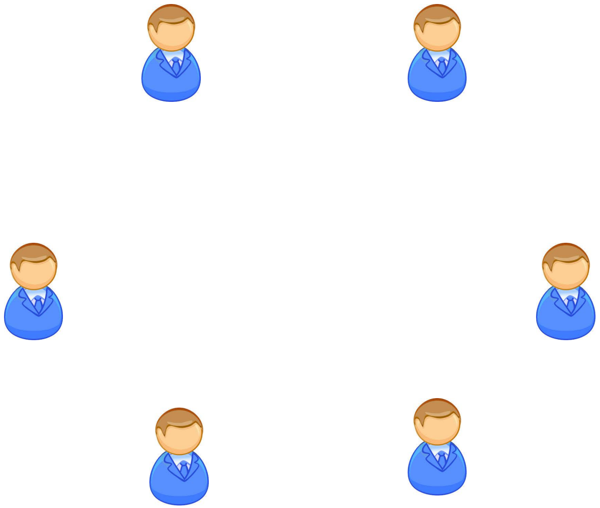
Better 2-round adaptive MPC

Ran Canetti, Oxana Poburinnaya

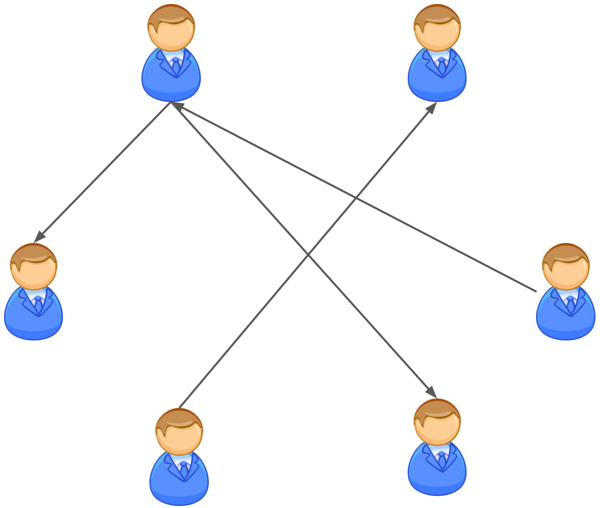
TAU and BU BU

Adaptive Security of MPC



Adaptive corruptions:
adversary can decide who to corrupt adaptively during the execution

Adaptive Security of MPC

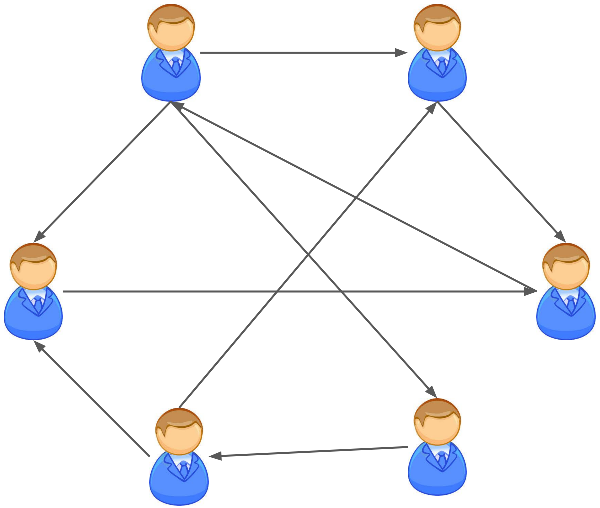


Adaptive corruptions:
adversary can decide who to corrupt adaptively during the execution

Simulator:

1. simulate communication (without knowing $x_1, \ldots, x_n$)

Adaptive Security of MPC

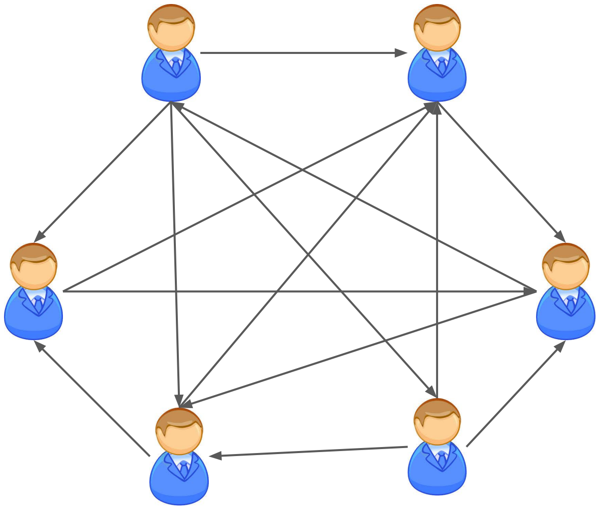


Adaptive corruptions:
adversary can decide who to corrupt adaptively during the execution

Simulator:

1. simulate communication (without knowing $x_1, \ldots, x_n$)

Adaptive Security of MPC

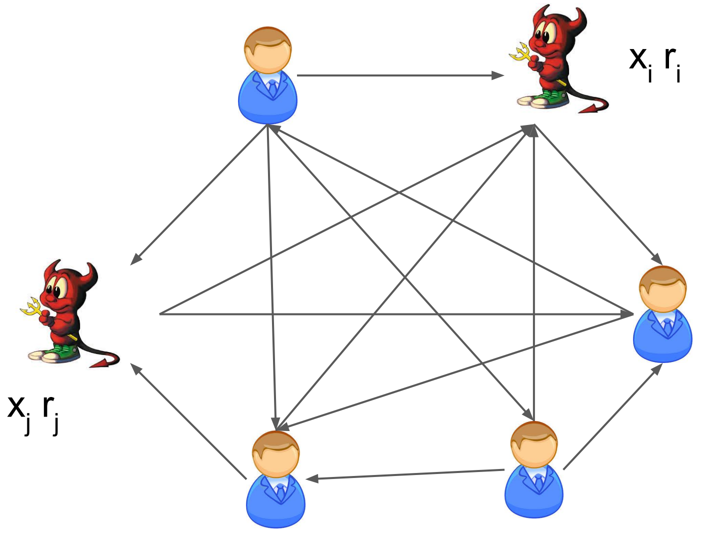


Adaptive corruptions:
adversary can decide who to corrupt adaptively during the execution

Simulator:

1. simulate communication (without knowing $x_1, \ldots, x_n$)

Adaptive Security of MPC

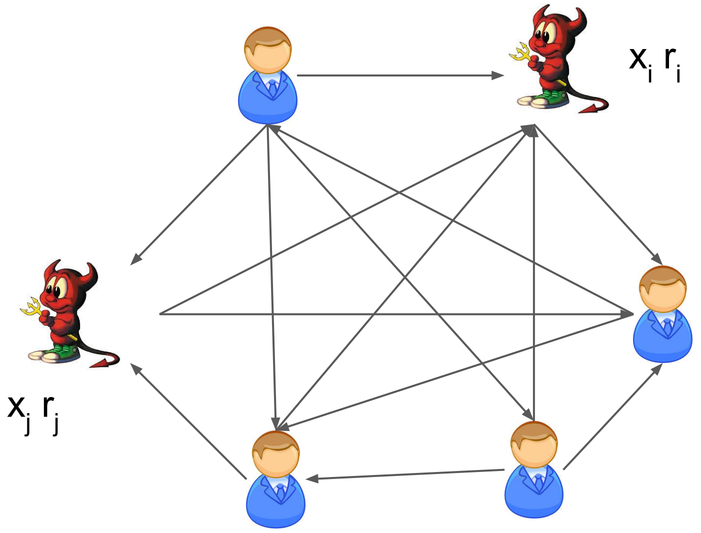


Adaptive corruptions:
adversary can decide who to corrupt adaptively during the execution

Simulator:

1. simulate communication (without knowing $x_1, \ldots, x_n$)
2. simulate r_i of corrupted parties, consistent with communication and x_i

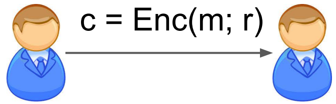
Adaptive Security of MPC



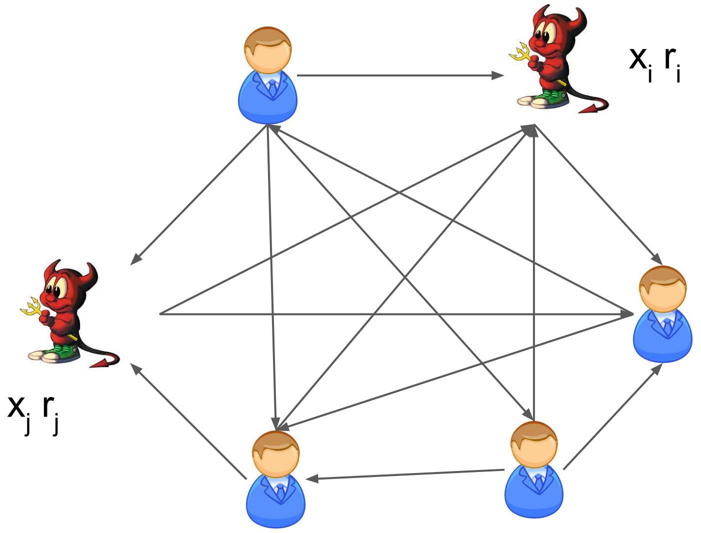
Adaptive corruptions:
adversary can decide who to corrupt adaptively during the execution

Simulator:

Example: encryption



Adaptive Security of MPC

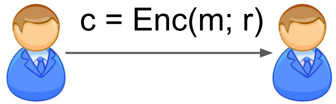


Adaptive corruptions:
adversary can decide who to corrupt adaptively during the execution

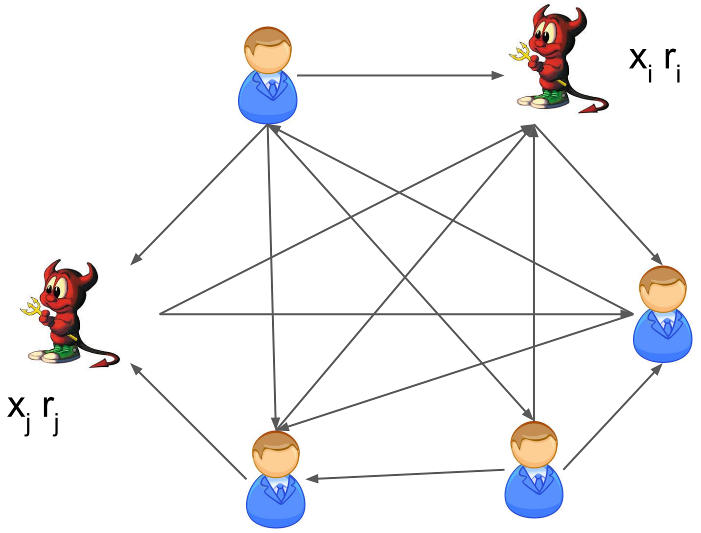
Simulator:

1. simulate fake ciphertext c (without knowing m)

Example: encryption



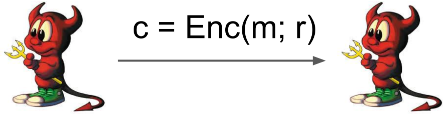
Adaptive Security of MPC



Adaptive corruptions:
adversary can decide who to corrupt adaptively during the execution

Simulator:
1. simulate fake ciphertext c (without knowing m)
2. upon corruption, learn m and provide consistent r, sk

Example: encryption



Full Adaptive Security

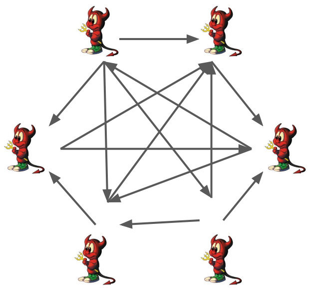
Full adaptive security:

- No erasures

Full Adaptive Security

Full adaptive security:

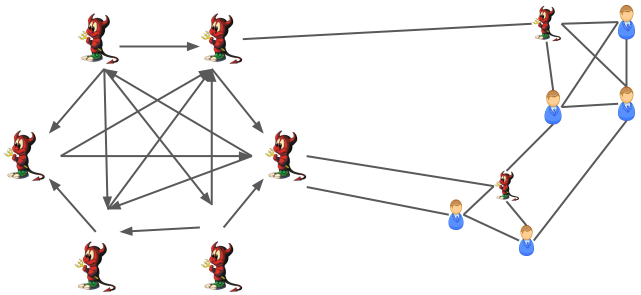
- No erasures
- Security even when all parties are corrupted



Full Adaptive Security

Full adaptive security:

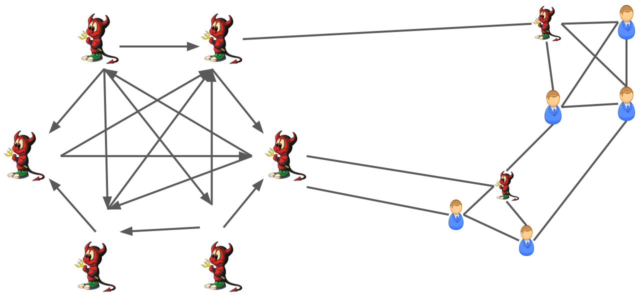
- No erasures
- Security even when all parties are corrupted



