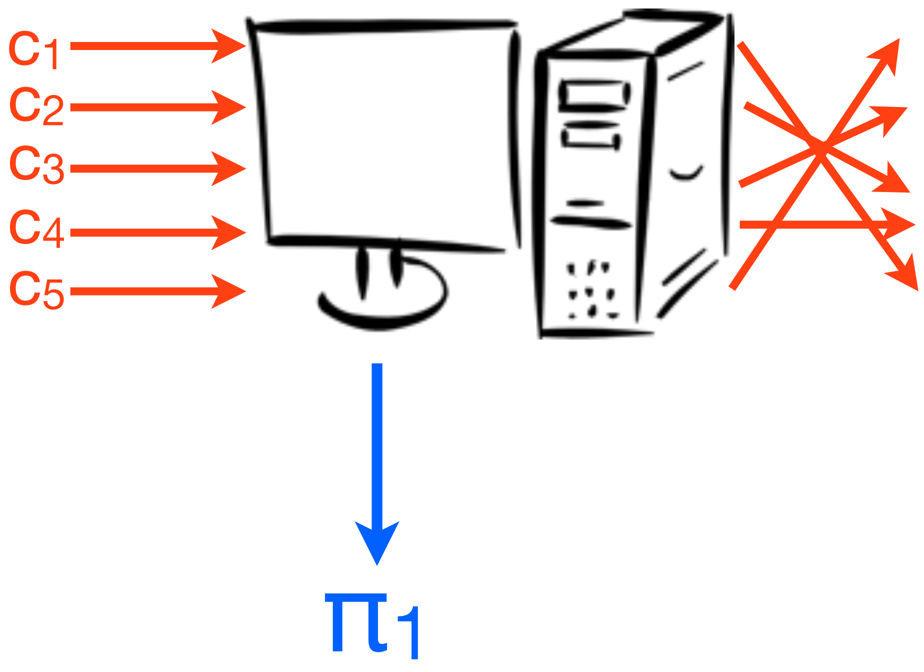


A verifiable shuffle [SK95,...,GL07]

Problem: How do we know these mix servers are behaving honestly?

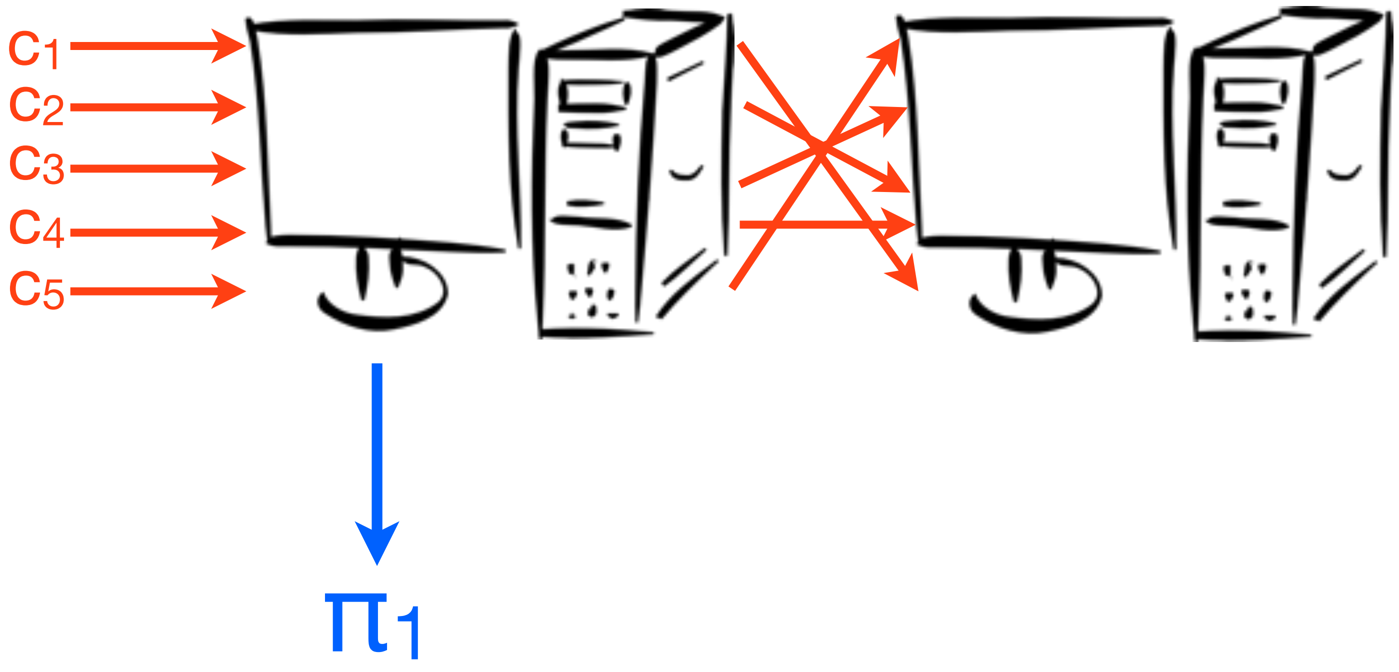
A verifiable shuffle [SK95,...,GL07]

Problem: How do we know these mix servers are behaving honestly?



