Zero-Knowledge Proofs for Verifiable Computation on Encrypted Data

Dario Fiore IMDEA Software Institute

Foundations and Applications of Zero-Knowledge Proofs | Edinburgh, UK | Sep 6, 2024

European Research Council
Established by the European Commission

Agenda

Outsourcing data and computation

Verifiable Computation with Privacy

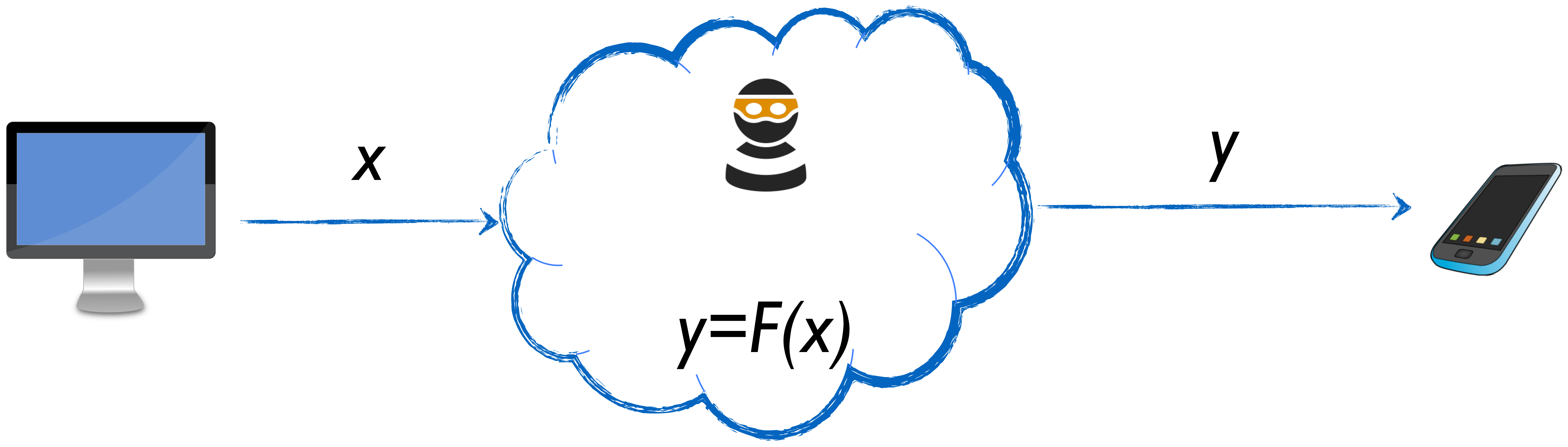
Efficiency challenges of proving FHE computations

SNARK for polynomial rings arithmetic

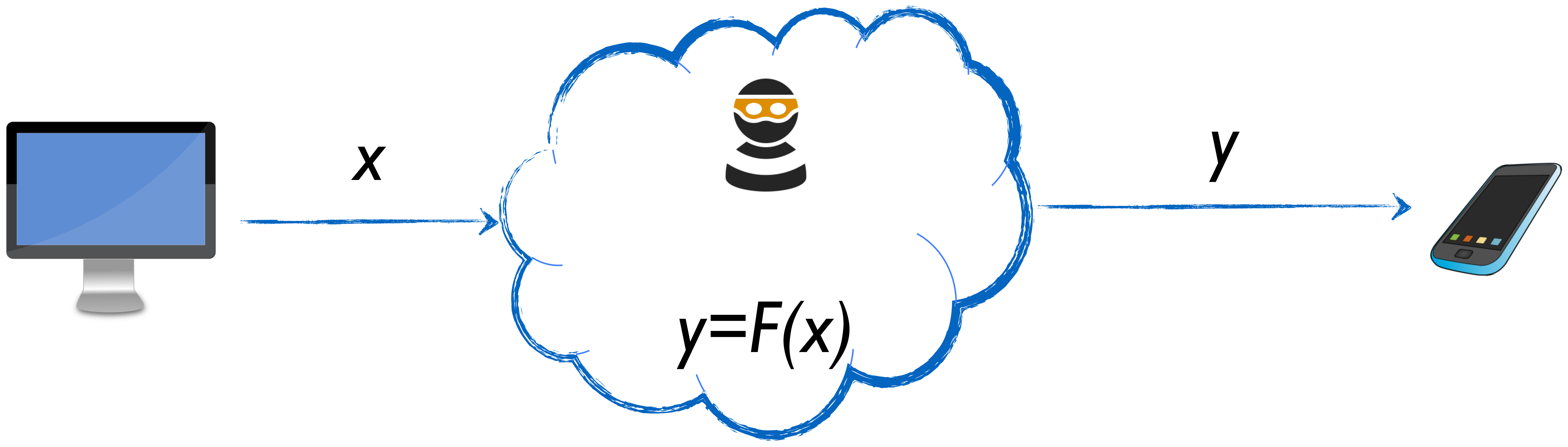
State of the art overview & Conclusions

Motivation: outsourcing data and computation

Motivation: outsourcing data and computation

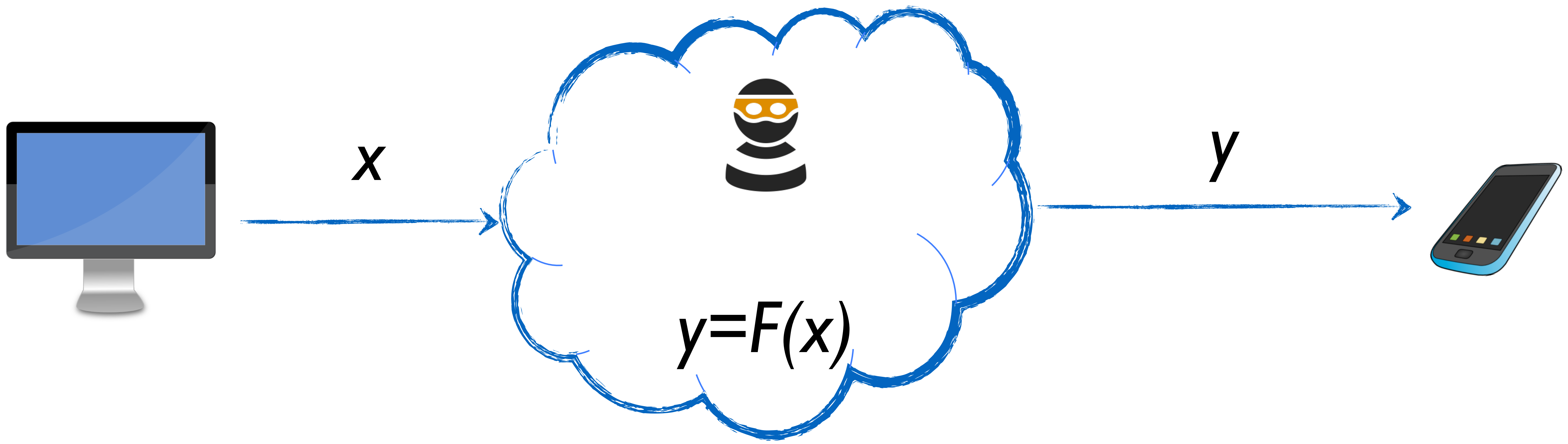


Motivation: outsourcing data and computation



Desired goals:

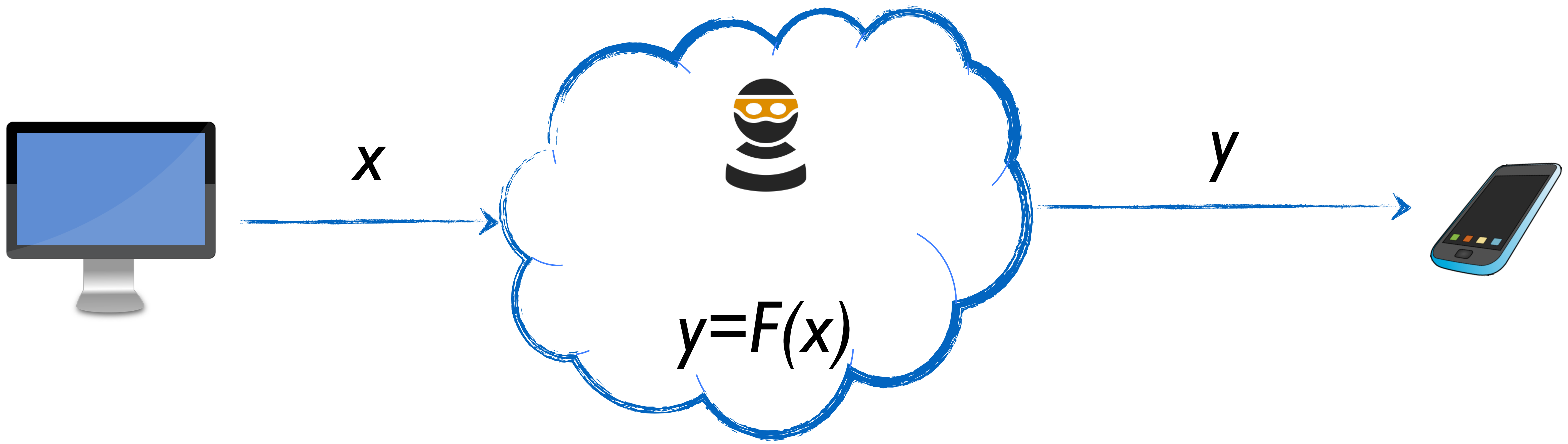
Motivation: outsourcing data and computation



Desired goals:

Integrity: the cloud should not be able to send **incorrect** results

Motivation: outsourcing data and computation

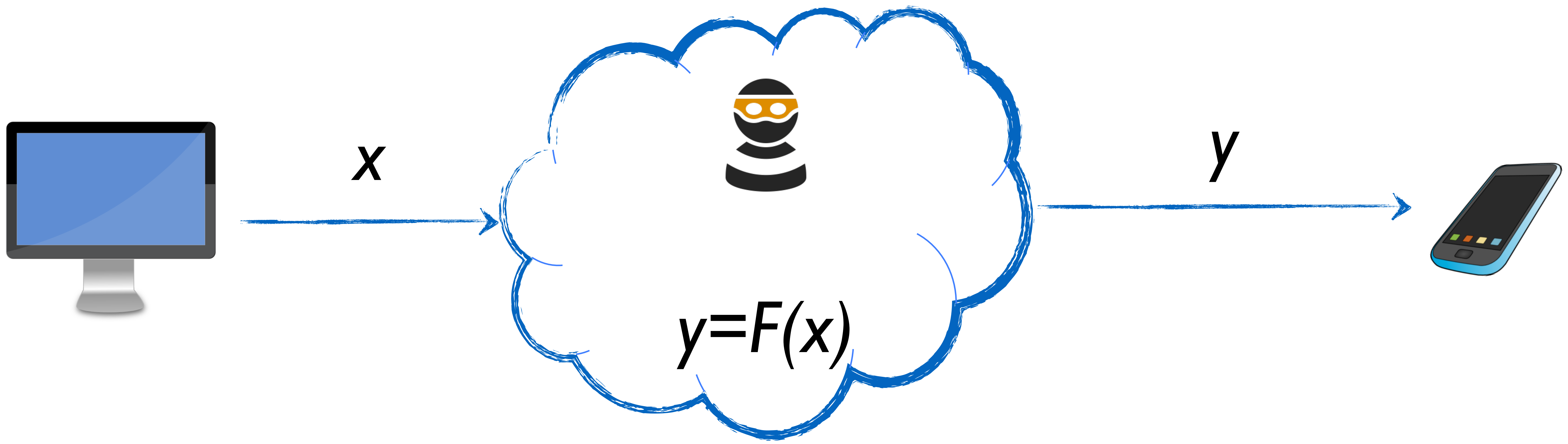


Desired goals:

Integrity: the cloud should not be able to send **incorrect** results

Privacy: the cloud **should not learn information** on the data

Motivation: outsourcing data and computation



Desired goals:

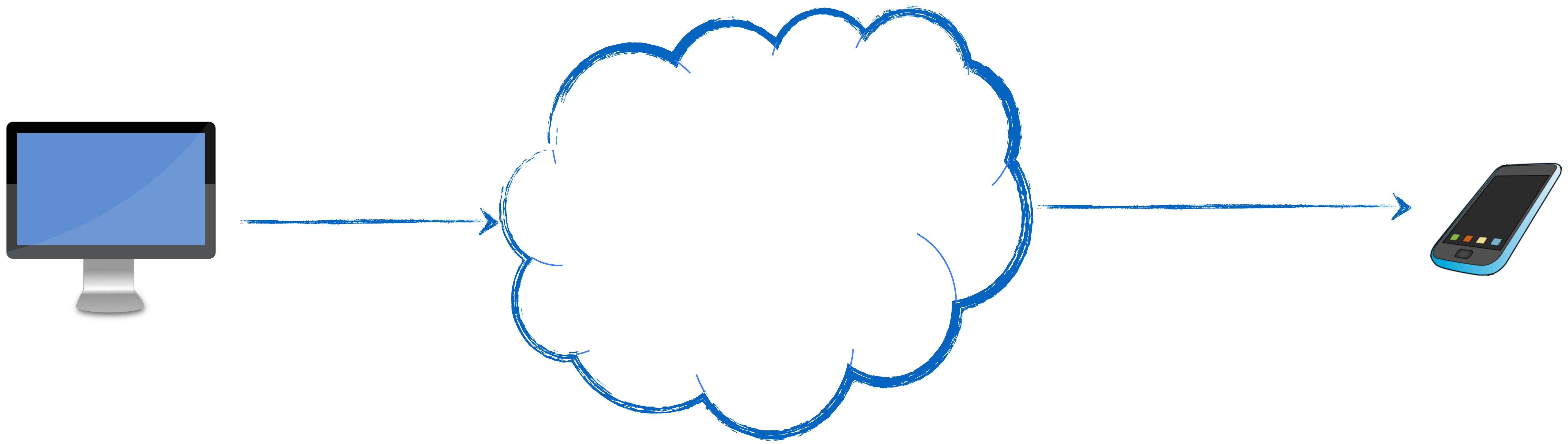
Integrity: the cloud should not be able to send **incorrect** results

Privacy: the cloud **should not learn information** on the data

Efficiency: communication and storage at client "minimal"

Verifiable Computation [GennaroGentryParno10,ParnoRaikovaVainkuntanathan12]

*Here publicly verifiable/delegatable notion



VC Scheme

KeyGen(F) $\rightarrow$ (PK_F, SK_F)

ProbGen(PK_F, x) $\rightarrow$ (σ_x, τ_x)

Compute(PK_F, σ_x) $\rightarrow$ σ_y

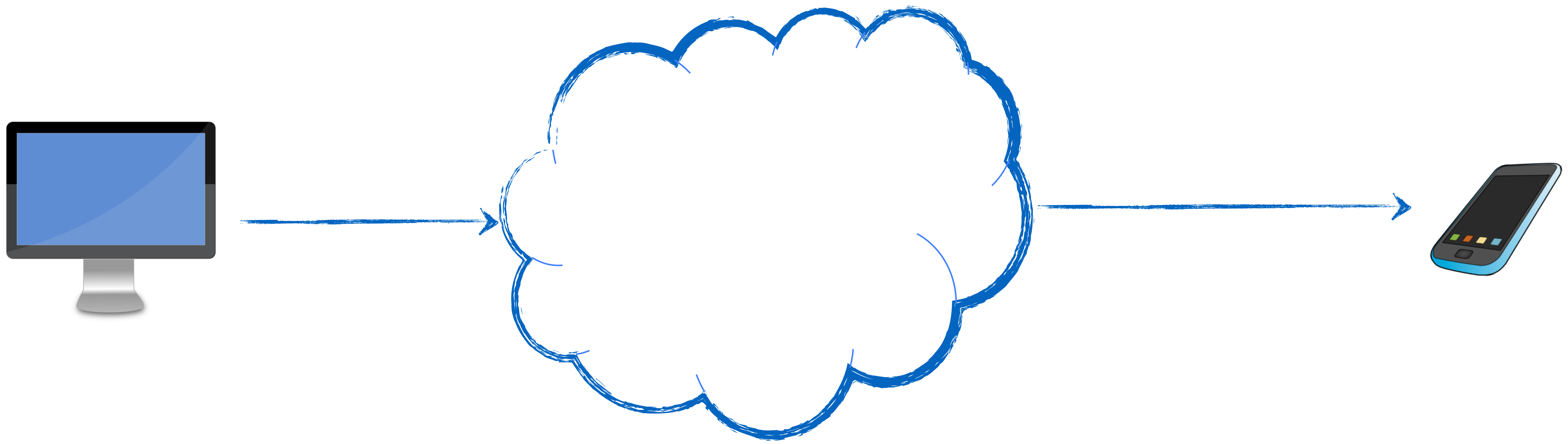
Ver(PK_F, τ_x, σ_y) $\rightarrow$ $acc \in \{0,1\}$

Decode(SK_F, σ_y) $\rightarrow$ y

Verifiable Computation [GennaroGentryParno10,ParnoRaikovaVainkuntanathan12]

*Here publicly verifiable/delegatable notion

$(PK_F, SK_F) \leftarrow \text{KeyGen}(F)$



VC Scheme

$\text{KeyGen}(F) \rightarrow (PK_F, SK_F)$

$\text{ProbGen}(PK_F, x) \rightarrow (\sigma_x, \tau_x)$

$\text{Compute}(PK_F, \sigma_x) \rightarrow \sigma_y$

$\text{Ver}(PK_F, \tau_x, \sigma_y) \rightarrow acc \in \{0,1\}$

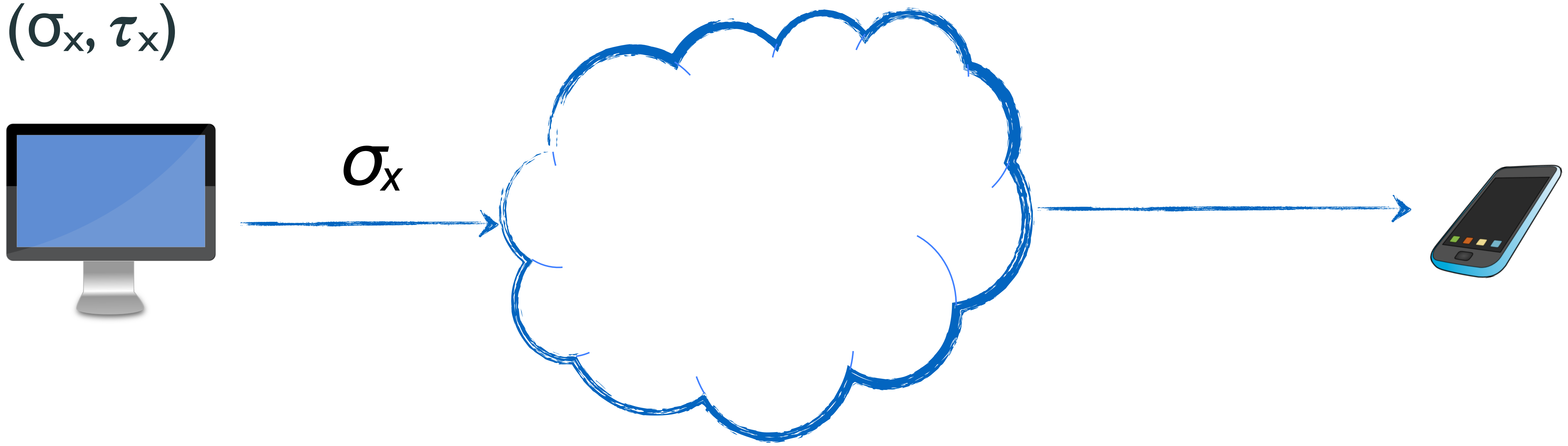
$\text{Decode}(SK_F, \sigma_y) \rightarrow y$

Verifiable Computation [GennaroGentryParno10,ParnoRaikovaVainkuntanathan12]

*Here publicly verifiable/delegatable notion

$(PK_F, SK_F) \leftarrow \text{KeyGen}(F)$

$\text{ProbGen}(PK_F, x) \rightarrow (\sigma_x, \tau_x)$



VC Scheme

$\text{KeyGen}(F) \rightarrow (PK_F, SK_F)$

$\text{ProbGen}(PK_F, x) \rightarrow (\sigma_x, \tau_x)$

$\text{Compute}(PK_F, \sigma_x) \rightarrow \sigma_y$

$\text{Ver}(PK_F, \tau_x, \sigma_y) \rightarrow acc \in \{0,1\}$

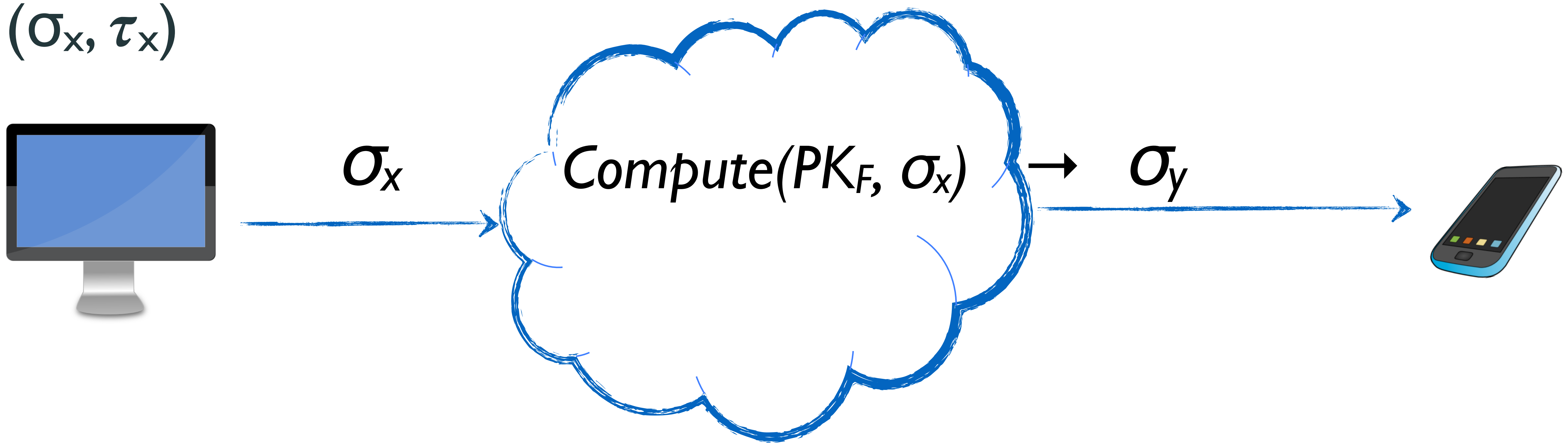
$\text{Decode}(SK_F, \sigma_y) \rightarrow y$

Verifiable Computation [GennaroGentryParno10,ParnoRaikovaVainkuntanathan12]

*Here publicly verifiable/delegatable notion

$(PK_F, SK_F) \leftarrow \text{KeyGen}(F)$

$\text{ProbGen}(PK_F, x) \rightarrow (\sigma_x, \tau_x)$



VC Scheme

$\text{KeyGen}(F) \rightarrow (PK_F, SK_F)$

$\text{ProbGen}(PK_F, x) \rightarrow (\sigma_x, \tau_x)$

$\text{Compute}(PK_F, \sigma_x) \rightarrow \sigma_y$

$\text{Ver}(PK_F, \tau_x, \sigma_y) \rightarrow acc \in \{0,1\}$

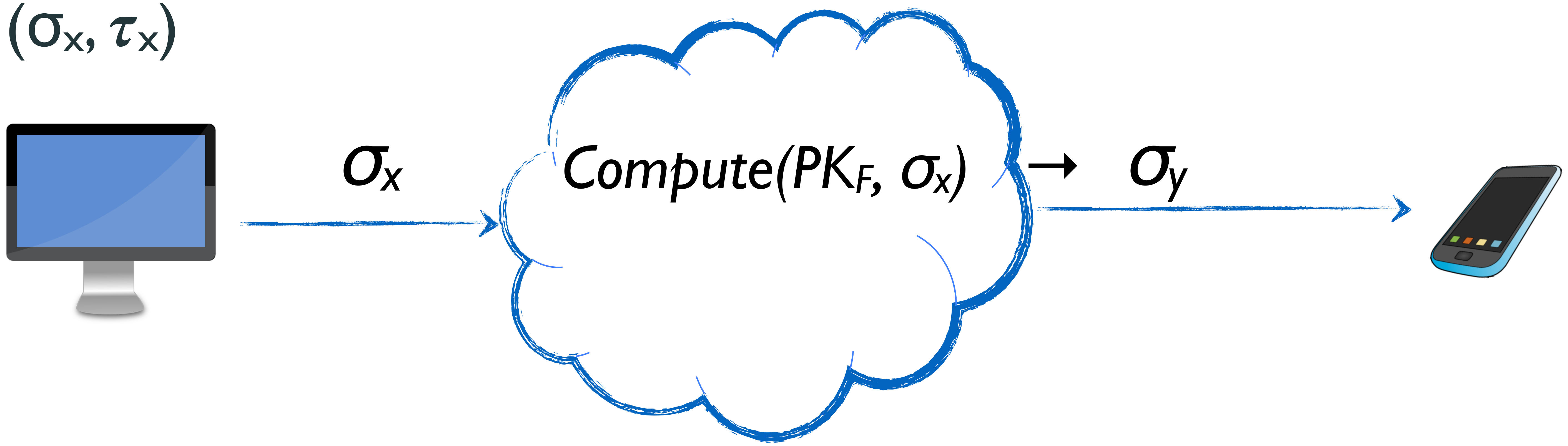
$\text{Decode}(SK_F, \sigma_y) \rightarrow y$

Verifiable Computation [GennaroGentryParno10,ParnoRaikovaVainkuntanathan12]

*Here publicly verifiable/delegatable notion

$(PK_F, SK_F) \leftarrow \text{KeyGen}(F)$

$\text{ProbGen}(PK_F, x) \rightarrow (\sigma_x, \tau_x)$



If $\text{Ver}(PK_F, \tau_x, \sigma_y) = 1$

$y \leftarrow \text{Decode}(SK_F, \sigma_y)$

VC Scheme

$\text{KeyGen}(F) \rightarrow (PK_F, SK_F)$

$\text{ProbGen}(PK_F, x) \rightarrow (\sigma_x, \tau_x)$

$\text{Compute}(PK_F, \sigma_x) \rightarrow \sigma_y$

$\text{Ver}(PK_F, \tau_x, \sigma_y) \rightarrow acc \in \{0,1\}$

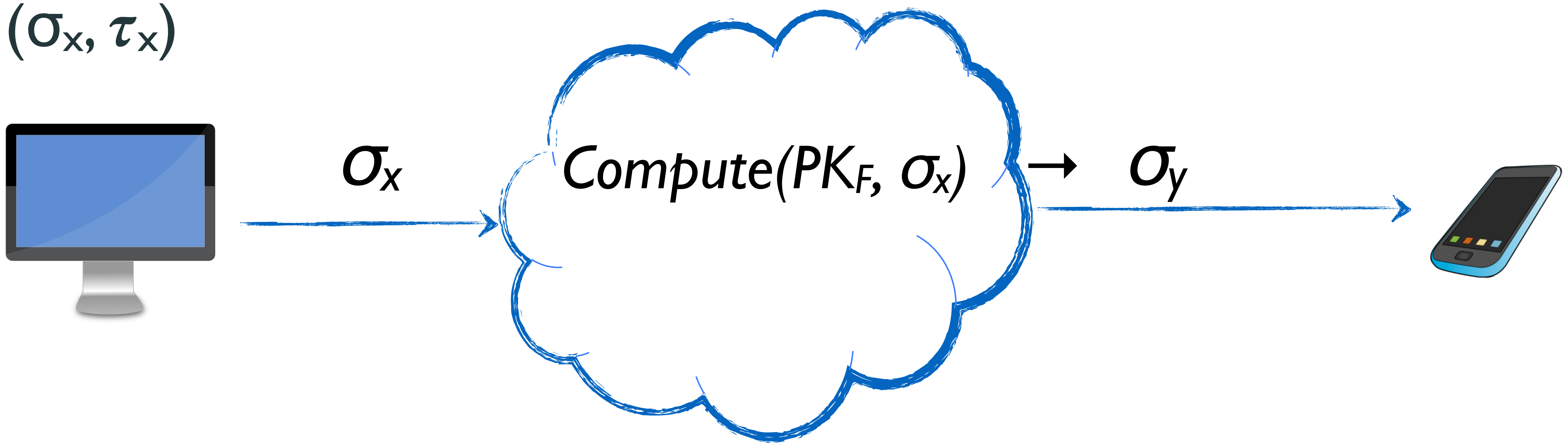
$\text{Decode}(SK_F, \sigma_y) \rightarrow y$

Verifiable Computation [GennaroGentryParno10,ParnoRaikovaVainkuntanathan12]

*Here publicly verifiable/delegatable notion

$(PK_F, SK_F) \leftarrow \text{KeyGen}(F)$

$\text{ProbGen}(PK_F, x) \rightarrow (\sigma_x, \tau_x)$



If $\text{Ver}(PK_F, \tau_x, \sigma_y)=1$

$y \leftarrow \text{Decode}(SK_F, \sigma_y)$

VC Scheme

$\text{KeyGen}(F) \rightarrow (PK_F, SK_F)$

$\text{ProbGen}(PK_F, x) \rightarrow (\sigma_x, \tau_x)$

$\text{Compute}(PK_F, \sigma_x) \rightarrow \sigma_y$

$\text{Ver}(PK_F, \tau_x, \sigma_y) \rightarrow acc \in \{0,1\}$

$\text{Decode}(SK_F, \sigma_y) \rightarrow y$

Correctness. If $(PK_F, SK_F) \leftarrow \text{KeyGen}(F)$, $(\sigma_x, \tau_x) \leftarrow \text{ProbGen}(PK_F, x)$ and $\sigma_y \leftarrow \text{Compute}(PK_F, \sigma_x)$, then

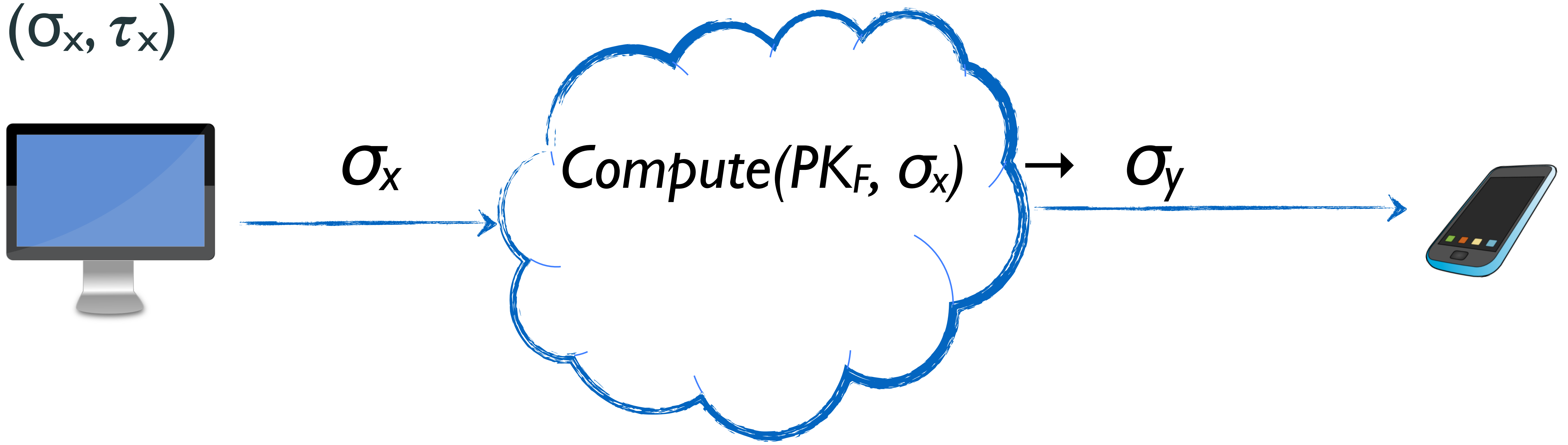
$\text{Ver}(PK_F, \tau_x, \sigma_y) = 1$ and $\text{Decode}(SK_F, \sigma_y) = F(x)$

Verifiable Computation [GennaroGentryParno10,ParnoRaikovaVainkuntanathan12]

*Here publicly verifiable/delegatable notion

$(PK_F, SK_F) \leftarrow \text{KeyGen}(F)$

$\text{ProbGen}(PK_F, x) \rightarrow (\sigma_x, \tau_x)$



If $\text{Ver}(PK_F, \tau_x, \sigma_y)=1$

$y \leftarrow \text{Decode}(SK_F, \sigma_y)$

VC Scheme

$\text{KeyGen}(F) \rightarrow (PK_F, SK_F)$

$\text{ProbGen}(PK_F, x) \rightarrow (\sigma_x, \tau_x)$

$\text{Compute}(PK_F, \sigma_x) \rightarrow \sigma_y$

$\text{Ver}(PK_F, \tau_x, \sigma_y) \rightarrow acc \in \{0,1\}$

$\text{Decode}(SK_F, \sigma_y) \rightarrow y$

Correctness. If $(PK_F, SK_F) \leftarrow \text{KeyGen}(F)$, $(\sigma_x, \tau_x) \leftarrow \text{ProbGen}(PK_F, x)$ and $\sigma_y \leftarrow \text{Compute}(PK_F, \sigma_x)$, then

$\text{Ver}(PK_F, \tau_x, \sigma_y) = 1$ and $\text{Decode}(SK_F, \sigma_y) = F(x)$

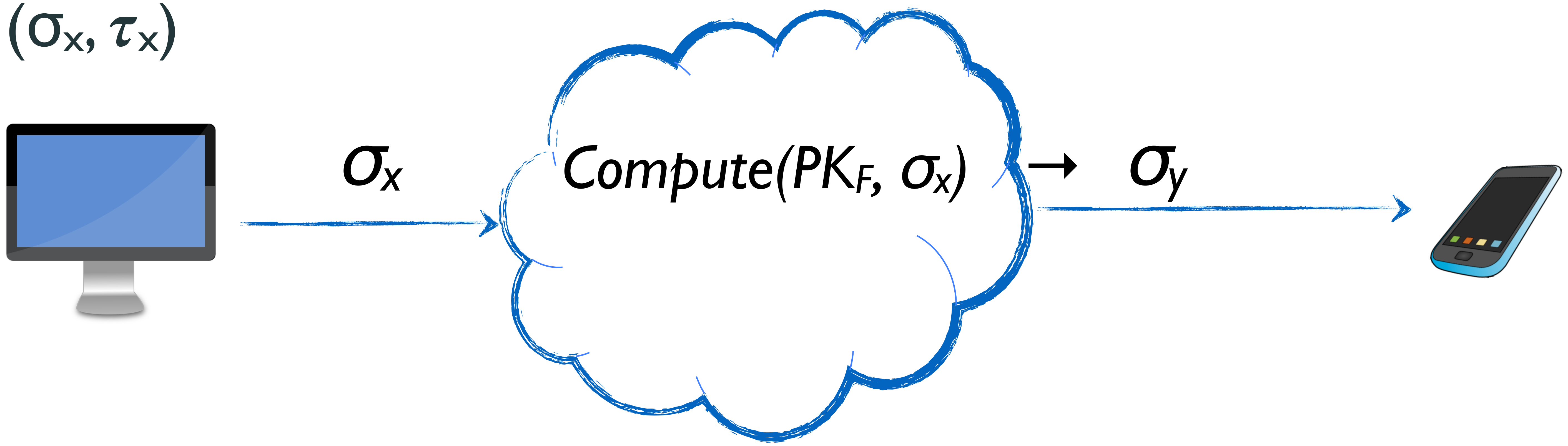
Efficiency. $T(\text{ProbGen})+T(\text{Ver})+T(\text{Decode}) = o(\,T(F)\,)$