Full Adaptive Security

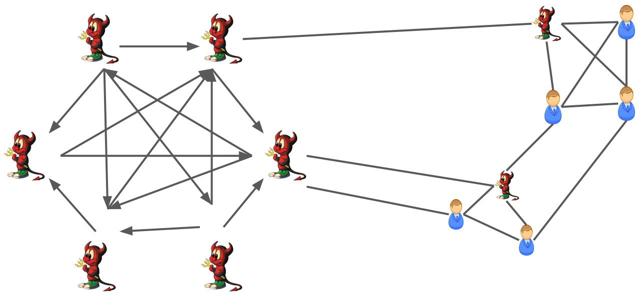
Full adaptive security:

- No erasures
- Security even when all parties are corrupted



Fully adaptively secure, constant rounds protocols appeared only recently: CGP15, DKR15, GP15.
Before: number of rounds ~ depth of the circuit (e.g. CLOS02)

Full Adaptive Security



Full adaptive security:

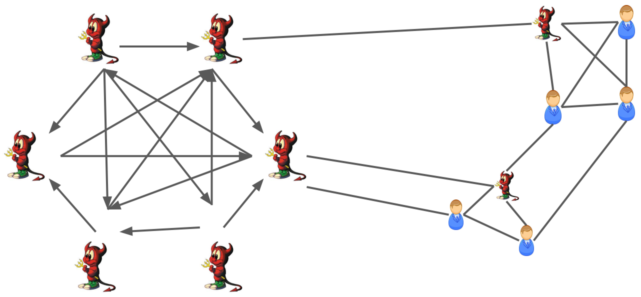
- No erasures
- Security even when all parties are corrupted

Fully adaptively secure, constant rounds protocols appeared only recently: CGP15, DKR15, GP15.
Before: number of rounds ~ depth of the circuit (e.g. CLOS02)

Full adaptive security for randomized functionalities:

- Randomness of the computation remains hidden even when all parties are corrupted

Full Adaptive Security



Full adaptive security:

- No erasures
- Security even when all parties are corrupted

Fully adaptively secure, constant rounds protocols appeared only recently: CGP15, DKR15, GP15.
Before: number of rounds ~ depth of the circuit (e.g. CLOS02)

Full adaptive security for randomized functionalities:

- Randomness of the computation remains hidden even when all parties are corrupted

Example: F internally chooses random primes p, q, and outputs N = pq.
Most protocols (e.g. CLOS02) reveal p, q, when all parties are corrupted.

Full Adaptive Security

	# of parties	# of rounds	assumptions
Canetti, Goldwasser, Poburinnaya’15	2	2	OWF subexp iO
Dachman-Soled, Katz, Rao’15	n	4	OWF iO
Garg, Polychroniadou’15	n	2	TDP subexp. iO

Only 3 fully adaptively secure protocols with constant rounds - but with a CRS*
Only one of them is 2 round MPC.

*need a CRS even for HBC case!

Full Adaptive Security

	# of parties	# of rounds	assumptions
Canetti, Goldwasser, Poburinnaya’15	2	2	OWF subexp iO
Dachman-Soled, Katz, Rao’15	n	4	OWF iO
Garg, Polychroniadou’15	n	2	TDP subexp. iO

Q1: can we build 2 round MPC with **global (non-programmable)** CRS?

Full Adaptive Security

	# of parties	# of rounds	assumptions	global CRS
Canetti, Goldwasser, Poburinnaya'15	2	2	OWF subexp iO	+
Dachman-Soled, Katz, Rao'15	n	4	OWF iO	+
Garg, Polychroniadou'15	n	2	TDP subexp. iO	- (even in HBC case)

Q1: can we build 2 round MPC with **global (non-programmable)** CRS?

Full Adaptive Security

	# of parties	# of rounds	assumptions	global CRS
Canetti, Goldwasser, Poburinnaya’15	2	2	OWF subexp iO	+
Dachman-Soled, Katz, Rao’15	n	4	OWF iO	+
Garg, Polychroniadou’15	n	2	TDP subexp. iO	- (even in HBC case)

Q1: can we build 2 round MPC with **global (non-programmable)** CRS?

Q2: can we compute **all randomized functionalities** (even not adaptively well formed, e.g. N = pq)?

Full Adaptive Security

	# of parties	# of rounds	assumptions	global CRS	randomized functionalities
Canetti, Goldwasser, Poburinnaya'15	2	2	OWF subexp iO	+	+
Dachman-Soled, Katz, Rao'15	n	4	OWF iO	+	+
Garg, Polychroniadou'15	n	2	TDP subexp. iO	- (even in HBC case)	-

Q1: can we build 2 round MPC with **global (non-programmable)** CRS?

Q2: can we compute **all randomized functionalities** (even not adaptively well formed, e.g. N = pq)?

Full Adaptive Security

	# of parties	# of rounds	assumptions	global CRS	randomized functionalities
Canetti, Goldwasser, Poburinnaya’15	2	2	OWF subexp iO	+	+
Dachman-Soled, Katz, Rao’15	n	4	OWF iO	+	+
Garg, Polychroniadou’15	n	2	TDP subexp. iO	- (even in HBC case)	-

Q1: can we build 2 round MPC with **global (non-programmable)** CRS?

Q2: can we compute **all randomized functionalities** (even not adaptively well formed, e.g. N = pq)?

Q3: can we build 2 round MPC from **weaker assumptions?** (e.g. remove the need for subexp. iO)

Full Adaptive Security

	# of parties	# of rounds	assumptions	global CRS	randomized functionalities
Canetti, Goldwasser, Poburinnaya’15	2	2	OWF subexp iO	+	+
Dachman-Soled, Katz, Rao’15	n	4	OWF iO	+	+
Garg, Polychroniadou’15	n	2	TDP subexp. iO	- (even in HBC case)	-
This work	**n**	**2**	**injective OWF iO**	**+**	**+ (comp. close)**

Q1: can we build 2 round MPC with **global (non-programmable)** CRS?

Q2: can we compute **all randomized functionalities** (even not adaptively well formed, e.g. N = pq)?

Q3: can we build 2 round MPC from **weaker assumptions?** (e.g. remove the need for subexp. iO)

Our results :

Part I:

Theorem (informal):
Assuming indistinguishability obfuscation for circuits and injective one way functions,
there exists **2-round, fully-adaptively-secure, RAM-efficient semi-honest** MPC protocol where:

- the CRS is global;
- even randomized functionalities can be computed.

Our results :

Part I:

Theorem (informal):
Assuming indistinguishability obfuscation for circuits and injective one way functions,
there exists **2-round, fully-adaptively-secure, RAM-efficient semi-honest** MPC protocol where:

- the CRS is global;
- even randomized functionalities can be computed.

The first two-round fully adaptive MPC without subexp. iO assumption;
The first two-round fully adaptive MPC with global CRS.

Our results :

Part I:

Theorem (informal):
Assuming indistinguishability obfuscation for circuits and injective one way functions,
there exists **2-round, fully-adaptively-secure, RAM-efficient semi-honest** MPC protocol where:

- the CRS is global;
- even randomized functionalities can be computed.

The first two-round fully adaptive MPC without subexp. iO assumption;
The first two-round fully adaptive MPC with global CRS.

Part II:

Theorem (informal):
Assuming iO for circuits and TDPs, there exists **RAM-efficient statistically sound NIZK**.

Our results :

Part I:

Theorem (informal):
Assuming indistinguishability obfuscation for circuits and injective one way functions,
there exists **2-round, fully-adaptively-secure, RAM-efficient semi-honest** MPC protocol where:

- the CRS is global;
- even randomized functionalities can be computed.

The first two-round fully adaptive MPC without subexp. iO assumption;
The first two-round fully adaptive MPC with global CRS.

Part II:

Theorem (informal):
Assuming iO for circuits and TDPs, there exists **RAM-efficient statistically sound NIZK**.

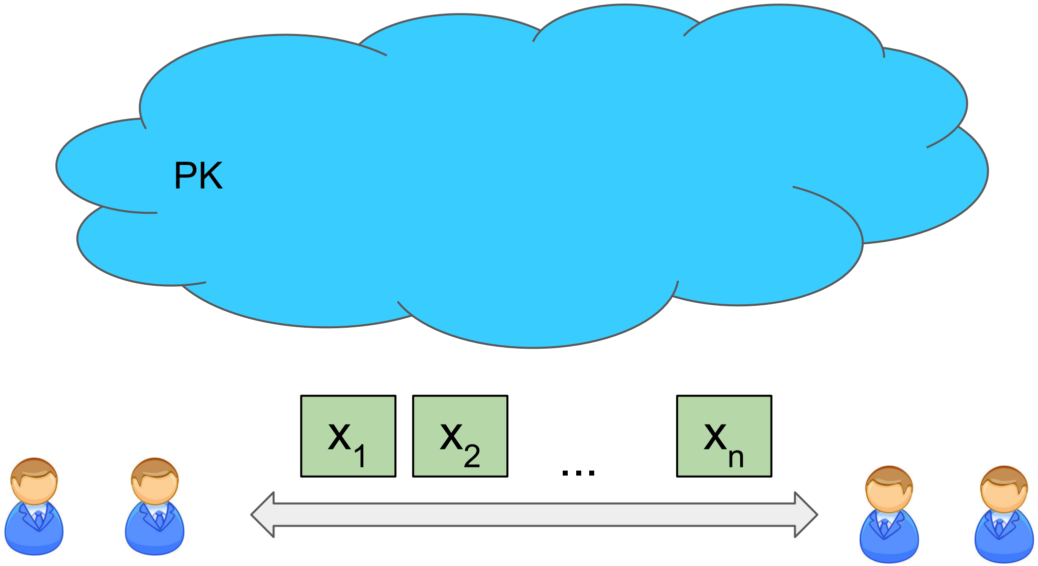
⬇

Theorem (GP15, our work):
Assuming subexp. iO for circuits and RAM-efficient statistically sound NIZK,
there exists **2-round, fully-adaptively-secure, RAM-efficient byzantine** MPC protocol.