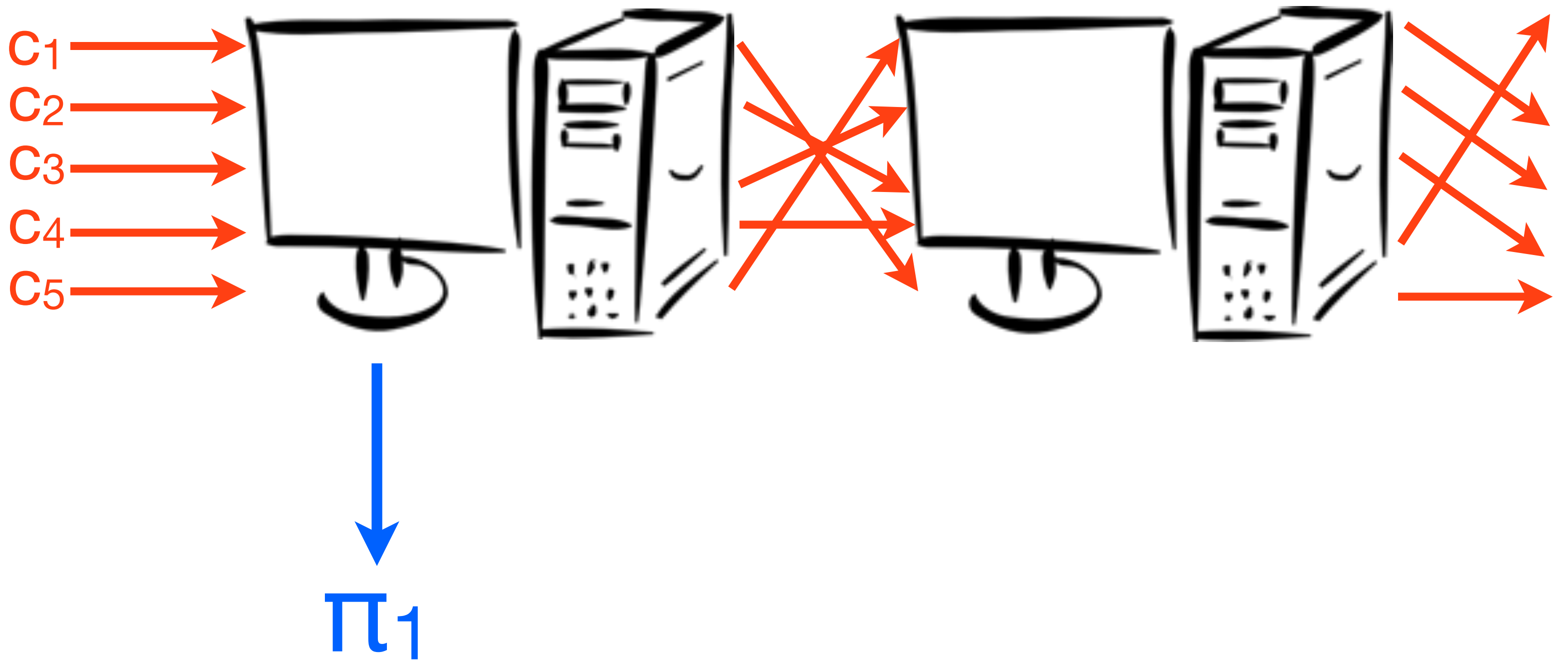
A verifiable shuffle [SK95,...,GL07]

Problem: How do we know these mix servers are behaving honestly?



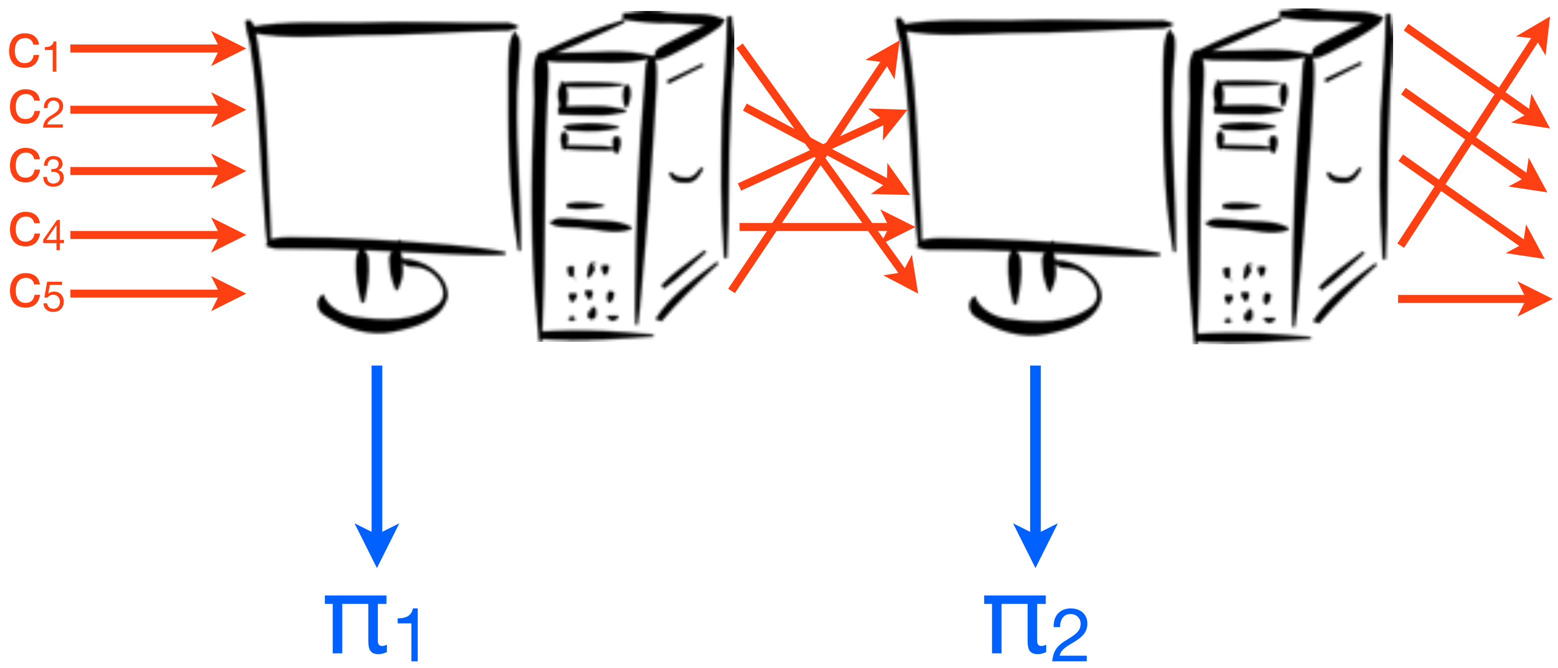
A verifiable shuffle [SK95,...,GL07]