Verifiable Computation [GennaroGentryParno10,ParnoRaikovaVainkuntanathan12]

*Here publicly verifiable/delegatable notion

$(PK_F, SK_F) \leftarrow \mathrm{KeyGen}(F)$

$\mathrm{ProbGen}(PK_F, x) \rightarrow (\sigma_x, \tau_x)$



If $\mathrm{Ver}(PK_F, \tau_x, \sigma_y)=1$

$y \leftarrow \mathrm{Decode}(SK_F, \sigma_y)$

VC Scheme

$\mathrm{KeyGen}(F) \rightarrow (PK_F, SK_F)$

$\mathrm{ProbGen}(PK_F, x) \rightarrow (\sigma_x, \tau_x)$

$\mathrm{Compute}(PK_F, \sigma_x) \rightarrow \sigma_y$

$\mathrm{Ver}(PK_F, \tau_x, \sigma_y) \rightarrow acc \in \{0,1\}$

$\mathrm{Decode}(SK_F, \sigma_y) \rightarrow y$

Correctness. If $(PK_F, SK_F) \leftarrow \mathrm{KeyGen}(F)$, $(\sigma_x, \tau_x) \leftarrow \mathrm{ProbGen}(PK_F, x)$ and $\sigma_y \leftarrow \mathrm{Compute}(PK_F, \sigma_x)$, then

$\mathrm{Ver}(PK_F, \tau_x, \sigma_y) = 1$ and $\mathrm{Decode}(SK_F, \sigma_y) = F(x)$

Efficiency. $T(\mathrm{ProbGen})+T(\mathrm{Ver})+T(\mathrm{Decode}) = o(\,T(F)\,)$

Security, Privacy: … next slides

VC Security

Hard to produce an accepting proof for a false result

VC Security

Hard to produce an accepting proof for a false result

PK_F

$(PK_F, SK_F) \leftarrow \text{KeyGen}(F)$

VC Security

Hard to produce an accepting proof for a false result

PK_F ←

$(PK_F, SK_F) \leftarrow \text{KeyGen}(F)$

x →

$(\sigma_x, \tau_x) \leftarrow \text{ProbGen}(PK_F, x)$

(σ_x, τ_x) ←

VC Security

Hard to produce an accepting proof for a false result

PK_F ←

x →

(σ_x, τ_x) ←

σ_y →

$(PK_F, SK_F) \leftarrow \text{KeyGen}(F)$

$(\sigma_x, \tau_x) \leftarrow \text{ProbGen}(PK_F, x)$

Win = "$\text{Ver}(PK_F, \tau_x, \sigma_y) = 1$ and $\text{Decode}(SK_F, \sigma_y) \neq F(x)$"

VC Security

Hard to produce an accepting proof for a false result

PK_F ←

x →

(σ_x, τ_x) ←

σ_y →

$(PK_F, SK_F) \leftarrow \text{KeyGen}(F)$

$(\sigma_x, \tau_x) \leftarrow \text{ProbGen}(PK_F, x)$

Win = "$\text{Ver}(PK_F, \tau_x, \sigma_y) = 1$ and $\text{Decode}(SK_F, \sigma_y) \neq F(x)$"

Def. VC is <u>secure</u> if for any PPT adversary Pr[Win]=negl

VC Privacy

Cloud learns no information on the client's data

VC Privacy

Cloud learns no information on the client's data

PK_F ←

$(PK_F, SK_F) \leftarrow \text{KeyGen}(F)$

VC Privacy

Cloud learns no information on the client's data

PK_F ←

$(PK_F, SK_F) \leftarrow \text{KeyGen}(F)$

x_0, x_1 →

(σ_b, τ_b) ←

$(\sigma_b, \tau_b) \leftarrow \text{ProbGen}(PK_F, x_b)$ $\quad b \leftarrow \$\{0,1\}$

VC Privacy

Cloud learns no information on the client's data

PK_F ← — $(PK_F, SK_F) \leftarrow$ KeyGen(F)

x_0, x_1 → — $(\sigma_b, \tau_b) \leftarrow$ ProbGen(PK_F, x_b) — $b \leftarrow \$\{0,1\}$

(σ_b, τ_b) ←

b' → — if $b'=b$ Win

VC Privacy

Cloud learns no information on the client's data

PK_F ← $(PK_F, SK_F) \leftarrow \text{KeyGen}(F)$

x_0, x_1 → $(\sigma_b, \tau_b) \leftarrow \text{ProbGen}(PK_F, x_b)$ $b \leftarrow \$\{0,1\}$

(σ_b, τ_b) ←

b' → if $b'=b$ Win

Def. VC is *private* if for any PPT adversary Pr[Win]=1/2 + negl *(essentially semantic security)*

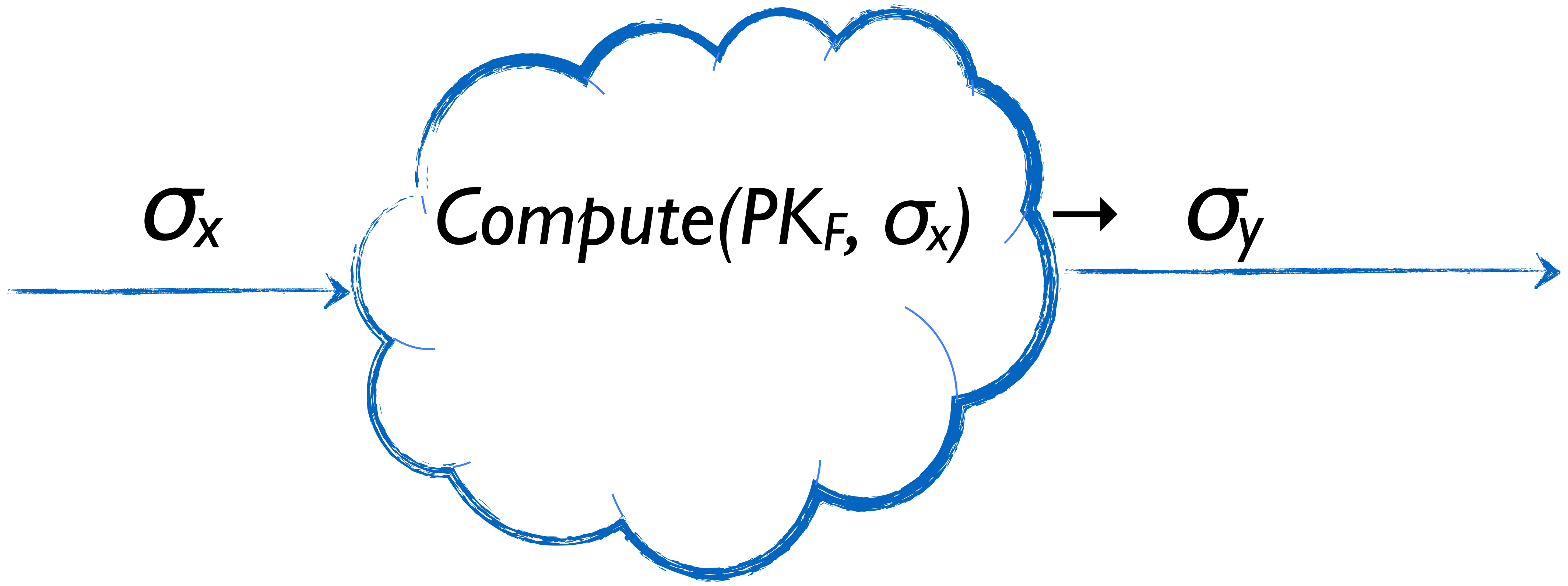
Note: for private verifiable schemes, privacy notion is more complex as the adversary can ask verifications

Outsourcing Data and Computation using VC

*Here publicly verifiable/delegatable notion

$(PK_F, SK_F) \leftarrow \text{KeyGen}(F)$

$\text{ProbGen}(PK_F, x) \rightarrow (\sigma_x, \tau_x)$



If $\text{Ver}(PK_F, \tau_x, \sigma_y)=1$

$y \leftarrow \text{Decode}(SK_F, \sigma_y)$

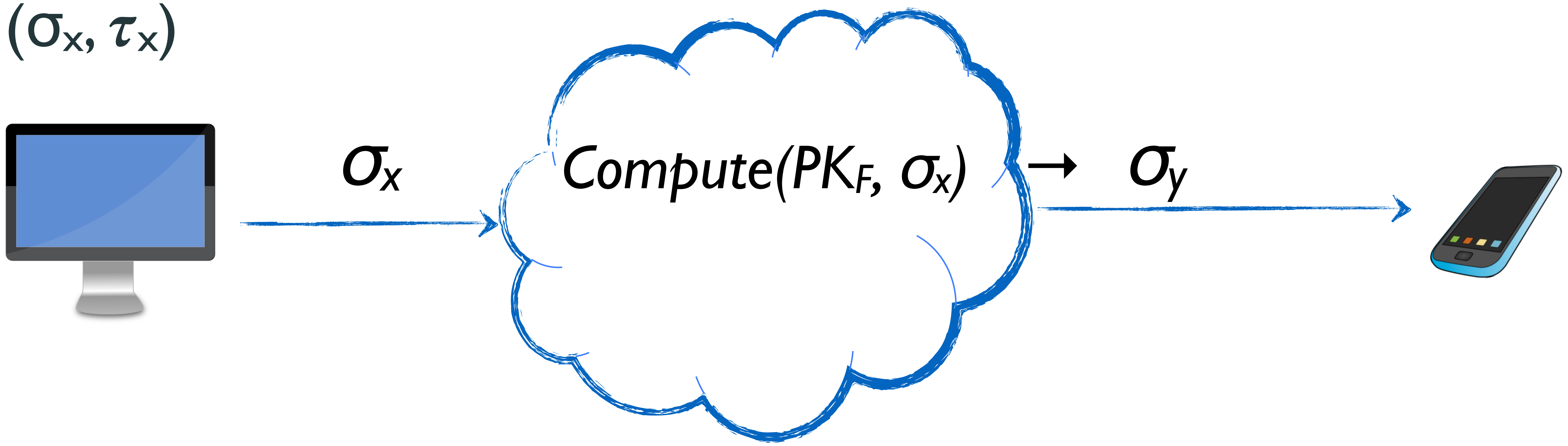
Desired goals:

Outsourcing Data and Computation using VC

*Here publicly verifiable/delegatable notion

$(PK_F, SK_F) \leftarrow \text{KeyGen}(F)$

$\text{ProbGen}(PK_F, x) \rightarrow (\sigma_x, \tau_x)$



If $\text{Ver}(PK_F, \tau_x, \sigma_y) = 1$

$y \leftarrow \text{Decode}(SK_F, \sigma_y)$

Desired goals:

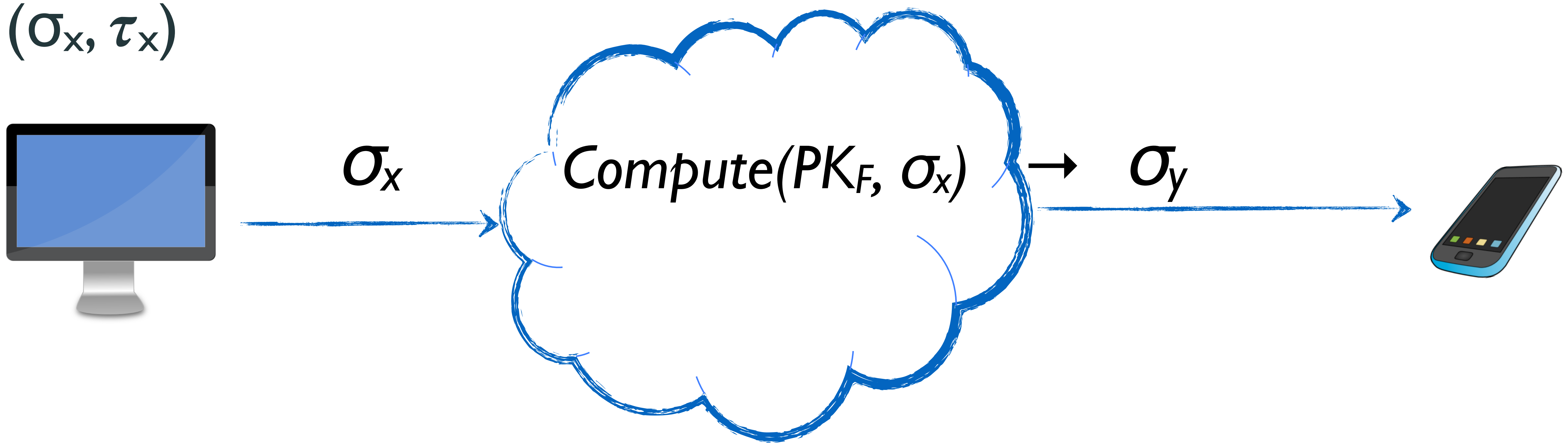
Integrity: the cloud should not be able to send **incorrect** results ⬅ **VC Security**

Outsourcing Data and Computation using VC

*Here publicly verifiable/delegatable notion

$(PK_F, SK_F) \leftarrow \text{KeyGen}(F)$

$\text{ProbGen}(PK_F, x) \rightarrow (\sigma_x, \tau_x)$



If $\text{Ver}(PK_F, \tau_x, \sigma_y)=1$

$y \leftarrow \text{Decode}(SK_F, \sigma_y)$

Desired goals:

Integrity: the cloud should not be able to send **incorrect** results **⬅ VC Security**

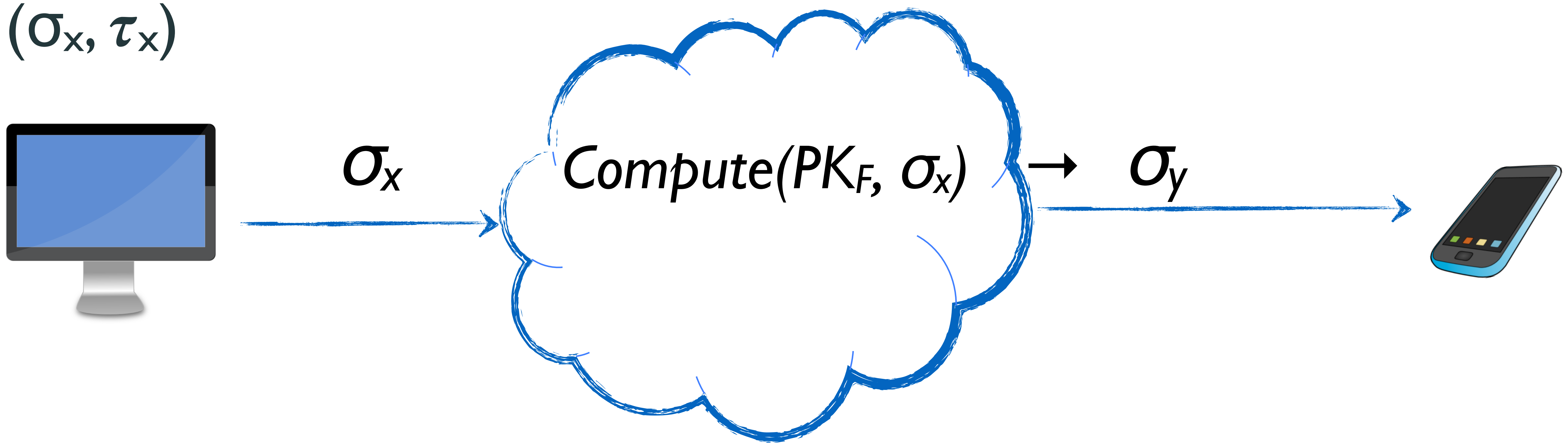
Privacy: the cloud **should not learn information** on the data **⬅ VC Privacy**

Outsourcing Data and Computation using VC

*Here publicly verifiable/delegatable notion

$(PK_F, SK_F) \leftarrow \text{KeyGen}(F)$

$\text{ProbGen}(PK_F, x) \rightarrow (\sigma_x, \tau_x)$



If $\text{Ver}(PK_F, \tau_x, \sigma_y) = 1$

$y \leftarrow \text{Decode}(SK_F, \sigma_y)$

Desired goals:

Integrity: the cloud should not be able to send **incorrect** results ⬅ **VC Security**

Privacy: the cloud **should not learn information** on the data ⬅ **VC Privacy**

Efficiency: communication and storage at client "minimal" ⬅ **VC Efficiency**

How to construct VC

How to construct VC

[GennaroGentryPastro10] first VC proposal based on FHE + garbled circuits

need full power of FHE

How to construct VC

[GennaroGentryPastro10] first VC proposal based on FHE + garbled circuits

need full power of FHE

[GoldwasserKalaiPopaVaikuntanathanZeldovich13] based on single-key Functional Encryption

inherently limited to functions w/ 1-bit outputs, need several ABE for expressive predicates

How to construct VC

[GennaroGentryPastro10] first VC proposal based on FHE + garbled circuits

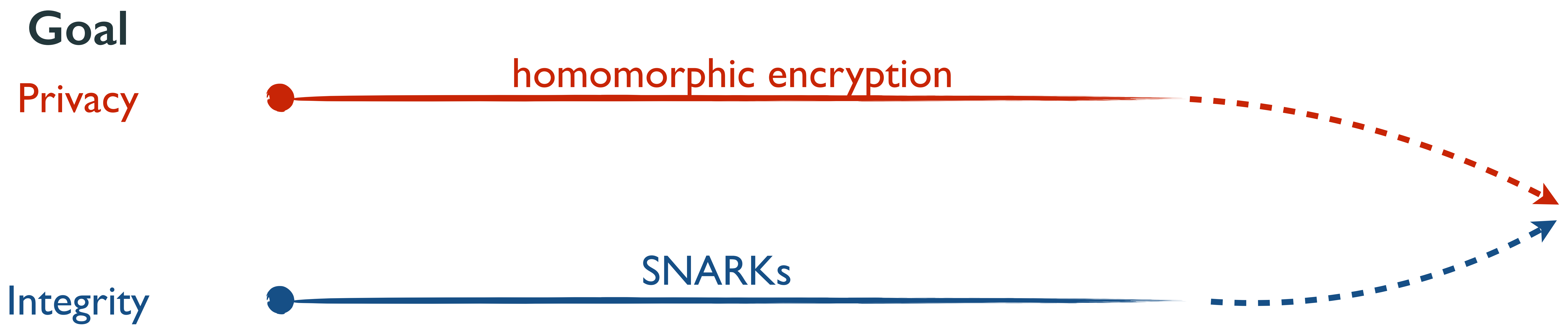
need full power of FHE

[GoldwasserKalaiPopaVaikuntanathanZeldovich13] based on single-key Functional Encryption

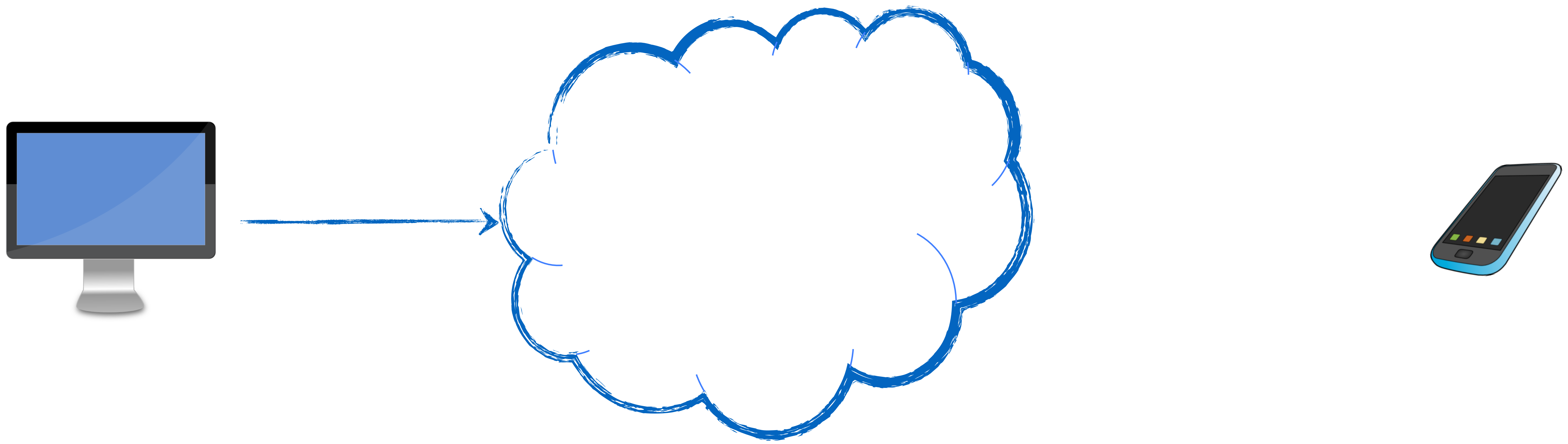
inherently limited to functions w/1-bit outputs, need several ABE for expressive predicates

[FioreGennaroPastro14] generic solution FHE + (non-private) VC

↑ this talk



Solving Privacy&Efficiency using FHE



Fully Homomorphic Encryption

HE.KG() → (ek, dk)

Enc(ek, x) → ct_x

Dec(dk, ct_y) → y

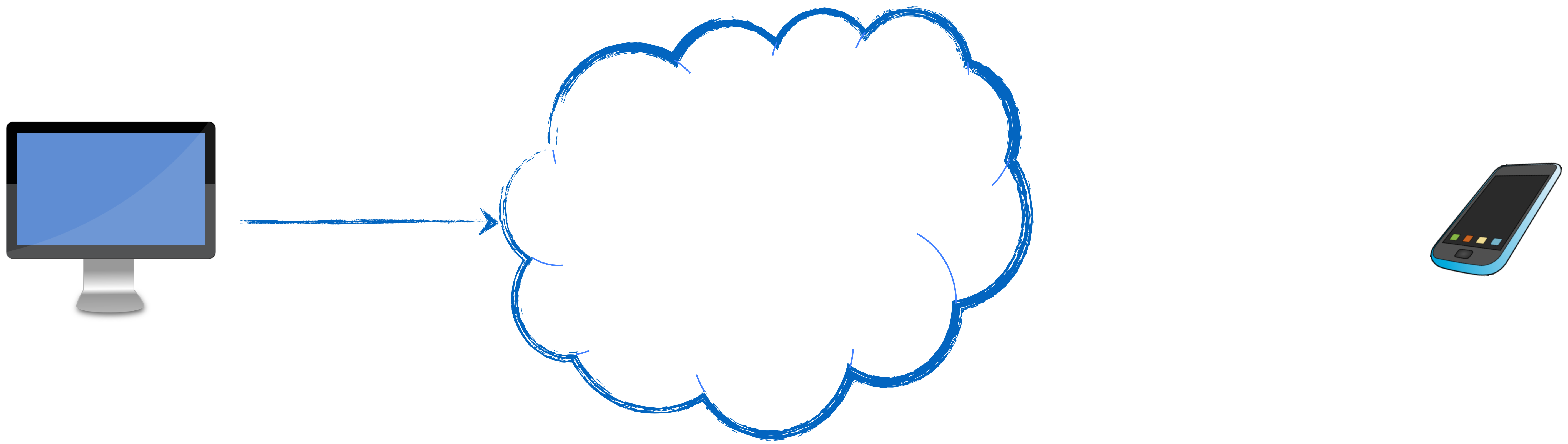
Eval(ek, F, $ct_1, \ldots, ct_n$) → ct

Correctness.

$$\text{Dec}(sk, \text{Eval}(F, \text{Enc}(x_1), \ldots, \text{Enc}(x_n))) = F(x_1, \ldots, x_n)$$

Solving Privacy&Efficiency using FHE

$(ek,dk) \leftarrow HE.KG()$



Fully Homomorphic Encryption

$HE.KG() \rightarrow (ek, dk)$

$Enc(ek, x) \rightarrow ct_x$

$Dec(dk, ct_y) \rightarrow y$

$Eval(ek, F, ct_1, \ldots, ct_n) \rightarrow ct$

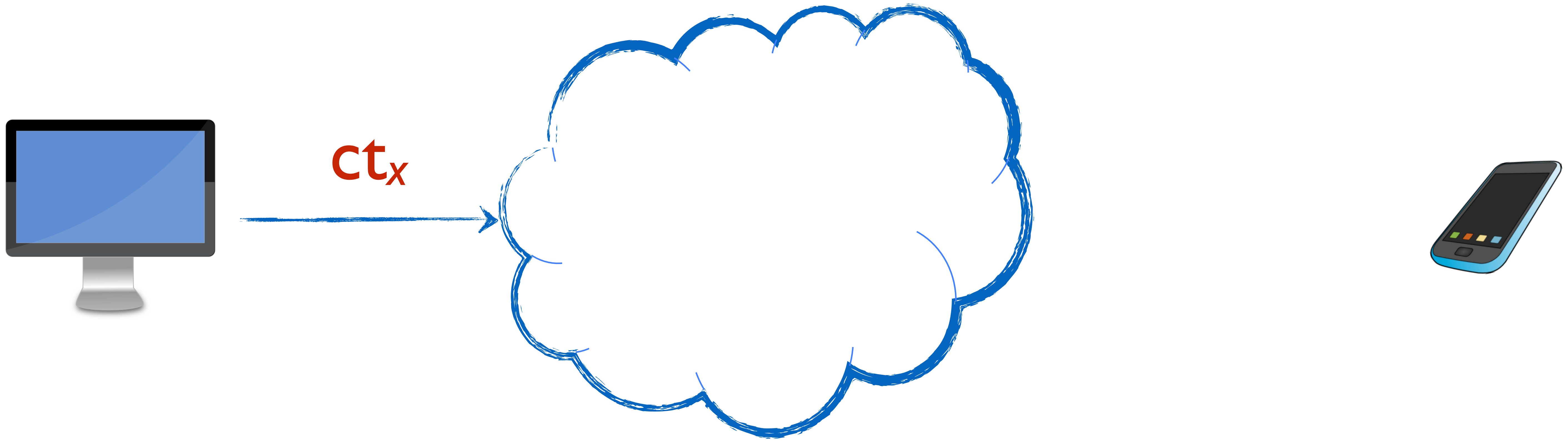
Correctness.

$$Dec(sk, Eval(F, Enc(x_1), \ldots, Enc(x_n))) = F(x_1, \ldots, x_n)$$

Solving Privacy&Efficiency using FHE

$(ek,dk) \leftarrow$ HE.KG()

Enc(ek, x) $\rightarrow$ ct_x



Fully Homomorphic Encryption

HE.KG() $\rightarrow$ (ek, dk)

Enc(ek, x) $\rightarrow$ ct_x

Dec(dk, ct_y) $\rightarrow$ y

Eval(ek, F, $ct_1,\ldots, ct_n$) $\rightarrow$ ct

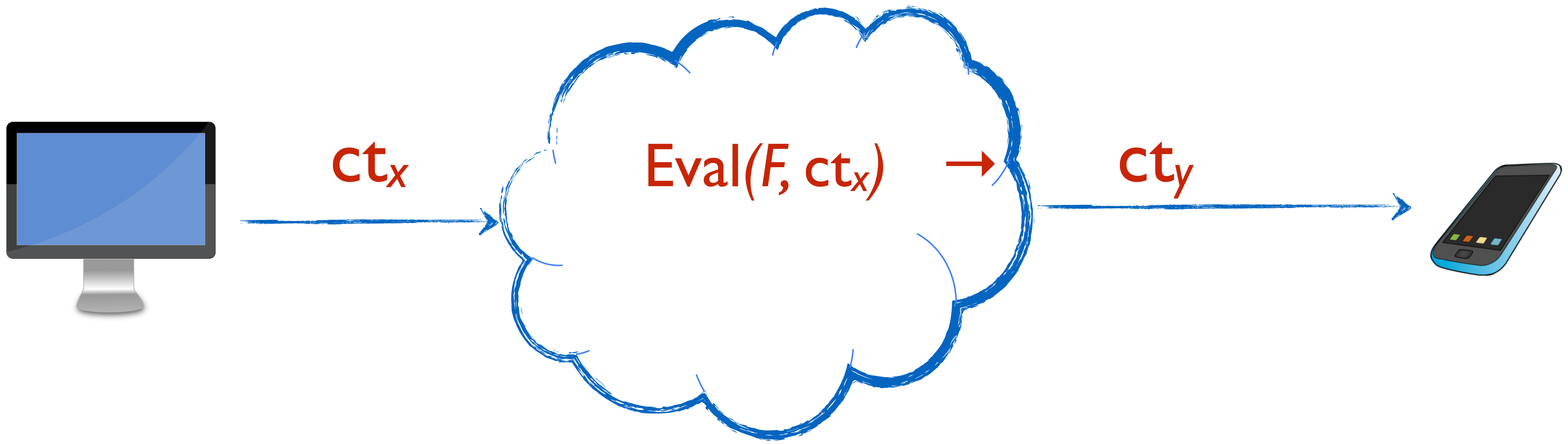
Correctness.

$$\text{Dec}(sk, \text{Eval}(F, \text{Enc}(x_1), \ldots, \text{Enc}(x_n))) = F(x_1, \ldots, x_n)$$

Solving Privacy&Efficiency using FHE

(ek,dk)←HE.KG()

Enc(ek, x) → ct_x



Fully Homomorphic Encryption

HE.KG() → (ek, dk)

Enc(ek, x) → ct_x

Dec(dk, ct_y) → y

Eval(ek, F, $ct_1, \ldots, ct_n$) → ct

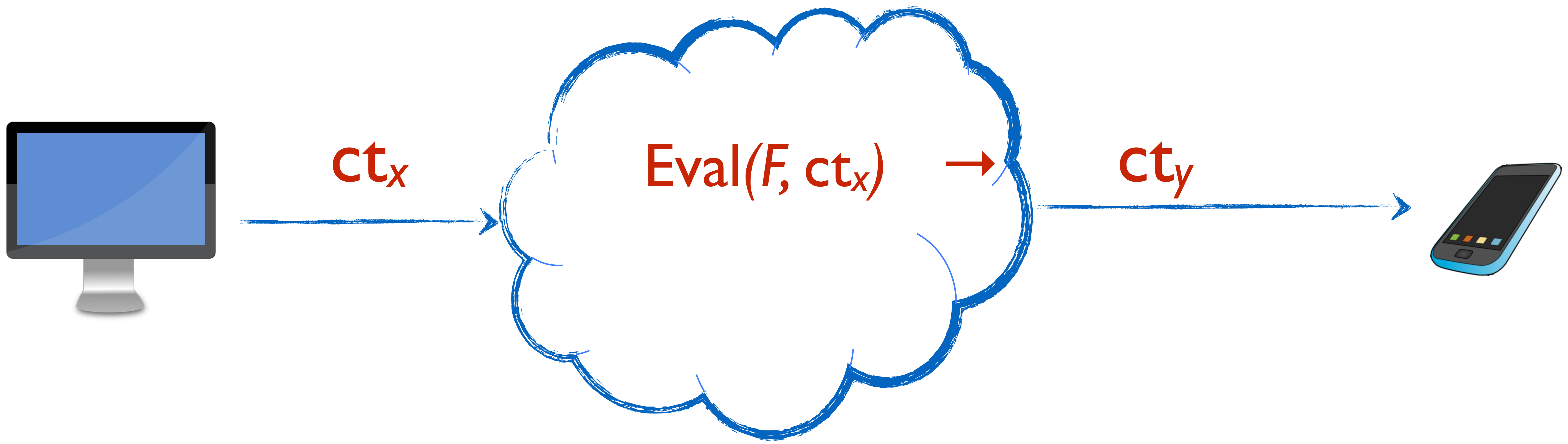
Correctness.

$$\mathrm{Dec}(sk, \mathrm{Eval}(F, \mathrm{Enc}(x_1), \ldots, \mathrm{Enc}(x_n))) = F(x_1, \ldots, x_n)$$

Solving Privacy&Efficiency using FHE

(ek,dk)←HE.KG()

Enc(ek, x) → ct_x



y←Dec(dk, ct_y)

Fully Homomorphic Encryption

HE.KG() → (ek, dk)

Enc(ek, x) → ct_x

Dec(dk, ct_y) → y

Eval(ek, F, $\mathrm{ct}_1, \ldots, \mathrm{ct}_n$) → ct

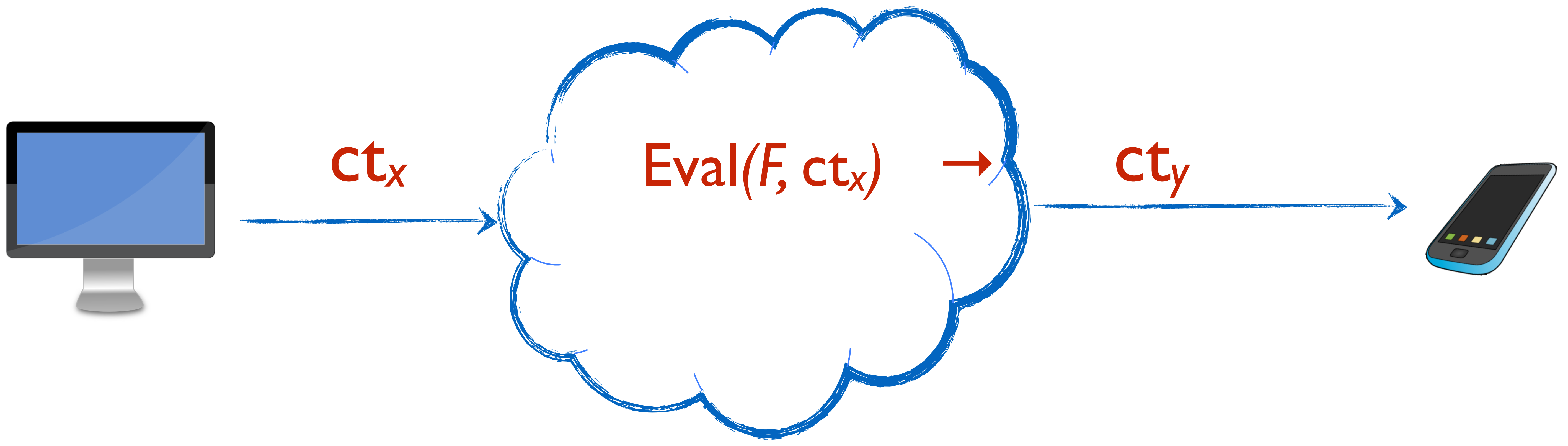
Correctness.

$$\mathrm{Dec}(sk, \mathrm{Eval}(F, \mathrm{Enc}(x_1), \ldots, \mathrm{Enc}(x_n))\ ) = F(x_1, \ldots, x_n)$$

Solving Privacy&Efficiency using FHE

$(ek, dk) \leftarrow HE.KG()$

$Enc(ek, x) \rightarrow ct_x$



$y \leftarrow Dec(dk, ct_y)$

Fully Homomorphic Encryption

$HE.KG() \rightarrow (ek, dk)$

$Enc(ek, x) \rightarrow ct_x$

$Dec(dk, ct_y) \rightarrow y$

$Eval(ek, F, ct_1, \ldots, ct_n) \rightarrow ct$

Correctness.