Part I: HBC protocol with global CRS

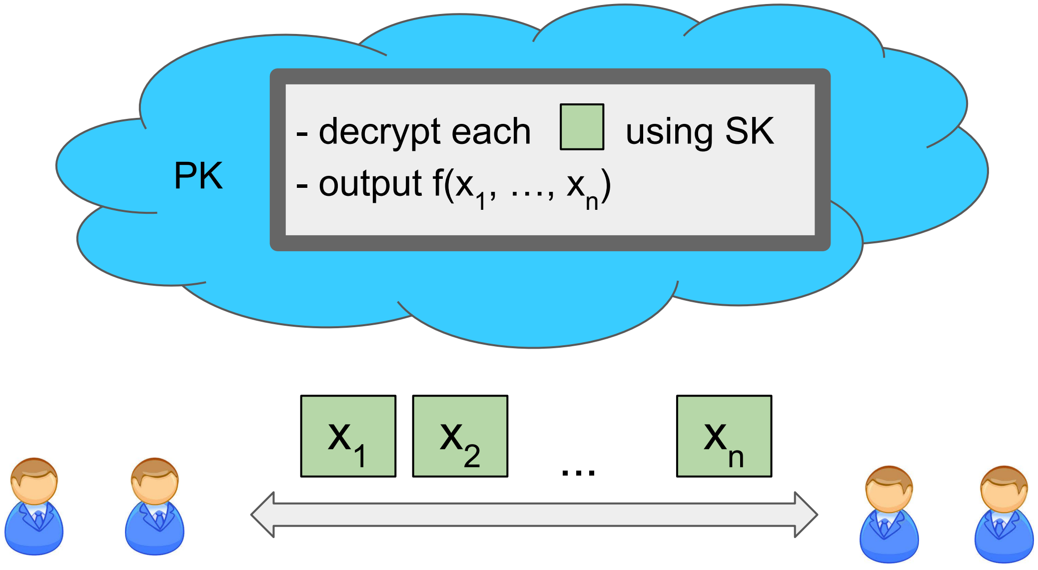
First attempt

$$\boxed{x_i} = \mathrm{Enc}_{PK}(x_i)$$



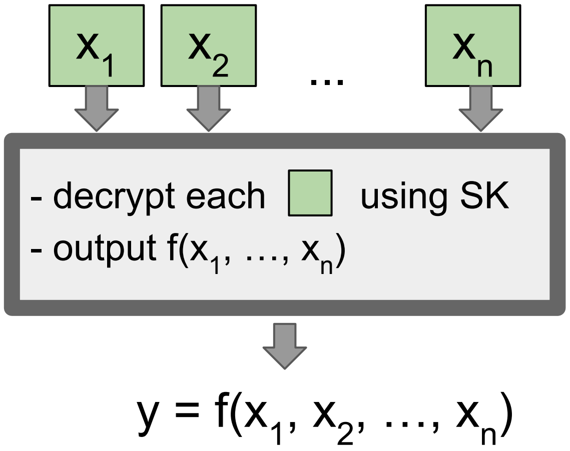
First attempt

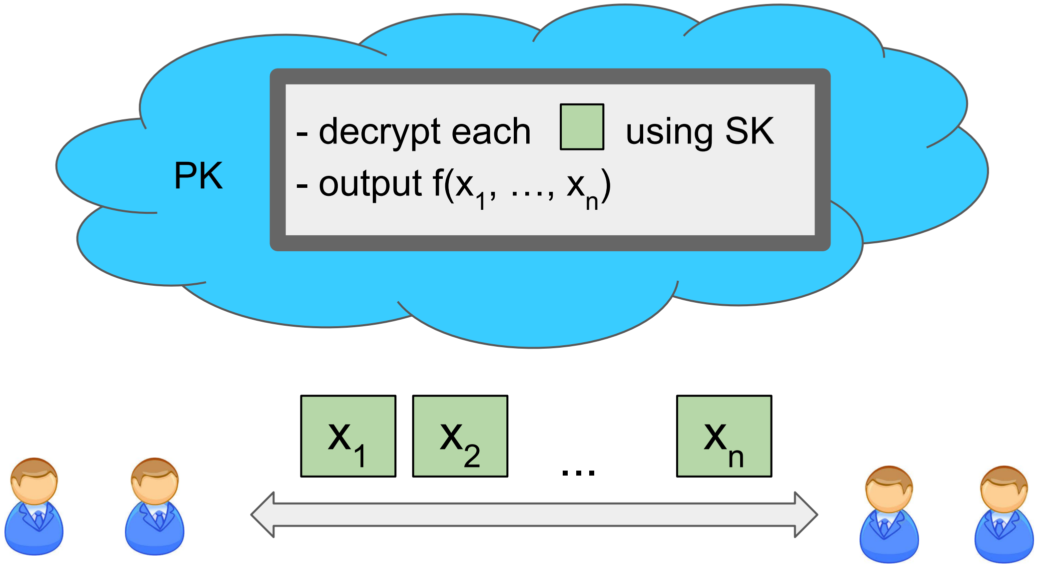
$\boxed{x_i} = \mathrm{Enc}_{PK}(x_i)$



First attempt

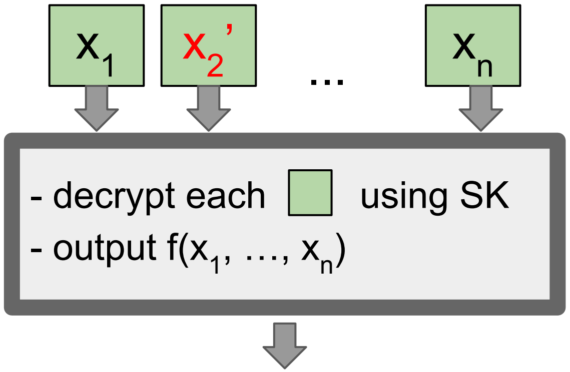
$\boxed{x_i} = Enc_{PK}(x_i)$



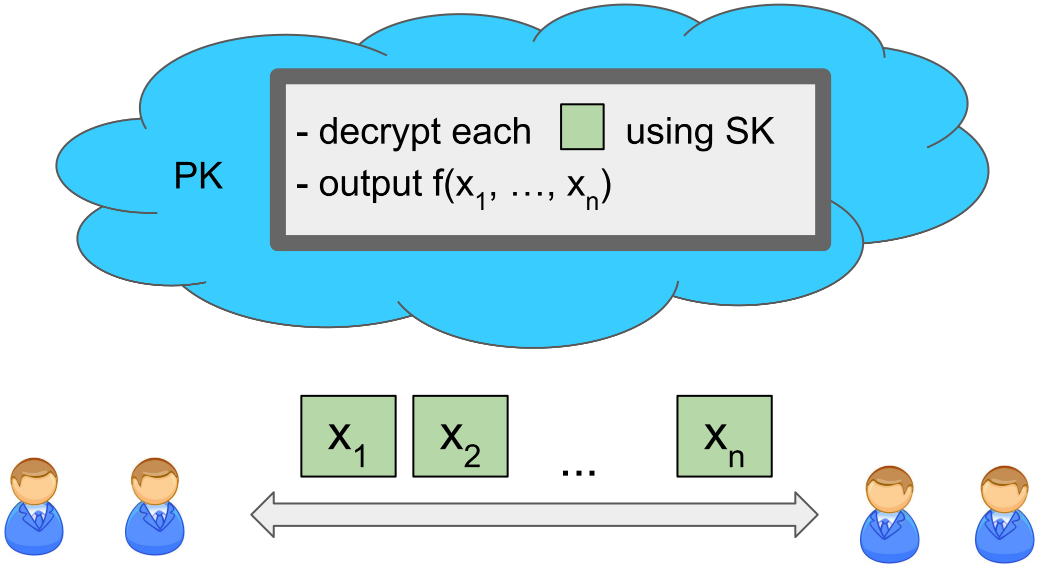


First attempt

$\boxed{x_i} = Enc_{PK}(x_i)$



$y' = f(x_1, x_2'\ldots, x_n)$

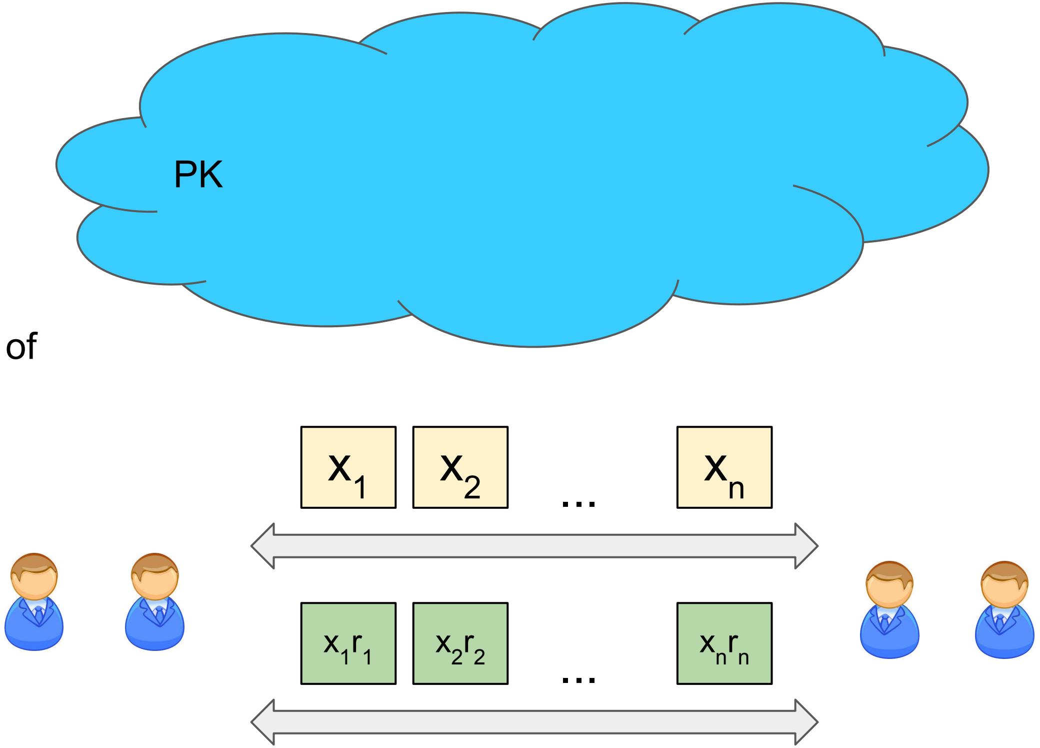


Second attempt

$\boxed{x_i} = \mathrm{Commit}(x_i; r_i)$

$\boxed{x_i r_i} = \mathrm{Enc}_{PK}(x_i || r_i)$

opening of comm

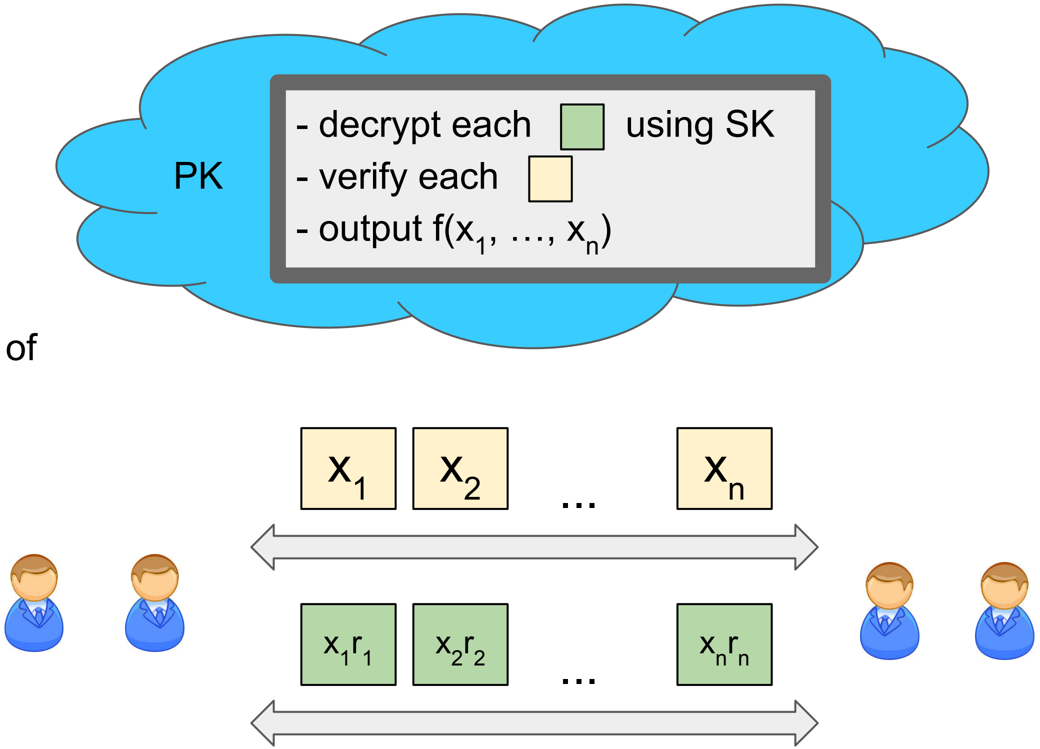


Second attempt

x_i = Commit(x_i; r_i)

$x_i r_i$ = $Enc_{PK}(x_i||r_i)$

opening of comm

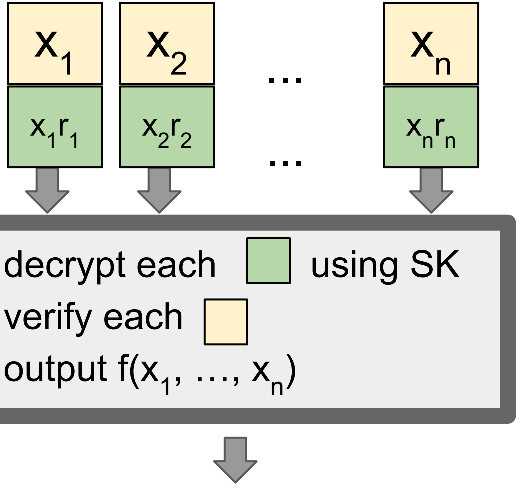


Second attempt

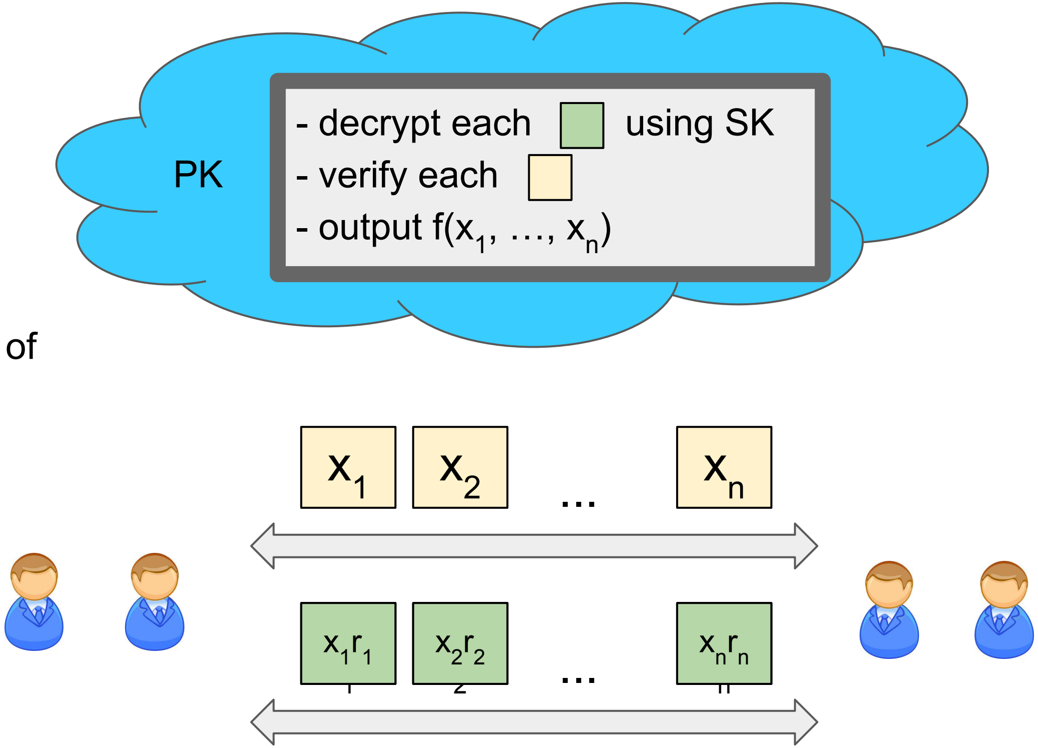
$\boxed{x_i} = \mathrm{Commit}(x_i;\, r_i)$

