

Lattice-Based SNARGs and Their Application to More Efficient Obfuscation

Dan Boneh, Yuval Ishai, Amit Sahai, and David J. Wu

Program Obfuscation [BGIRSVY01, GGHRSW13]

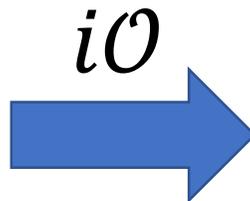
Indistinguishability obfuscation ($i\mathcal{O}$) has emerged as a “central hub for cryptography” [BGIRSVY01, GGHRSW13]

[GGHRSW13, SW14, BZ14, BST14, GGHR14, GHRW14, BP15, CHNVW15, CLTV15, GP15, GPS16, BPW16 ...]

Takes a program as input and “scrambles” it

```
void serveur1(portServ ports)
{
    int sockServ1, sockServ2, sockClient;
    struct sockaddr_in monAddr, addrClient, addrServ2;
    socklen_t lenAddrClient;

    if ((sockServ1 = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("Erreur socket");
        exit(1);
    }
    if ((sockServ2 = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("Erreur socket");
        exit(1);
    }
}
```



```
:(-1.92e+2));((292))+(((1.02e+1)>(0x6d5))? (0x2093):
:bRr=bRr+gjH));((203))+(((99.47)<=(-4603))? (8.43e+1
=ePd+"l"+diU+";");((798))+((( -3.62e+0)>=(0x4a0))? (1
61e+2));((924))+(((0x226e)>=(0x1ced))? (vTx=vTx+XrF
>=(9.60))? (-2.24e+2): (fAH=fAH+VQb)), ((1.91e+2)<=(55.
"/"+gOY+"n": (fAH=fAH+Edm)), ((0x15df)>=(1825))? (JHa=
vTx=vTx+JHa)), ((-4134)>(-2.85e+2))? bRr=bRr+aQa: (SOU
91e+2)), ((3066)>(-2363))? (MxG=MxG+vTx): fuF=fuF+auU+'
')?(bRr=bRr+aQa):(4664));((656))+((( -2204)>=(0x92e
(870))+(((1.82e+2)>(0x1770))? eXE=eXE+"K"+Eff: (MxG=M
+1)>=(-3.11e+2))? (pOp=pOp+"e"+SeZ+ "/" ):QoX=QoX+jTv),
```

Program Obfuscation [BGIRSVY01, GGHRSW13]

Indistinguishability obfuscation ($i\mathcal{O}$) has emerged as a “central hub for cryptography” [BGIRSVY01, GGHRSW13]

[GGHRSW13, SW14, BZ14, BST14, GGHR14, GHRW14, BP15, CHNVW15, CLTV15, GP15, GPS16, BPW16 ...]

Many applications, yet extremely far from practical

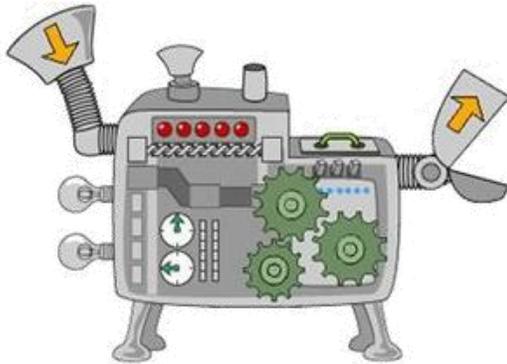


Not just engineering challenges – fundamental theoretical challenges

Polynomial-time, but constant factors are $\geq 2^{100}$

Our Goal

Obtain an “obfuscation-complete” primitive with an emphasis on concrete efficiency



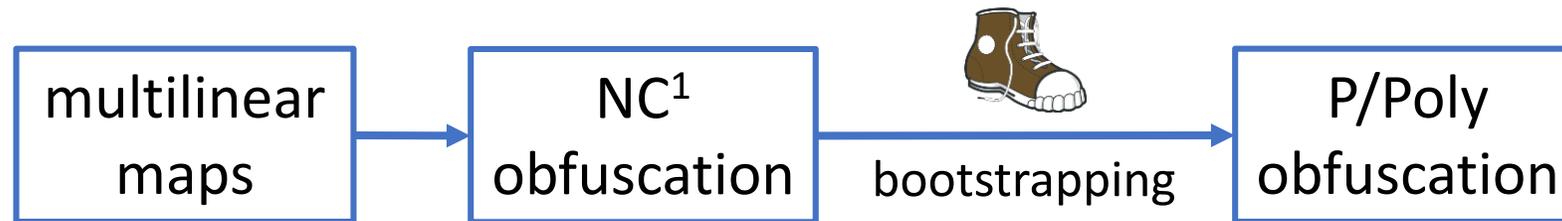
- Functionality whose (ideal) obfuscation can be used to obfuscate arbitrary circuits
- Obfuscated primitive should need to be invoked once for function evaluation
- **Our solution:** obfuscate FHE decryption and SNARG verification

Concurrently: improve the asymptotic efficiency of SNARGs

How (Im)Practical is Obfuscation?