$$Dec(sk, Eval(F, Enc(x_1), \ldots, Enc(x_n))) = F(x_1, \ldots, x_n)$$

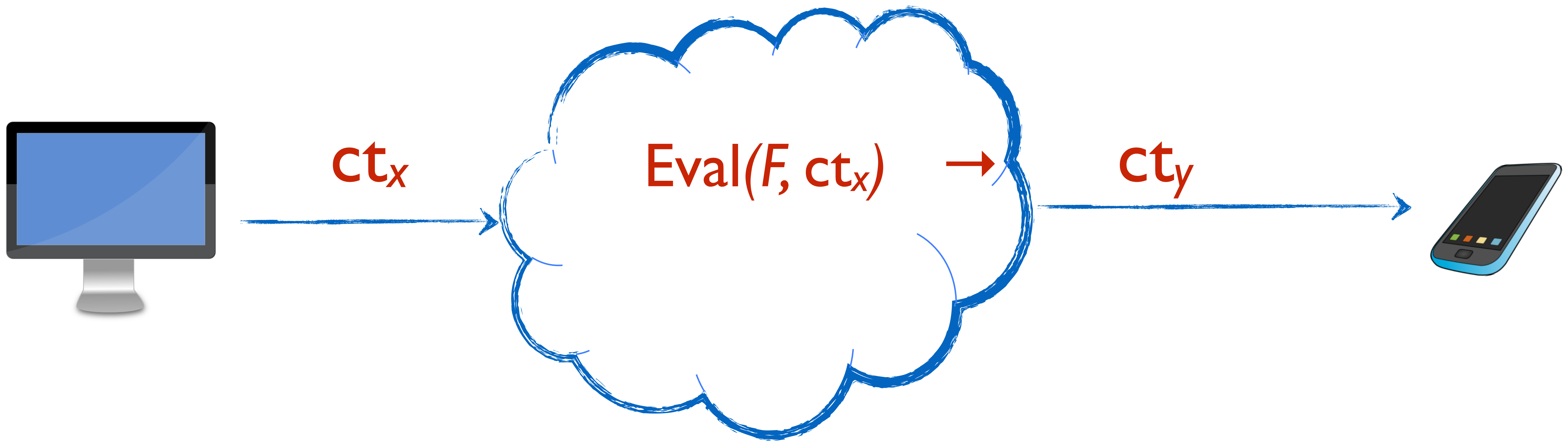
Semantic Security

$$Pr[\, A(Enc(x_b)) = b \mid (x_0, x_1) \leftarrow A(ek);\ b \leftarrow^{\$} \{0,1\}\,] = 1/2 + negl$$

Solving Privacy&Efficiency using FHE

$(ek,dk) \leftarrow HE.KG()$

$Enc(ek, x) \rightarrow ct_x$



$y \leftarrow Dec(dk, ct_y)$

Fully Homomorphic Encryption

$HE.KG() \rightarrow (ek, dk)$

$Enc(ek, x) \rightarrow ct_x$

$Dec(dk, ct_y) \rightarrow y$

$Eval(ek, F, ct_1, \ldots, ct_n) \rightarrow ct$

Correctness.

$$Dec(sk, Eval(F, Enc(x_1), \ldots, Enc(x_n))) = F(x_1, \ldots, x_n)$$

Semantic Security

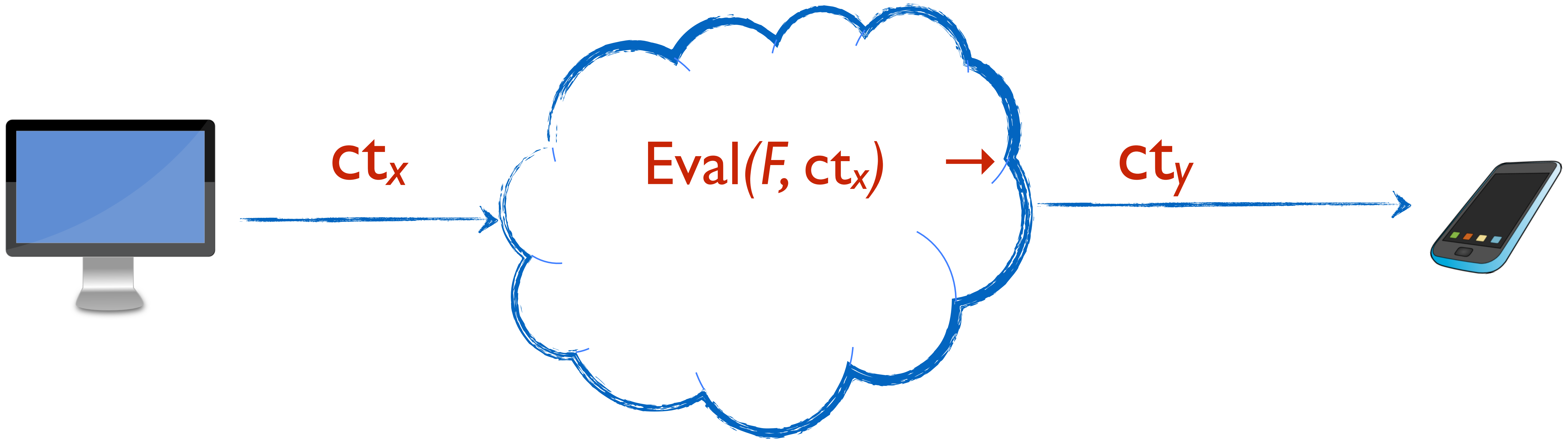
$$\Pr[\, A(Enc(x_b)) = b \mid (x_0, x_1) \leftarrow A(ek);\ b \leftarrow^{\$} \{0,1\} \,] = 1/2 + negl$$

Compactness. $T(Dec) = poly(\lambda)$

Solving Privacy&Efficiency using FHE

(ek,dk)←HE.KG()

Enc(ek, x) → ct_x



y←Dec(dk, ct_y)

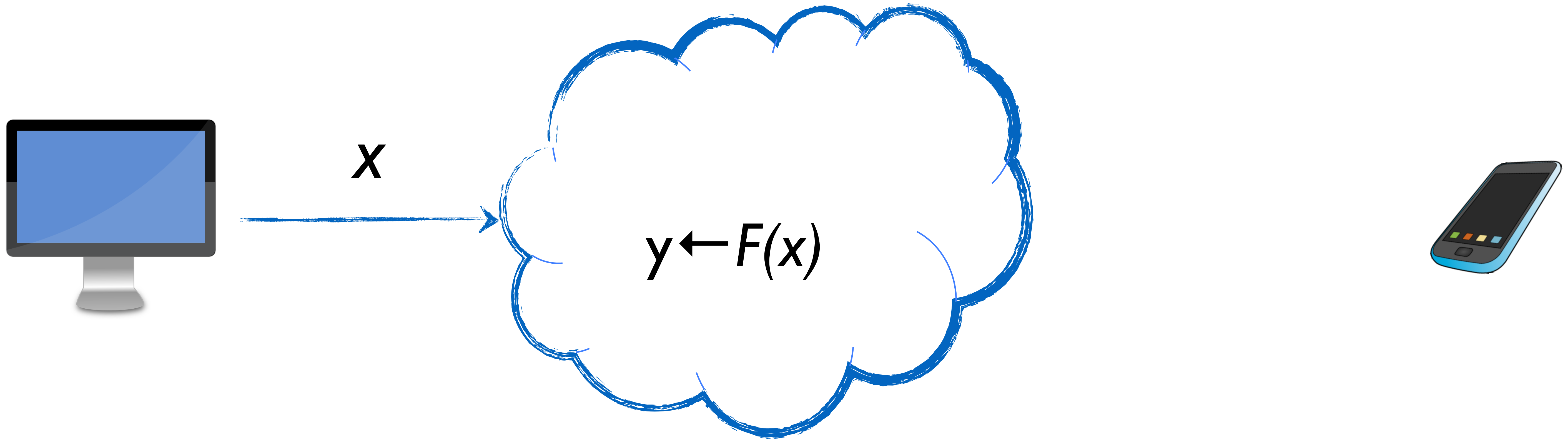
Desired goals:

~~**Integrity:** the cloud should not be able to send **incorrect** results~~

Privacy: the cloud **should not learn information** on the data **⬅ FHE Semantic Sec.**

Efficiency: communication and storage at client "minimal" **⬅ FHE Compactness**

Solving Integrity&Efficiency using SNARGs



SNARGs

Setup(R) → crs

Prove(crs, $\mathbb{x}$, $\mathbb{w}$) → π

Ver(crs, $\mathbb{x}$, π) → 0/1

Correctness. $\forall(\mathbb{x},\mathbb{w})\in R$: Ver(crs, $\mathbb{x}$, Prove(crs, $\mathbb{x}$, $\mathbb{w}$))=1

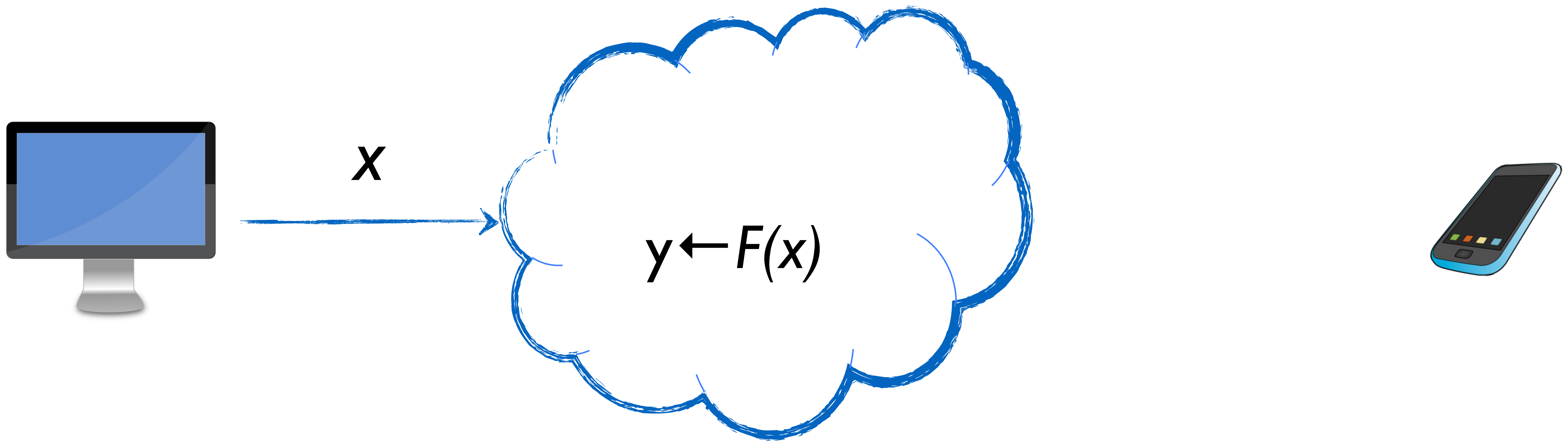
Soundness: Pr[Ver(crs, $\mathbb{x}$, π)=1 $\wedge$ $\nexists\mathbb{w}$: $(\mathbb{x},\mathbb{w})\in R$ | $(\mathbb{x}, \pi)\leftarrow$A(crs)] = negl

Succinctness. T(Ver) = poly(|$\mathbb{x}$|, log|$\mathbb{w}$|)

Solving Integrity&Efficiency using SNARGs

$R_F = \{\, (x, y) : y = F(x) \}$

crs ← Setup(R_F)



SNARGs

Setup(R) → crs

Prove(crs, $\mathbb{x}$, $\mathbb{w}$) → π

Ver(crs, $\mathbb{x}$, π) → 0/1

Correctness. $\forall(\mathbb{x},\mathbb{w})\in R$: Ver(crs, $\mathbb{x}$, Prove(crs, $\mathbb{x}$, $\mathbb{w}$))= 1

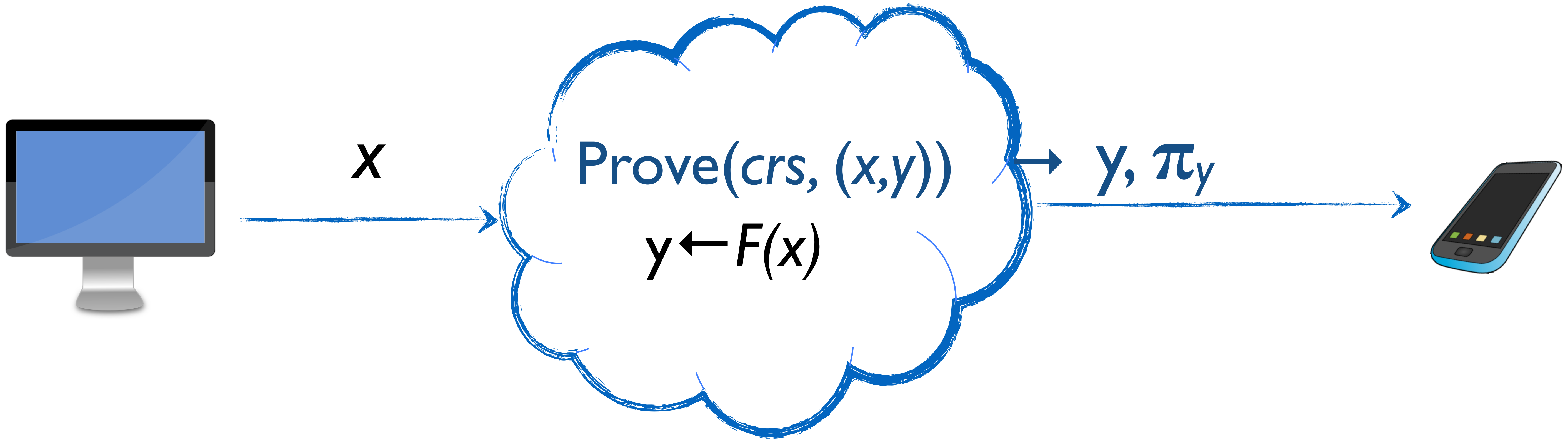
Soundness: Pr[Ver(crs, $\mathbb{x}$, π)=1 $\wedge$ $\nexists\mathbb{w}$: $(\mathbb{x},\mathbb{w})\in R$ | $(\mathbb{x}, \pi)$←A(crs)] = negl

Succinctness. T(Ver) = poly($|\mathbb{x}|$, log$|\mathbb{w}|$)

Solving Integrity&Efficiency using SNARGs

$R_F = \{\ (x, y) : y = F(x) \}$

crs←Setup(R_F)



SNARGs

Setup(R) → crs

Prove(crs, $\mathbb{x}$, $\mathbb{w}$) → π

Ver(crs, $\mathbb{x}$, π) → 0/1

Correctness. $\forall(\mathbb{x},\mathbb{w})\in R$: Ver(crs, $\mathbb{x}$, Prove(crs, $\mathbb{x}$, $\mathbb{w}$))=1

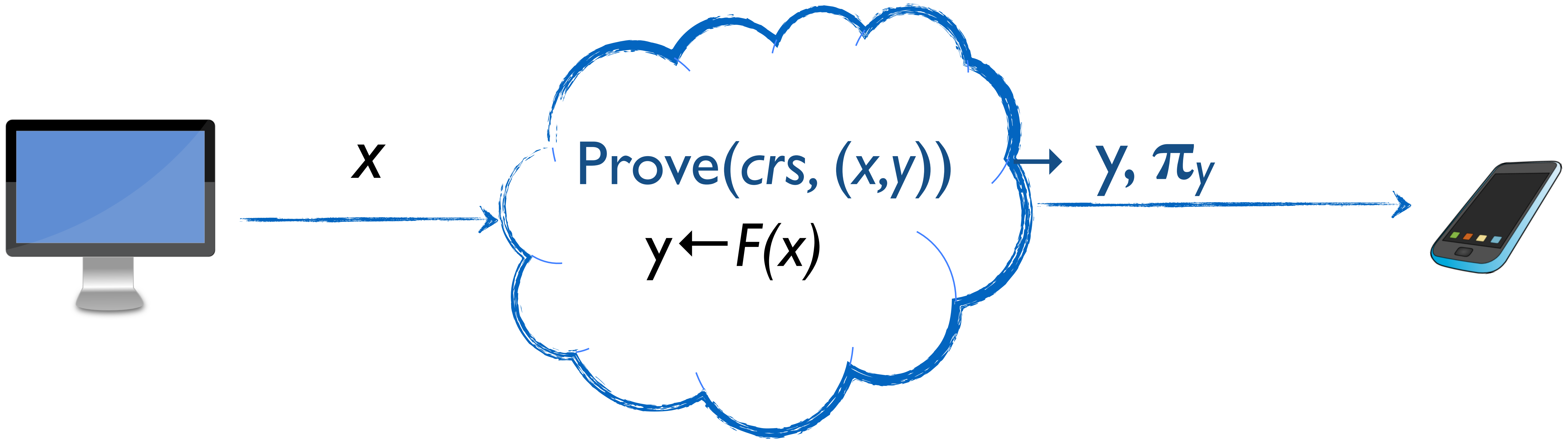
Soundness: Pr[Ver(crs, $\mathbb{x}$, π)=1 $\wedge$ $\nexists\mathbb{w}$: $(\mathbb{x},\mathbb{w})\in R$ | $(\mathbb{x}, \pi)$←A(crs)] = negl

Succinctness. T(Ver) = poly(|$\mathbb{x}$|, log|$\mathbb{w}$|)

Solving Integrity&Efficiency using SNARGs

$R_F = \{ (x, y) : y=F(x)\}$

crs←Setup(R_F)



Check
Ver(crs,(x,y),π_y)=1

SNARGs

Setup(R) → crs

Prove(crs, $\mathbb{x}$, $\mathbb{w}$) → π

Ver(crs, $\mathbb{x}$, π) → 0/1

Correctness. $\forall(\mathbb{x},\mathbb{w})\in R$: Ver(crs, $\mathbb{x}$, Prove(crs, $\mathbb{x}$, $\mathbb{w}$))=1

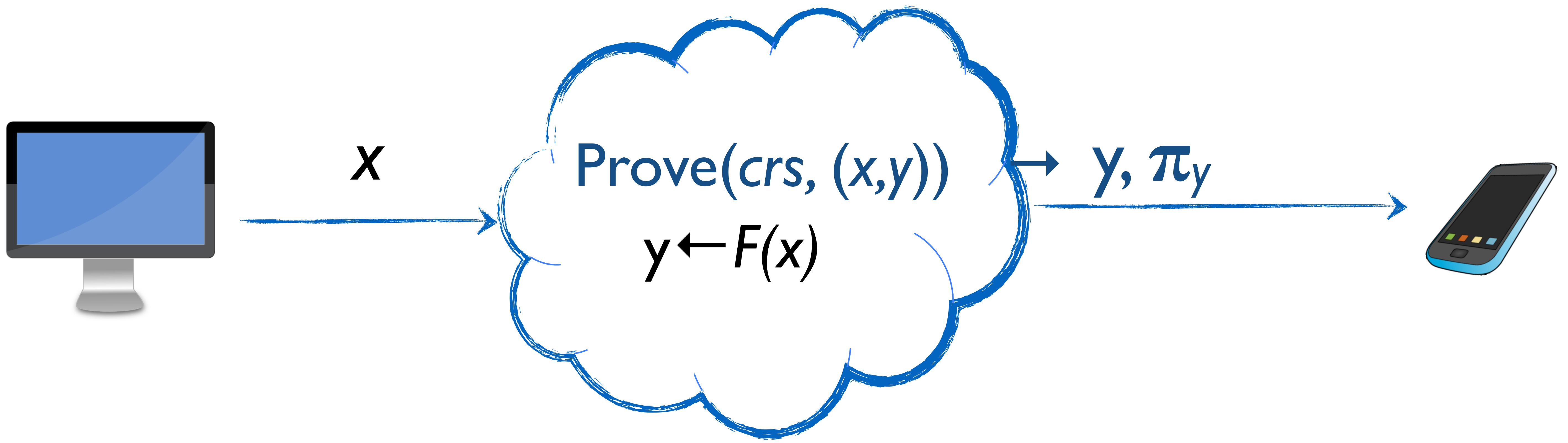
Soundness: Pr[Ver(crs, $\mathbb{x}$, π)=1 $\wedge$ $\nexists\mathbb{w}$: $(\mathbb{x},\mathbb{w})\in R$ | $(\mathbb{x}, \pi)$←A(crs)] = negl

Succinctness. T(Ver) = poly(|$\mathbb{x}$|, log|$\mathbb{w}$|)

Solving Integrity&Efficiency using SNARGs

$R_F = \{ (x, y) : y=F(x)\}$

crs←Setup(R_F)



Check
Ver(crs,(x,y),π_y)=1

Desired goals:

Integrity: the cloud should not be able to send **incorrect** results ← **SNARG Soundness**

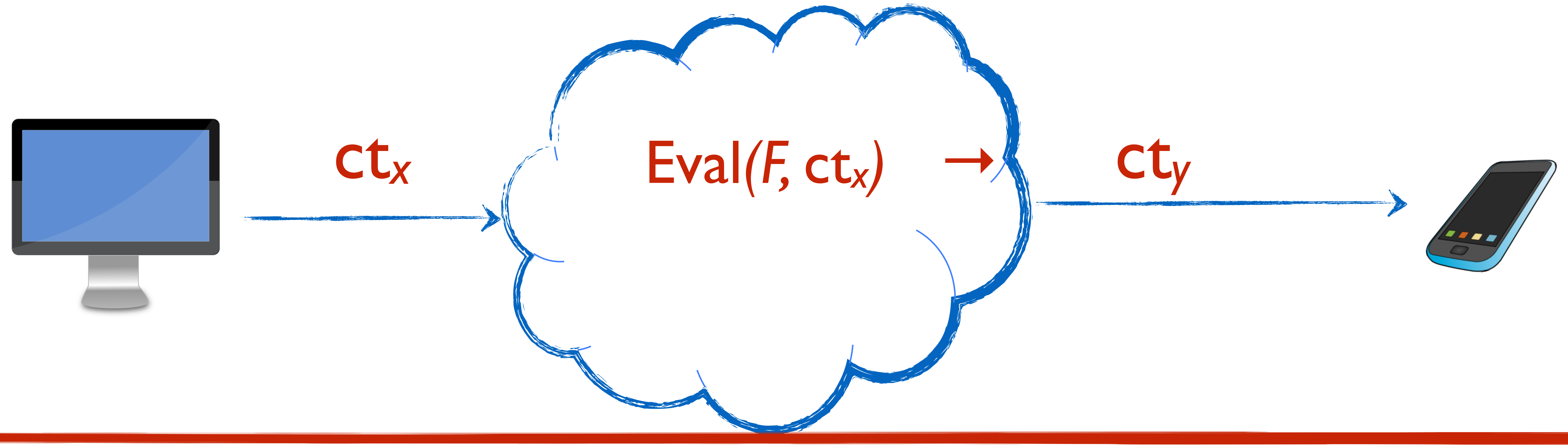
~~**Privacy:** the cloud **should not learn information** on the data~~

Efficiency: communication and storage at client "minimal" ← **SNARG succinctness**

Solving Integrity&Privacy&Efficiency using FHE+SNARGs

(ek,dk)←HE.KG()

Enc(ek, x) → ct_x



y←Dec(dk, ct_y)

Main idea

Start from the FHE solution, and add a SNARG proof that ct_y=Eval(ek, F, ct_x)

Interesting note: the converse is also possible (compute SNARG proof under FHE) but privacy holds with a caveat (secret-key verification, no queries allowed)

VC from FHE + SNARGs

Tools: **HE for F**

SNARG for
$R'_F = \{(ct_x, ct_y) : ct_y = \text{Eval}(ek, F, ct_x)\}$

VC scheme

VC from FHE + SNARGs

Tools: **HE for F** **SNARG** for $R'_F = \{(ct_x, ct_y) : ct_y = \text{Eval}(ek, F, ct_x)\}$

VC scheme

- KeyGen(F)→(PK_F, SK_F) : (ek, dk)←HE.KG(); crs←Setup(R'_F); PK_F=(ek, crs), SK_F=dk

VC from FHE + SNARGs

Tools: **HE for F** | **SNARG** for $R'_F = \{(ct_x, ct_y) : ct_y = \mathrm{Eval}(ek, F, ct_x)\}$

<u>VC scheme</u>

- $\mathrm{KeyGen}(F) \rightarrow (PK_F, SK_F)$: $(ek, dk) \leftarrow \mathrm{HE.KG}()$; $crs \leftarrow \mathrm{Setup}(R'_F)$; $PK_F = (ek, crs)$, $SK_F = dk$
- $\mathrm{ProbGen}(PK_F, x) \rightarrow (\sigma_x, \tau_x)$: $\sigma_x = \tau_x = ct_x \leftarrow \mathrm{Enc}(ek, x)$

VC from FHE + SNARGs

Tools: **HE for F** **SNARG** for $R'_F = \{(ct_x, ct_y) : ct_y = \text{Eval}(ek, F, ct_x)\}$

VC scheme

- KeyGen(F)→(PK_F, SK_F) : (ek, dk)←HE.KG(); crs←Setup(R'_F); PK_F=(ek, crs), SK_F=dk
- ProbGen(PK_F, x)→(σ_x, τ_x) : $\sigma_x = \tau_x = ct_x$←Enc(ek, x)
- Compute(PK_F, σ_x) → σ_y : ct_y←Eval(ek, F, σ_x), π_y←Prove(crs, (ct_x, ct_y)); σ_y =(ct_y, π_y)

VC from FHE + SNARGs

Tools: **HE for F** | **SNARG** for $R'_F = \{(ct_x, ct_y) : ct_y = \mathrm{Eval}(ek, F, ct_x)\}$

<u>VC scheme</u>

- KeyGen(F)→(PK_F, SK_F) : (ek, dk)←HE.KG(); crs←Setup(R'_F); PK_F=(ek, crs), SK_F=dk
- ProbGen(PK_F, x)→(σ_x, τ_x) : $\sigma_x = \tau_x = ct_x$←Enc(ek, x)
- Compute(PK_F, σ_x) → σ_y : ct_y←Eval(ek, F, σ_x), π_y←Prove(crs, (ct_x, ct_y)); σ_y =(ct_y, π_y)
- Ver(PK_F, τ_x, σ_y) = Ver(crs, (ct_x, ct_y), π_y)

VC from FHE + SNARGs

Tools: **HE for F** | **SNARG** for $R'_F = \{(ct_x, ct_y) : ct_y = \mathsf{Eval}(ek, F, ct_x)\}$

VC scheme

- KeyGen(F)→(PK_F, SK_F) : (ek, dk)←HE.KG(); crs←Setup(R'_F); PK_F=(ek, crs), SK_F=dk
- ProbGen(PK_F, x)→(σ_x, τ_x) : $\sigma_x = \tau_x = ct_x$←Enc(ek, x)
- Compute(PK_F, σ_x) → σ_y : ct_y←Eval(ek, F, σ_x), π_y←Prove(crs, (ct_x, ct_y)); σ_y =(ct_y, π_y)
- Ver(PK_F, τ_x, σ_y) = Ver(crs, (ct_x, ct_y), π_y)
- Decode(SK_F, σ_y) = Dec(dk, ct_y)

VC from FHE + SNARGs

Tools: **HE for F**

SNARG for
$R'_F = \{(ct_x, ct_y) : ct_y = \text{Eval}(ek, F, ct_x)\}$

<u>VC scheme</u>

- KeyGen(F)→(PK_F, SK_F) : (ek, dk)←HE.KG(); crs←Setup(R'_F); PK_F=(ek, crs), SK_F=dk
- ProbGen(PK_F, x)→(σ_x, τ_x) : $\sigma_x = \tau_x = ct_x$←Enc(ek, x)
- Compute(PK_F, σ_x) → σ_y : ct_y←Eval(ek, F, σ_x), π_y←Prove(crs, (ct_x, ct_y)); σ_y =(ct_y, π_y)
- Ver(PK_F, τ_x, σ_y) = Ver(crs, (ct_x, ct_y), π_y)
- Decode(SK_F, σ_y) = Dec(dk, ct_y)

Efficiency: T(ProbGen)+T(Ver)+T(Decode) = poly($|x|$) + poly($|ct_x|+|ct_y|$, log$|F|$) + poly(λ)

VC from FHE + SNARGs

Tools: **HE for F** | **SNARG** for $R'_F = \{(ct_x, ct_y) : ct_y = \mathrm{Eval}(ek, F, ct_x)\}$

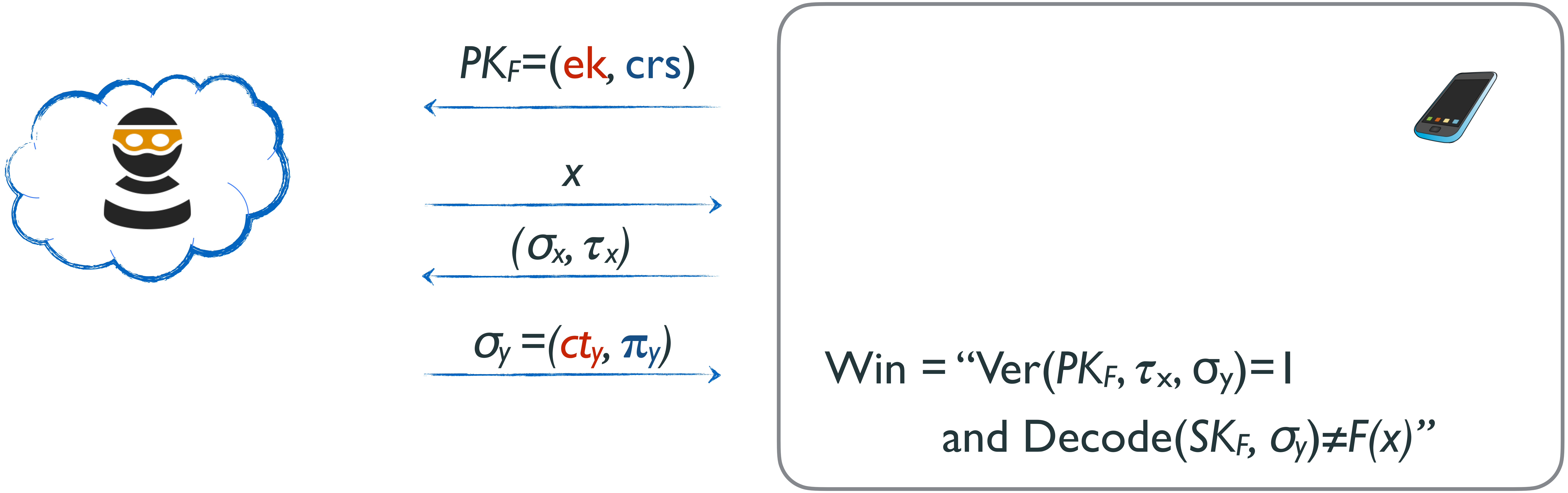
<u>VC scheme</u>

- KeyGen(F)→(PK_F, SK_F) : (ek, dk)←HE.KG(); crs←Setup(R'_F); PK_F=(ek, crs), SK_F=dk
- ProbGen(PK_F, x)→(σ_x, τ_x) : $\sigma_x = \tau_x = ct_x$←Enc(ek, x)
- Compute(PK_F, σ_x) → σ_y : ct_y←Eval(ek, F, σ_x), π_y←Prove(crs, (ct_x, ct_y)); σ_y =(ct_y, π_y)
- Ver(PK_F, τ_x, σ_y) = Ver(crs, (ct_x, ct_y), π_y)
- Decode(SK_F, σ_y) = Dec(dk, ct_y)

Efficiency: T(ProbGen)+T(Ver)+T(Decode) = poly(|x|) + poly(|ct_x|+|ct_y|, log|F|) + poly(λ)

Privacy: straightforward from FHE semantic security

Security



Theorem. If FHE is *correct* and SNARG is *sound*, then the VC is *secure*.

Security

$PK_F = (ek, crs)$ ←

x →

(σ_x, τ_x) ←

$\sigma_y = (ct_y, \pi_y)$ →

Win = "$\mathrm{Ver}(PK_F, \tau_x, \sigma_y) = 1$

and $\mathrm{Decode}(SK_F, \sigma_y) \neq F(x)$"

Theorem. If FHE is _correct_ and SNARG is _sound_, then the VC is _secure_.

$\Pr[\mathrm{Win}] = \Pr[\mathrm{Win} \wedge ct_y \neq \mathrm{Eval}(ek, F, ct_x)] + \Pr[\mathrm{Win} \wedge ct_y = \mathrm{Eval}(ek, F, ct_x)]$

Security

PK_F=(ek, crs) ←

x →

(σ_x, τ_x) ←

$\sigma_y = (ct_y, \pi_y)$ →

Win = "$\mathrm{Ver}(PK_F, \tau_x, \sigma_y)=1$

and $\mathrm{Decode}(SK_F, \sigma_y) \neq F(x)$"

Theorem. If FHE is *correct* and SNARG is *sound*, then the VC is *secure*.

$$\Pr[\mathrm{Win}] = \Pr[\mathrm{Win} \wedge ct_y \neq \mathrm{Eval}(ek, F, ct_x)] + \Pr[\mathrm{Win} \wedge ct_y = \mathrm{Eval}(ek, F, ct_x)]$$

$$= \Pr[\mathrm{SndWin}] \qquad + \qquad 0$$

Security

$PK_F = (\text{ek}, \text{crs})$

x

(σ_x, τ_x)

$\sigma_y = (ct_y, \pi_y)$

Win = "Ver(PK_F, τ_x, σ_y)=1 and Decode(SK_F, σ_y)$\neq F(x)$"

crs

Theorem. If FHE is *correct* and SNARG is *sound*, then the VC is *secure*.

$$\Pr[\text{Win}] = \Pr[\text{Win} \wedge ct_y \neq \text{Eval}(\text{ek}, F, ct_x)] + \Pr[\text{Win} \wedge ct_y = \text{Eval}(\text{ek}, F, ct_x)]$$

$$= \Pr[\text{SndWin}] \qquad + \qquad 0$$

Security

PK_F=(ek, crs) ←

x →

(σ_x, τ_x) ←

σ_y =(ct_y, π_y) →

(ek, dk)←HE.KG()

$\sigma_x=\tau_x=ct_x$←Enc(ek, x)

Win = "Ver(PK_F, τ_x, σ_y)=1
and Decode(SK_F, σ_y)≠$F(x)$"

↑ crs

Theorem. If FHE is *correct* and SNARG is *sound*, then the VC is *secure*.

Pr[Win]=Pr[Win ∧ ct_y≠Eval(ek, F, ct_x)]+Pr[Win ∧ ct_y=Eval(ek, F, ct_x)]

= Pr[SndWin] + 0

Security

PK_F=(ek, crs)

x

(σ_x, τ_x)

σ_y =(ct_y, π_y)

(ek, dk)←HE.KG()

σ_x=τ_x=ct_x←Enc(ek, x)

Win = "Ver(PK_F, τ_x, σ_y)=1
and Decode(SK_F, σ_y)≠$F(x)$"

(ct_x, ct_y), π_y ↓ ↑ crs

Theorem. If FHE is *correct* and SNARG is *sound*, then the VC is *secure*.