$\boxed{x_i r_i} = \mathrm{Enc}_{PK}(x_i || r_i)$

opening of comm



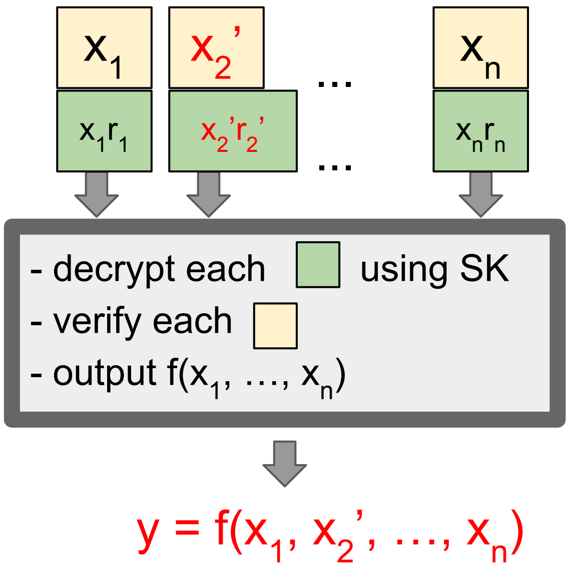
$y = f(x_1, x_2, \ldots, x_n)$

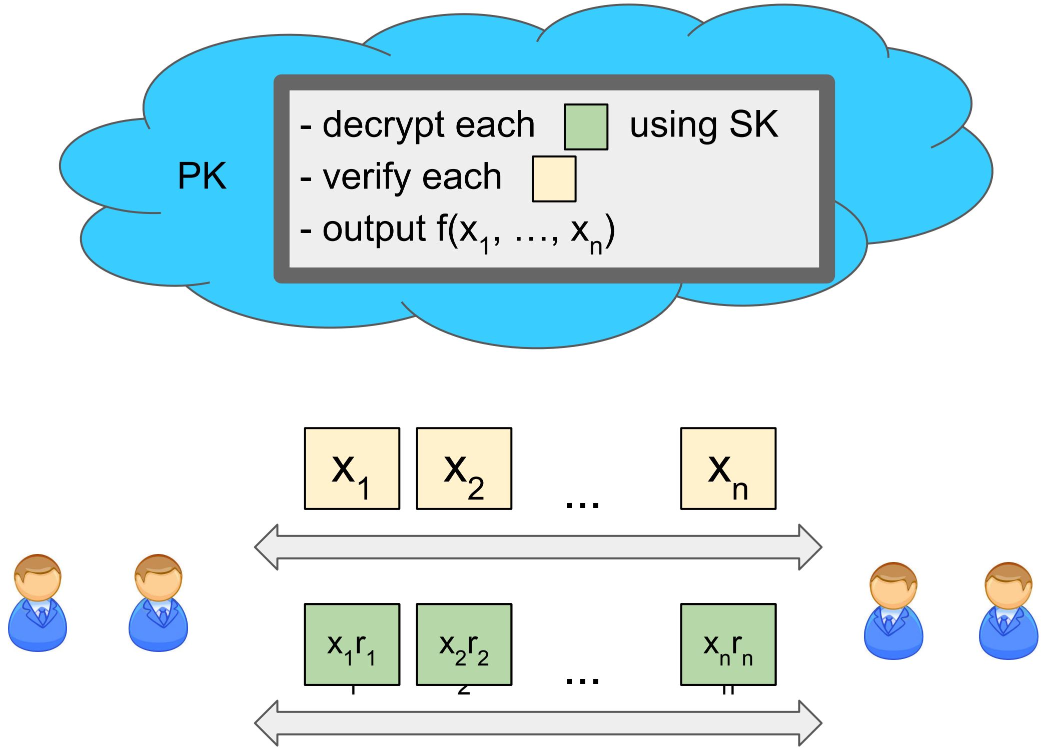


Second attempt

$\boxed{x_i} = \text{Commit}(x_i; r_i)$

$\boxed{x_i r_i} = \text{Enc}_{PK}(x_i || r_i)$

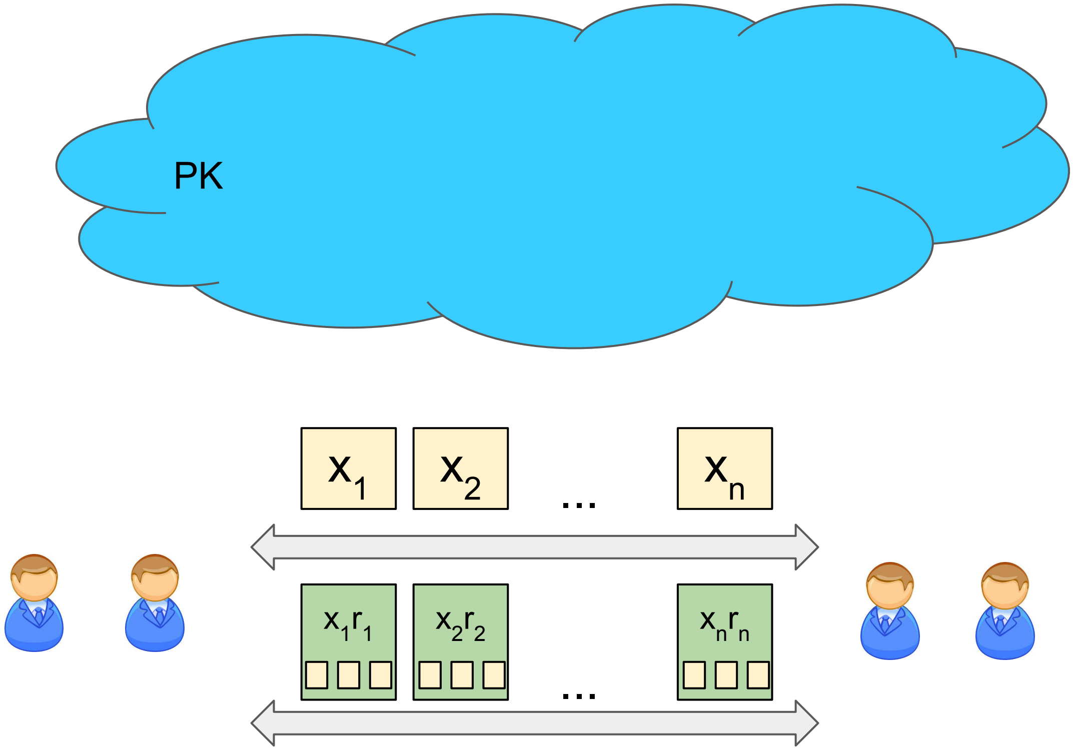




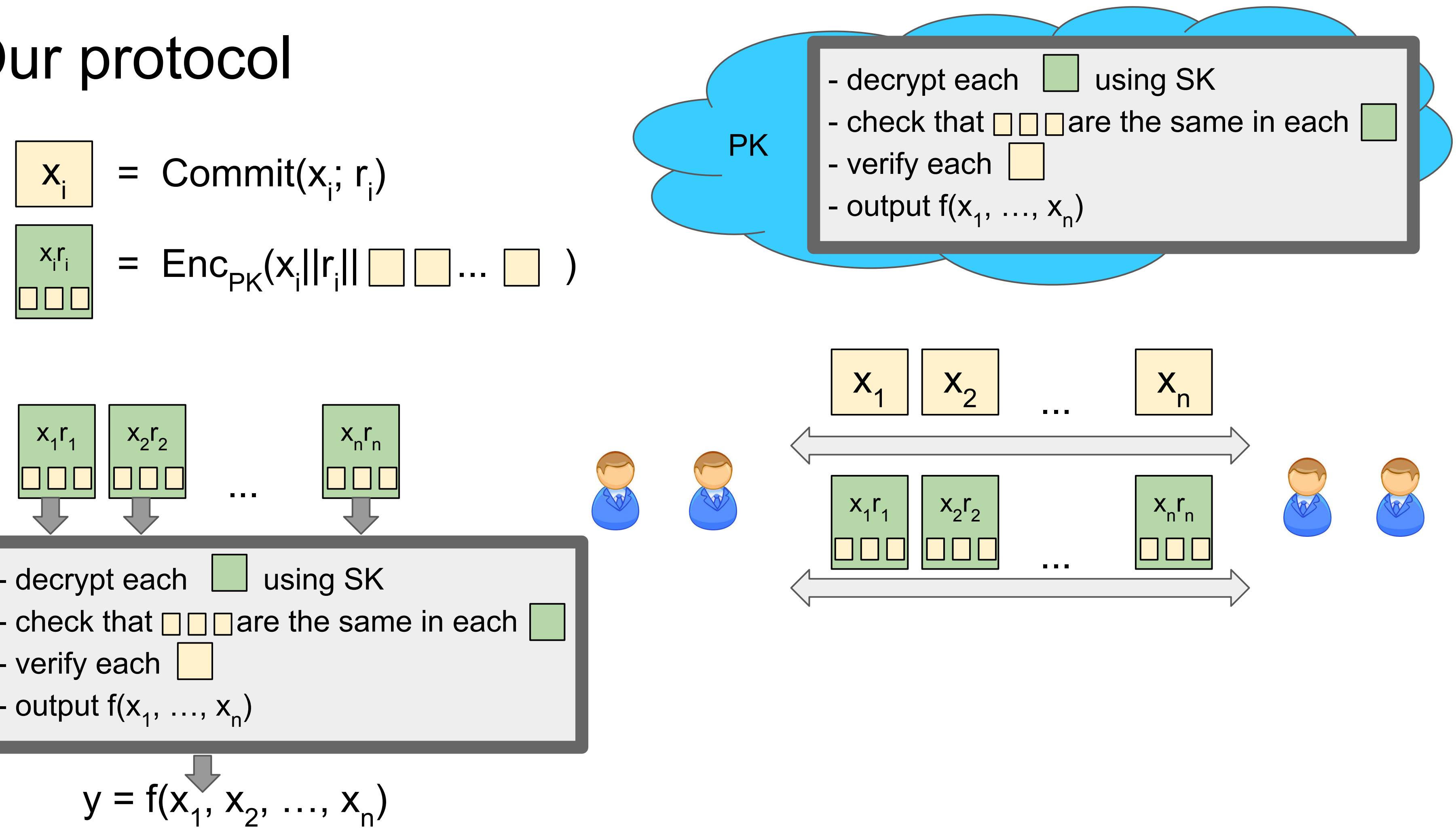
Our protocol

$\boxed{x_i} = \text{Commit}(x_i;\, r_i)$

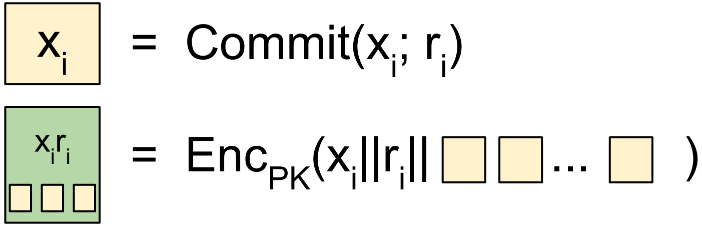
$\boxed{x_i r_i \;\square\,\square\,\square} = \text{Enc}_{PK}(x_i || r_i || \square\;\square \dots \square\;)$