Problem: How do we know these mix servers are behaving honestly?



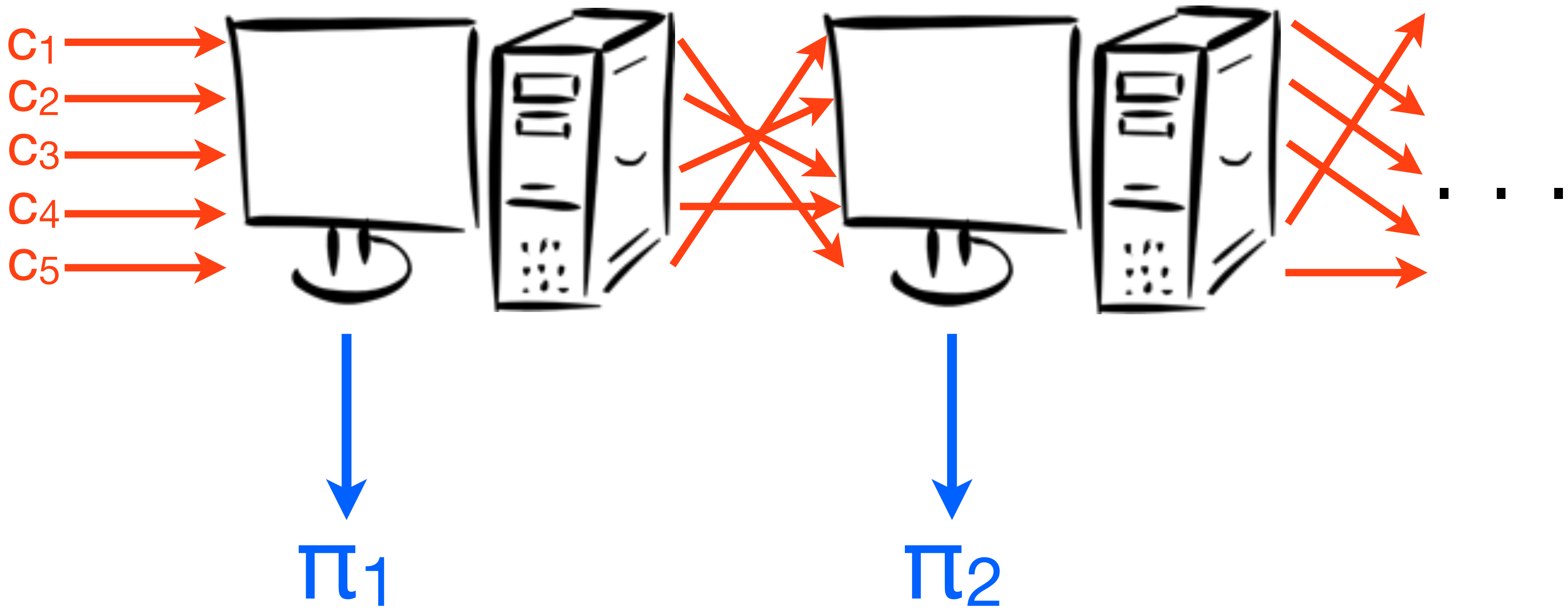
A verifiable shuffle [SK95,...,GL07]

Problem: How do we know these mix servers are behaving honestly?