$\Pr[\text{Win}]=\Pr[\text{Win} \wedge ct_y \neq \text{Eval}(ek, F, ct_x)]+\Pr[\text{Win} \wedge ct_y=\text{Eval}(ek, F, ct_x)]$

$= \Pr[\text{SndWin}] \quad + \quad 0$

SndWin = "Ver(crs, (ct_x, ct_y), π_y)=1
and ct_y≠Eval(ek, F, ct_x)"

Practical efficiency challenges of the generic VC

- KeyGen(F)$\rightarrow$(PK_F, SK_F) : (ek, dk)$\leftarrow$HE.KG(); crs$\leftarrow$Setup(R'_F); PK_F=(ek, crs), SK_F=dk
- ProbGen(PK_F, x)$\rightarrow$(σ_x, τ_x) : σ_x=τ_x=$ct_x$$\leftarrow$Enc(ek, x)
- Compute(PK_F, σ_x) $\rightarrow$ σ_y : $ct_y$$\leftarrow$Eval(ek, F, σ_x), $\pi_y$$\leftarrow$Prove(crs, ($ct_x$, ct_y)); σ_y =(ct_y, π_y)
- Ver(PK_F, τ_x, σ_y) = Ver(crs, (ct_x, ct_y), π_y)
- Decode(SK_F, σ_y) = Dec(dk, ct_y)

Building blocks' efficiency

- FHE is executed "natively" (as if no integrity is needed) — *virtually optimal*
- SNARG's efficiency depends on FHE **Eval(ek, *F*, .)** — *potential blowup*

Compatibility challenges

Blueprint of RingLWE-based HE [BV, BFV, BGV]

Compatibility challenges

Blueprint of RingLWE-based HE [BV, BFV, BGV]

$R = \mathbb{Z}[X]/(X^d + 1)$, message space $R_p = R/pR$

Compatibility challenges

Blueprint of RingLWE-based HE [BV, BFV, BGV]

$R = \mathbb{Z}[X]/(X^d + 1),$ message space $R_p = R/pR$

Encryption

$R_p \ni m \longmapsto \mathsf{ct} = (\mathsf{ct}[0] + \mathsf{ct}[1]Y) \in R_q[Y]$ //ciphertxt space

Compatibility challenges

Blueprint of RingLWE-based HE [BV, BFV, BGV]

$R = \mathbb{Z}[X]/(X^d + 1),$ message space $R_p = R/pR$

Encryption

$R_p \ni m \longmapsto \mathsf{ct} = (\mathsf{ct}[0] + \mathsf{ct}[1]Y) \in R_q[Y]$ //ciphertxt space

Addition

Eval(+, ct_1, ct_2)→ct_1+ct_2 $\in R_q[Y]$

Compatibility challenges

Blueprint of RingLWE-based HE [BV, BFV, BGV]

$R = \mathbb{Z}[X]/(X^d + 1)$, message space $R_p = R/pR$

Encryption

$R_p \ni m \longmapsto \text{ct} = (\text{ct}[0] + \text{ct}[1]Y) \in R_q[Y]$ //ciphertxt space

Addition

Eval(+, ct_1, ct_2)→ct_1+ct_2 $\in R_q[Y]$

Basic multiplication

Eval(x, ct_1, ct_2)→$\text{ct}_1 \cdot \text{ct}_2 \in R_q[Y]$ // $\deg_Y$ increases

Compatibility challenges

Blueprint of RingLWE-based HE [BV, BFV, BGV]

$R = \mathbb{Z}[X]/(X^d + 1)$, message space $R_p = R/pR$

Encryption

$R_p \ni m \longmapsto \mathsf{ct} = (\mathsf{ct}[0] + \mathsf{ct}[1]Y) \in R_q[Y]$ //ciphertxt space

Addition

Eval(+, ct_1, ct_2)→ct_1+ct_2 $\in R_q[Y]$

Basic multiplication

Eval(x, ct_1, ct_2)→$\mathsf{ct}_1 \cdot \mathsf{ct}_2 \in R_q[Y]$ // $\deg_Y$ increases

Relinearization + mod switch /noise reduction

$$\mathsf{ct} \longmapsto \mathsf{ct}' = \sum_{i=0}^{deg_Y(\mathsf{ct})} \mathsf{ct}[i] \cdot \mathsf{rk}[i] \bmod q \mapsto \lceil \frac{q'}{q} \mathsf{ct} \rfloor$$

Compatibility challenges

Blueprint of RingLWE-based HE [BV, BFV, BGV]

$R = \mathbb{Z}[X]/(X^d + 1)$, message space $R_p = R/pR$

Encryption

$R_p \ni m \longmapsto \mathsf{ct} = (\mathsf{ct}[0] + \mathsf{ct}[1]Y) \in R_q[Y]$ //ciphertxt space

Addition

$\mathsf{Eval}(+, \mathsf{ct}_1, \mathsf{ct}_2) \rightarrow \mathsf{ct}_1 + \mathsf{ct}_2 \in R_q[Y]$

Basic multiplication

$\mathsf{Eval}(\times, \mathsf{ct}_1, \mathsf{ct}_2) \rightarrow \mathsf{ct}_1 \cdot \mathsf{ct}_2 \in R_q[Y]$ // $\deg_Y$ increases

Relinearization + mod switch /noise reduction

$$\mathsf{ct} \longmapsto \mathsf{ct}' = \sum_{i=0}^{deg_Y(\mathsf{ct})} \mathsf{ct}[i] \cdot \mathsf{rk}[i] \bmod q \mapsto \lceil \frac{q'}{q} \mathsf{ct} \rfloor$$

SNARGs

Best schemes for computations over large finite field $\mathbb{F}$

Compatibility challenges

Blueprint of RingLWE-based HE [BV, BFV, BGV]

$R = \mathbb{Z}[X]/(X^d + 1)$, message space $R_p = R/pR$

Encryption

$R_p \ni m \longmapsto \mathsf{ct} = (\mathsf{ct}[0] + \mathsf{ct}[1]Y) \in R_q[Y]$ //ciphertxt space

Addition

$\mathsf{Eval}(+, \mathsf{ct}_1, \mathsf{ct}_2) \rightarrow \mathsf{ct}_1 + \mathsf{ct}_2 \in R_q[Y]$

Basic multiplication

$\mathsf{Eval}(\times, \mathsf{ct}_1, \mathsf{ct}_2) \rightarrow \mathsf{ct}_1 \cdot \mathsf{ct}_2 \in R_q[Y]$ // deg_Y increases

Relinearization + mod switch /noise reduction

$$\mathsf{ct} \longmapsto \mathsf{ct}' = \sum_{i=0}^{deg_Y(\mathsf{ct})} \mathsf{ct}[i] \cdot \mathsf{rk}[i] \bmod q \mapsto \lfloor \frac{q'}{q} \mathsf{ct} \rceil$$

SNARGs

Best schemes for computations over large finite field $\mathbb{F}$

Challenges

1) Ciphertext expansion

unless optimized packing, $deg_X(m) \ll d$

2) Ciphertext modulus

q usually not prime

3) Non-algebraic operations

noise control techniques require divisions and rounding

Ciphertext and circuit expansion (extreme case)

plaintexts ciphertexts

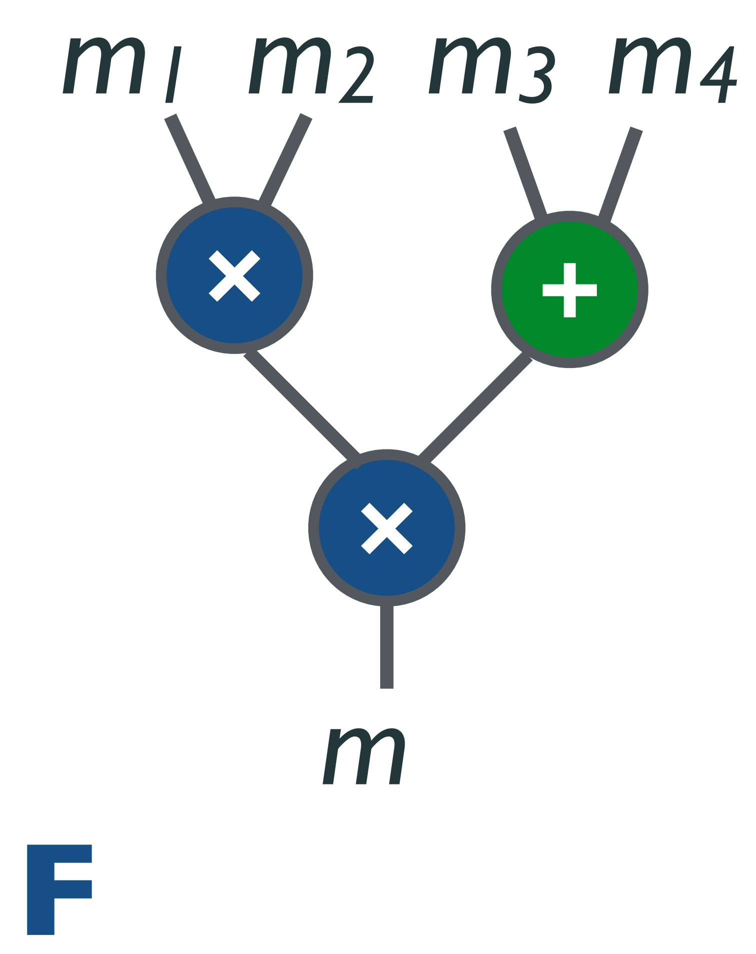
$$\mathbb{Z}_p \ni m \longmapsto \mathsf{ct} = (\mathsf{ct}[0] + \mathsf{ct}[1]Y) \in R_q[Y] = \mathbb{Z}_q[X]/(X^d + 1)$$

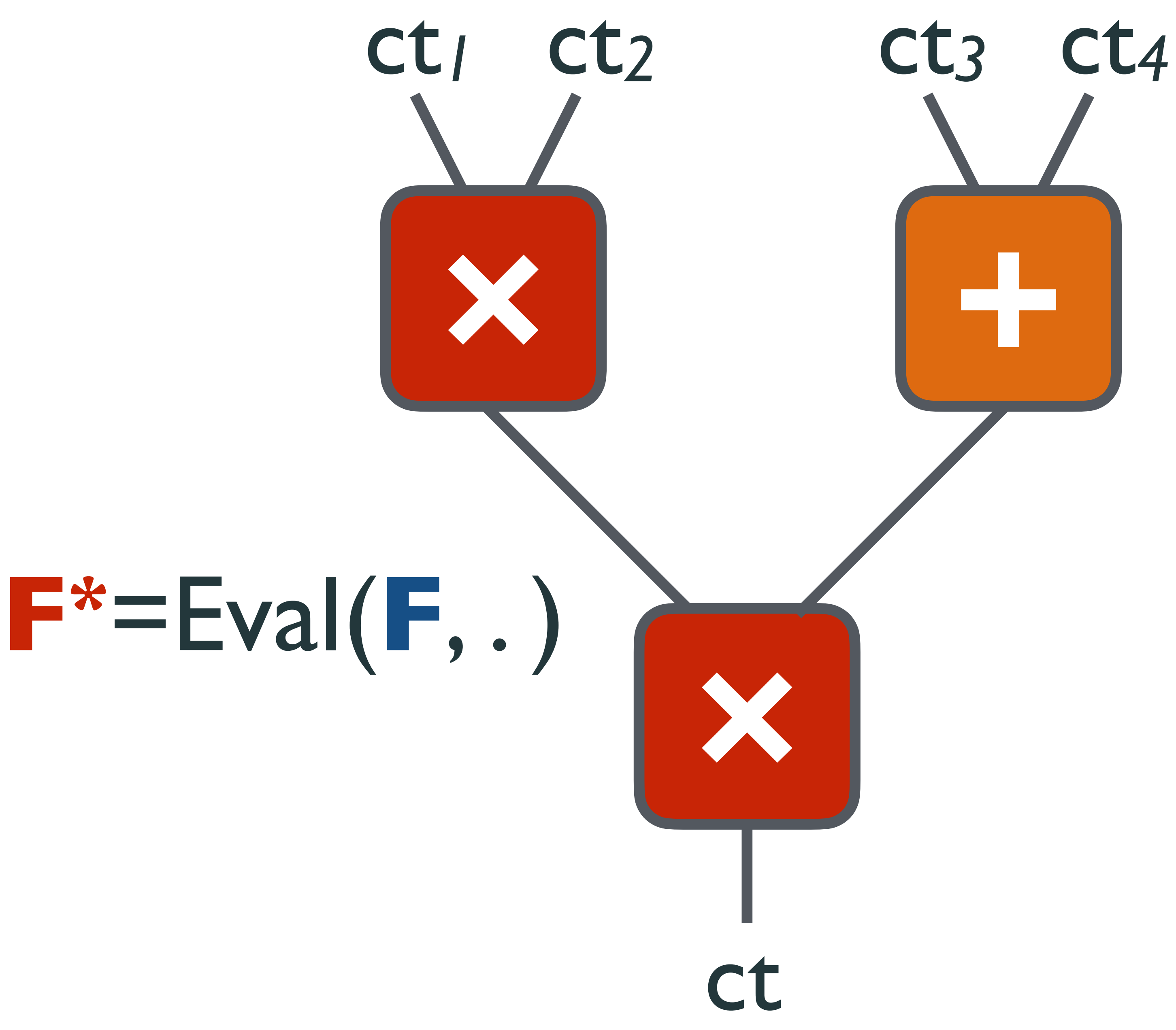
Ciphertext and circuit expansion (extreme case)

plaintexts

ciphertexts

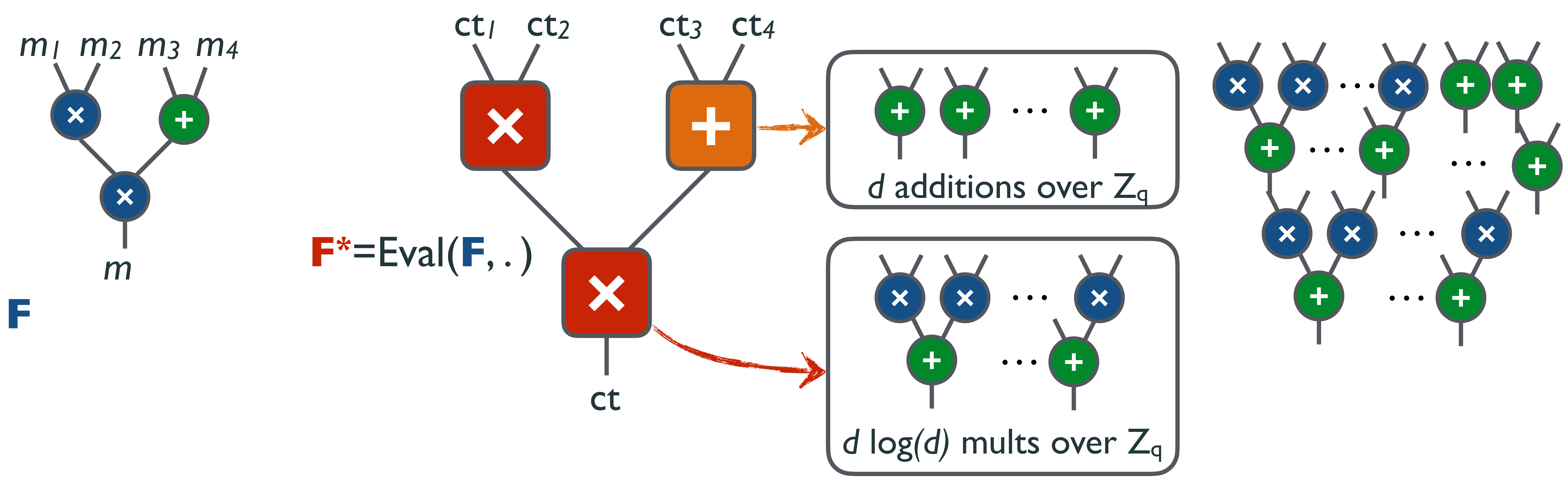
$\mathbb{Z}_p \ni m \longmapsto \mathrm{ct} = (\mathrm{ct}[0] + \mathrm{ct}[1]Y) \in R_q[Y] = \mathbb{Z}_q[X]/(X^d + 1)$





Ciphertext and circuit expansion (extreme case)

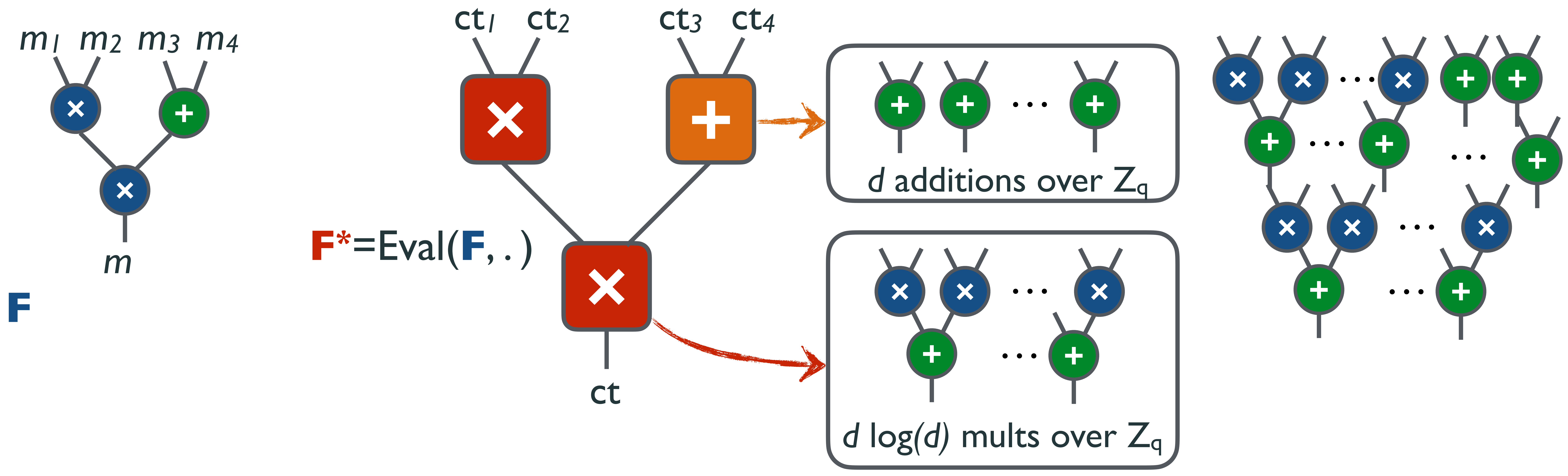
plaintexts $\mathbb{Z}_p \ni m \longmapsto$ ciphertexts $\text{ct} = (\text{ct}[0] + \text{ct}[1]Y) \in R_q[Y] = \mathbb{Z}_q[X]/(X^d + 1)$



Ciphertext and circuit expansion (extreme case)

plaintexts → ciphertexts

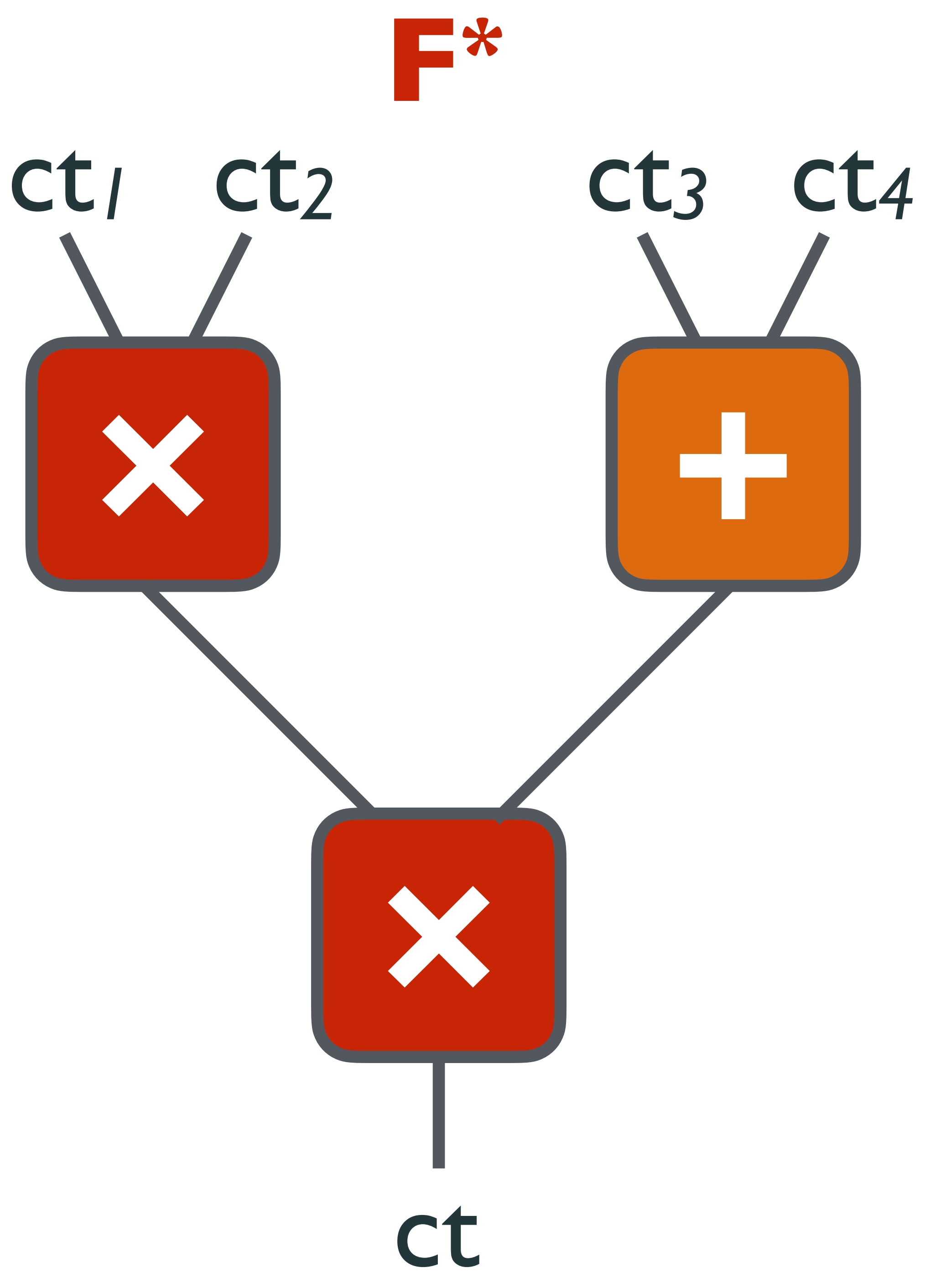
$\mathbb{Z}_p \ni m \longmapsto ct = (ct[0] + ct[1]Y) \in R_q[Y] = \mathbb{Z}_q[X]/(X^d + 1)$



d depends on RingLWE security, e.g., for q≈256-bits, d≈8000

Impact of expansion on the SNARG

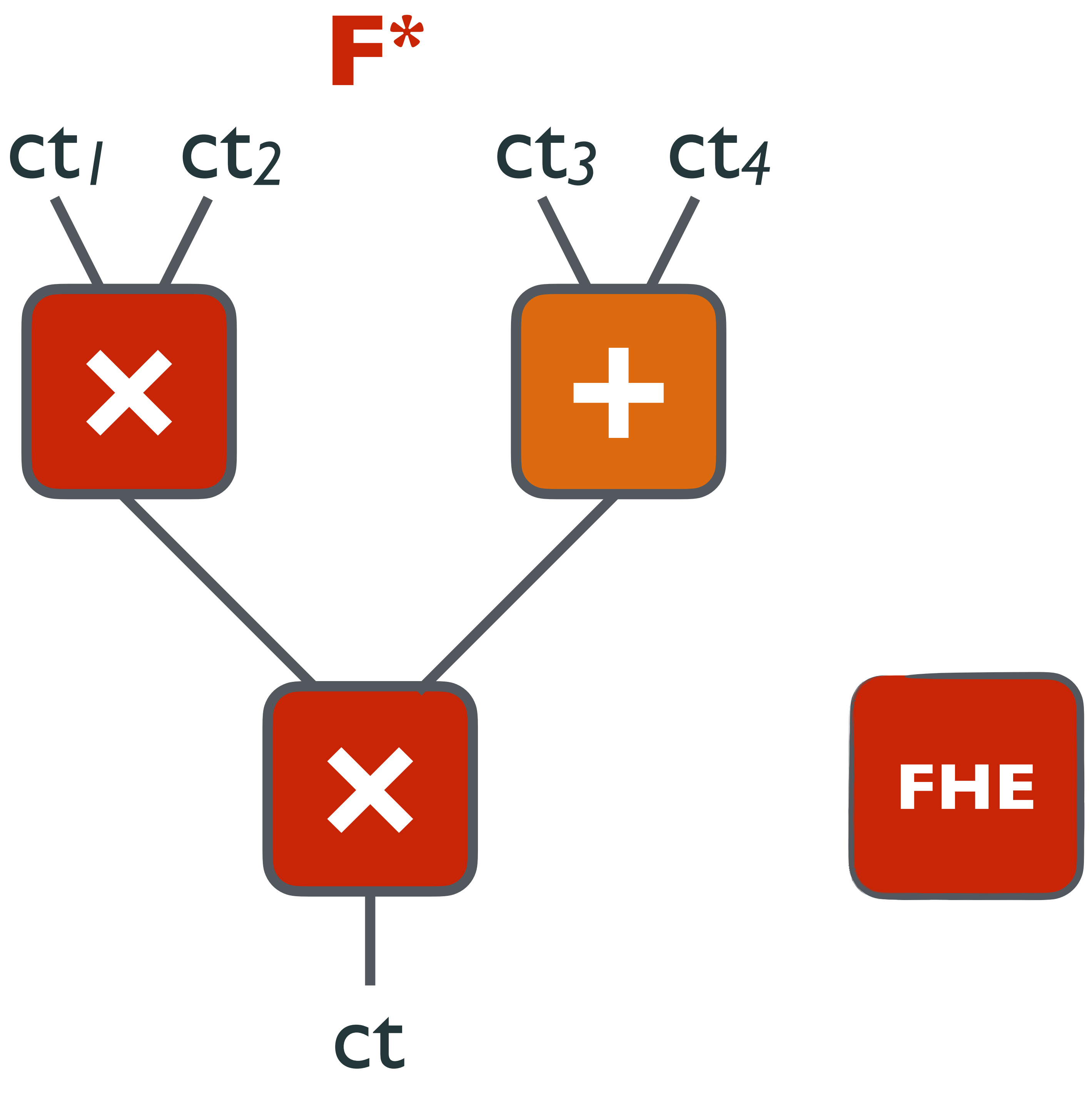
Goal. prove that ct= **F***(ct_1, …, ct_4)



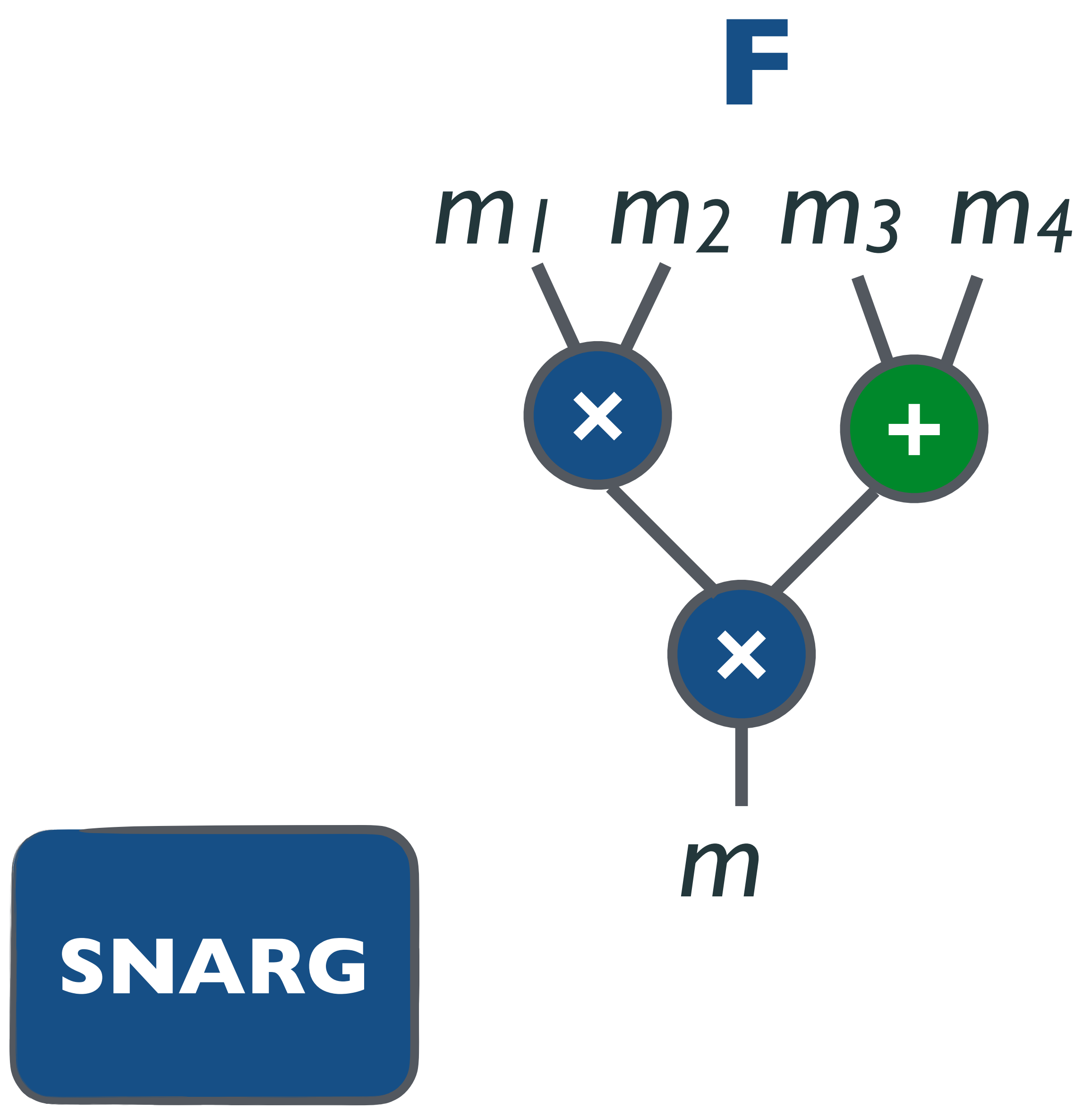
Can we avoid that the **proving complexity depends on *d*?**

Impact of expansion on the SNARG

Goal. prove that ct= **F***(ct$_1$, …, ct$_4$)



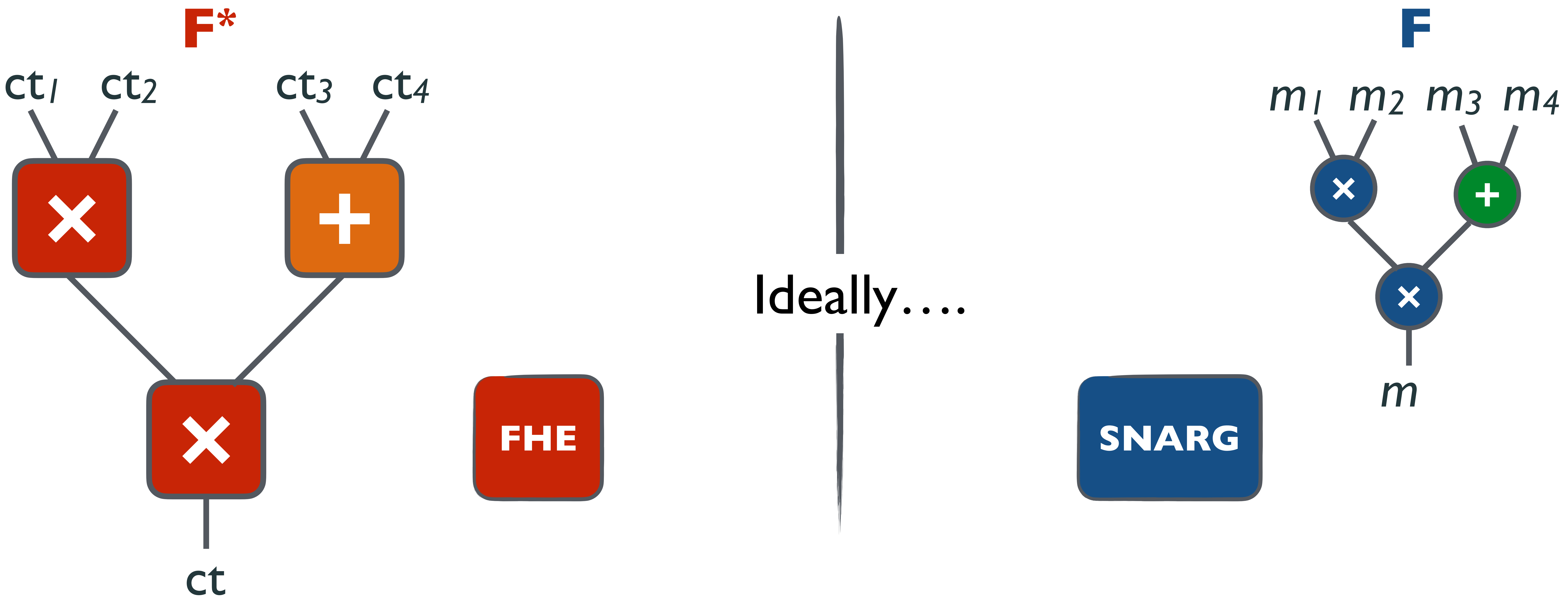
Ideally….