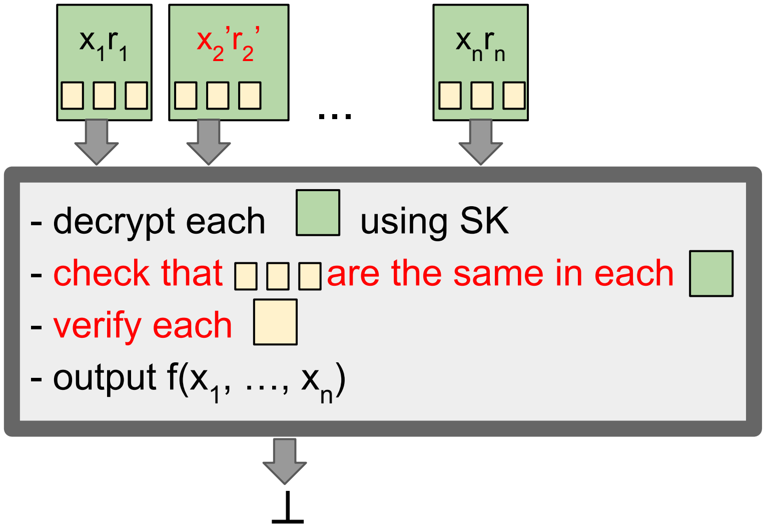


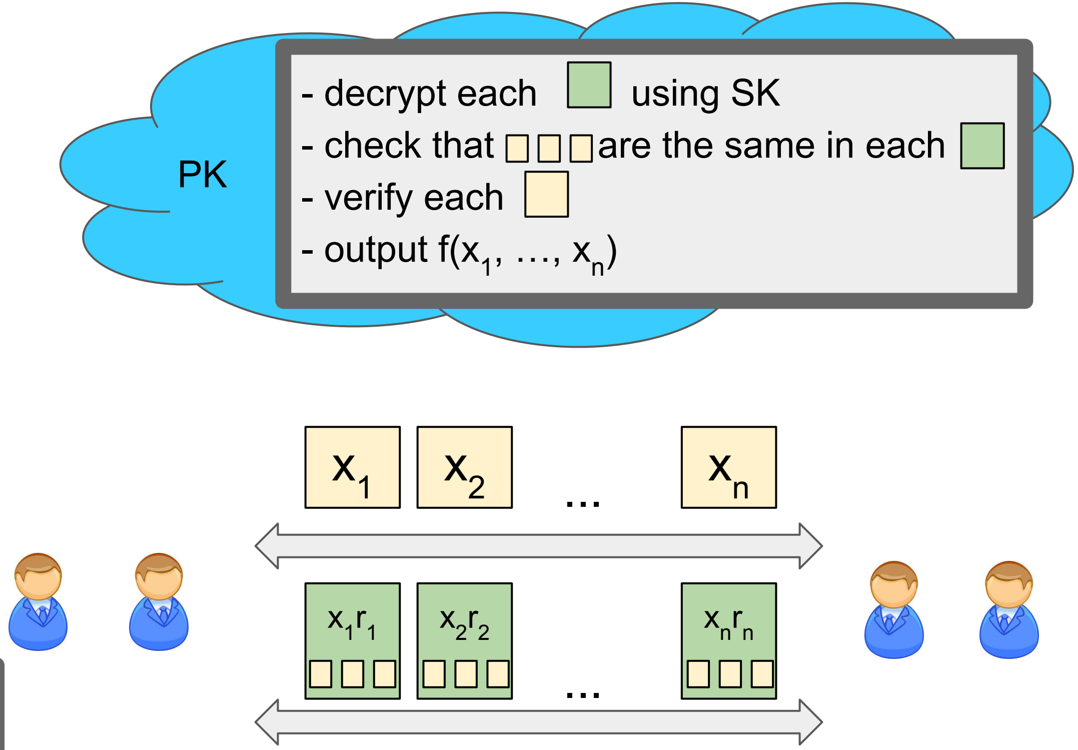
Our protocol



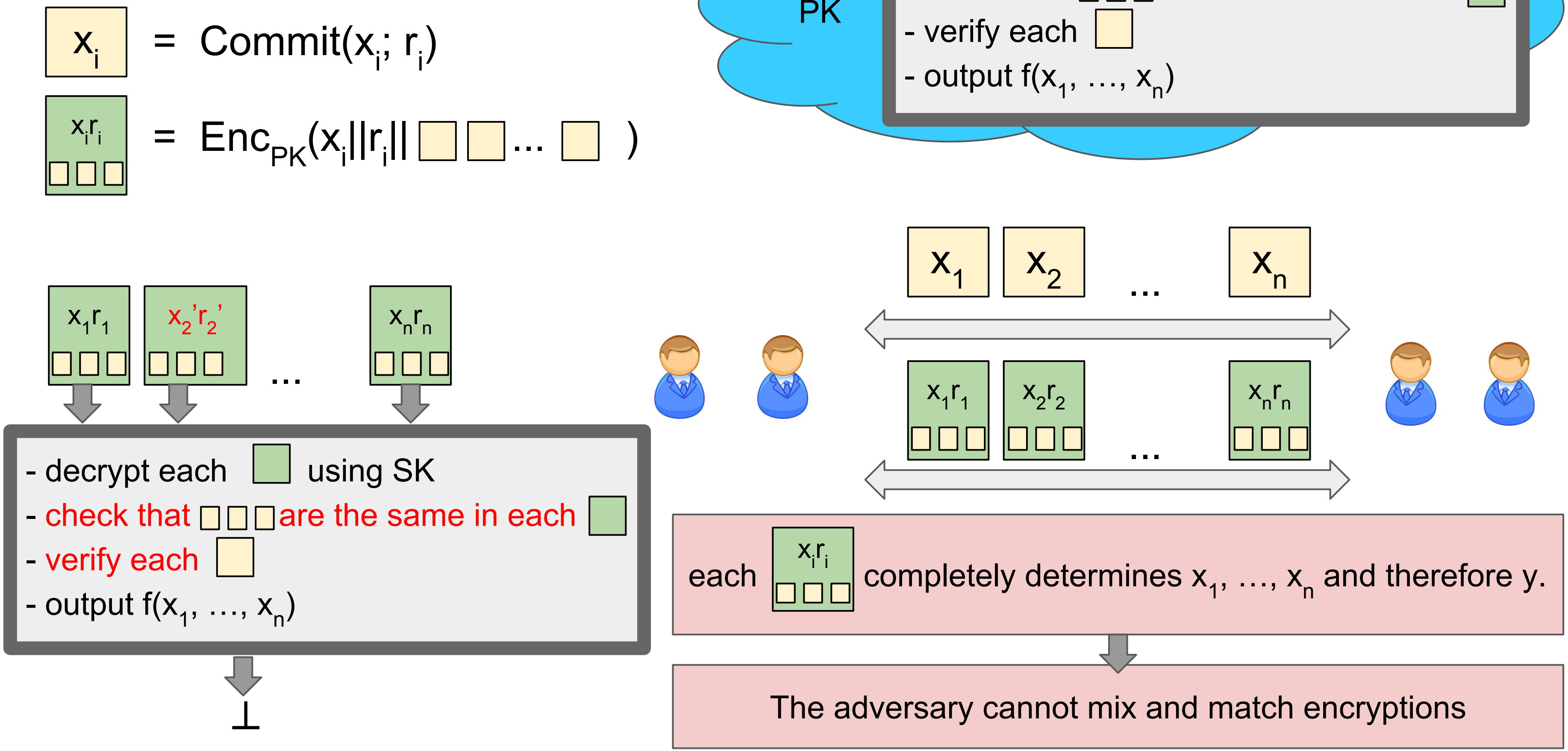
Our protocol



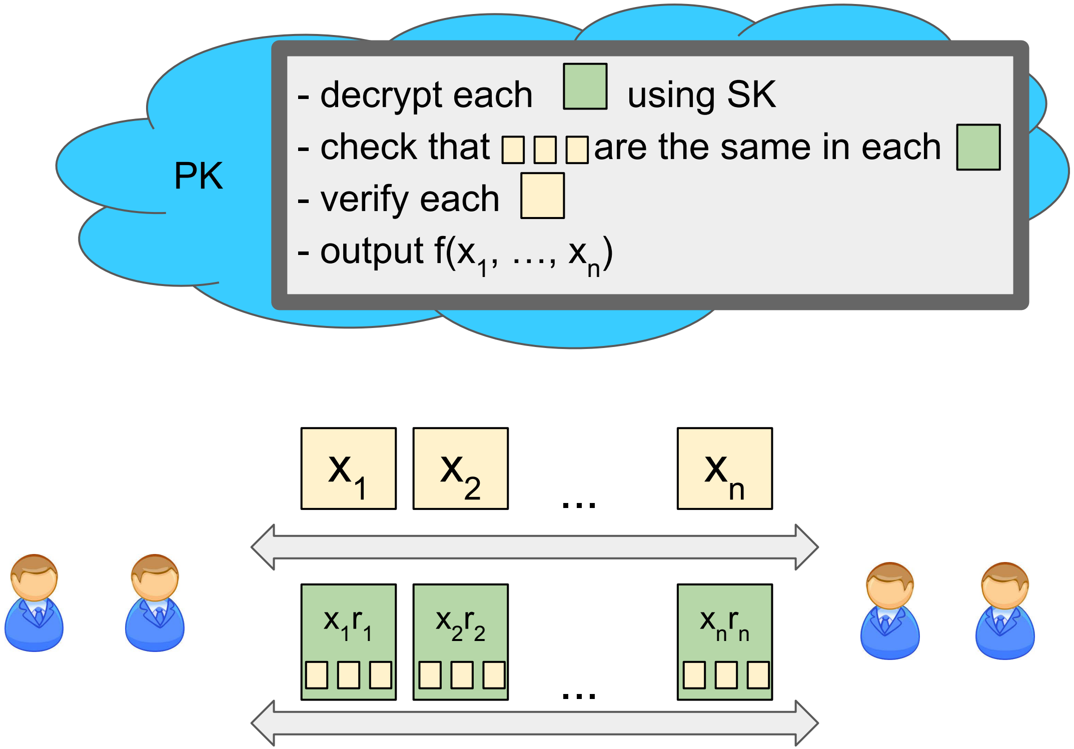




Our protocol



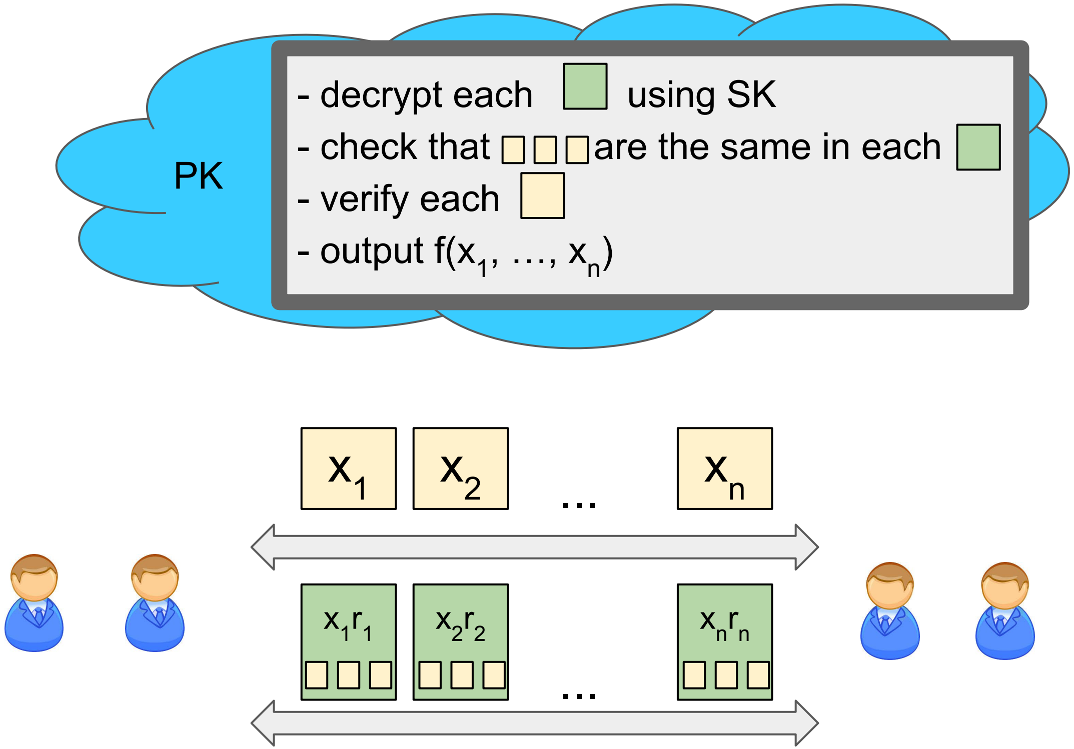
Required primitives



Required primitives

Commitments

Problem:
equivocal commitments require **local** CRS



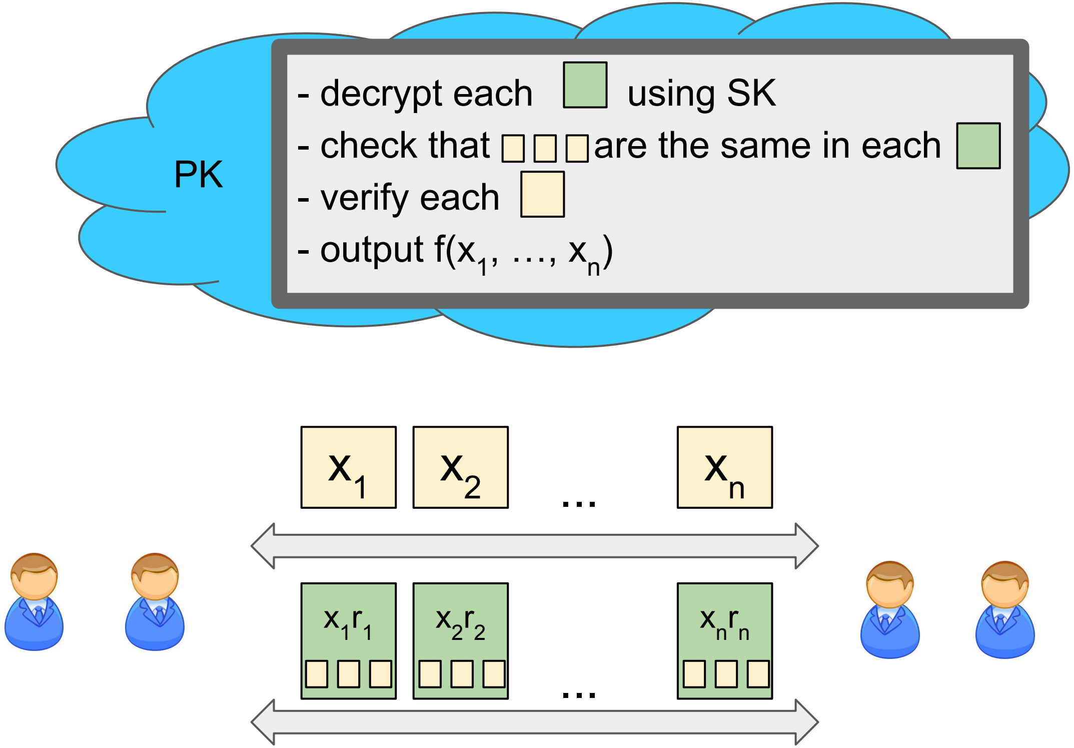
Required primitives

Commitments

Problem:
equivocal commitments require **local** CRS

Solution:
semi-honest commitments (no CRS)
Com(0) = (r, prg(s)); Com(1) = (prg(s), r)