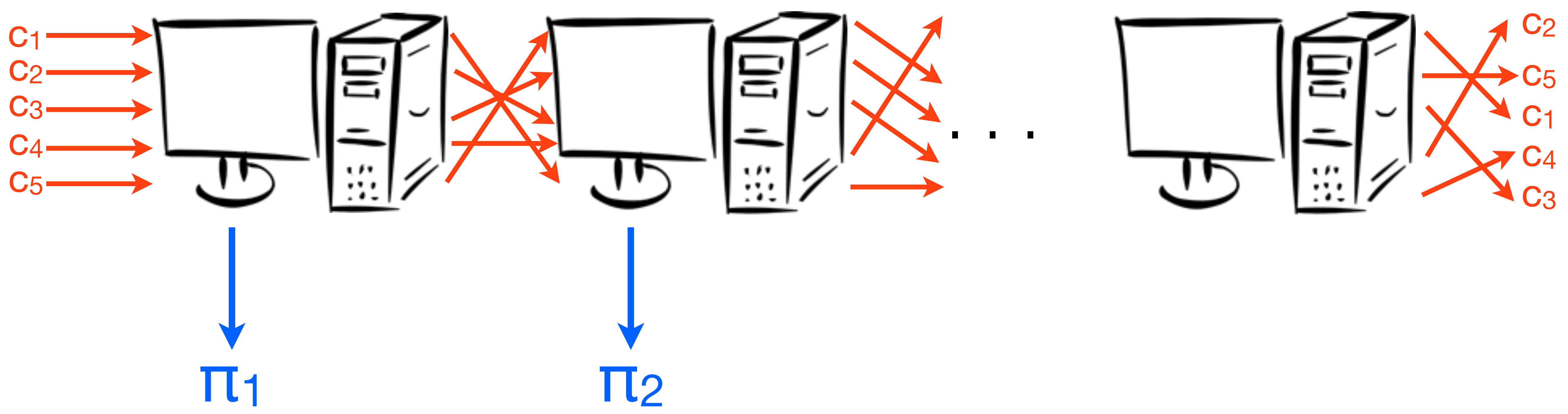
A verifiable shuffle [SK95,...,GL07]

Problem: How do we know these mix servers are behaving honestly?



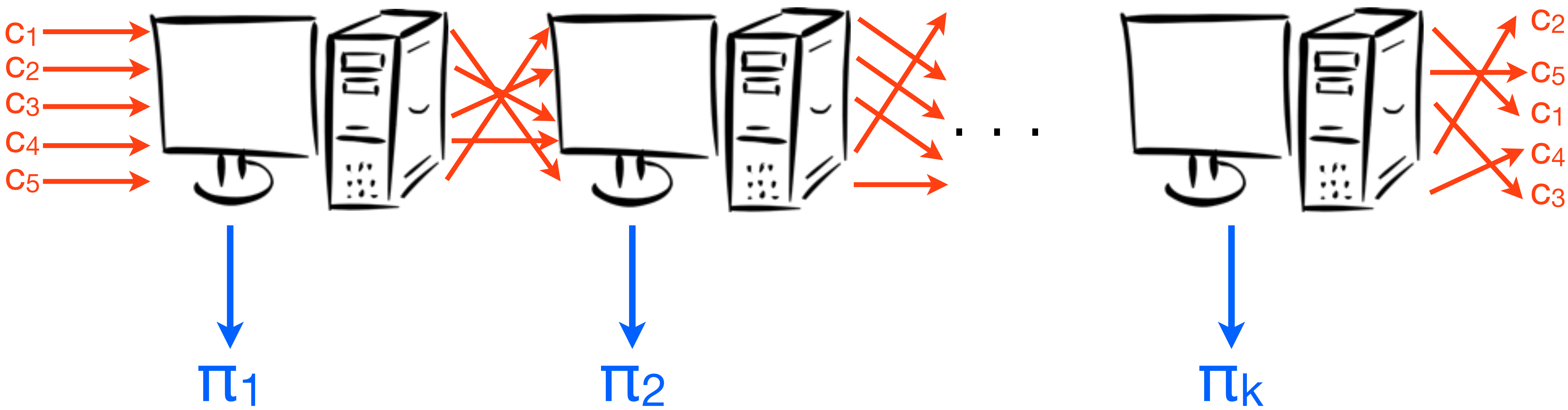
A verifiable shuffle [SK95,...,GL07]

Problem: How do we know these mix servers are behaving honestly?



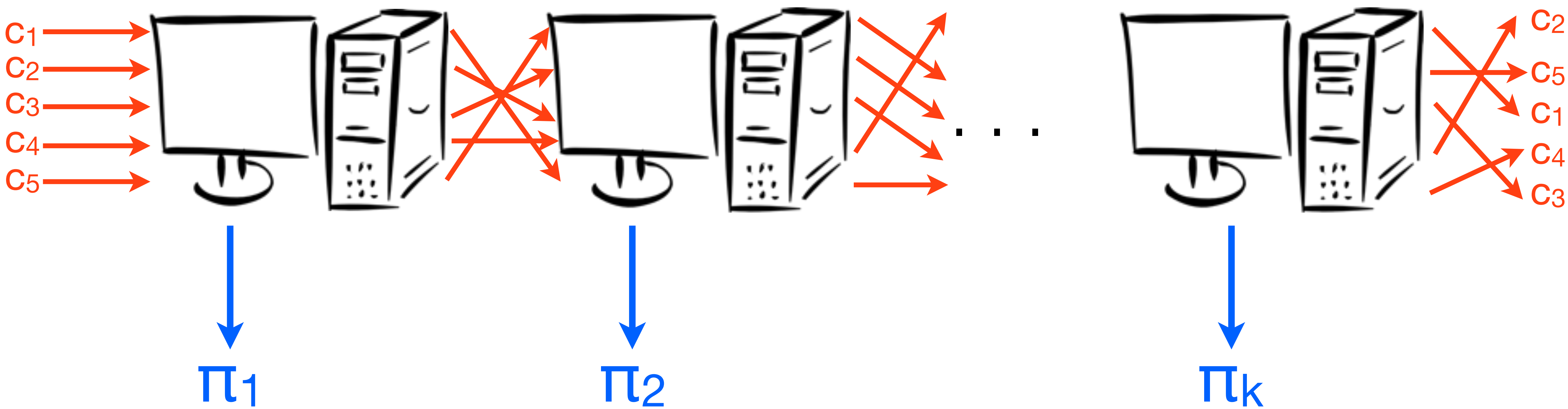
A verifiable shuffle [SK95,...,GL07]