Can we avoid that the **proving complexity depends on *d*?**

Impact of expansion on the SNARG

Goal. prove that ct= **F***(ct$_1$, …, ct$_4$)



Can we avoid that the **proving complexity depends on *d*?**

not really… at least must read inputs/output. We'll see how to achieve O(dn + |F|) instead of O(d log(d) |F|)

Tackling ciphertext/circuit expansion [FNP20]

BV11 HE

$R = \mathbb{Z}[X]/(X^d+1)$, message space $R_p = R/pR$

Encryption

$R_p \ni m \longmapsto \mathsf{ct} = (\mathsf{ct}[0] + \mathsf{ct}[1]Y) \in R_q[Y]$ // ciphertxt space

Addition

$\mathsf{Eval}(+, \mathsf{ct}_1, \mathsf{ct}_2) \rightarrow \mathsf{ct}_1 + \mathsf{ct}_2 \in R_q[Y]$

Basic multiplication

$\mathsf{Eval}(\times, \mathsf{ct}_1, \mathsf{ct}_2) \rightarrow \mathsf{ct}_1 \cdot \mathsf{ct}_2 \in R_q[Y]$ // deg_Y increases

Relinearization + mod switch /noise reduction

$\mathsf{ct} \longmapsto \mathsf{ct}' = \sum_{i=0}^{deg_Y(\mathsf{ct})} \mathsf{ct}[i] \cdot \mathsf{rk}[i] \bmod q \mapsto [\frac{q'}{q}\mathsf{ct}]$

Set q prime s.t. $\mathbb{Z}_q = \mathbb{F}$ supported by the SNARK

SNARK Rq-Π

$R_{F^*} = \{\mathbb{x} = (\mathsf{cm}_x, \mathsf{ct}_y), \mathbb{w} = (\mathsf{ct}_x) :$
$\mathsf{ct}_y = F^*(\mathsf{ct}_x)\ \mathsf{cm}_x = \mathsf{Com}(\mathsf{ct}_x)\}$

Challenges

1) Ciphertext expansion

unless optimized packing, $deg_X(m) \ll d$

~~2) Ciphertext modulus~~

~~q usually not prime~~

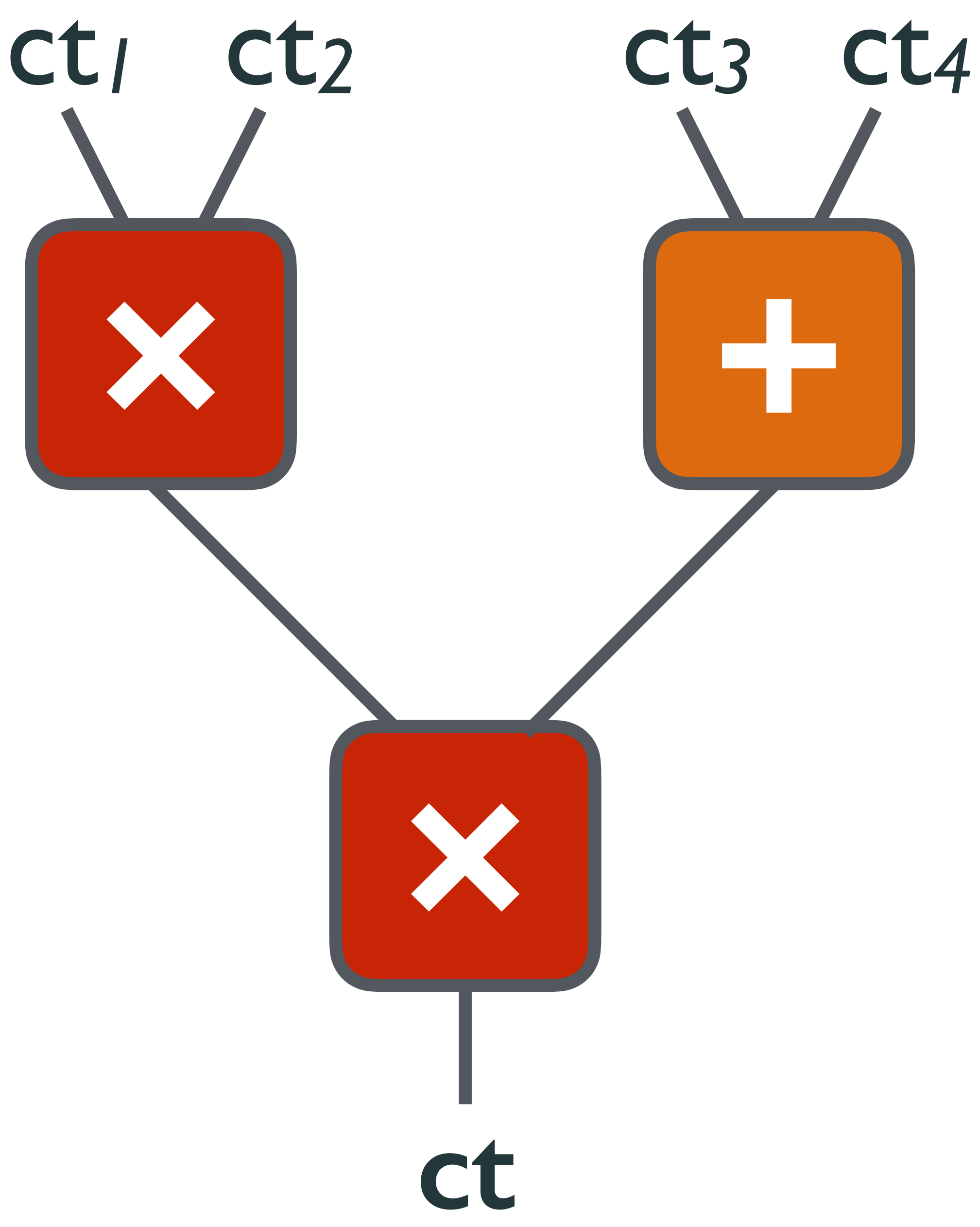
~~3) Non-algebraic operations~~

~~noise control techniques require divisions and rounding~~

[FNP20] D. Fiore. A. Nitulescu, D. Pointcheval. *Boosting Verifiable Computation on Encrypted Data*. PKC 2020

Basic idea of Rq-Π

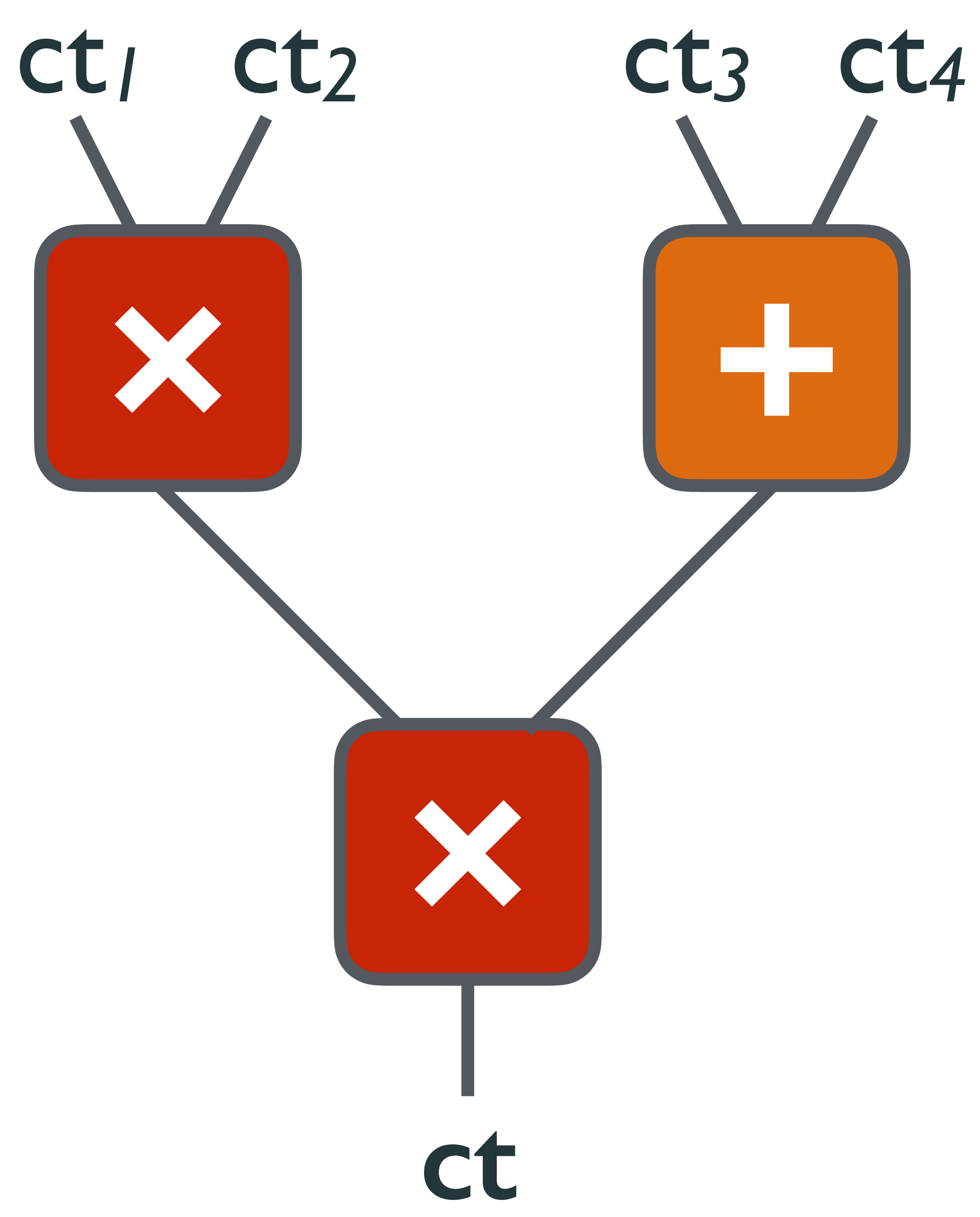
$F^* : R_q^{2n} \rightarrow R_q^{D+1}$

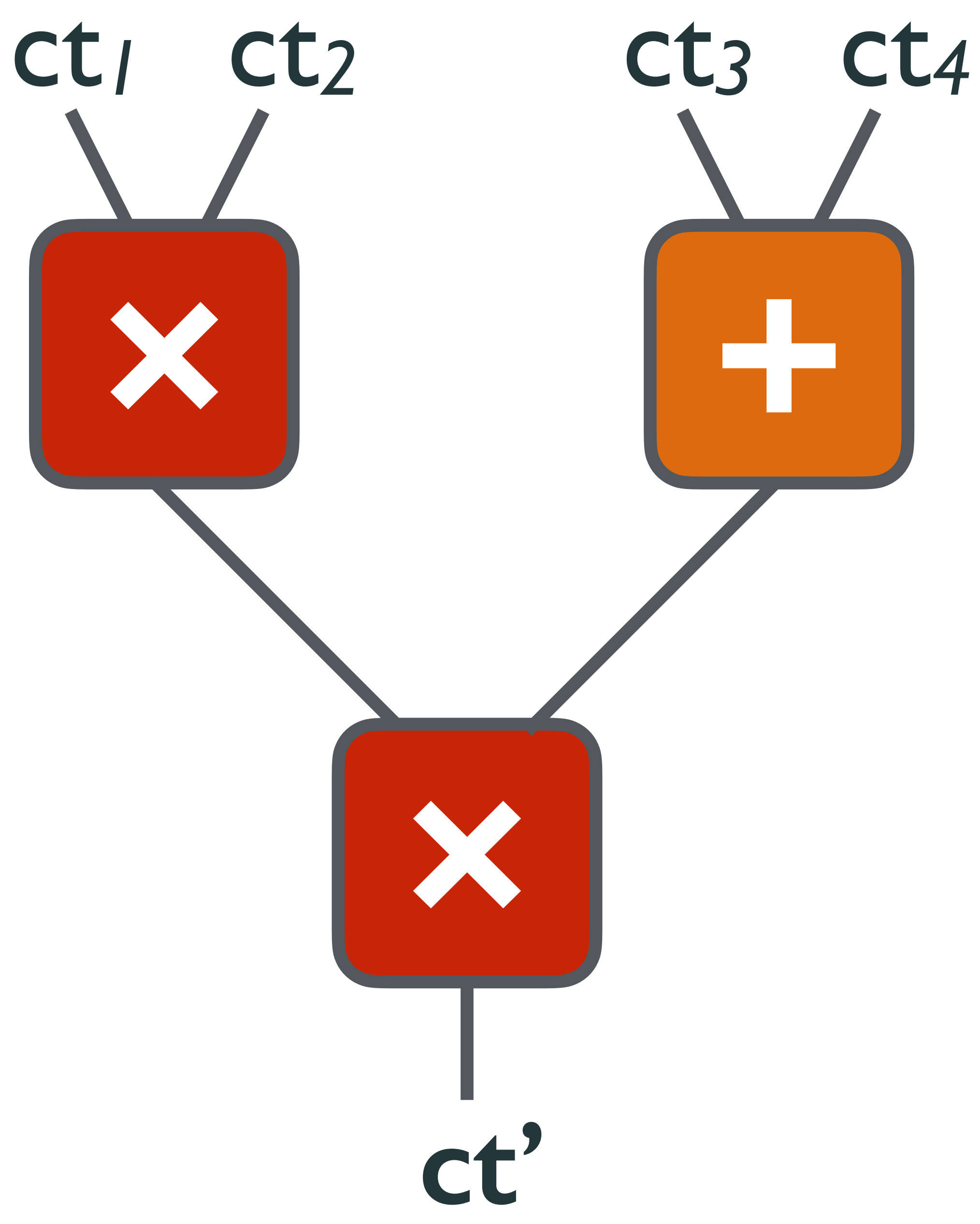


Basic idea of Rq-Π

$F^* : R_q^{2n} \to R_q^{D+1}$

Let $F' : \mathbb{Z}_q[X]^{2n} \to \mathbb{Z}_q[X]^{D+1}$
be as F^* w/o $\bmod X^d + 1$





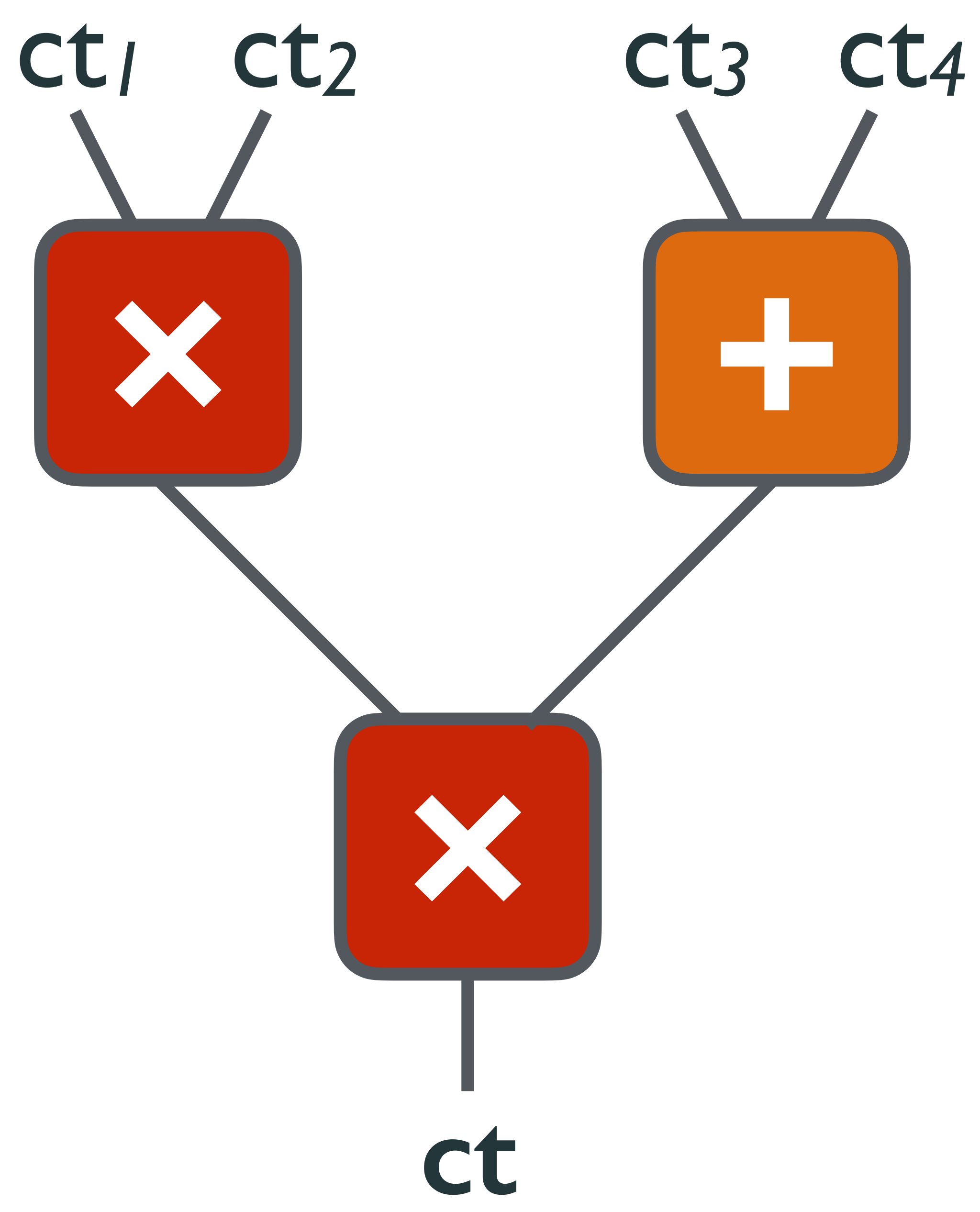
$\mathsf{ct}(X) = F^*(\{\mathsf{ct}_j(X)\}_j) \iff \exists H(X) : \mathsf{ct}(X) = F'(\{\mathsf{ct}_j(X)\}_j) - H(X)(X^d + 1)$

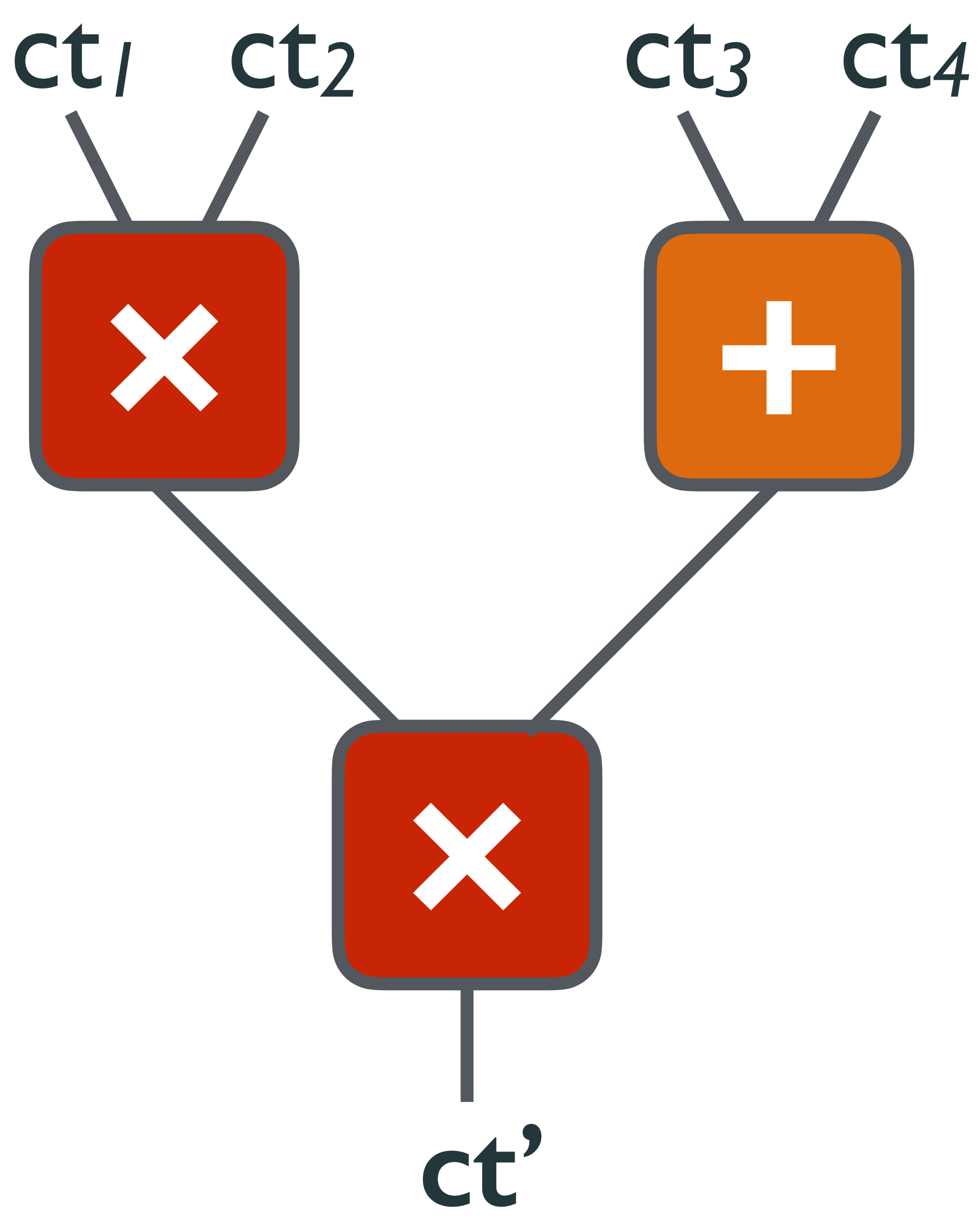
Basic idea of Rq-Π

$F^* : R_q^{2n} \to R_q^{D+1}$

Let $F' : \mathbb{Z}_q[X]^{2n} \to \mathbb{Z}_q[X]^{D+1}$ be as F^* w/o $\bmod X^d + 1$

“compress” & prove



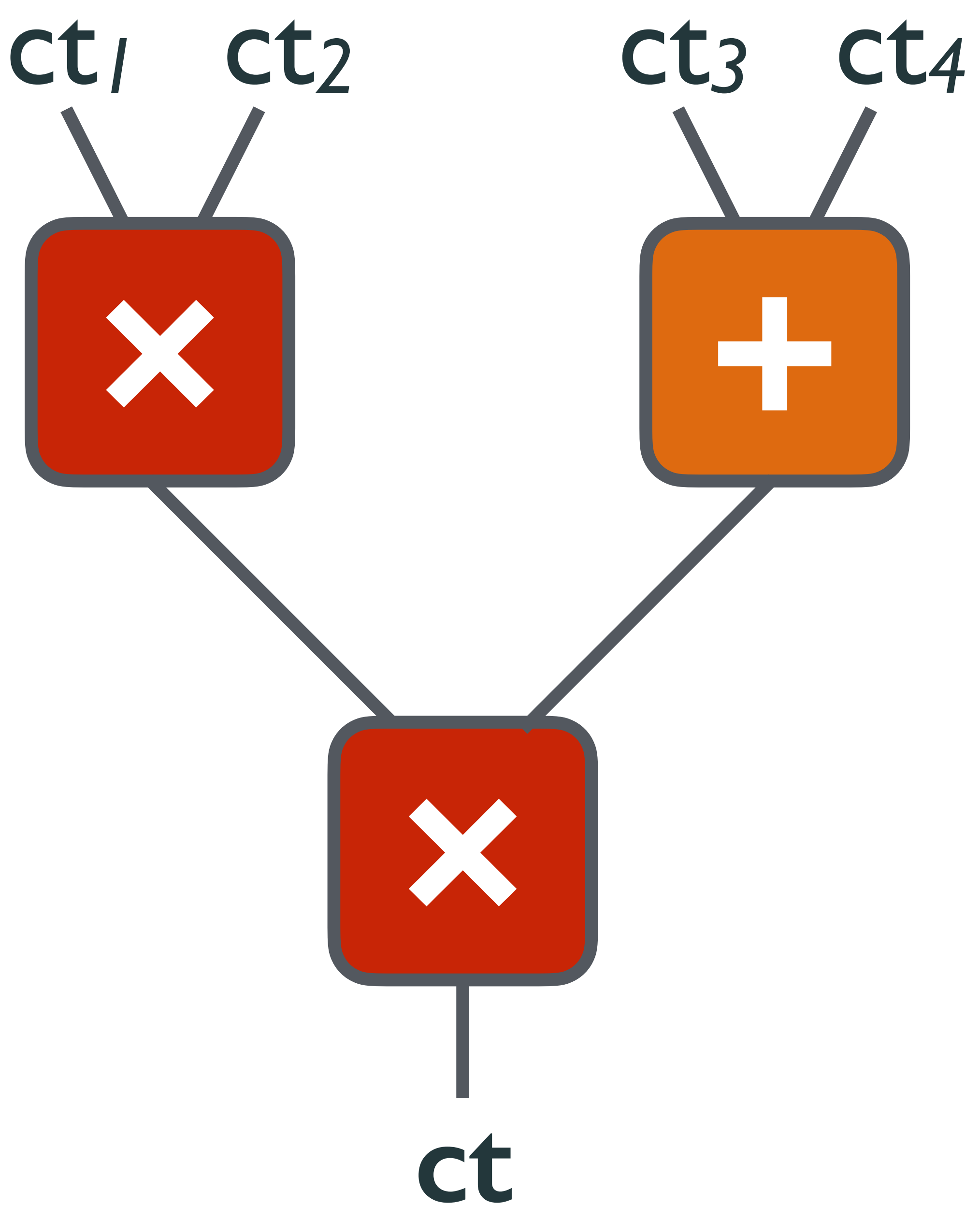


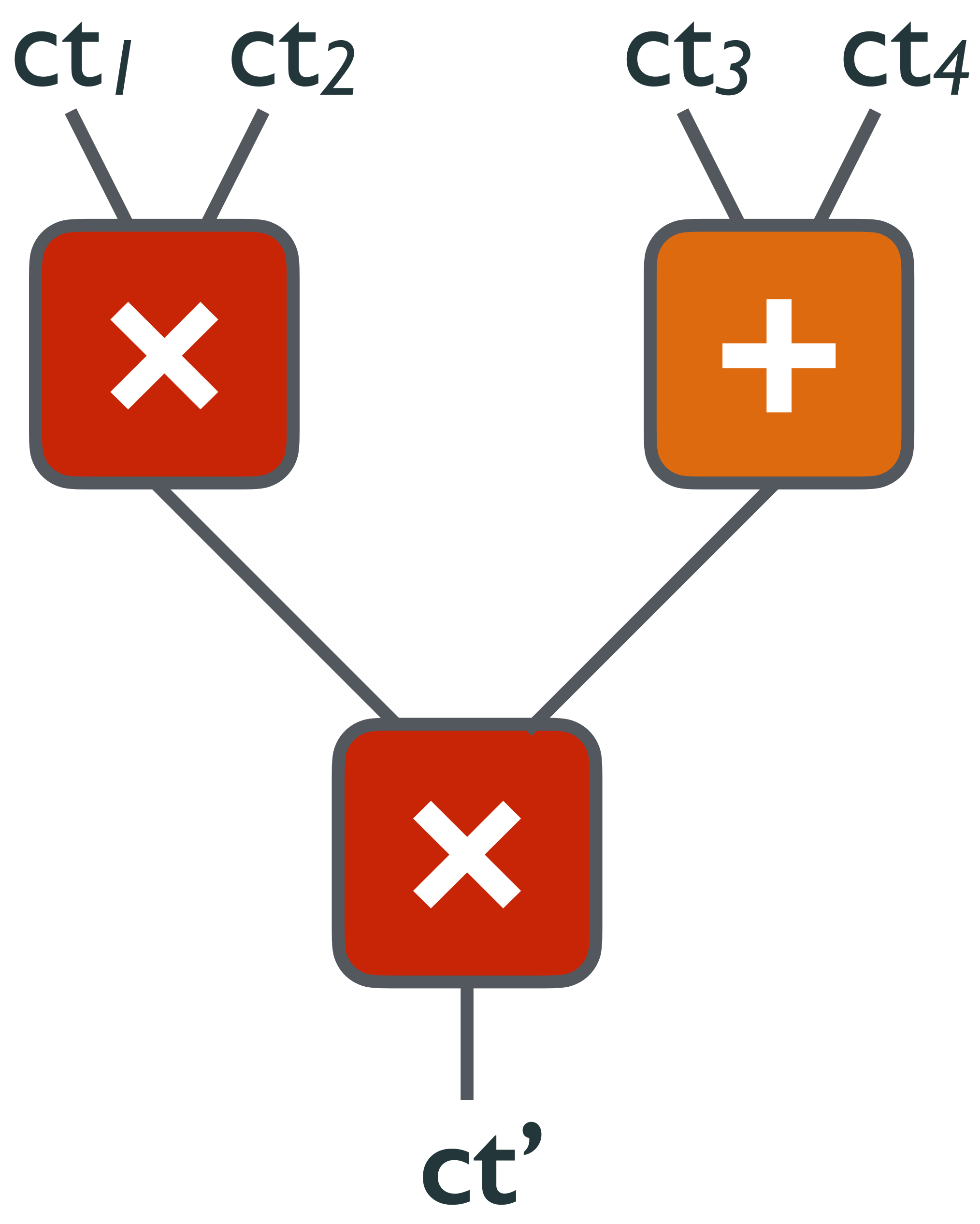
$$\mathrm{ct}(X) = F^*(\{\mathrm{ct}_j(X)\}_j) \iff \exists H(X) : \mathrm{ct}(X) = F'(\{\mathrm{ct}_j(X)\}_j) - H(X)(X^d + 1)$$

Basic idea of Rq-Π

$F^* : R_q^{2n} \to R_q^{D+1}$

Let $F' : \mathbb{Z}_q[X]^{2n} \to \mathbb{Z}_q[X]^{D+1}$ be as F^* w/o $\mathrm{mod}\ X^d + 1$





"compress"
&
prove

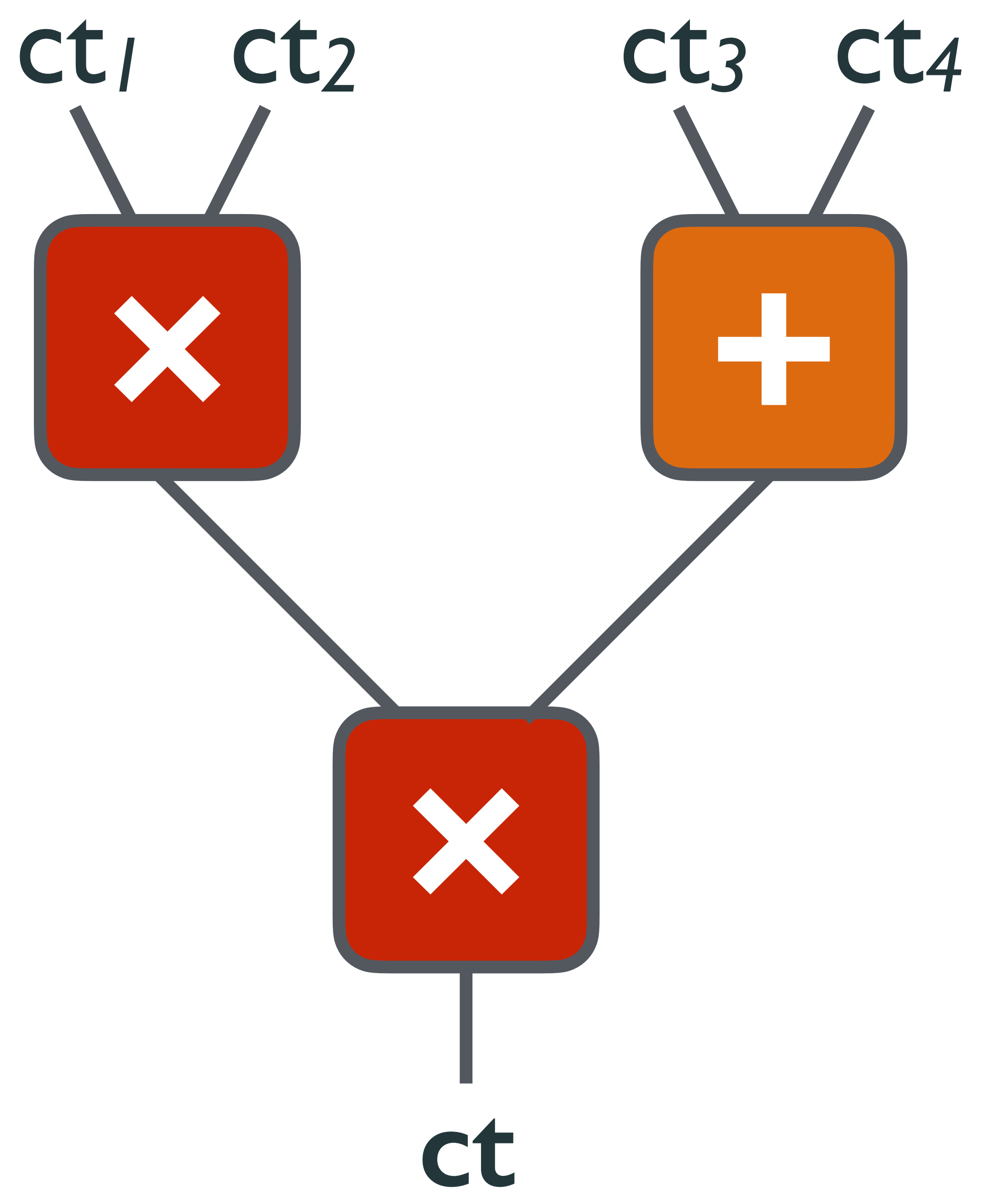
$\{\mathrm{ct}_j\}_j$, ct, H →

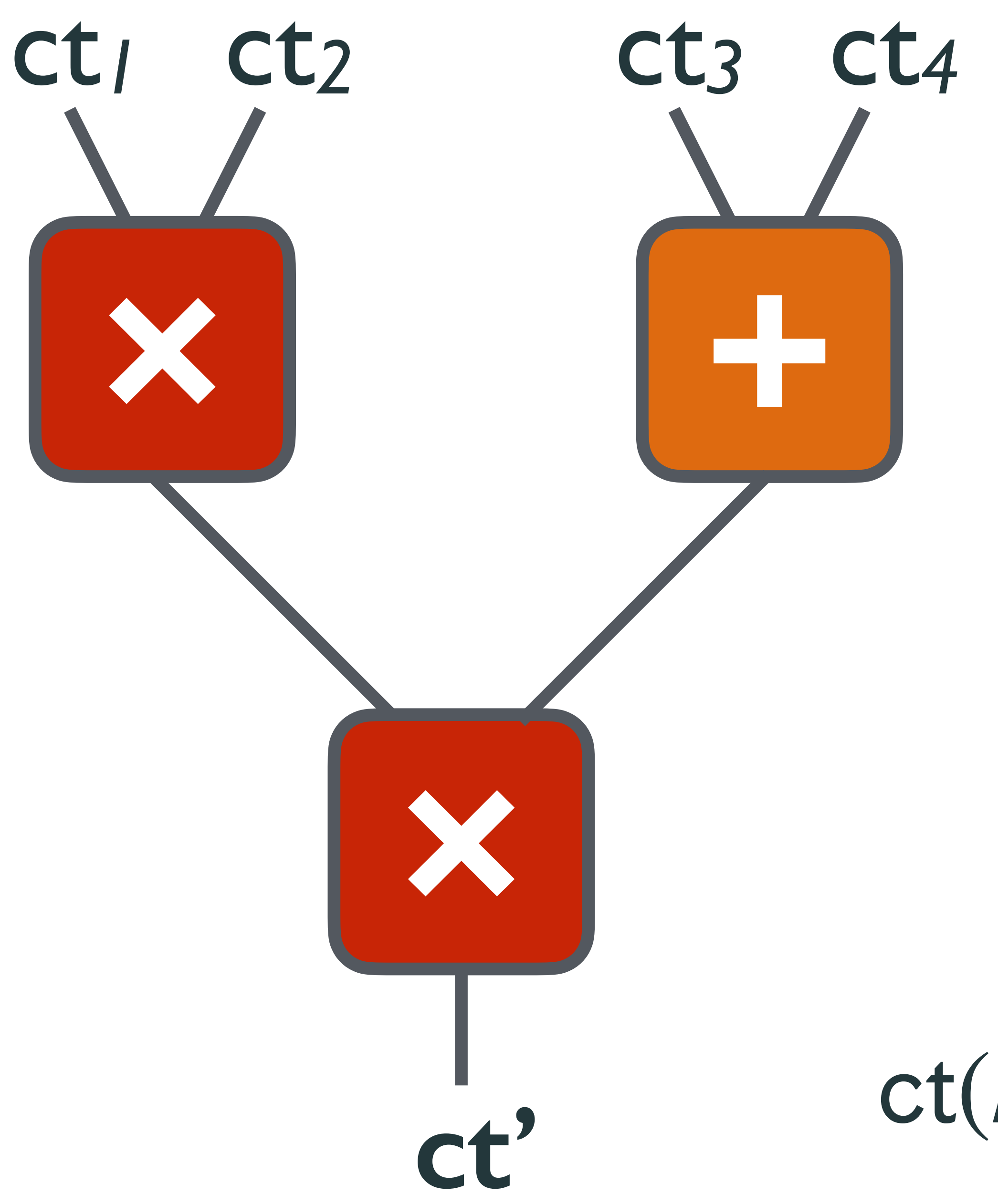
$$\mathrm{ct}(X) = F^*(\{\mathrm{ct}_j(X)\}_j) \iff \exists H(X) : \mathrm{ct}(X) = F'(\{\mathrm{ct}_j(X)\}_j) - H(X)(X^d + 1)$$

Basic idea of Rq-Π

$F^* : R_q^{2n} \to R_q^{D+1}$

Let $F' : \mathbb{Z}_q[X]^{2n} \to \mathbb{Z}_q[X]^{D+1}$ be as F^* w/o $\bmod X^d + 1$