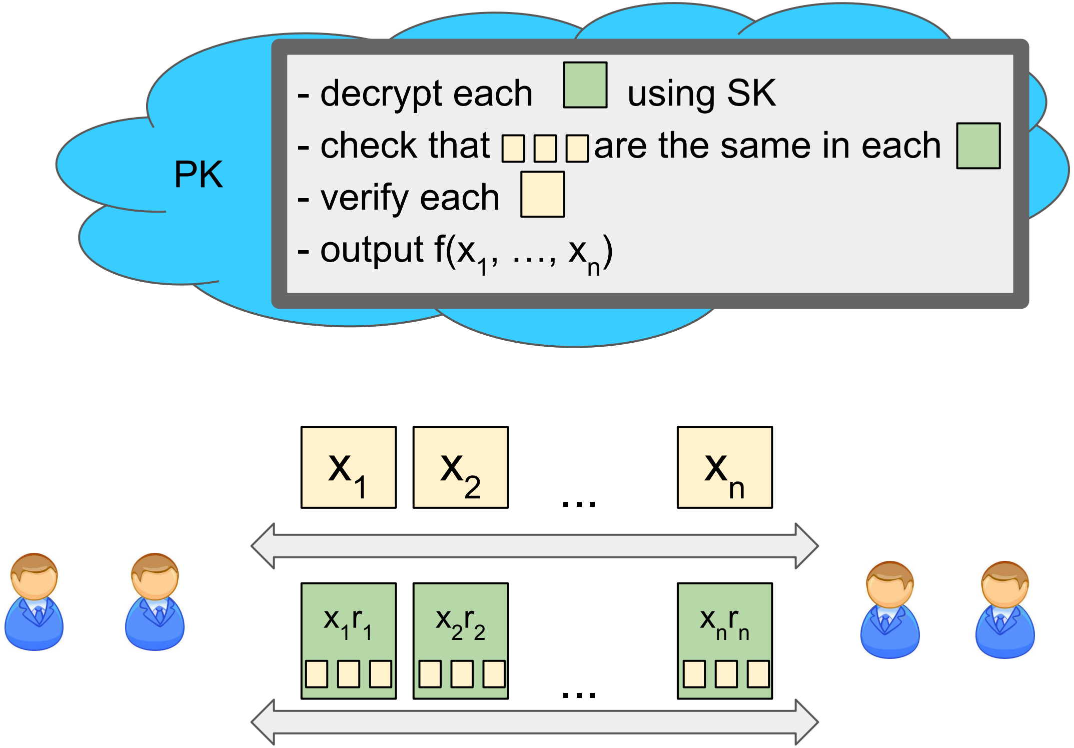
Property:
honestly generated ☐ is statistically binding.



Required primitives

Encryption

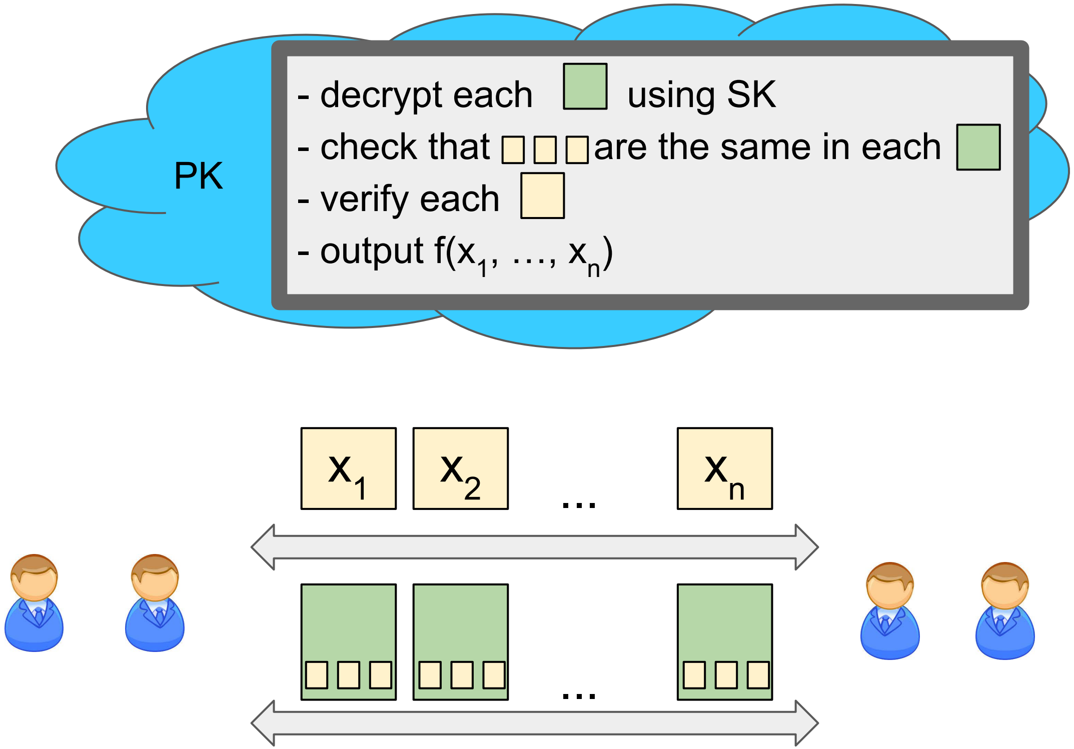
Problem:
cannot use security of encryption
since SK is in the program



Required primitives

Encryption

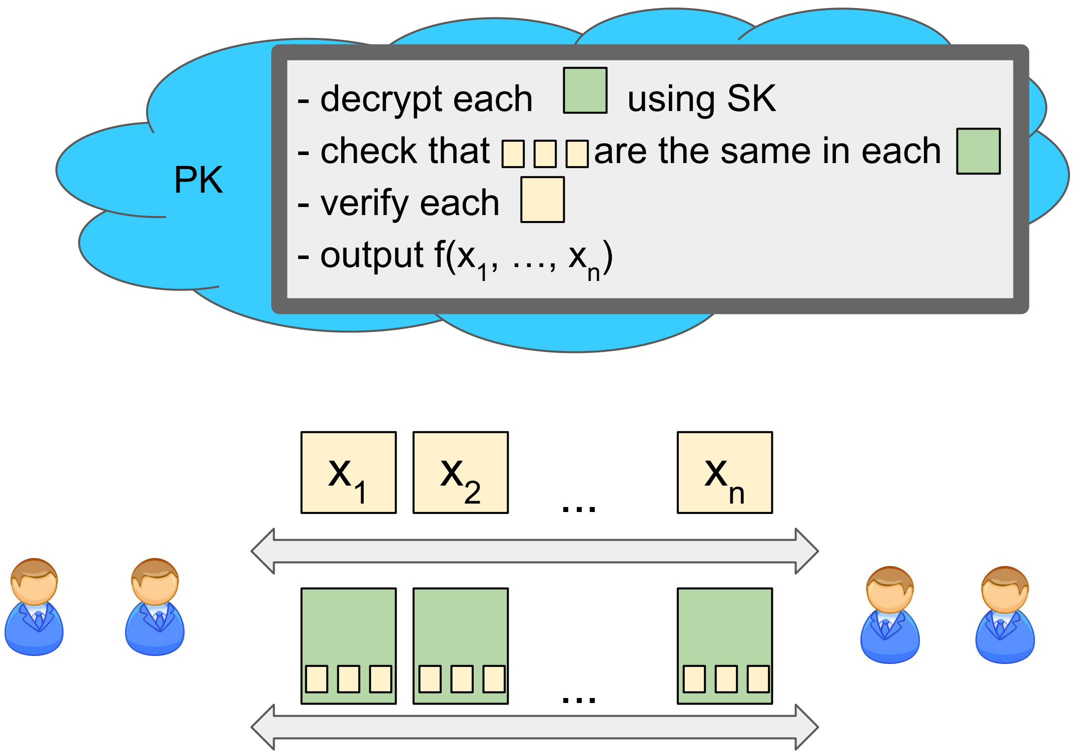
Problem:
cannot use security of encryption
since SK is in the program

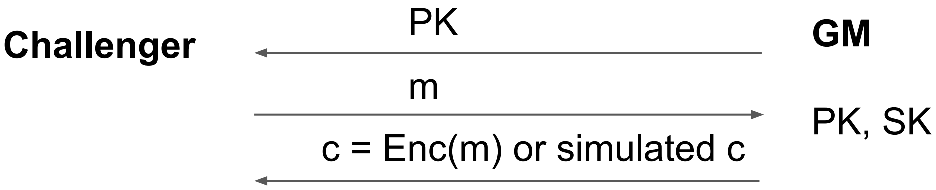


Required primitives

Encryption

Problem:
cannot use security of encryption
since SK is in the program

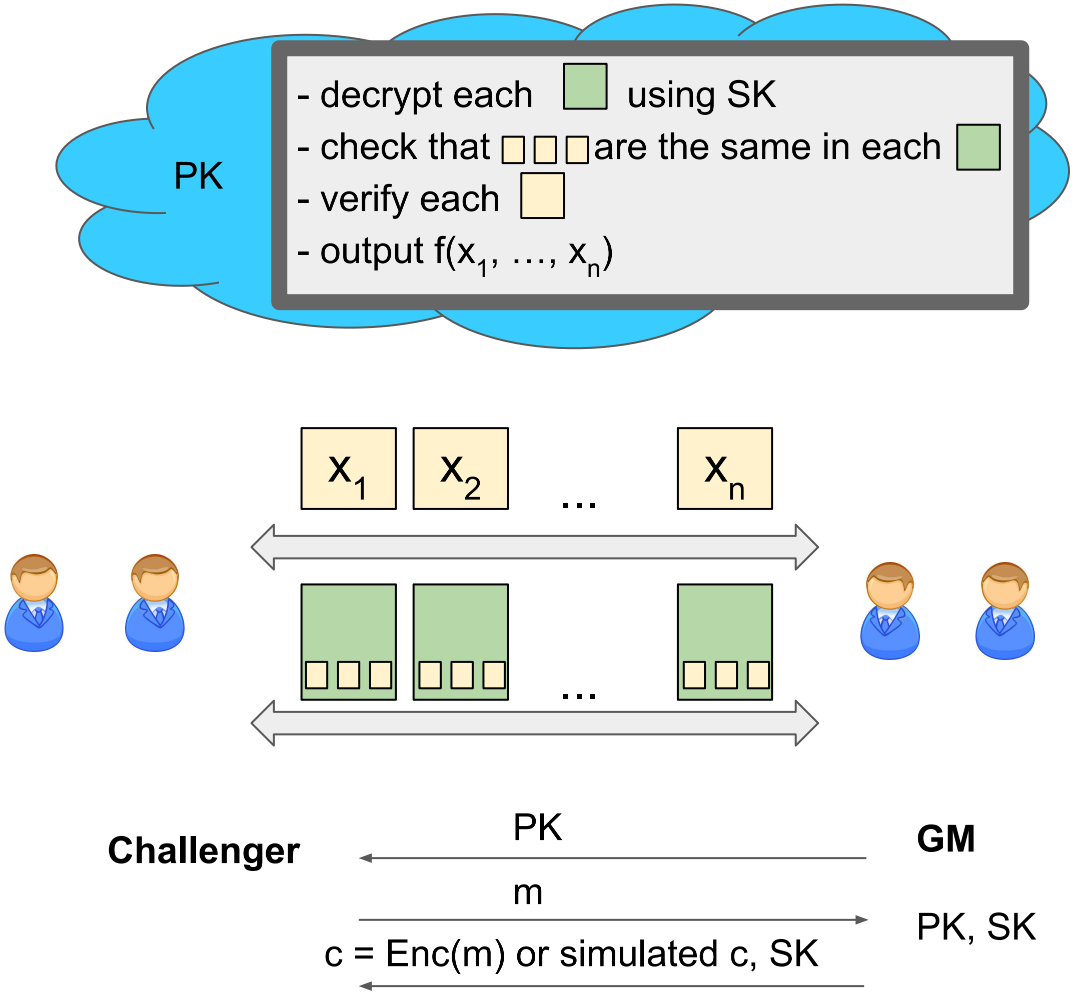




Required primitives

Encryption

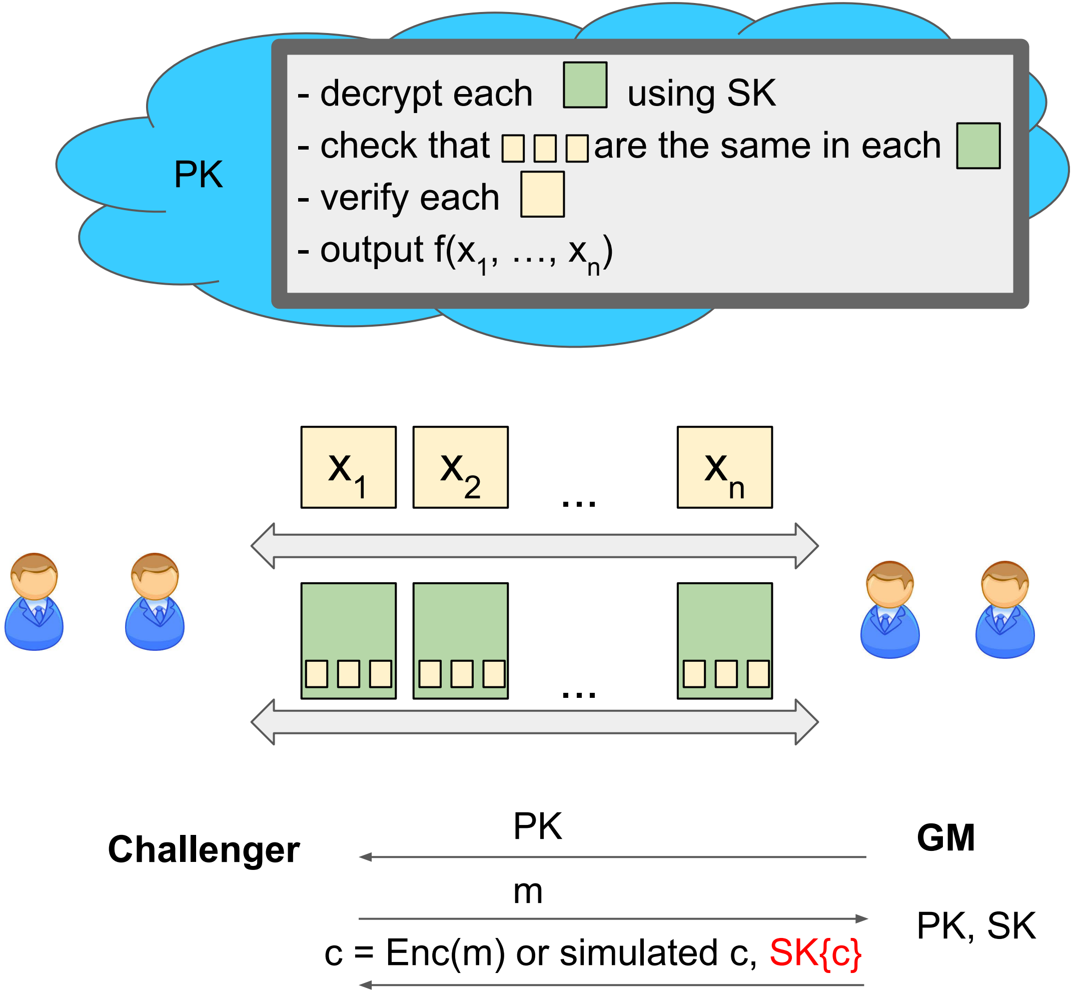
Problem:
cannot use security of encryption
since SK is in the program