Problem: How do we know these mix servers are behaving honestly?



A verifiable shuffle [SK95,...,GL07]

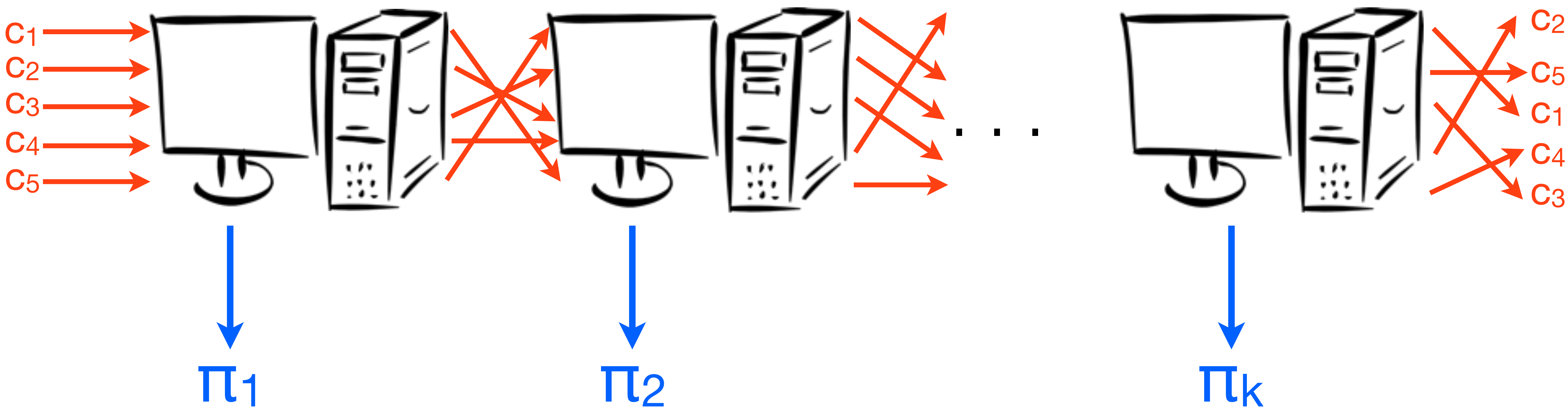
Problem: How do we know these mix servers are behaving honestly?



Each server now proves that it is honestly shuffling the ciphertexts, and so the shuffle is said to be verifiable

A verifiable shuffle [SK95,...,GL07]

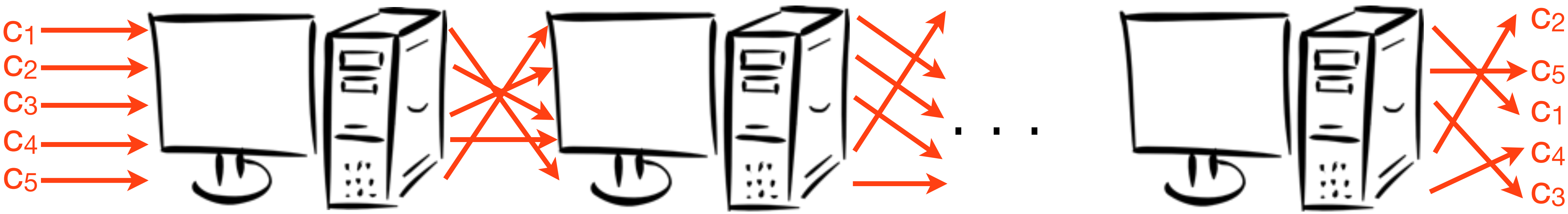
Problem: How do we know these mix servers are behaving honestly?