"compress" & prove

$\{\mathsf{ct}_j\}_j, \mathsf{ct}, H$ →

$k \leftarrow_\$ \mathbb{Z}_q$ ←

Prove that

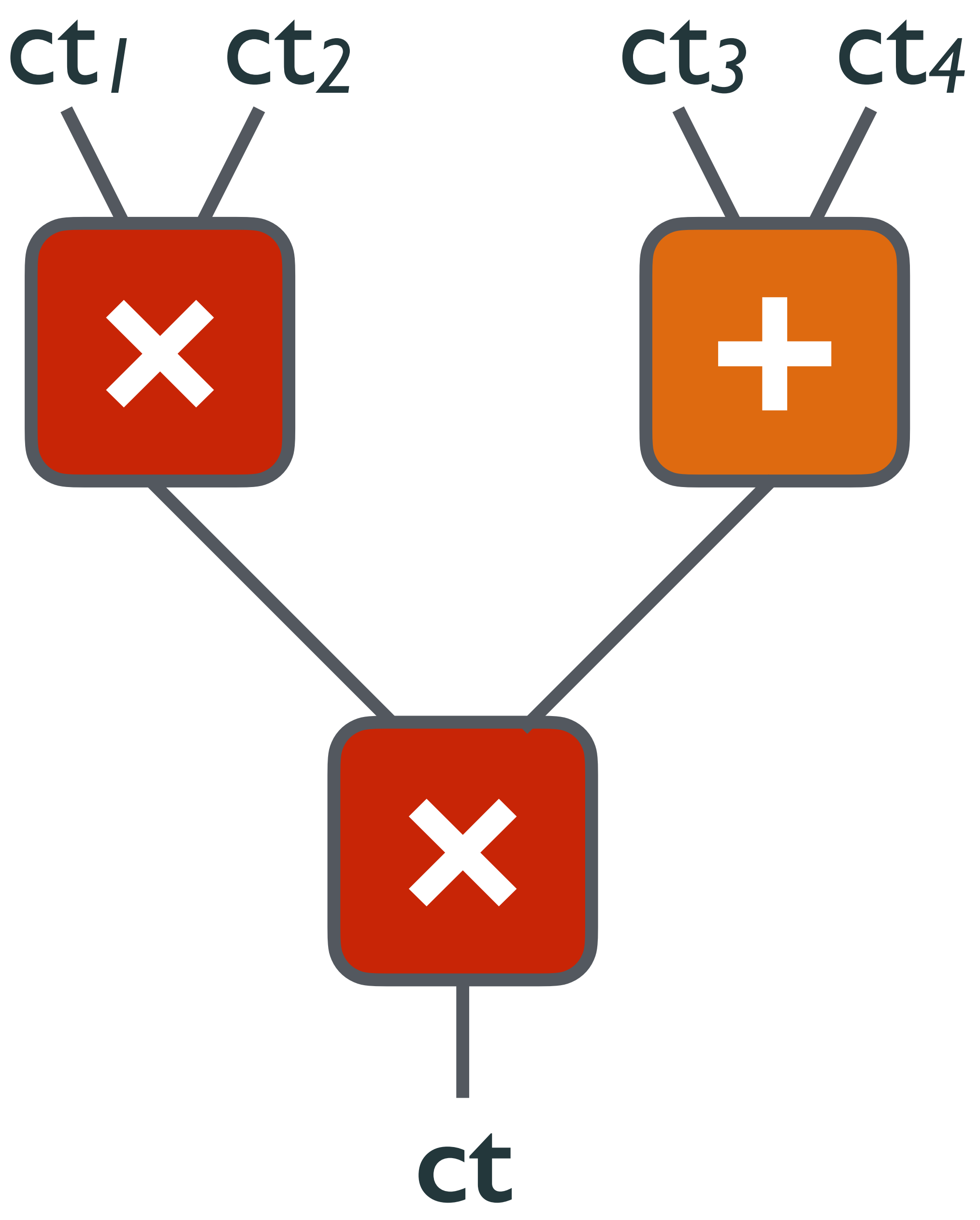
$\mathsf{ct}(k) + H(k)(k^d + 1) = F(\{\mathsf{ct}_j(k)\}_j)$ →

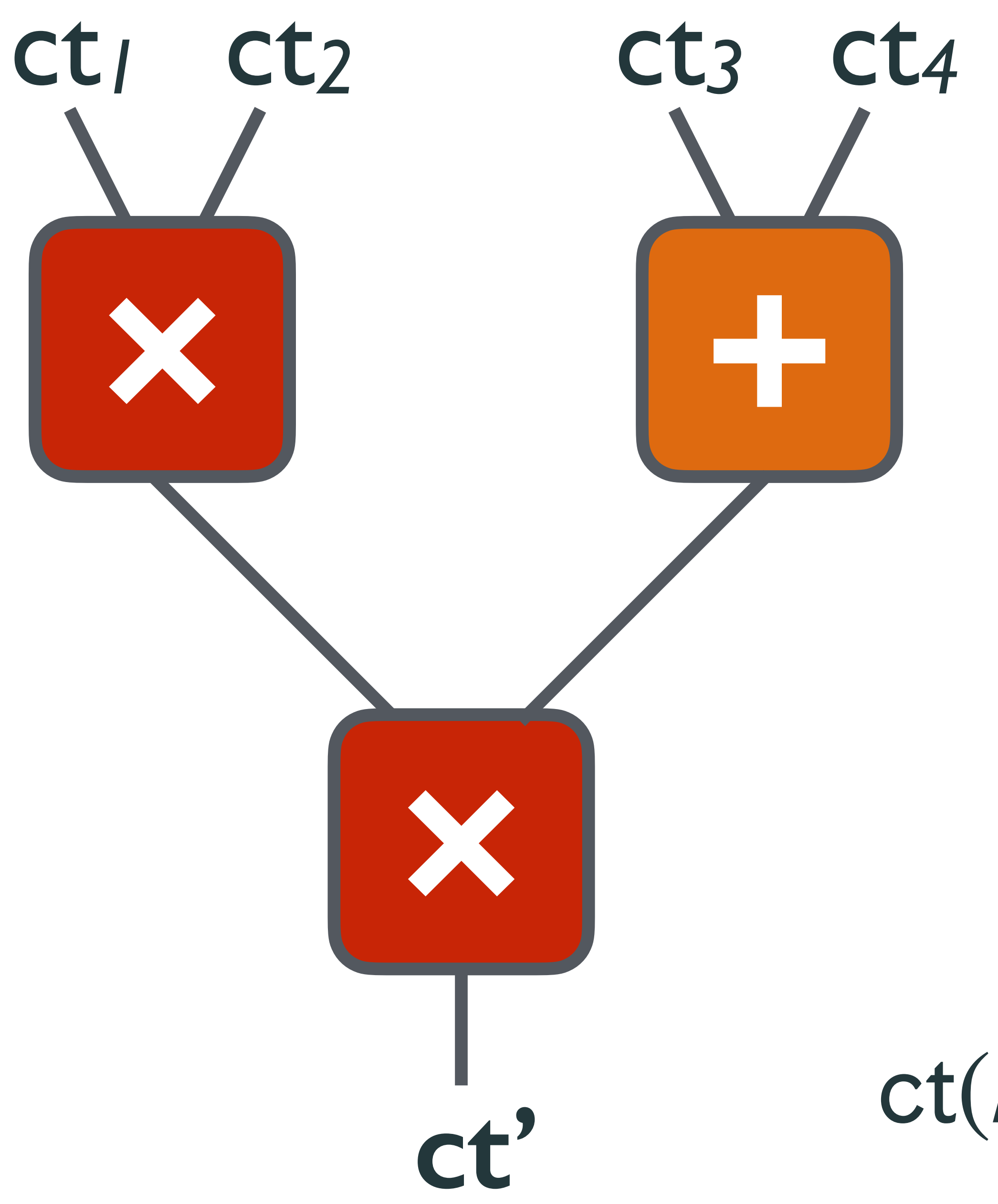
$$\mathsf{ct}(X) = F^*(\{\mathsf{ct}_j(X)\}_j) \iff \exists H(X) : \mathsf{ct}(X) = F'(\{\mathsf{ct}_j(X)\}_j) - H(X)(X^d + 1)$$

Basic idea of Rq-Π

$F^* : R_q^{2n} \to R_q^{D+1}$

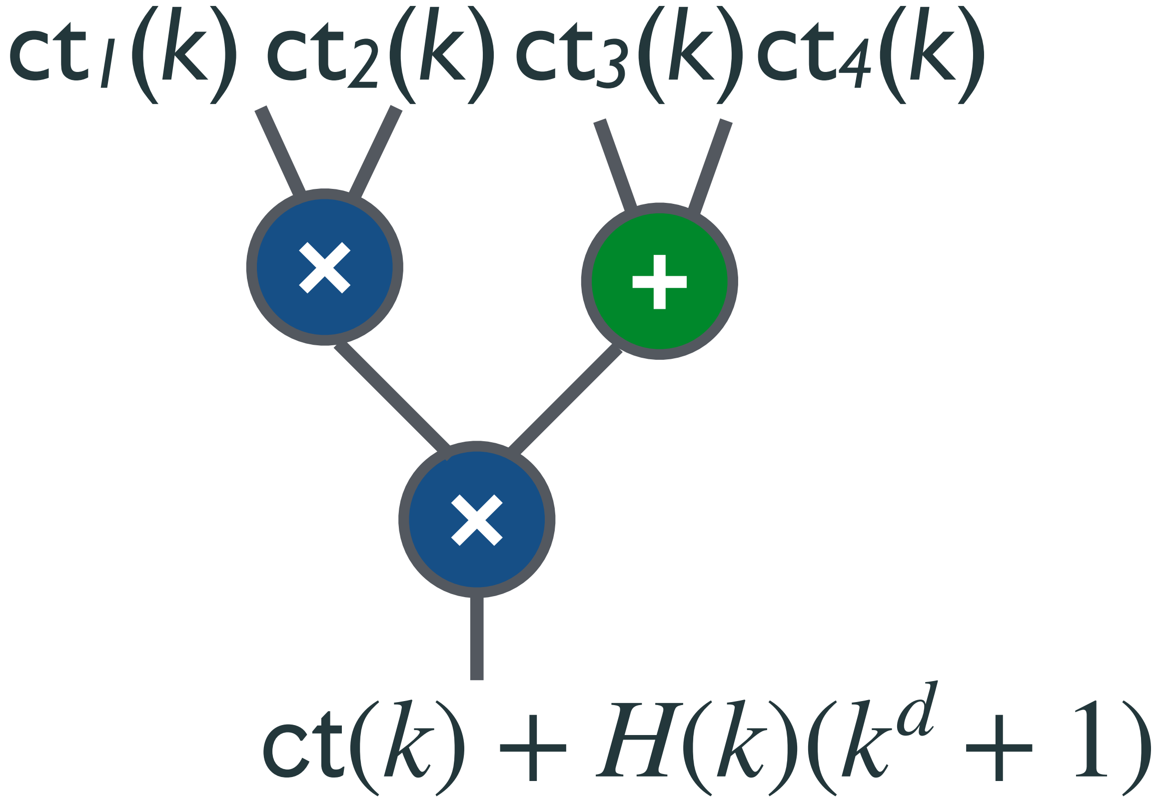
Let $F' : \mathbb{Z}_q[X]^{2n} \to \mathbb{Z}_q[X]^{D+1}$ be as F^* w/o $\bmod X^d + 1$





"compress" & prove

F



$\{\text{ct}_j\}_j$, ct, H →

← $k \leftarrow_{\$} \mathbb{Z}_q$

Prove that

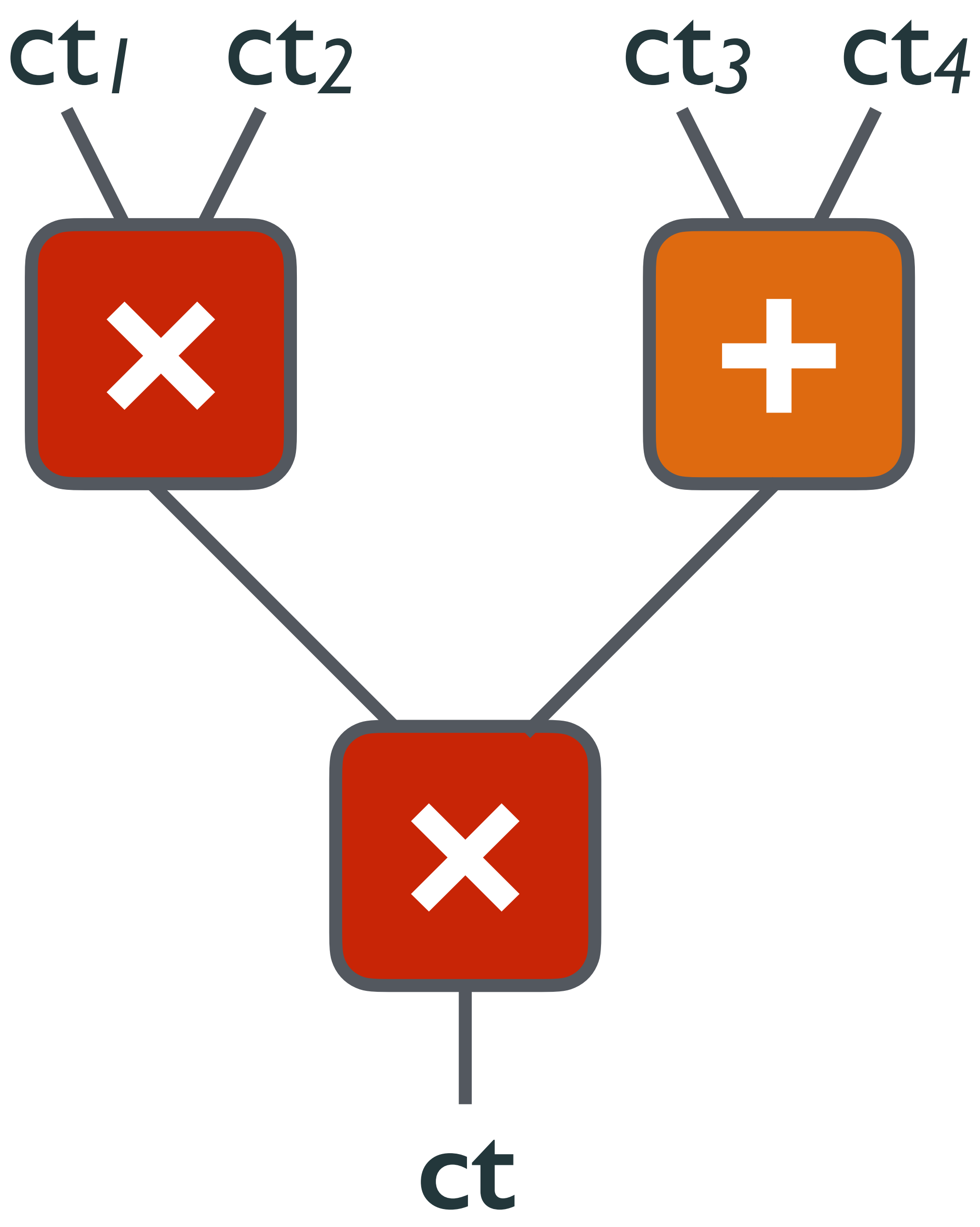
$\text{ct}(k) + H(k)(k^d + 1) = F(\{\text{ct}_j(k)\}_j)$ →

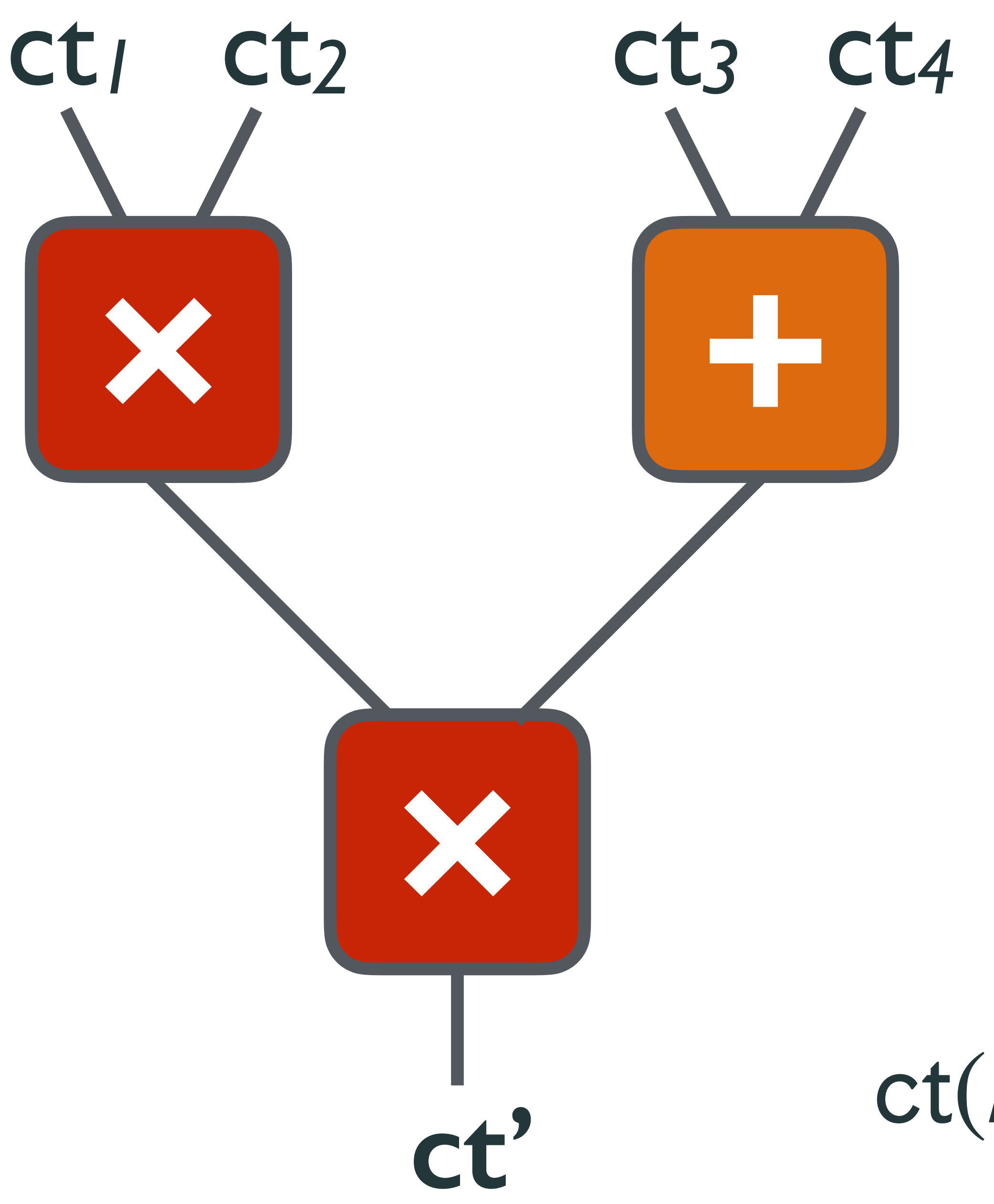
$$\text{ct}(X) = F^*(\{\text{ct}_j(X)\}_j) \iff \exists H(X) : \text{ct}(X) = F'(\{\text{ct}_j(X)\}_j) - H(X)(X^d + 1)$$

Basic idea of Rq-Π

$F^* : R_q^{2n} \to R_q^{D+1}$

Let $F' : \mathbb{Z}_q[X]^{2n} \to \mathbb{Z}_q[X]^{D+1}$ be as F^* w/o $\bmod X^d + 1$





"compress"
&
prove

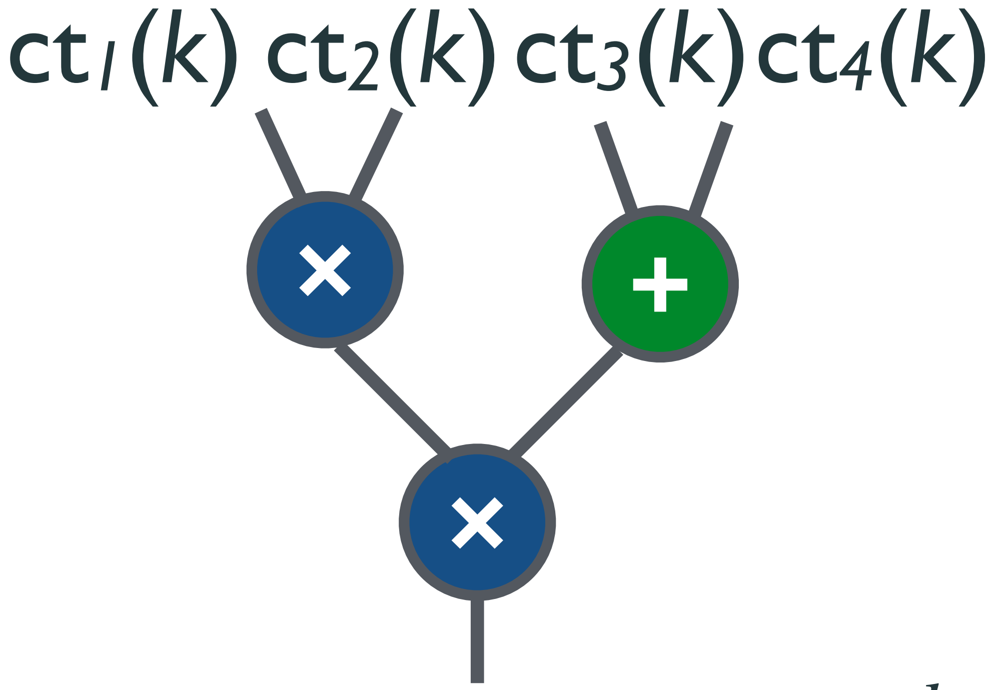
$\{\mathsf{ct}_j\}_j$, ct, H →

$k \leftarrow_\$ \mathbb{Z}_q$ ←

Prove that

$\mathsf{ct}(k) + H(k)(k^d+1) = F(\{\mathsf{ct}_j(k)\}_j)$ →

F



$\mathsf{ct}(k) + H(k)(k^d+1)$

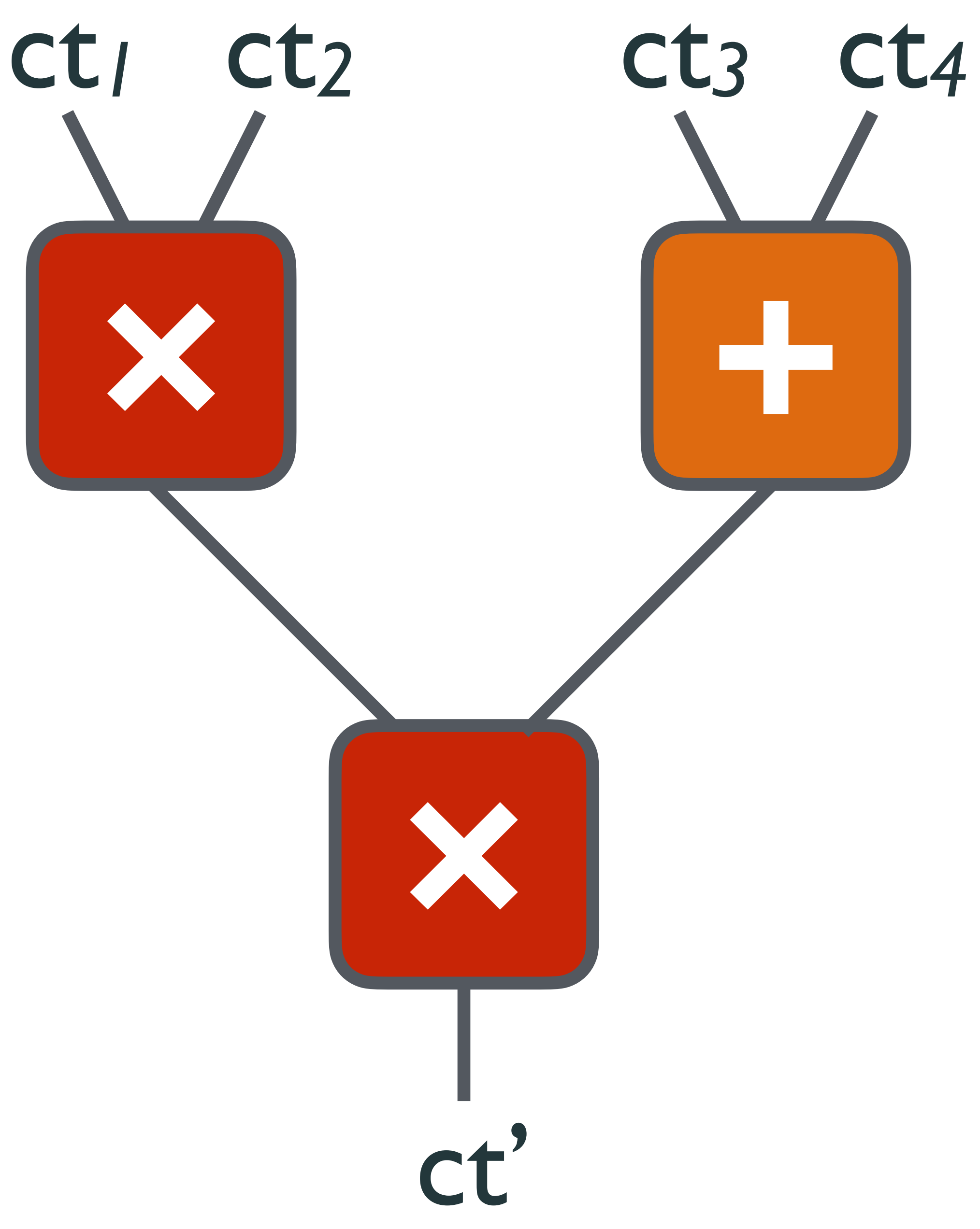
based on hom property of evaluation map $\mathbb{Z}_q[X] \overset{k}{\longmapsto} \mathbb{Z}_q$

$\mathsf{ct}(X) = F^*(\{\mathsf{ct}_j(X)\}_j) \iff \exists H(X) : \mathsf{ct}(X) = F'(\{\mathsf{ct}_j(X)\}_j) - H(X)(X^d+1)$

➡ $\mathsf{ct}(k) + H(k)(k^d+1) = F(\{\mathsf{ct}_j(k)\}_j)$

Modular realization of Rq-Π

$F' : \mathbb{Z}_q[X]^{2n} \to \mathbb{Z}_q[X]^{D+1}$



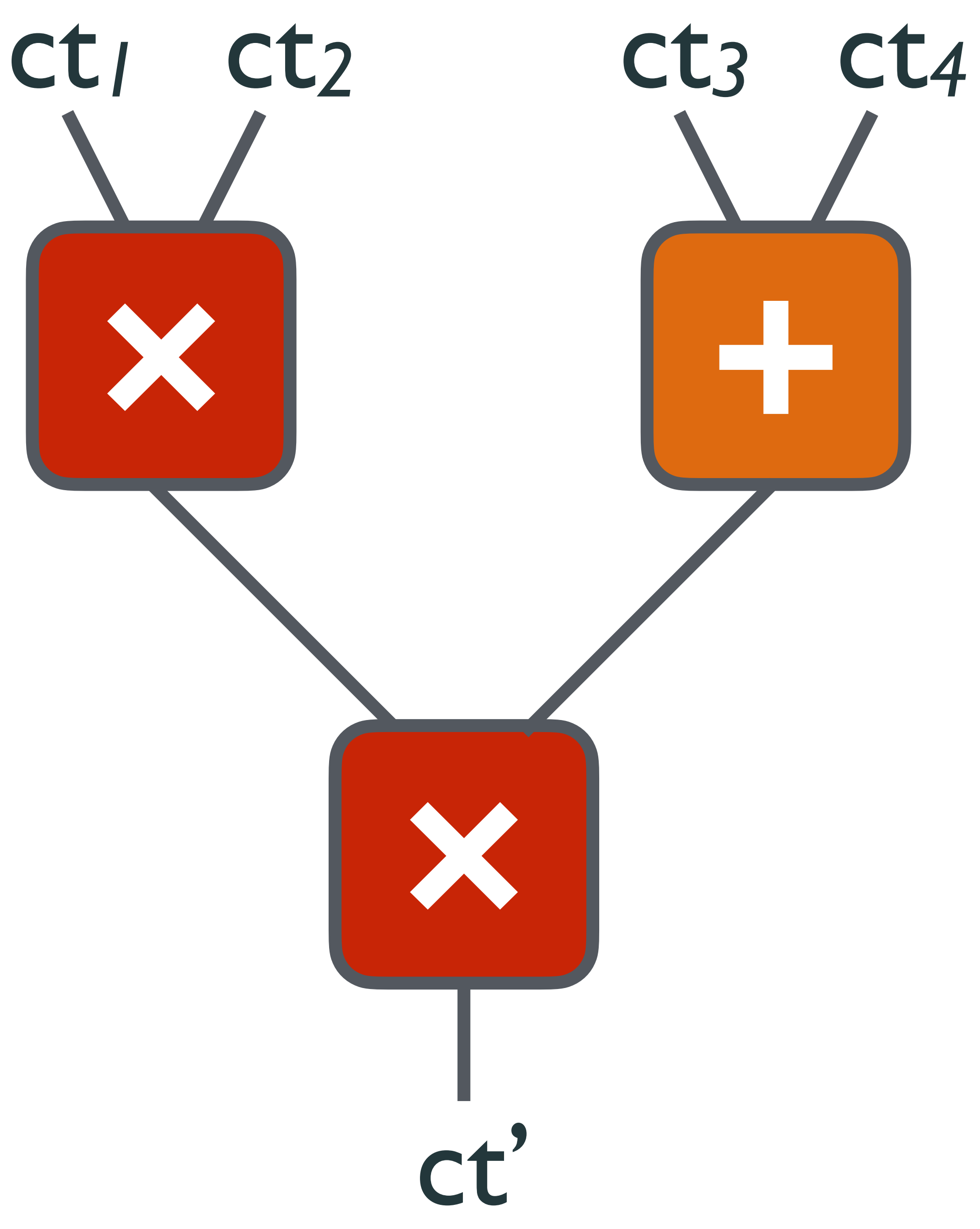
"**commit,** compress"

&

prove

Modular realization of Rq-Π

$F' : \mathbb{Z}_q[X]^{2n} \to \mathbb{Z}_q[X]^{D+1}$



"**commit,** **compress**"

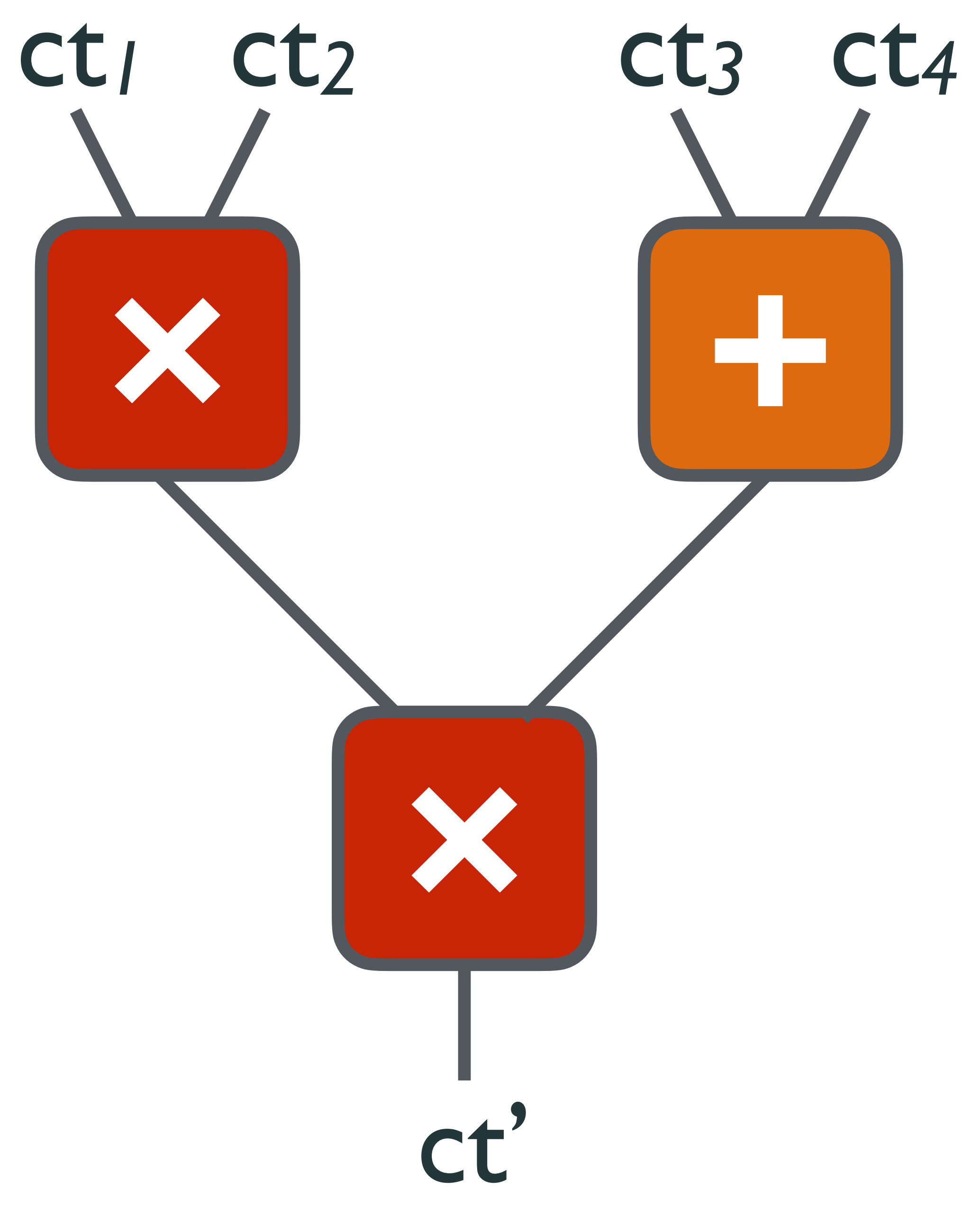
&

prove

Com($\{ct_j\}_j$, ct, H)

Modular realization of Rq-Π

$F' : \mathbb{Z}_q[X]^{2n} \to \mathbb{Z}_q[X]^{D+1}$



"**commit,** **compress**"

&

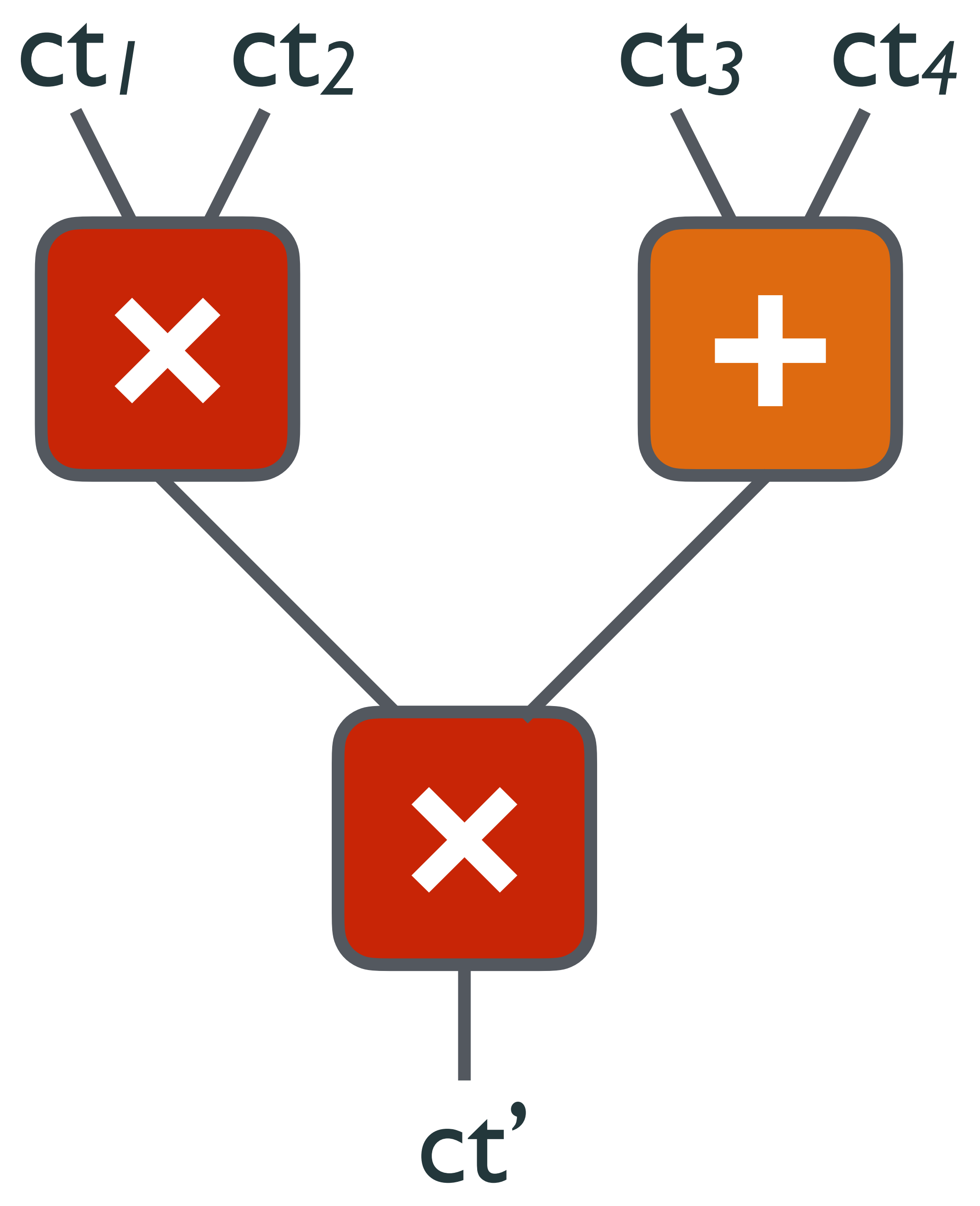
prove

Com($\{ct_j\}_j$, ct, H)