Required primitives

Encryption

Problem:
cannot use security of encryption
since SK is in the program



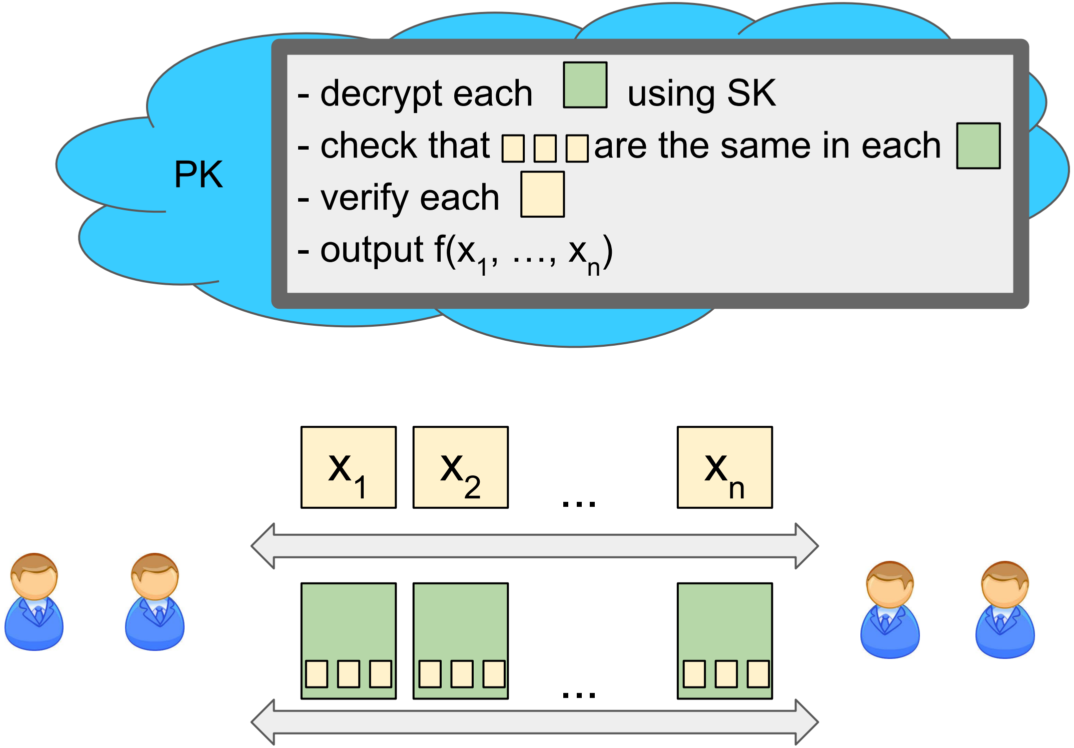
Required primitives

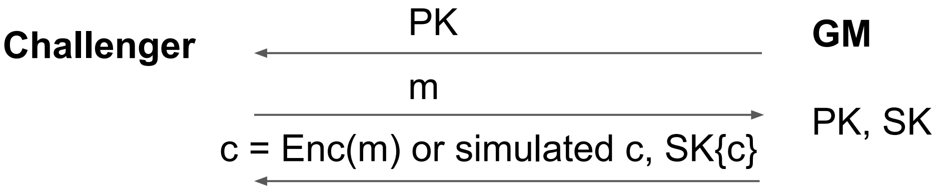
Encryption

Problem:
cannot use security of encryption
since SK is in the program

Solution:
Puncturable randomized encryption (PRE)
(from iO and injective OWFs)

Property:
simulation-secure
even when almost all SK is known





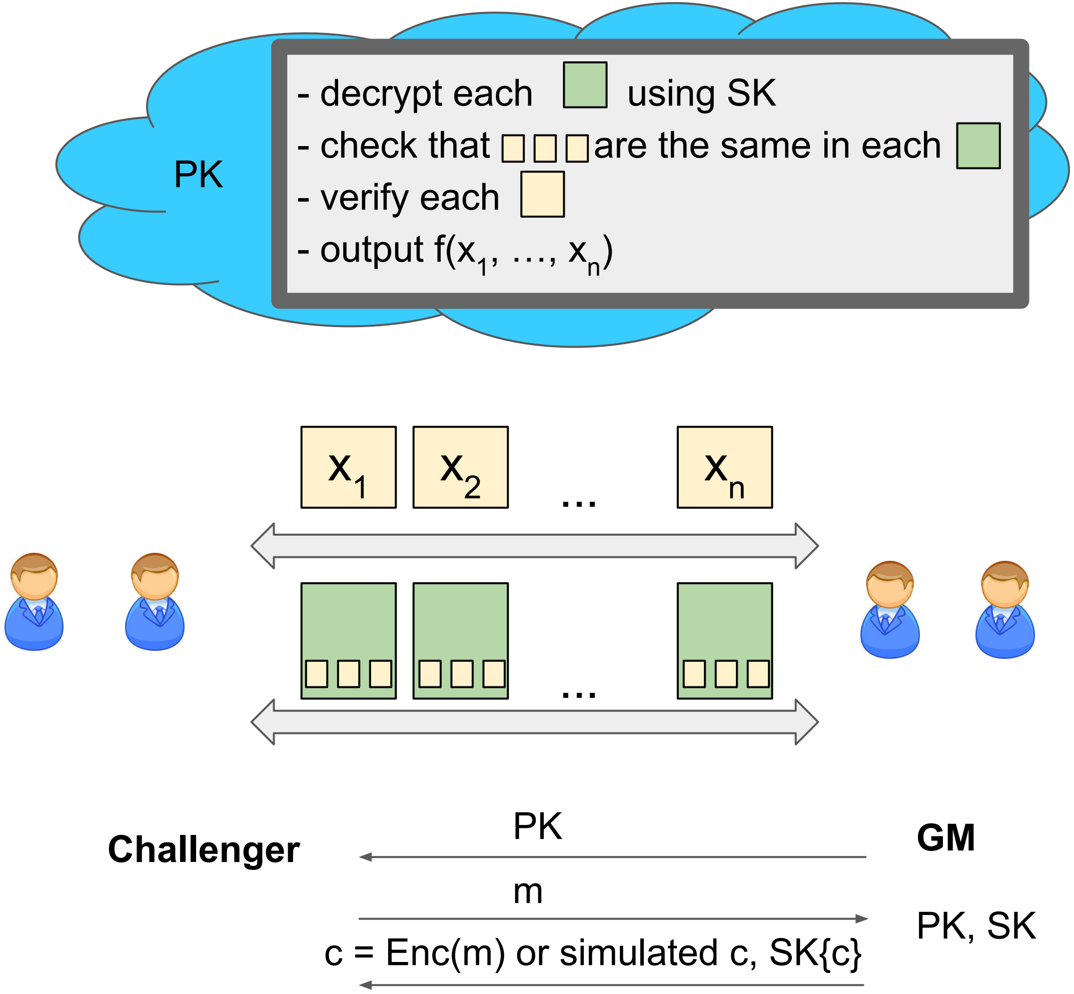
Required primitives

Encryption

Problem:
cannot use security of encryption
since SK is in the program

Solution:
Puncturable randomized encryption (PRE)
(from iO and injective OWFs)

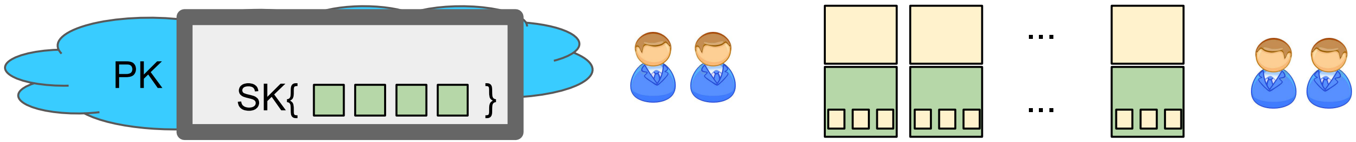
Property:
simulation-secure
even when almost all SK is known*



*: Simulation-secure analog of Sahai-Waters PDE

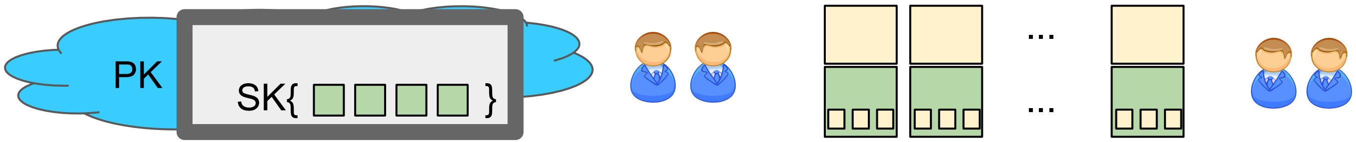
Achieving globality and full adaptive security

Simulation: not global



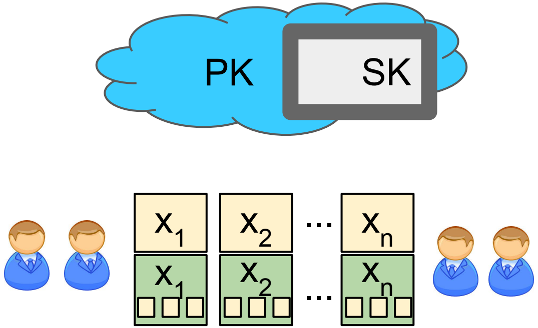
Achieving globality and full adaptive security

Simulation: not global



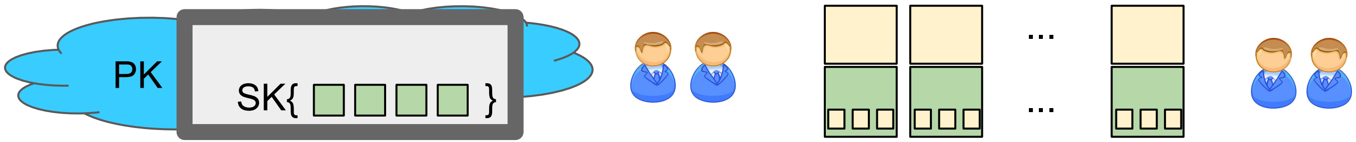
Solution: Modify the protocol to choose PK, 

during the execution.

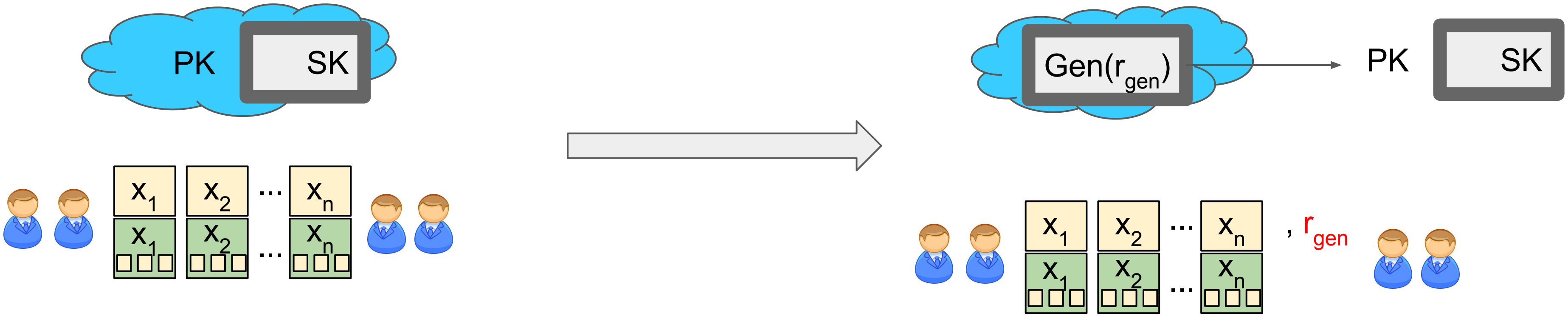


Achieving globality and full adaptive security

Simulation: not global



Solution: Modify the protocol to choose PK, SK during the execution.