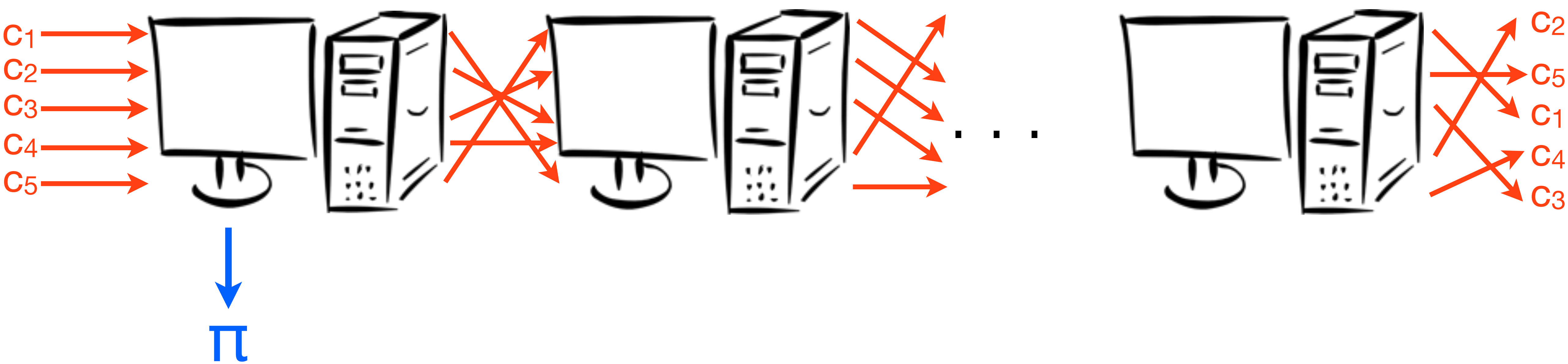
Each server now proves that it is honestly shuffling the ciphertexts, and so the shuffle is said to be verifiable

New problem: The size of this proof grows with the number of mix servers

Using malleability to shrink the overall proof size

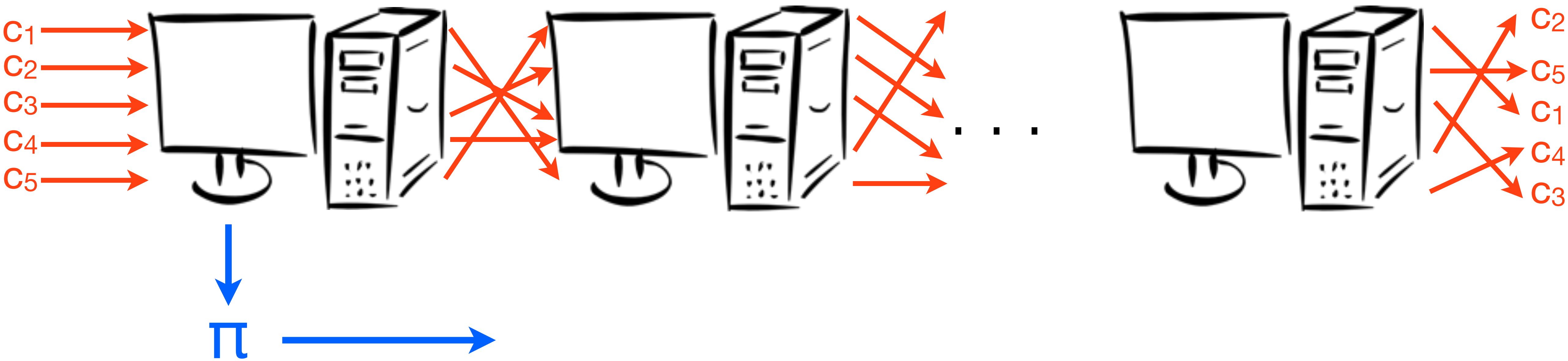


Using malleability to shrink the overall proof size



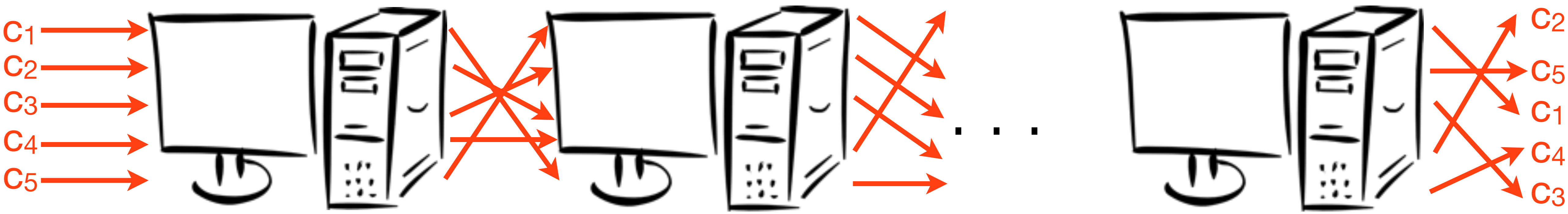
Initial mix server still outputs a fresh proof π, but now subsequent servers will "maul" this proof using permutation ϕ_i, re-randomization R_i, and public key pk_i

Using malleability to shrink the overall proof size



Initial mix server still outputs a fresh proof π, but now subsequent servers will "maul" this proof using permutation ϕ_i, re-randomization R_i, and public key pk_i