$k \leftarrow_\$ \mathbb{Z}_q$

Modular realization of Rq-Π

$F' : \mathbb{Z}_q[X]^{2n} \rightarrow \mathbb{Z}_q[X]^{D+1}$



"**commit,** **compress**"

&

prove

Com($\{ct_j\}_j$, ct, H) →

$k \leftarrow_{\$} Z_q$ ←

Com($\{c_j\}_j$, c, h)

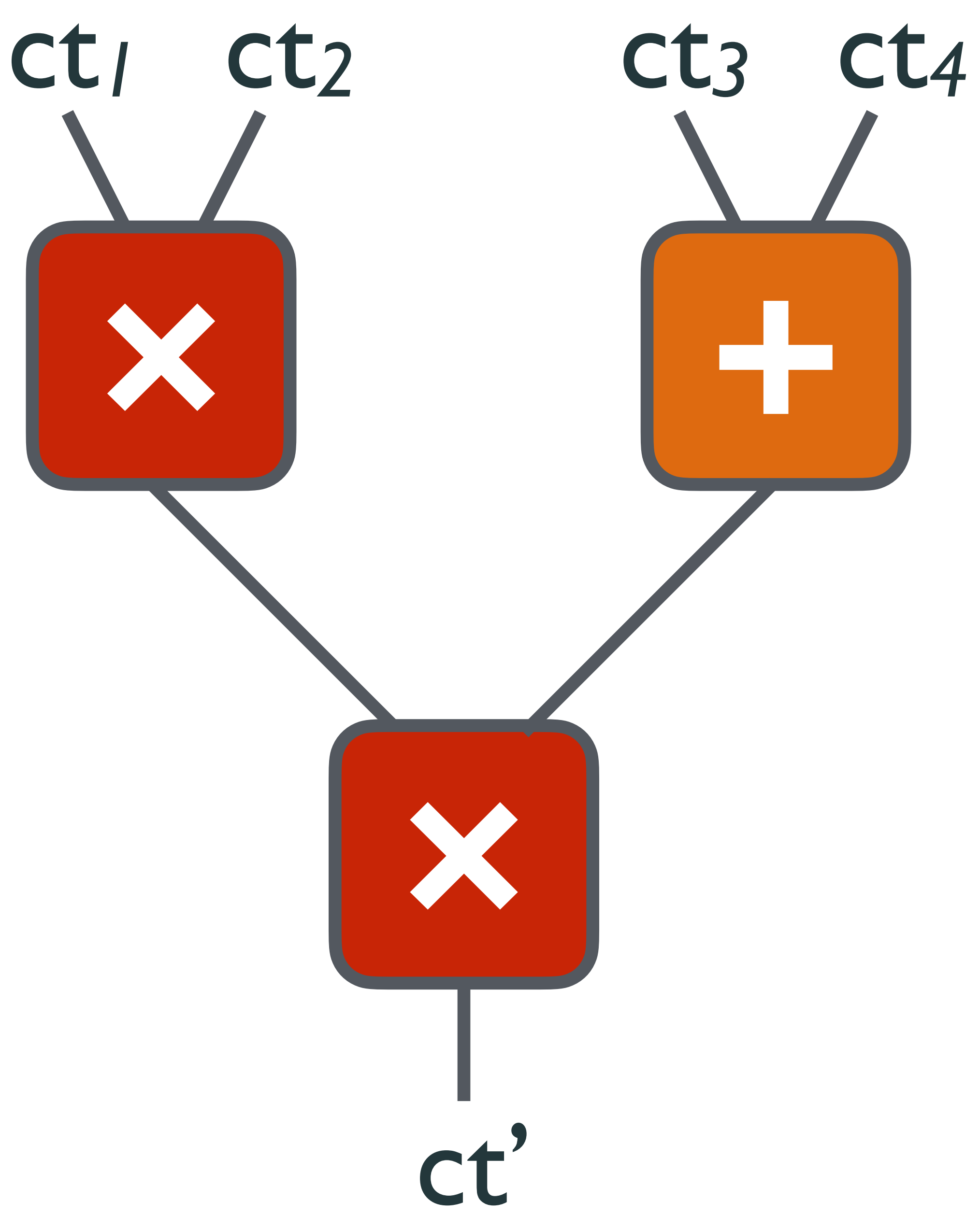
π_{ev} →

prove

$\forall j$: $c_j = ct_j(k)$, $c = ct(k)$, $h = H(k)$

Modular realization of Rq-Π

$F' : \mathbb{Z}_q[X]^{2n} \rightarrow \mathbb{Z}_q[X]^{D+1}$



"commit, compress"

&

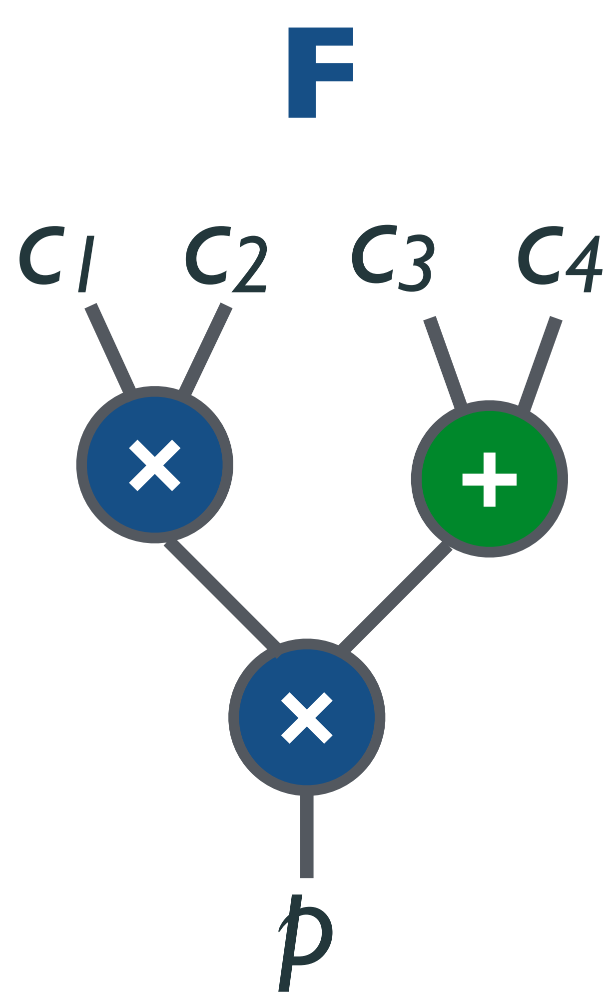
prove

Com($\{ct_j\}_j$, ct, H)

$k \leftarrow_\$ \mathrm{Z_q}$

Com($\{c_j\}_j$, c, h)

π_{ev} π_F



prove

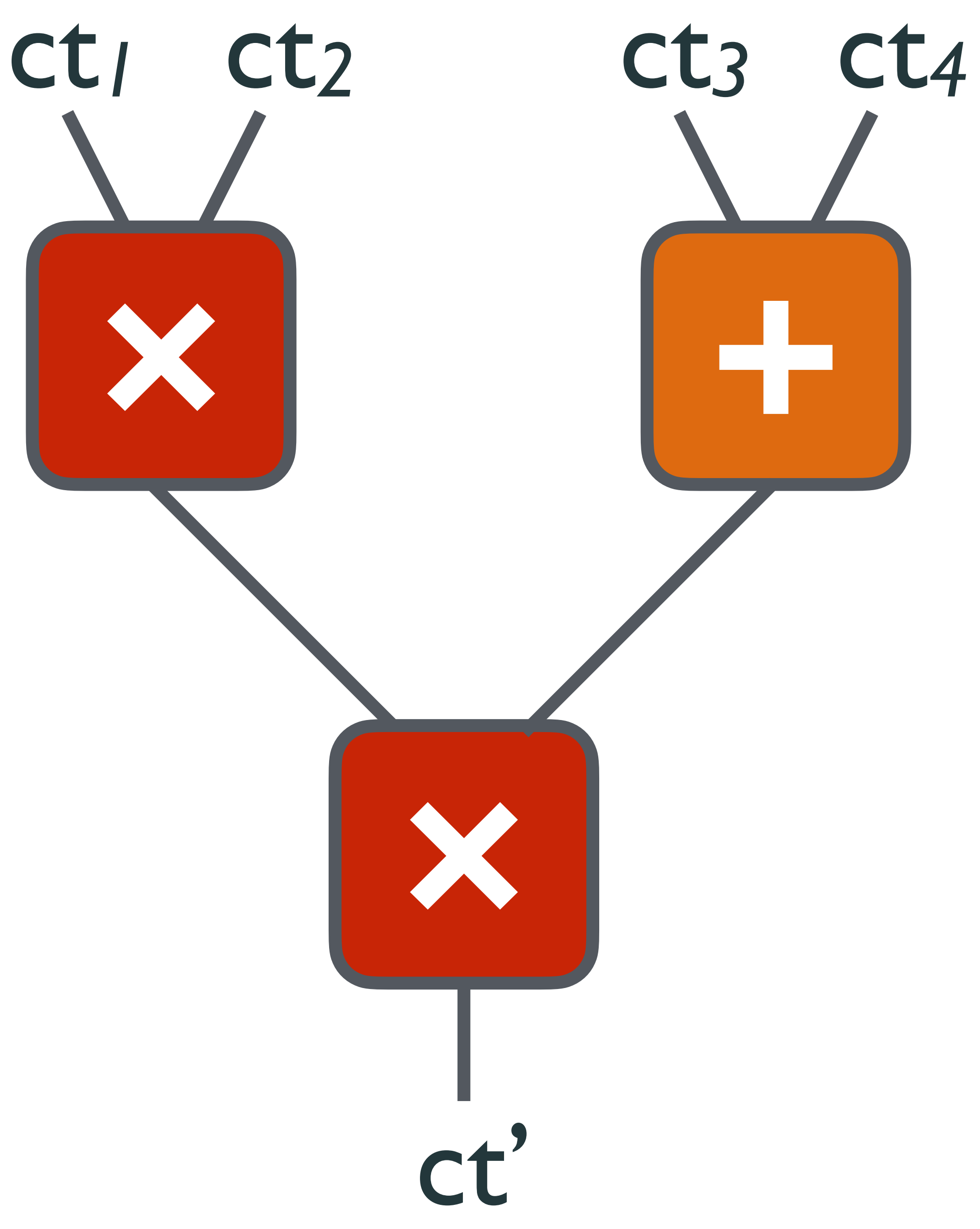
$\forall j$: $c_j = ct_j(k)$, $c = ct(k)$, $h = H(k)$

prove

$c = \boldsymbol{F}(\{c_j\}) - h(k^d + 1)$

Modular realization of Rq-Π

$F' : \mathbb{Z}_q[X]^{2n} \rightarrow \mathbb{Z}_q[X]^{D+1}$



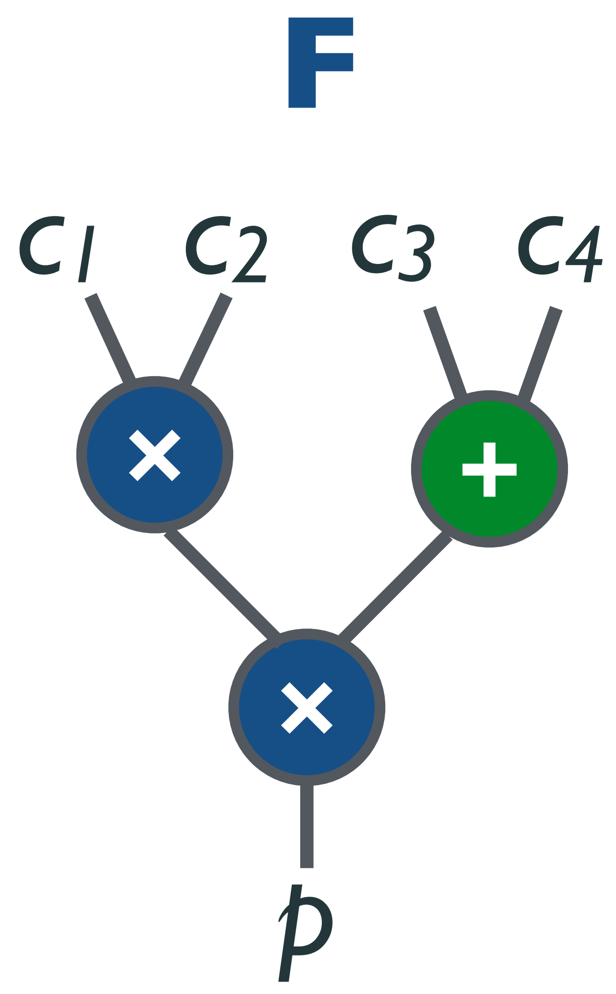
"**commit,** **compress**"
&
prove

Com({ct$_j$}$_j$, ct, H) →

$k \leftarrow_\$ \mathbb{Z}_q$ ←

Com({c_j}$_j$, c, h)

π_{ev} π_F →



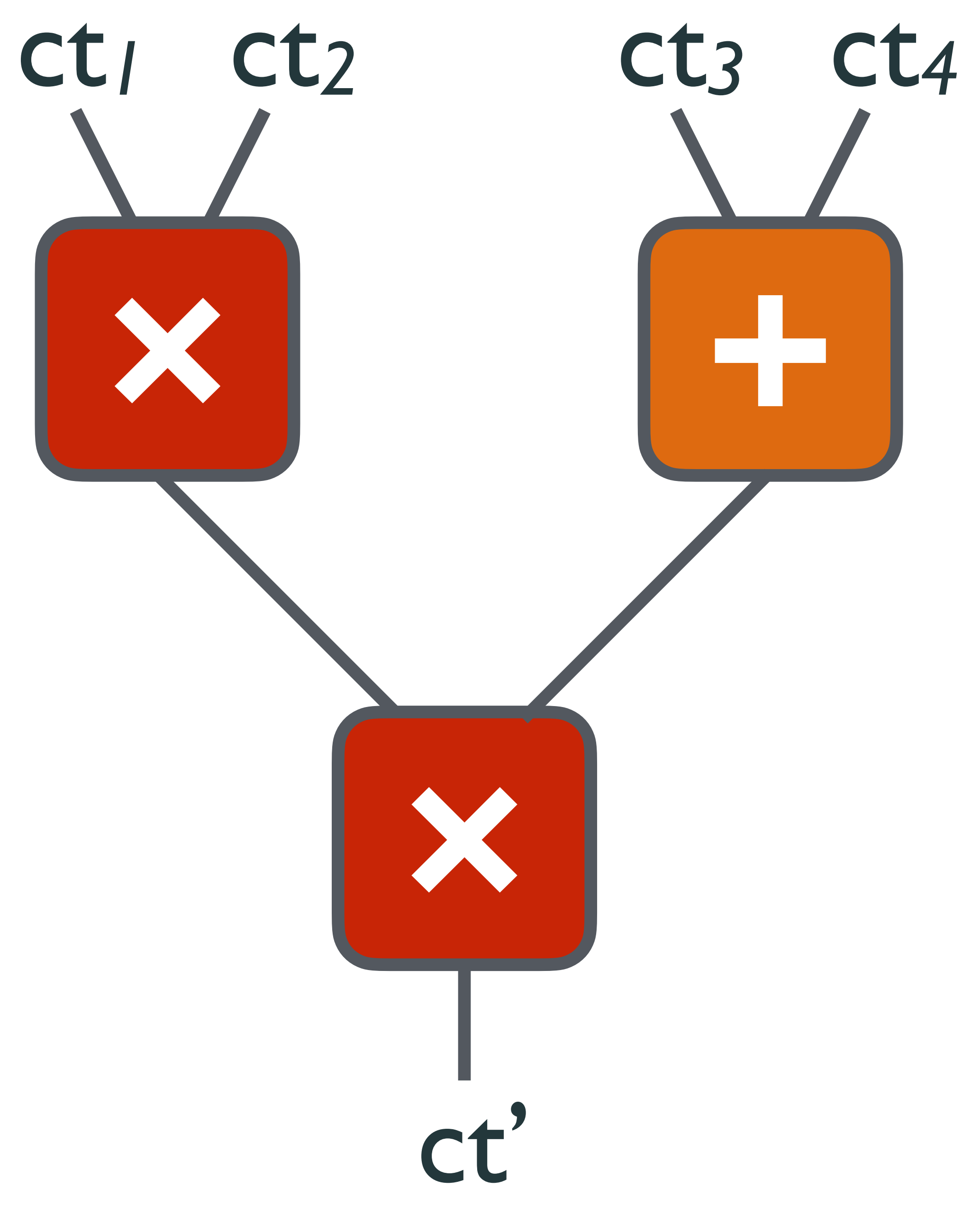
prove
$\forall j$: c_j =ct$_j$(k), c=ct(k), h=H(k)

prove
c=$\boldsymbol{F}$({c_j}) - h(k^d+1)

circuit complexity $O(\mathrm{n} \cdot \mathrm{d})$ $O(|\boldsymbol{F}|)$

Modular realization of Rq-Π

$F' : \mathbb{Z}_q[X]^{2n} \rightarrow \mathbb{Z}_q[X]^{D+1}$



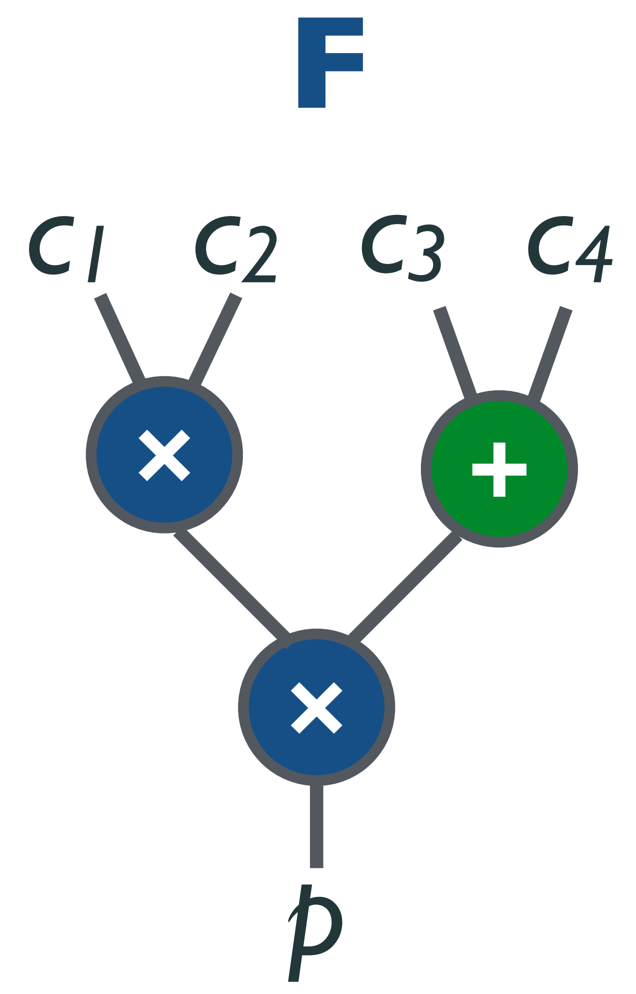
"commit, compress"
&
prove

Com($\{ct_j\}_j$, ct, H) →

$k \leftarrow_\$ Z_q$ ←

Com($\{c_j\}_j$, c, h)

π_{ev} π_F →



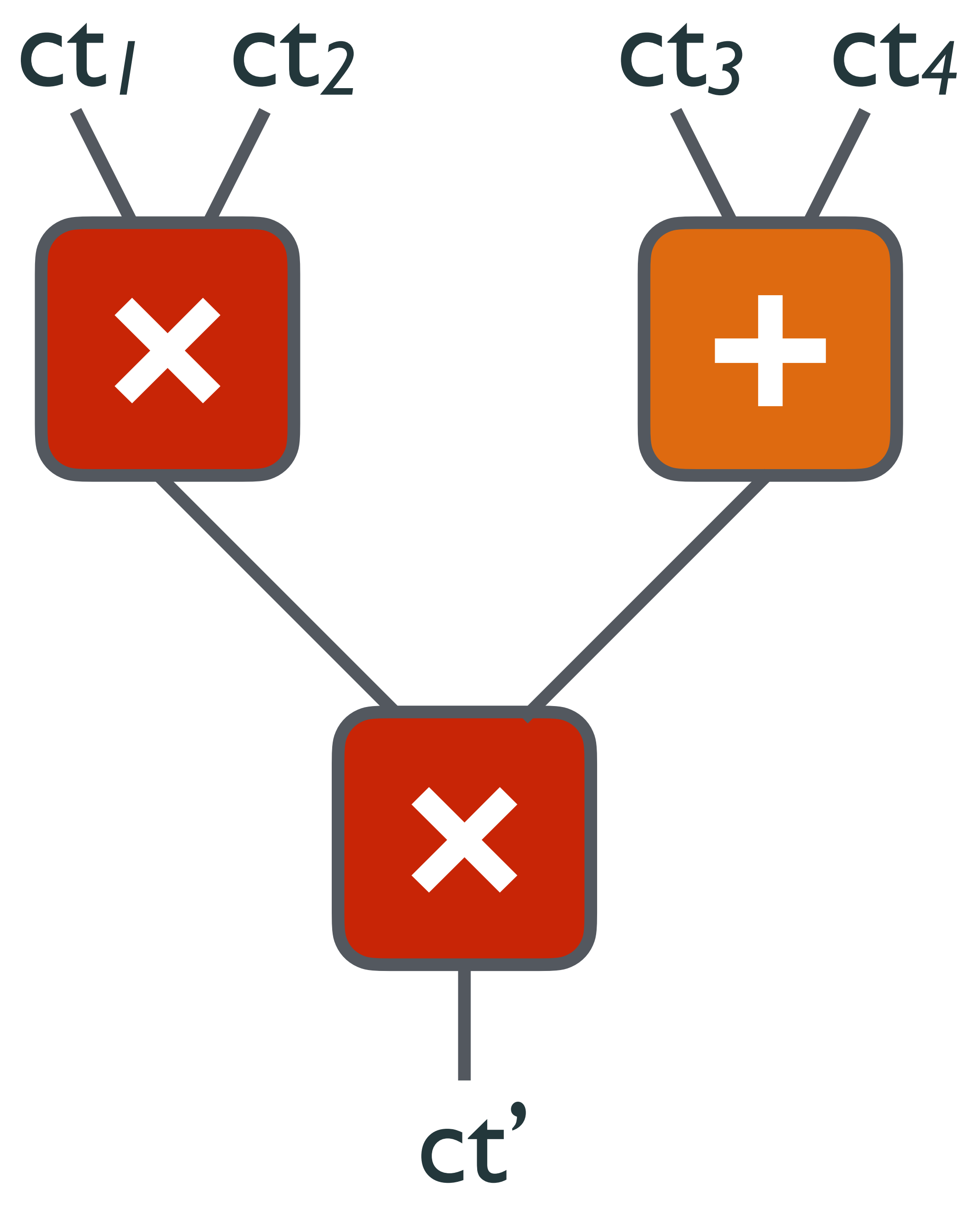
prove	prove (AC-Π)
$\forall j$: $c_j = ct_j(k)$, $c = ct(k)$, $h = H(k)$	$c = \boldsymbol{F}(\{c_j\}) - h(k^d + 1)$

can be instantiated with SNARK for Z_q

circuit complexity	$O(n \cdot d)$	$O(\lvert \boldsymbol{F} \rvert)$

Modular realization of Rq-Π

$F' : \mathbb{Z}_q[X]^{2n} \rightarrow \mathbb{Z}_q[X]^{D+1}$



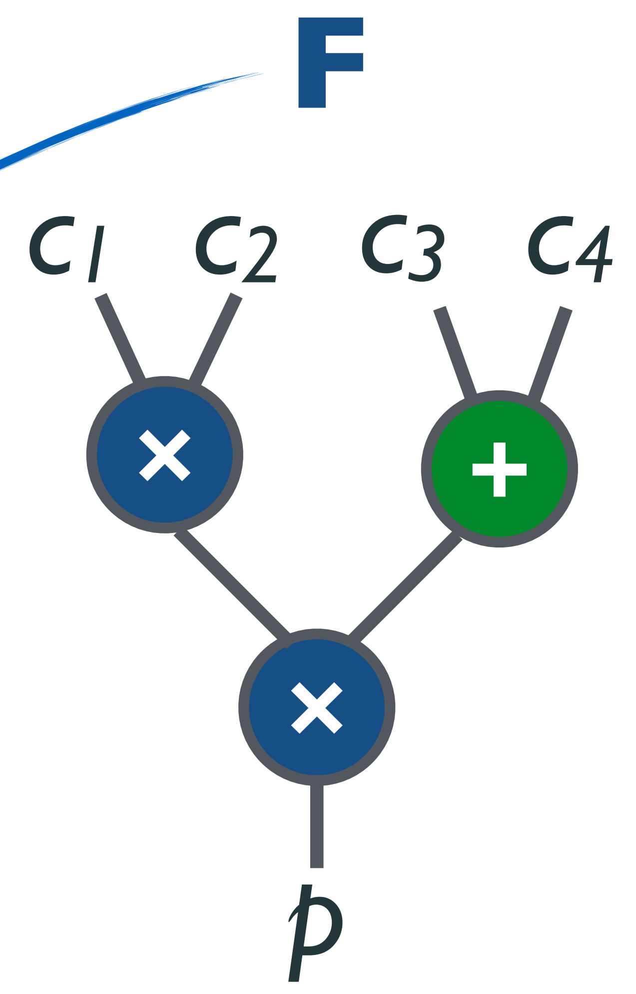
"**commit,** **compress**"
&
prove

Com($\{ct_j\}_j$, ct, H)

$k \leftarrow_\$ Z_q$

Com($\{c_j\}_j$, c, h)

π_{ev} π_F



main technical realization

MUniEv-Π
prove
$\forall j$: $c_j = ct_j(k)$, $c = ct(k)$, $h = H(k)$

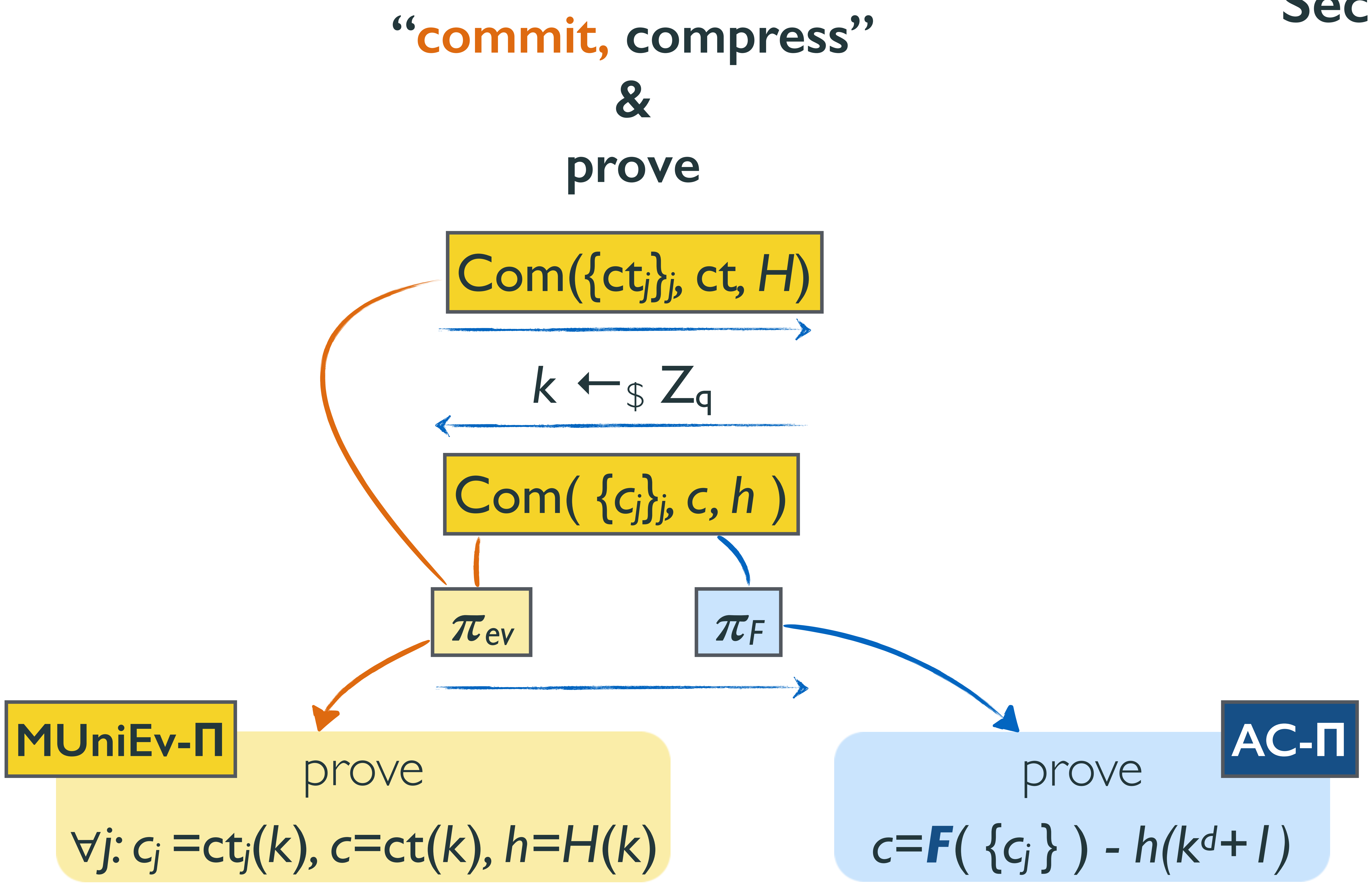
AC-Π
prove
$c = \mathbf{F}(\{c_j\}) - h(k^d + 1)$

can be instantiated with SNARK for Z_q

circuit complexity $O(n \cdot d)$ $O(|\mathbf{F}|)$

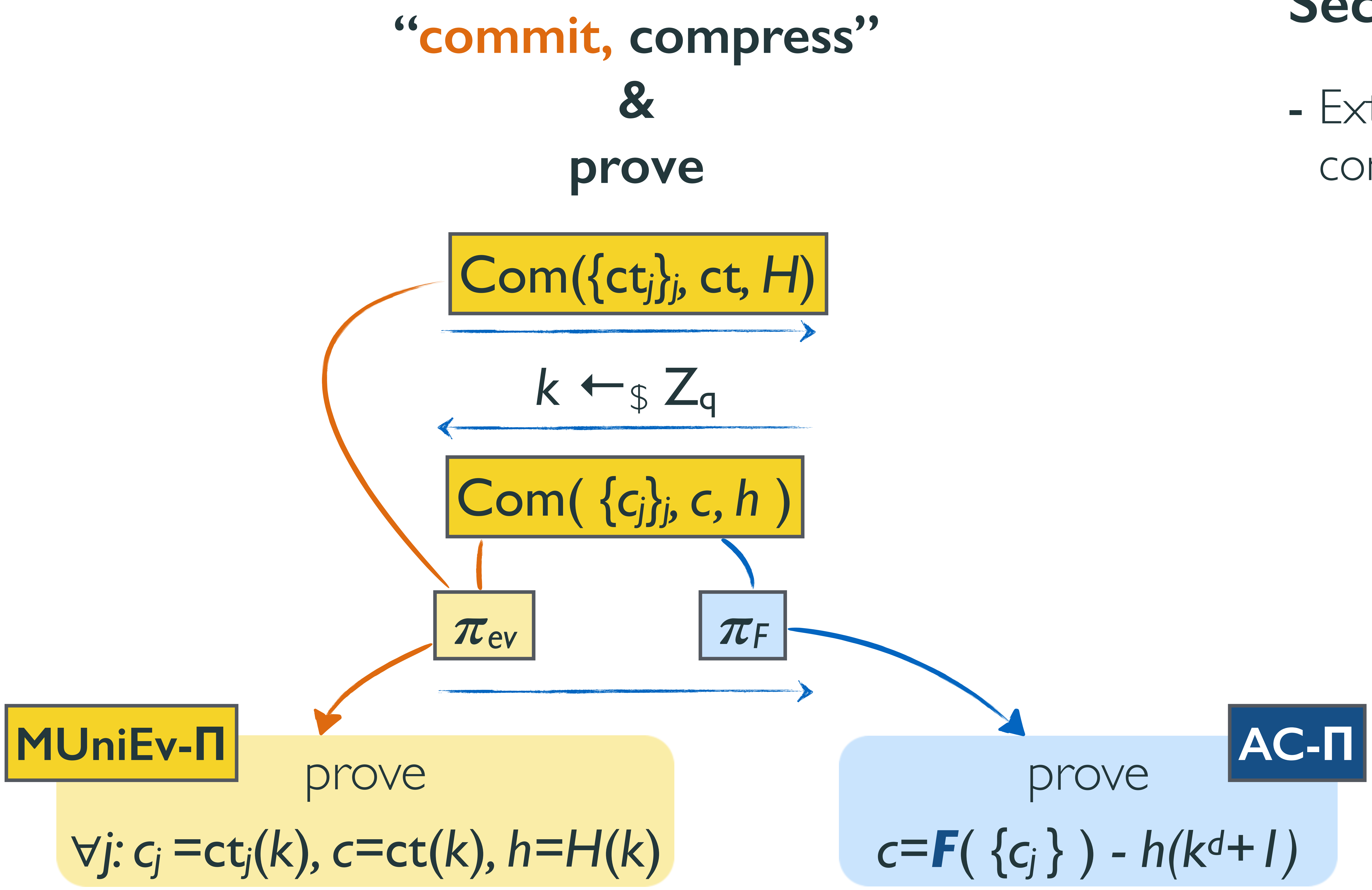
Modular realization of Rq-Π

Security of Rq-Π



Security intuition.