How to make the protocol RAM-efficient

Ishai-Kushilevitz paradigm:

use MPC to evaluate garbling:

$F(x_1, \ldots, x_n; r)$ = garbled f, garbled $x_1, \ldots, x_n$.

How to make the protocol RAM-efficient

Ishai-Kushilevitz paradigm:
use MPC to evaluate garbling:
$F(x_1, \ldots, x_n; r)$ = garbled f, garbled $x_1, \ldots, x_n$.

Any MPC protocol + RAM-efficient garbling (e.g. CH'16) = RAM-efficient protocol

How to make the protocol RAM-efficient

Ishai-Kushilevitz paradigm:
use MPC to evaluate garbling:
$F(x_1, \ldots, x_n; r)$ = garbled f, garbled $x_1, \ldots, x_n$.

Any MPC protocol + RAM-efficient garbling (e.g. CH'16) = RAM-efficient protocol

Only works for n-1 corruptions!

How to make the protocol RAM-efficient

Ishai-Kushilevitz paradigm:
use MPC to evaluate garbling:
$F(x_1, \ldots, x_n; r) = \text{garbled } f, \text{garbled } x_1, \ldots, x_n.$

Any MPC protocol + RAM-efficient garbling (e.g. CH'16) = RAM-efficient protocol

Only works for n-1 corruptions!
For full adaptive security:

Any **randomness-hiding** MPC protocol + RAM-efficient garbling (e.g. CH'16) = RAM-efficient protocol

How to make the protocol RAM-efficient: **two ways**

Our MPC protocol (which is randomness-hiding) + RAM-efficient garbling (e.g. CH’16) = RAM-efficient protocol

How to make the protocol RAM-efficient: **two ways**

Our MPC protocol (which is randomness-hiding) + RAM-efficient garbling (e.g. CH'16) = RAM-efficient protocol

or

Our MPC protocol + iO for RAM = RAM-efficient protocol

(requires subexp. iO)

Part II: Byzantine protocol and NIZK for RAM

Part II: Byzantine protocol and NIZK for RAM

Any **randomness-hiding** MPC protocol	+	RAM-efficient garbling (e.g. CH'16)	=	RAM-efficient protocol

GP'15 doesn't compute randomness-hiding functionalities, i.e.IK02 approach doesn't work.

Malicious case

Observation: GP'15 works with circuits only because of NIZK proof of the statement $f(x_1, \ldots, x_n) = y$.
In all NIZK proofs so far: the work of verifier ~ circuit size of f.

Malicious case

Observation: GP'15 works with circuits only because of NIZK proof of the statement $f(x_1, \ldots, x_n) = y$.
In all NIZK proofs so far: the work of verifier ~ circuit size of f.

Theorem (Garg-Polychroniadou'15):
Assuming iO for RAM, one way functions, and **NIZK proofs for RAM,**
there exists **2-round, fully-adaptively-secure, RAM-efficient** MPC protocol against **malicious adversaries.**

Malicious case

Observation: GP'15 works with circuits only because of NIZK proof of the statement $f(x_1, \ldots, x_n) = y$.
In all NIZK proofs so far: the work of verifier ~ circuit size of f.

Theorem (Garg-Polychroniadou'15):
Assuming iO for RAM, one way functions, and **NIZK proofs for RAM,**
there exists **2-round, fully-adaptively-secure, RAM-efficient** MPC protocol against **malicious adversaries.**

Theorem (Our work):
Assuming garbling scheme for RAM and NIZK proofs for circuits, there exists **statistically sound NIZK proof system** for RAM.

Defs: NIZK, Garbling

NIZK proof system:
Let language L be defined by relation R(x; w)
Prove(x, w) → $\boldsymbol{\pi}$
Verify(x, $\boldsymbol{\pi}$) → accept / reject

Defs: NIZK, Garbling

NIZK proof system:
Let language L be defined by relation R(x; w)
Prove(x, w) → $\boldsymbol{\pi}$
Verify(x, $\boldsymbol{\pi}$) → accept / reject

Completeness;
Statistical soundness;
Zero-knowledge;
RAM-efficient*:

- work of P only depends on $|R|_{RAM}$
- $|\boldsymbol{\pi}|$ only depends on $|R|_{RAM}$
- work of V depends on RAM complexity of R

*: everything also depends on |x|, |w|.

Defs: NIZK, Garbling

NIZK proof system:
Let language L be defined by relation R(x; w)
Prove(x, w) → $\boldsymbol{\pi}$
Verify(x, $\boldsymbol{\pi}$) → accept / reject

Completeness;
Statistical soundness;
Zero-knowledge;
RAM-efficient*:
- work of P only depends on $|R|_{RAM}$
- $|\boldsymbol{\pi}|$ only depends on $|R|_{RAM}$
- work of V depends on RAM complexity of R

Garbling scheme:
KeyGen(r) → k
GarbleProg(k, f) → $\boxed{f}$
GarbleInput(k, x) → $\boxed{x}$

*: everything also depends on |x|, |w|.

Defs: NIZK, Garbling

NIZK proof system:
Let language L be defined by relation R(x; w)
Prove(x, w) → $\boldsymbol{\pi}$
Verify(x, $\boldsymbol{\pi}$) → accept / reject

Completeness;
Statistical soundness;
Zero-knowledge;
RAM-efficient*:
- work of P only depends on $|R|_{RAM}$
- $|\boldsymbol{\pi}|$ only depends on $|R|_{RAM}$
- work of V depends on RAM complexity of R

*: everything also depends on |x|, |w|.

Garbling scheme:
KeyGen(r) → k
GarbleProg(k, f) → $\boxed{f}$
GarbleInput(k, x) → $\boxed{x}$

Correctness: can compute f(x)
Security: garbled values only reveal f(x)
RAM-efficient*:
- work of the garbler only depends on $|f|_{RAM}$
- size of garbled values depends on $|f|_{RAM}$
- work of the evaluator depends on RAM complexity of f

*: everything also depends on |x|

Defs: NIZK, Garbling

NIZK proof system:
Let language L be defined by relation R(x; w)
Prove(x, w) → $\boldsymbol{\pi}$
Verify(x, $\boldsymbol{\pi}$) → accept / reject

Completeness;
Statistical soundness;
Zero-knowledge;
RAM-efficient*:
- work of P only depends on $|R|_{RAM}$
- $|\boldsymbol{\pi}|$ only depends on $|R|_{RAM}$
- work of V depends on RAM complexity of R

*: everything also depends on |x|, |w|.

Garbling scheme:
KeyGen(r) → k
GarbleProg(k, f) → $\boxed{f}$
GarbleInput(k, x) → $\boxed{x}$

Correctness: can compute f(x)
Security: garbled values only reveal f(x)
RAM-efficient*:
- work of the garbler only depends on $|f|_{RAM}$
- size of garbled values depends on $|f|_{RAM}$
- work of the evaluator depends on RAM complexity of f

Exists under iO for circuits + OWFs (Canetti-Holmgren'16)

*: everything also depends on |x|

NIZK + Garbled RAM → NIZK for RAM

Attempt 1

Convince that ∃ w such that $R(x; w) = 1$

Prover

$x \in L$
w

Verifier

$x \in L$

NIZK + Garbled RAM → NIZK for RAM

Attempt 1

Convince that $\exists$ w such that R(x; w) = 1

Prover

$x \in L$
w

Verifier

$x \in L$

KeyGen(r) → k
GarbleProg(k, R) → R(*,*)
GarbleInput(k, (xw)) → x, w

NIZK + Garbled RAM → NIZK for RAM

Attempt 1

Convince that ∃ w such that R(x; w) = 1

Prover

$x \in L$
w

Proof π = R(*,*) x, w

Verifier

$x \in L$

KeyGen(r) → k
GarbleProg(k, R) → R(*,*)
GarbleInput(k, (xw)) → x, w

Accept if Eval(R(*,*) x, w) = 1

NIZK + Garbled RAM → NIZK for RAM

Attempt 1

Convince that ∃ w such that R(x; w) = 1

Prover

x ∈ L
w

Proof π = R(*,*) x, w

Verifier

x ∈ L

Accept if Eval(R(*,*) x, w) = 1

KeyGen(r) → k
GarbleProg(k, R) → R(*,*)
GarbleInput(k, (xw)) → x, w

- Verifier doesn't learn anything about w

NIZK + Garbled RAM → NIZK for RAM

Attempt 1

Convince that ∃ w such that R(x; w) = 1

Prover

x ∈ L
w

Proof π = R(*,*) x, w

Verifier

x ∈ L

Accept if Eval(R(*,*) x, w) = 1

KeyGen(r) → k
GarbleProg(k, R) → R(*,*)
GarbleInput(k, (xw)) → x, w

- Verifier doesn't learn anything about w
- Malicious prover can garble all-one function

NIZK + Garbled RAM → NIZK for RAM

Attempt 2

Convince that $\exists$ w such that R(x; w) = 1

Prover

x ∈ L
w

R(*,*) | x, w

NIZK proof: “garbling done correctly, for correct R and x”

Verifier
x ∈ L

Accept if Eval(R(*,*) | x, w) = 1

and if NIZK verifies.

KeyGen(r) → k
GarbleProg(k, R) → R(*,*)
GarbleInput(k, (xw)) → x, w

NIZK + Garbled RAM → NIZK for RAM

Attempt 2

Convince that $\exists$ w such that R(x; w) = 1

Prover

x ∈ L
w

R(*,*) | x, w

NIZK proof: “garbling done correctly, for correct R and x”

→

Verifier
x ∈ L

Accept if Eval(R(*,*) | x, w) = 1

and if NIZK verifies.

KeyGen(r) → k
GarbleProg(k, R) → R(*,*)
GarbleInput(k, (xw)) → x, w

- Verifier doesn’t learn anything about w
- Correctness of garbling guaranteed by NIZK: idea works for **perfectly correct** garbling scheme for RAM

NIZK + Garbled RAM → NIZK for RAM

Attempt 2

Convince that ∃ w such that R(x; w) = 1

Prover

x ∈ L

w

R(*,*) | x, w

NIZK proof: "garbling done correctly, for correct R and x"

Verifier

x ∈ L

Accept if Eval(R(*,*) x, w) = 1

and if NIZK verifies.

KeyGen(r) → k

GarbleProg(k, R) → R(*,*)

GarbleInput(k, (xw)) → x, w

- Verifier doesn't learn anything about w
- Correctness of garbling guaranteed by NIZK: idea works for **perfectly correct** garbling scheme for RAM
- Problem: don't have perfectly correct garbling scheme for RAM

NIZK + Garbled RAM → NIZK for RAM

Attempt 2

Convince that ∃ w such that $R(x; w) = 1$

Prover

$x \in L$
w

R(*,*) x, w

NIZK proof: “garbling done correctly, for correct R and x”

Verifier

$x \in L$

KeyGen(r) → k
GarbleProg(k, R) → R(*,*)
GarbleInput(k, (xw)) → x, w

Accept if Eval(R(*,*) x, w) = 1

and if NIZK verifies.

What might go wrong?

- Can verify that garbling was done correctly for some r
- cannot verify that r was chosen at random

NIZK + Garbled RAM → NIZK for RAM

Attempt 2

Convince that ∃ w such that R(x; w) = 1

Prover

$x \in L$
w

R(*,*) x, w

NIZK proof: “garbling done correctly, for correct R and x”

Verifier
$x \in L$

Accept if Eval(R(*,*) x, w) = 1

and if NIZK verifies.

KeyGen(r) → k
GarbleProg(k, R) → R(*,*)
GarbleInput(k, (xw)) → x, w

What might go wrong?
Consider garbling which is incorrect for one bad key k’:
- For k ≠ k’ the evaluation is always correct
- for k’ GarbleProg always outputs all-one function.

NIZK + Garbled RAM → NIZK for RAM

Attempt 2

Convince that ∃ w such that R(x; w) = 1

Malicious Prover

$x \notin L$

output 1 | x, 0

NIZK proof: "garbling done correctly, for correct R and x"

Verifier accepts

$x \in L$

Accept if Eval(R(*,*) | x, w) = 1

and if NIZK verifies.

KeyGen(r') → k'

GarbleProg(k', R) → output 1

GarbleInput(k', x, 0) → x, 0

What might go wrong?

Consider garbling which is incorrect for one bad key k':

- For k ≠ k' the evaluation is always correct
- for k' GarbleProg always outputs all-one function.

NIZK + Garbled RAM → NIZK for RAM

Attempt 2

Convince that ∃ w such that R(x; w) = 1

Malicious Prover

$x \notin L$

output 1 | x, 0

NIZK proof: “garbling done correctly, for correct R and x”

Verifier accepts

$x \in L$

Accept if Eval(R(*,*) | x, w) = 1

and if NIZK verifies.

KeyGen(r’) → k’

GarbleProg(k’, R) → output 1

GarbleInput(k’, x, 0) → x, 0

Crucial observation:

the garbling scheme of CH15 is **perfectly correct with abort**, i.e.:
for **any key** k evaluation of garbled program on garbled input wither gives correct output, or ⊥ .

Summary: two round adaptively secure protocols

Semi-honest case:

- global CRS
- RAM-efficient
- computes randomized functionalities
- from iO and injective OWFs (no subexp iO)

Malicious case (GP15 + our RAM efficient NIZK):

- RAM-efficient
- from subexp iO and TDP

Open questions

Fully adaptive constant round HBC protocol **without a CRS?**
Fully adaptive constant round malicious protocol **without subexp iO?**

Questions?