Using malleability to shrink the overall proof size

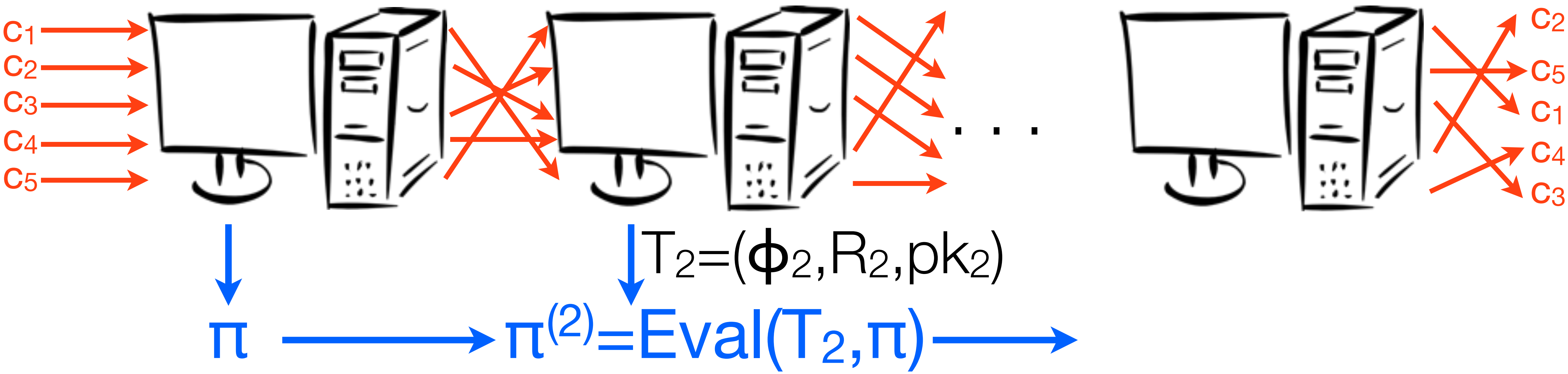


$T_2=(\phi_2,R_2,pk_2)$

$\pi \longrightarrow \pi^{(2)}=\text{Eval}(T_2,\pi)$

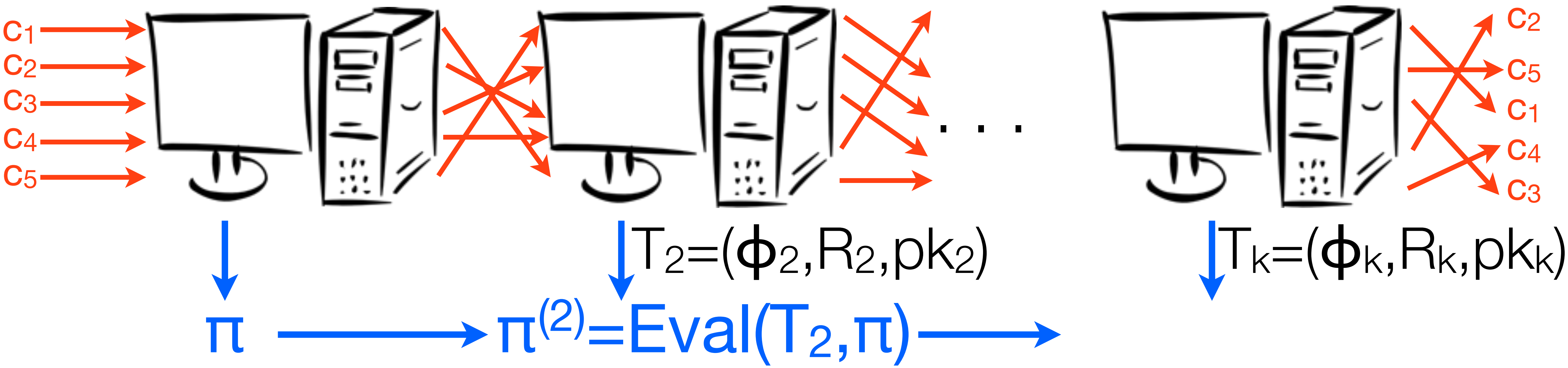
Initial mix server still outputs a fresh proof π, but now subsequent servers will "maul" this proof using permutation ϕ_i, re-randomization R_i, and public key pk_i

Using malleability to shrink the overall proof size



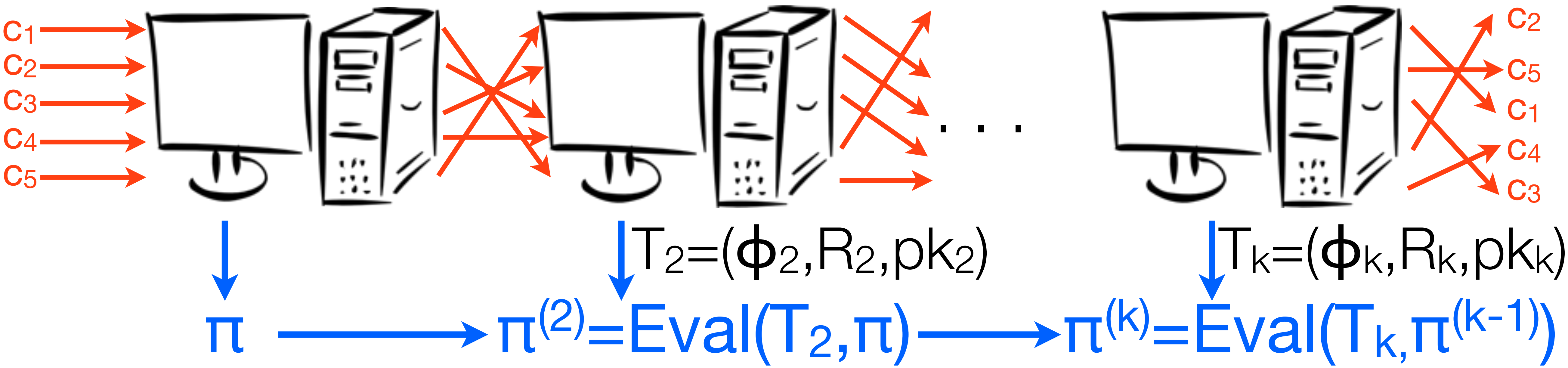
Initial mix server still outputs a fresh proof π, but now subsequent servers will "maul" this proof using permutation ϕ_i, re-randomization R_i, and public key pk_i