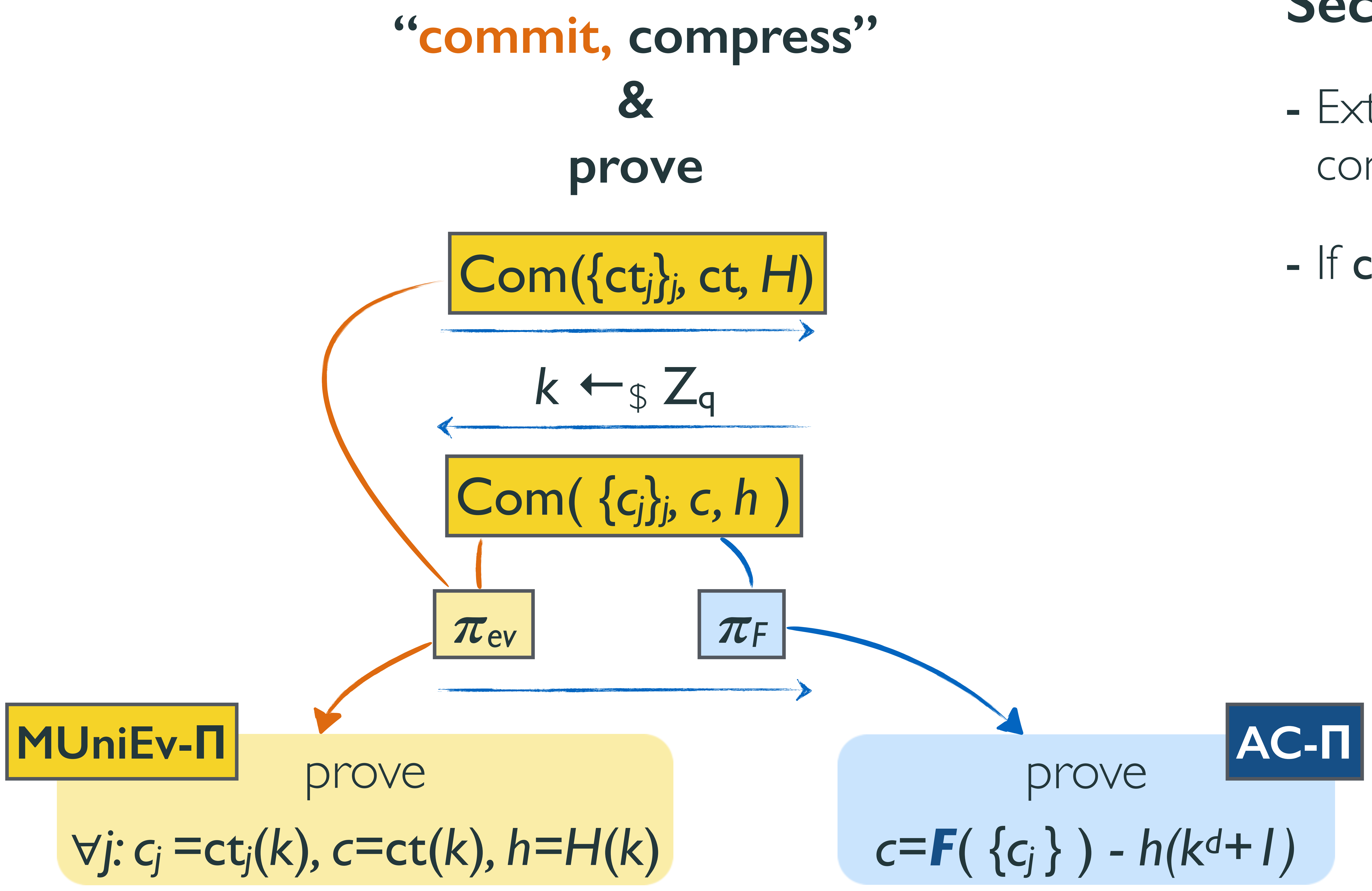
Security of Rq-Π



Security intuition.

- Extract **{ct$_j$}$_j$, ct, H, {c$_j$}$_j$, c, h** from the commitments

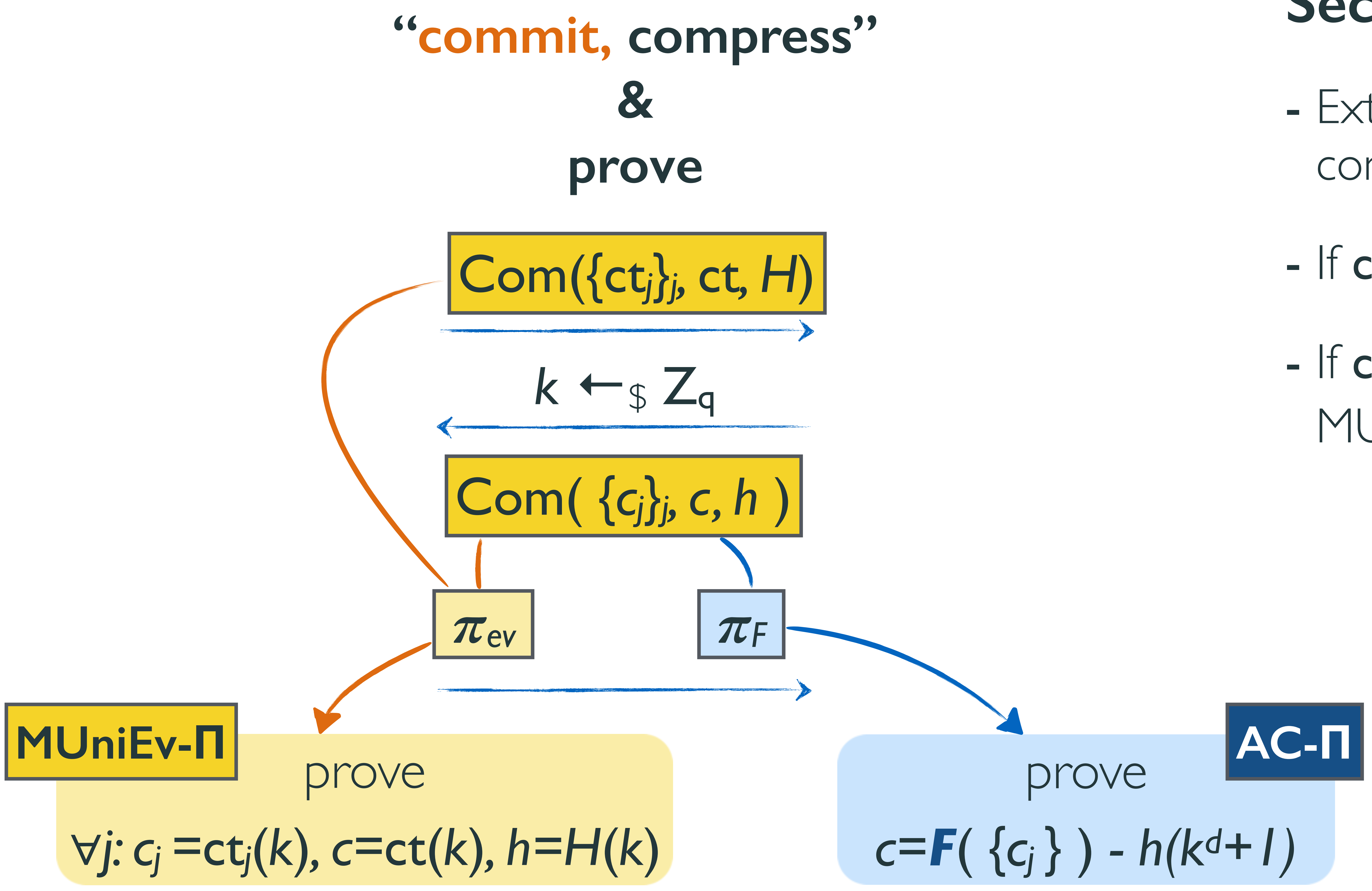
Security of Rq-Π



Security intuition.

- Extract **{ct$_j$}$_j$, ct, H, {c$_j$}$_j$, c, h** from the commitments
- If **c≠F({c$_j$}) - h(k^d+1)** break AC-Π

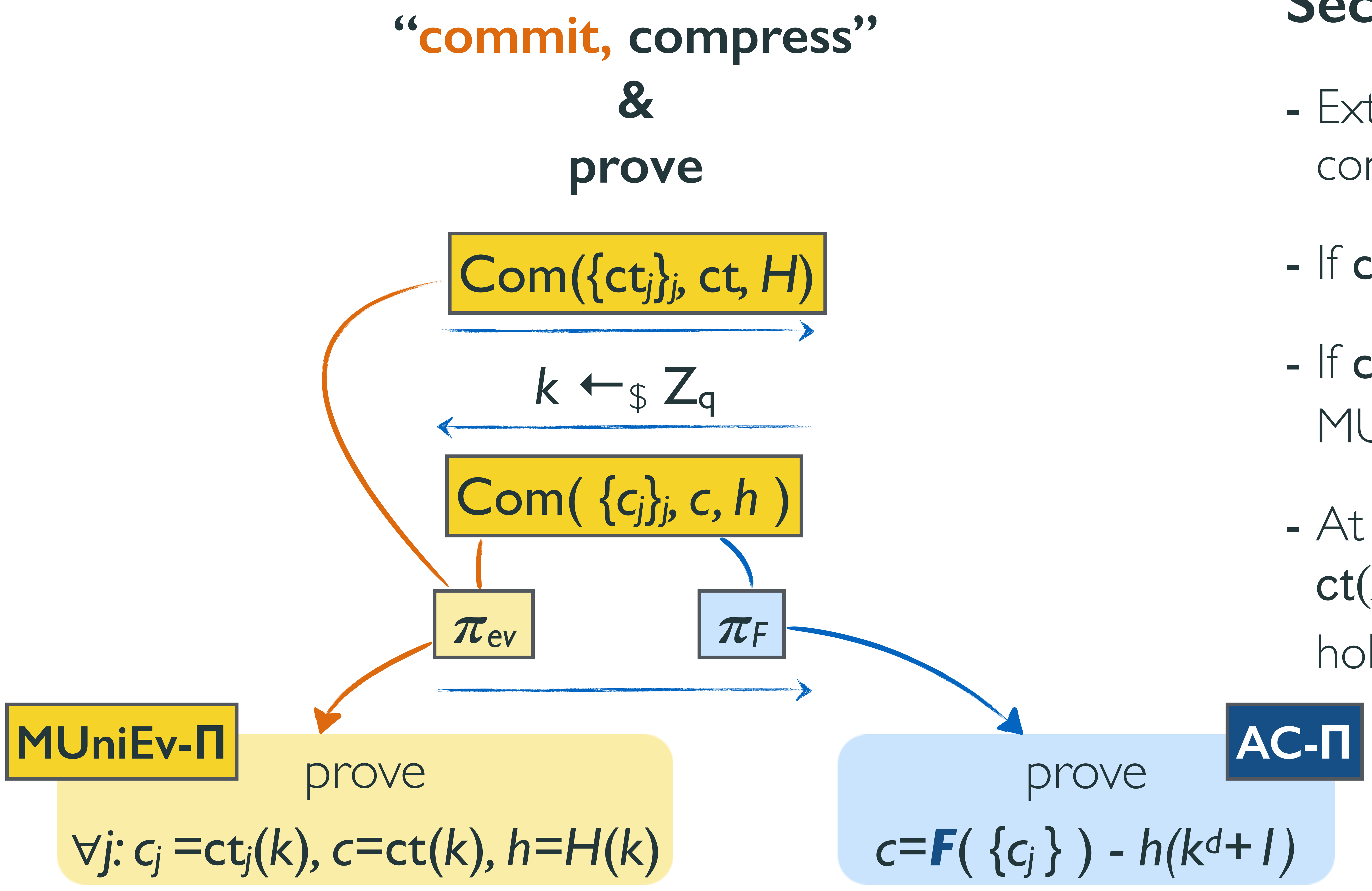
Security of Rq-Π



Security intuition.

- Extract **$\{ct_j\}_j$, ct, H, $\{c_j\}_j$, c, h** from the commitments
- If **$c \neq \mathbf{F}(\{c_j\}) - h(k^d+1)$** break AC-Π
- If **$c \neq ct(k)$, $h \neq H(k)$** or **$\exists j$: $c_j \neq ct_j(k)$**, break MUniEv-Π

Security of Rq-Π



Security intuition.

- Extract **$\{ct_j\}_j$, ct, H, $\{c_j\}_j$, c, h** from the commitments
- If **c≠F($\{c_j\}$) - h(k^d+1)** break AC-Π
- If **c≠ct(k), h≠H(k)** or **∃j: c_j ≠ct_j(k)**, break MUniEv-Π
- At this point, over the randomness of **k**, $\mathsf{ct}(X) \neq F'(\{\mathsf{ct}_j(X)\}_j) - H(X)(X^d + 1)$ holds with prob. ≈dD/q

MUniEv-Π: CP-SNARK for multiple polynomial evaluations

$R_{uni}(\ (C, C', k) \ , (\{ct_j\}_j, \{c_j\}_j \ , \varrho, \varrho') :$ $C = \mathrm{Com}(\{ct_j\}_j \ , \varrho)$ $C' = \mathrm{Com}(\{c_j\}_j, \ \varrho')$ $\forall j: c_j = ct_j(k)$ $)$

MUniEv-Π: CP-SNARK for multiple polynomial evaluations

$R_{uni}((C, C', k), (\{ct_j\}_j, \{c_j\}_j, \varrho, \varrho') :$ $C = Com(\{ct_j\}_j, \varrho)$, $C' = Com(\{c_j\}_j, \varrho')$, $\forall j: c_j = ct_j(k)$ $)$

MUniEv-Π ⇐ BivPE-Π reduce to partial evaluation of one bivariate polynomial

MUniEv-Π: CP-SNARK for multiple polynomial evaluations

$R_{uni}(\ (C, C', k\ ), (\ \{ct_j\}_j, \{c_j\}_j\ , \varrho, \varrho'\ ) :$ $C = Com(\ \{ct_j\}_j\ , \varrho)$ $C' = Com(\ \{c_j\}_j,\ \varrho')$ $\forall j: c_j = ct_j(k)$)

MUniEv-Π ⇐ BivPE-Π reduce to partial evaluation of one bivariate polynomial

Bivariate polynomial encoding. $(ct_0(X), \ldots, ct_n(X)) \Rightarrow ct(X,Y) = ct_0(X) + ct_1(X)Y + \ldots + ct_n(X)\, Y^n$

Bivariate polynomial com. $Com(ct_0(X), \ldots, ct_n(X)) = Com(ct(X,Y))$, $Com(c_0, \ldots, c_n) = Com(c(Y))$

MUniEv-Π: CP-SNARK for multiple polynomial evaluations

R_{uni}((C, C', k) , ($\{ct_j\}_j$, $\{c_j\}_j$, ϱ, ϱ') : C= Com($\{ct_j\}_j$, ϱ) C'=Com($\{c_j\}_j$, ϱ') $\forall j: c_j = ct_j(k)$)

MUniEv-Π ⇐ BivPE-Π reduce to partial evaluation of one bivariate polynomial

Bivariate polynomial encoding. $(ct_0(X), \ldots, ct_n(X)) \Rightarrow ct(X,Y) = ct_0(X) + ct_1(X)Y + \ldots + ct_n(X)Y^n$

Bivariate polynomial com. $\mathrm{Com}(ct_0(X), \ldots, ct_n(X)) = \mathrm{Com}(ct(X,Y))$, $\mathrm{Com}(c_0, \ldots, c_n) = \mathrm{Com}(c(Y))$

$R_{uni} \Rightarrow R_{biv}(\ (C, C', k), (ct, c, \varrho, \varrho') : C = \mathrm{Com}(ct(X,Y), \varrho),\ C' = \mathrm{Com}(c(Y), \varrho'),\ c(Y) = ct(k,Y)\ )$

MUniEv-Π: CP-SNARK for multiple polynomial evaluations

$R_{uni}(\ (C, C', k\ ), (\ \{ct_j\}_j, \{c_j\}_j, \varrho, \varrho'\ ) :$ $C = Com(\ \{ct_j\}_j, \varrho)$ $C' = Com(\ \{c_j\}_j,\ \varrho')$ $\forall j: c_j = ct_j(k)$)

MUniEv-Π ⇐ BivPE-Π reduce to partial evaluation of one bivariate polynomial

Bivariate polynomial encoding. $(ct_0(X), \ldots, ct_n(X)) \Rightarrow ct(X,Y) = ct_0(X) + ct_1(X)Y + \ldots + ct_n(X)\,Y^n$

Bivariate polynomial com. $Com(ct_0(X), \ldots, ct_n(X)) = Com(ct(X,Y))$, $Com(c_0, \ldots, c_n) = Com(c(Y))$

$R_{uni} \Rightarrow R_{biv}(\ (\ C, C', k\ ), (\ ct, c, \varrho, \varrho'\ ) : C = Com(ct(X,Y), \varrho),\ C' = Com(c(Y), \varrho'),\ c(Y) = ct(k,Y)\ )$

Main construction. Com for bivariate polynomials + CP-SNARK BivPE-Π for partial evaluation

Biv commitment scheme

Basic idea. ck = $(\{[s^i\ t^j], [\alpha\ s^i\ t^j]\}_{i,j}, [h, \alpha h], [\alpha, s, sh])$,

$C=(c, c')=([P(s,t)+\varrho h], [\alpha\ (P(s,t) + \varrho h)])$

Biv commitment scheme

Basic idea. ck = ({[$s^i t^j$], [$\alpha s^i t^j$]}$_{i,j}$, [h, αh], [α, s, sh]),

C=(c, c')=([$P(s,t)+\varrho h$], [α ($P(s,t)$ + ϱh)])

$\underline{\mathsf{Biv.ComGen}(1^\lambda, d, \ell) \to \mathsf{ck}}$

1: $\mathsf{gk} \leftarrow \mathcal{G}(1^\lambda),\ \ g, h \leftarrow_\$ \mathbb{G}, \mathfrak{g} \leftarrow_\$ \mathfrak{G},\ \ \alpha, s, t \leftarrow_\$ \mathbb{Z}_q$
2: $\hat{g} := g^\alpha, \hat{h} := h^\alpha, \hat{\mathfrak{g}} := \mathfrak{g}^\alpha$
3: $g_{ij} := g^{s^i t^j},\ \ \hat{g}_{ij} := \hat{g}^{s^i t^j}\ \ \forall\ i < d, j < \ell$
4: $\mathfrak{g}_1 := \mathfrak{g}^s, h_1 := h^s$
5: return $\mathsf{ck} = \{\mathsf{gk}, (g_{ij})_{i,j=0}^{d,\ell}, (\hat{g}_{ij})_{i,j=0}^{d,\ell}; (h, \hat{h}); (\mathfrak{g}, \hat{\mathfrak{g}}); (\mathfrak{g}_1, h_1)\}$

$\underline{\mathsf{Biv.Com}(\mathsf{ck}, P) \to (C, \rho)}$

1: $P := \sum_{i,j=0}^{d,\ell} a_{ij} X^i Y^j$
2: $\rho \leftarrow_\$ \mathbb{Z}_q$
3: $c = h^\rho \prod_{i=0,j=0}^{d,\ell} g_{ij}^{a_{ij}}$
4: $\hat{c} = \hat{h}^\rho \prod_{i=0,j=0}^{d,\ell} \hat{g}_{ij}^{a_{ij}}$
5: $C \leftarrow (c, \hat{c})$
6: return (C, ρ)

$\underline{\mathsf{Biv.ComVer}(\mathsf{ck}, C) \to b}$

1: $C := (c, \hat{c})$
2: return $b := (\mathsf{e}(c, \hat{\mathfrak{g}}) = \mathsf{e}(\hat{c}, \mathfrak{g}))$

$\underline{\mathsf{Biv.OpenVer}(\mathsf{ck}, C, P, \rho) \to P}$

1: $C := (c, \hat{c}),\ \ P = \sum_{i,j=0}^{d,\ell} a_{ij} X^i Y^j$
2: $b_1 \leftarrow \mathsf{ComVer}(\mathsf{ck}, C)$
3: $b_2 \leftarrow (c = h^\rho \prod_{i,j=0}^{d,\ell} g_{ij}^{a_{ij}})$
4: return $(b_1\ \wedge\ b_2)$

BivPE-Π CP-SNARK

Goal. $p(Y) = P(k, Y)$

BivPE-Π CP-SNARK

Goal. $p(Y) = P(k,Y)$

KZG10 evaluation proof technique: for $b=P(a)$, give $[W(s)]$ where

$$W(X) = (P(X) - p(a))/(X - a)$$

and verifier tests $e(\ [W(s)], [s - a]\ ) = e(\ [P(s) - b], [1]\ )$

BivPE-Π CP-SNARK

Goal. $p(Y) = P(k, Y)$

KZG10 evaluation proof technique: for $b=P(a)$, give $[W(s)]$ where

$$W(X) = (P(X) - p(a))/(X - a)$$

and verifier tests $\mathbf{e}([W(s)], [s - a]) = \mathbf{e}([P(s) - b], [1])$

Partial evaluation of bivariate polynomial: prover → $[W(s,t)]$ where

$$W(X,Y) = (P(X,Y) - p(Y))/(X - k)$$

and verifier tests $\mathbf{e}([W(s,t)], [s - a]) = \mathbf{e}([P(s,t)] - [p(t)], [1])$

BivPE-Π CP-SNARK

Goal. $p(Y) = P(k,Y)$

KZG10 evaluation proof technique: for $b=P(a)$, give $[W(s)]$ where

$$W(X) = (P(X) - p(a))/(X - a)$$

and verifier tests $\mathbf{e}([W(s)], [s - a]) = \mathbf{e}([P(s) - b], [1])$

Partial evaluation of bivariate polynomial: prover → $[W(s,t)]$ where

$$W(X,Y) = (P(X,Y) - p(Y))/(X - k)$$

and verifier tests $\mathbf{e}([W(s,t)], [s - a]) = \mathbf{e}([P(s,t)] - [p(t)], [1])$

…doesn't work if commitments are hiding…

BivPE-Π CP-SNARK

Instance. $C=([P(s,t)+\varrho h], [\alpha\ (P(s,t) + \varrho h)])$, $C'=([p(t)+\varrho' h], [\alpha\ (p(t) + \varrho' h)])$, k

Goal: prove $p(Y) = P(k,Y)$

BivPE-Π CP-SNARK

Instance. $C=([P(s,t)+\varrho h], [\alpha (P(s,t) + \varrho h)])$, $C'=([p(t)+\varrho' h], [\alpha (p(t) + \varrho' h)])$, k

Goal: prove $p(Y) = P(k,Y)$

Main idea: compute $W(X,Y) = (P(X,Y) - p(Y))/(X - k)$ and give $D=[W(s,t)]$

verifier tests $e(D, [s - a]) = e(C - C', [1])$

BivPE-Π CP-SNARK

Instance. $C=([P(s,t)+\varrho h], [\alpha\,(P(s,t) + \varrho h)])$, $C'=([p(t)+\varrho' h], [\alpha\,(p(t) + \varrho' h)])$, k

Goal: prove $p(Y) = P(k,Y)$

Main idea: compute $W(X,Y) = (P(X,Y) - p(Y))/(X - k)$ and give $D=[W(s,t)]$

verifier tests $e(D, [s - a]) = e(C - C', [1])$

Problem 1: verification does not work

$$
\begin{aligned}
e(C - C', [1]) = e([P(s,t)+\varrho h] - [p(t)+\varrho' h], [1]) &= e(\,[P(s,t) - p(t) + (\varrho - \varrho')h]\,, [1]) \\
&= e([W(s,t)], [s-k])\; e([(\varrho - \varrho')h], [1]) \\
&= e(D, [s-k])\; e([(\varrho - \varrho')h], [1])
\end{aligned}
$$

BivPE-Π CP-SNARK

Instance. C=([P(s,t)+ϱh], [α (P(s,t) + ϱh)]), C'=([p(t)+ϱ'h], [α (p(t) + ϱ'h)]), k

Goal: prove p(Y) = P(k,Y)

Main idea: compute W(X,Y) = (P(X,Y) - p(Y))/(X - k) and give D=[W(s,t)]

verifier tests e(D, [s - a]) = e(C - C', [1])

Problem 1: verification does not work

e(C - C', [1]) = e([P(s,t)+ϱh] - [p(t)+ϱ'h], [1]) = e([P(s,t) - p(t) + (ϱ - ϱ')h] , [1])
= e([W(s,t)], [s-k]) e([(ϱ - ϱ')h], [1])
= e(D, [s-k]) e([(ϱ - ϱ')h], [1])

Problem 2: D does not hide W — solved by defining D=[W(s,t)+ωh]

BivPE-Π CP-SNARK

Instance. $C=([P(s,t)+\varrho h], [\alpha\,(P(s,t) + \varrho h)])$, $C'=([p(t)+\varrho' h], [\alpha\,(p(t) + \varrho' h)])$, k

Goal: prove $p(Y) = P(k,Y)$

Main idea: compute $W(X,Y) = (P(X,Y) - p(Y))/(X - k)$ and give $D=[W(s,t)]$

verifier tests $e(D, [s - a]) = e(C - C', [1])$

Problem 1: verification does not work

$$e(C - C', [1]) = e([P(s,t)+\varrho h] - [p(t)+\varrho' h], [1]) = e(\,[P(s,t) - p(t) + (\varrho - \varrho')h]\,, [1])$$
$$= e([W(s,t)], [s-k])\; e([(\varrho - \varrho')h], [1])$$
$$= e(D, [s-k])\; e([(\varrho - \varrho')h], [1])$$

Problem 2: D does not hide W — solved by defining $D=[W(s,t)+\omega h]$

Solution to (1): prove knowledge of $(\varrho\,, \varrho', \omega)$ s.t.

$$e(C - C', [1])/e(D,[s-k]) = e([h(\varrho - \varrho') - h(s-k)\omega], [1])$$

$$e([P(s,t)+\varrho h] - [p(t)+\varrho' h], [1]) \,/\, e([W(s,t)+\omega h], [s-k])$$

BivPE-Π CP-SNARK

Prove knowledge of $(\varrho, \varrho', \omega)$ s.t.

$$e(C - C', [1])/e(D,[s-k]) = e([h(\varrho - \varrho') - h(s-k)\omega], [1])$$

BivPE-Π CP-SNARK

Prove knowledge of (ϱ , ϱ', ω) s.t.

$$e(C - C', [1])/e(D,[s-k]) = e([h(\varrho - \varrho') - h(s-k)\omega], [1])$$

Define $[g]=[h(k-s)] = [h]k - [hs]$

$$e([h(\varrho - \varrho') - h(s-k)\omega], [1])= e([h](\varrho - \varrho')[g]\omega, [1])$$

BivPE-Π CP-SNARK

Prove knowledge of (ϱ , ϱ', ω) s.t.

$$e(C - C', [1])/e(D,[s-k]) = e([h(\varrho - \varrho') - h(s-k)\omega], [1])$$

Define $[g]=[h(k-s)] = [h]k - [hs]$

$$e([h(\varrho - \varrho') - h(s-k)\omega], [1])= e([h](\varrho - \varrho')[g]\omega, [1])$$

Build a Schnorr proof of knowledge of exponents (ϱ - ϱ') and ω s.t.

$$e(C - C', [1])/e(D,[s-k]) = A = e([h](\varrho - \varrho')[g]\omega, [1])$$

BivPE-Π CP-SNARK

Resulting CP-SNARK: obtained in ROM applying Fiat-Shamir to Schnorr proof

$\mathsf{BivPE\text{-}\Pi.Prove}(\mathsf{crs}, u, w)$

$(C, C', k) := u, (P, Q, \rho, \rho') := w$
$W := \big(P - Q\big)/(X - k)$
$(D, \omega) \leftarrow \mathsf{Biv.Com}(W)$
$\tilde{g} := h_1/h^k, \quad x, y \leftarrow_{\$} \mathbb{Z}_q$
$\mathbb{U} := \mathsf{e}(h^x \tilde{g}^y, \mathfrak{g})$
$e \leftarrow \mathsf{Hash}(u, D, \mathbb{U})$
$\sigma = x - (\rho' - \rho)e \mod q$
$\tau = y - \omega e \mod q$
return $\pi := (D, e, \sigma, \tau)$

$\mathsf{BivPE\text{-}\Pi.Ver}(\mathsf{crs}, u, \pi) \rightarrow b$

$(C, C', k) := u, (D, e, \sigma, \tau) := \pi$
$(c, \hat{c}) := C, (c', \hat{c}') := C', (d, \hat{d}) := D$
$b_1 \leftarrow \mathsf{Biv.ComVer}(C)$
$b_2 \leftarrow \mathsf{Biv.ComVer}(C')$
$b_3 \leftarrow \mathsf{Biv.ComVer}(D)$
$\mathbb{A} = \mathsf{e}(d, \mathfrak{g}_1/\mathfrak{g}^k) \cdot \mathsf{e}(c/c', \mathfrak{g})^{-1}$
$\mathbb{U} := \mathsf{e}(h^\sigma \tilde{g}^\tau, \mathfrak{g})\mathbb{A}^e$, s.t. $\tilde{g} := h_1/h^k$
$b_4 \leftarrow (e = \mathsf{Hash}(u, D, \mathbb{U}))$
return $(b_1 \wedge b_2 \wedge b_3 \wedge b_4)$

Tackling ciphertext/circuit expansion & modulus [BCFK21]

BV11 HE

$R_p = \mathbb{Z}_p[X]/(X^d + 1)$

Encryption

$R_p \ni m \longmapsto \mathsf{ct} = (\mathsf{ct}[0] + \mathsf{ct}[1]Y) \in R_q[Y]$

Addition

$\mathrm{Eval}(+, \mathsf{ct}_1, \mathsf{ct}_2) \to \mathsf{ct}_1 + \mathsf{ct}_2 \in R_q[Y]$

Basic multiplication

$\mathrm{Eval}(\times, \mathsf{ct}_1, \mathsf{ct}_2) \to \mathsf{ct}_1 \cdot \mathsf{ct}_2 \in R_q[Y]$

Relinearization + mod switch /noise reduction

$$\mathsf{ct} \longmapsto \mathsf{ct}' = \sum_{i=0}^{deg_Y(\mathsf{ct})} \mathsf{ct}[i] \cdot \mathsf{rk}[i] \bmod q \mapsto \lceil \frac{q'}{q} \mathsf{ct} \rfloor$$

q can be product of prime powers

BCFK21

Compress and prove over Galois rings

- Homomorphic hash $\mathbb{Z}_q[X] \to \mathbb{Z}_q[X]/h(X)$ for random irreducible $h(X)$ of degree <d
- GKR over $\mathbb{Z}_q[X]/h(X)$

Challenges

1) Ciphertext expansion

unless optimized packing, $deg_X(m) \ll d$

2) Ciphertext modulus

q usually not prime

~~3) Non-algebraic operations~~

~~noise control techniques require divisions and rounding~~

[BCFK21] A. Bois, I. Cascudo, D. Fiore. D. Kim. *Flexible and Efficient Verifiable Computation on Encrypted Data*. PKC 2021

State of the art on VC for FHE

	Delegation	Verif	Privacy	Mul depth	Ctxt modulus	Implemented | Practical?
FGP14	priv.	priv.	✔	1	prime $>2\lambda$	✔ | 5s for 1000-var deg-2 poly
FNP20	pub	pub	✔	O(1)	prime $>2\lambda$	—
BCFK21	pub	pub	✔	O(1)	any	—

Privacy no verif. means verification's outcome must be kept private
Practicality not apple-to-apple at all, just to give an idea of time

Active area, not yet a "full" solution

State of the art on VC for FHE

	Delegation	Verif	Privacy	Mul depth	Ctxt modulus	Implemented | Practical?
FGP14	priv.	priv.	✔	1	prime $>2\lambda$	✔ | 5s for 1000-var deg-2 poly
FNP20	pub	pub	✔	O(1)	prime $>2\lambda$	—
BCFK21	pub	pub	✔	O(1)	any	—
Rinocchio	pub	priv.	✔*	poly	any	✔ | 0.3s for 1 mult

Privacy no verif. means verification's outcome must be kept private
Practicality not apple-to-apple at all, just to give an idea of time

Active area, not yet a "full" solution

State of the art on VC for FHE

	Delegation	Verif	Privacy	Mul depth	Ctxt modulus	Implemented | Practical?
FGP14	priv.	priv.	✔	1	prime $>2\lambda$	✔ | 5s for 1000-var deg-2 poly
FNP20	pub	pub	✔	O(1)	prime $>2\lambda$	—
BCFK21	pub	pub	✔	O(1)	any	—
Rinocchio	pub	priv.	✔*	poly	any	✔ | 0.3s for 1 mult
HEliopolis	pub	priv.	no verif	any	any	✔ | 5s for HE-FRI on RS of size 4096
GGW24	pub	priv.	no verif	any	any	—

Privacy no verif. means verification's outcome must be kept private
Practicality not apple-to-apple at all, just to give an idea of time

Active area, not yet a "full" solution

State of the art on VC for FHE

	Delegation	Verif	Privacy	Mul depth	Ctxt modulus	Implemented | Practical?
FGP14	priv.	priv.	✔	1	prime $>2\lambda$	✔ | 5s for 1000-var deg-2 poly
FNP20	pub	pub	✔	O(1)	prime $>2\lambda$	—
BCFK21	pub	pub	✔	O(1)	any	—
Rinocchio	pub	priv.	✔*	poly	any	✔ | 0.3s for 1 mult
HEliopolis	pub	priv.	no verif	any	any	✔ | 5s for HE-FRI on RS of size 4096
GGW24	pub	priv.	no verif	any	any	—
TW24	pub	pub	✔	any	any	✔ | 20m for 1 bootstrapping

Privacy no verif. means verification's outcome must be kept private
Practicality not apple-to-apple at all, just to give an idea of time

Active area, not yet a "full" solution

State of the art on VC for FHE

	Delegation	Verif	Privacy	Mul depth	Ctxt modulus	Implemented | Practical?
FGP14	priv.	priv.	✔	1	prime $>2\lambda$	✔ | 5s for 1000-var deg-2 poly
FNP20	pub	pub	✔	O(1)	prime $>2\lambda$	—
BCFK21	pub	pub	✔	O(1)	any	—
Rinocchio	pub	priv.	✔*	poly	any	✔ | 0.3s for 1 mult
HEliopolis	pub	priv.	no verif	any	any	✔ | 5s for HE-FRI on RS of size 4096
GGW24	pub	priv.	no verif	any	any	—
TW24	pub	pub	✔	any	any	✔ | 20m for 1 bootstrapping

Privacy no verif. means verification's outcome must be kept private
Practicality not apple-to-apple at all, just to give an idea of time

Active area, not yet a "full" solution

Thanks! ***Questions?***

References

[FGP14] D. Fiore, R. Gennaro, V. Pastro. *Efficiently Verifiable Computation on Encrypted Data*. CCS 2014

[FNP20] D. Fiore. A. Nitulescu, D. Pointcheval. *Boosting Verifiable Computation on Encrypted Data*. PKC 2020

[BCFK21] A. Bois, I. Cascudo, D. Fiore. D. Kim. *Flexible and Efficient Verifiable Computation on Encrypted Data*. PKC 2021

[Rinocchio] C. Ganesh, A. Nitulescu, E. Soria-Vazquez. *Rinocchio: SNARKs for Ring Arithmetic*. Journal of Cryptology 2023

[Heliopolis] D. F. Aranha, A. Costache, A. Guimarães, E. Soria-Vazquez. *HELIOPOLIS: Verifiable Computation over Homomorphically Encrypted Data from Interactive Oracle Proofs is Practical*. ePrint 2023/1949

[GGW24] S. Garg, A. Goel, M. Wang. *How to prove statements obliviously?* CRYPTO 2024

[TW24] L. T. Thubault, M. Walter. *Towards Verifiable FHE in Practice: Proving Correct Execution of TFHE's Bootstrapping using plonky2*. ePrint 2024/451