Using malleability to shrink the overall proof size



Initial mix server still outputs a fresh proof π, but now subsequent servers will "maul" this proof using permutation ϕ_i, re-randomization R_i, and public key pk_i

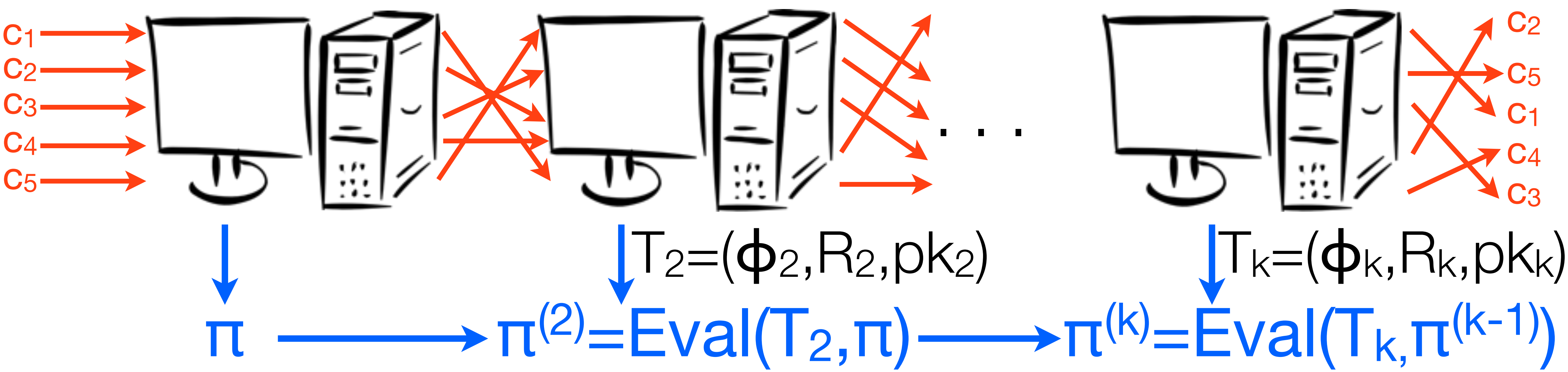
Using malleability to shrink the overall proof size



Initial mix server still outputs a fresh proof π, but now subsequent servers will "maul" this proof using permutation ϕ_i, re-randomization R_i, and public key pk_i

We call this shuffle compactly verifiable, as the last proof $\pi^{(k)}$ can now be used to verify the correctness of the whole shuffle (under an appropriate definition)

Using malleability to shrink the overall proof size



Initial mix server still outputs a fresh proof π, but now subsequent servers will "maul" this proof using permutation ϕ_i, re-randomization R_i, and public key pk_i

We call this shuffle compactly verifiable, as the last proof $\pi^{(k)}$ can now be used to verify the correctness of the whole shuffle (under an appropriate definition)

So if there are n ciphertexts and k servers, proof size can be $O(n+k)$ vs. $O(n*k)$

Using malleability to shrink the overall proof size

$T_2=(\phi_2,R_2,pk_2)$ $T_k=(\phi_k,R_k,pk_k)$

$\pi \longrightarrow \pi^{(2)}=\mathrm{Eval}(T_2,\pi) \longrightarrow \pi^{(k)}=\mathrm{Eval}(T_k,\pi^{(k-1)})$

Initial mix server still outputs a fresh proof π, but now subsequent servers will "maul" this proof using permutation ϕ_i, re-randomization R_i, and public key pk_i

We call this shuffle compactly verifiable, as the last proof $\pi^{(k)}$ can now be used to verify the correctness of the whole shuffle (under an appropriate definition)

So if there are n ciphertexts and k servers, proof size can be $O(n+k)$ vs. $O(n*k)$

- This bound isn't just theoretical: in this paper we get $O(n^2+k)$ but in a recent result we use new methods to achieve $O(n+k)$

Outline

Definitions

cm-NIZK construction

Applications

Conclusions

Conclusions and open problems

Conclusions and open problems

We defined notions of malleability for proof systems

Conclusions and open problems

We defined notions of malleability for proof systems

Saw that there are useful applications: CM-CCA and compact shuffles

Conclusions and open problems

We defined notions of malleability for proof systems

Saw that there are useful applications: CM-CCA and compact shuffles

Saw that Groth-Sahai proofs have meaningful malleability properties

Conclusions and open problems

We defined notions of malleability for proof systems

Saw that there are useful applications: CM-CCA and compact shuffles

Saw that Groth-Sahai proofs have meaningful malleability properties

Did a whole lot more at `eprint.iacr.org/2012/012`!

Conclusions and open problems

We defined notions of malleability for proof systems

Saw that there are useful applications: CM-CCA and compact shuffles

Saw that Groth-Sahai proofs have meaningful malleability properties

Did a whole lot more at `eprint.iacr.org/2012/012`!

Thanks!
Any questions?