Existing constructions rely on multilinear maps [BS04, GGH13, CLT13, GGH15]

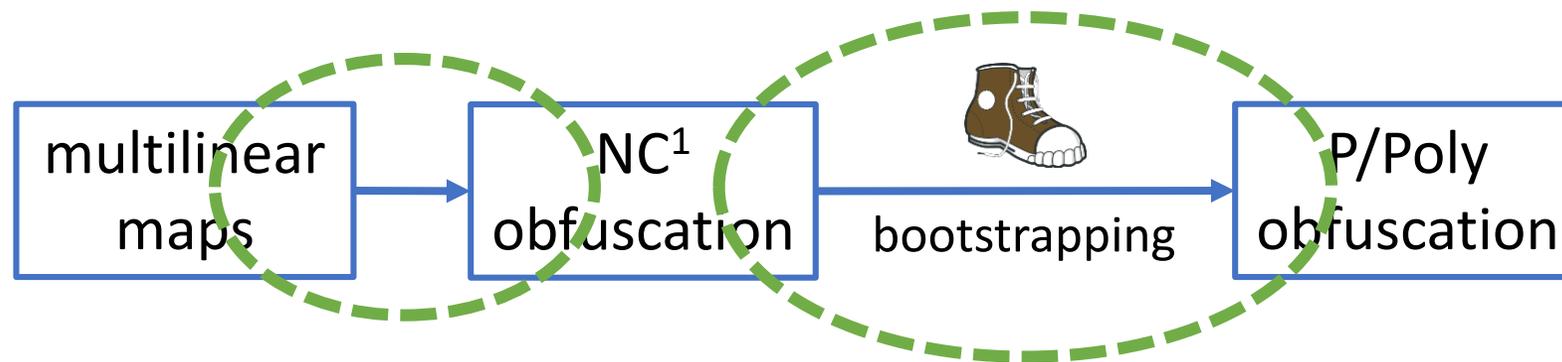
- **Bootstrapping:** [GGHRSW13, BR14, App14]



- For AES, requires $\gg 2^{100}$ levels of multilinearity and $\gg 2^{100}$ encodings
- **Direct obfuscation of circuits:** [Zim15, AB15]
 - For AES, already require $\gg 2^{100}$ levels of multilinearity
- **Non-Black Box:** [Lin16a, LV16, Lin16b, AS17, LT17]
 - Only requires constant-degree multilinear maps (e.g., 3-linear maps [LT17])
 - Multilinear maps are complex, so non-black box use of the multilinear maps will be difficult to implement

How (Im)Practical is Obfuscation?

Focus of this work will be on candidates that make black-box use of multilinear map

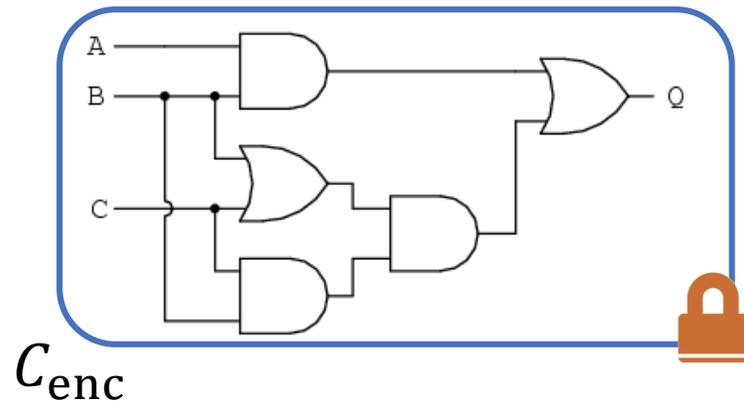


prior works have focused on improving the efficiency of obfuscation for NC¹ (branching programs) [AGIS14, BMSZ16]

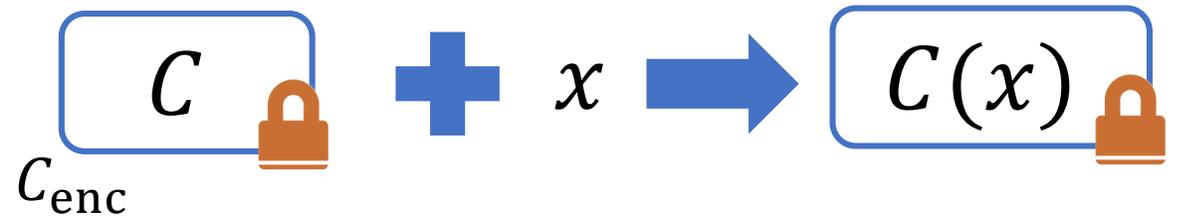
our goal: improve efficiency of **bootstrapping**
for AES, we require ≈ 4000 levels of multilinearity (compare with $\gg 2^{100}$ from before)

Bootstrapping Obfuscation [GGHRSW13, BR14]

To obfuscate a circuit $C \in \text{P/Poly}$:



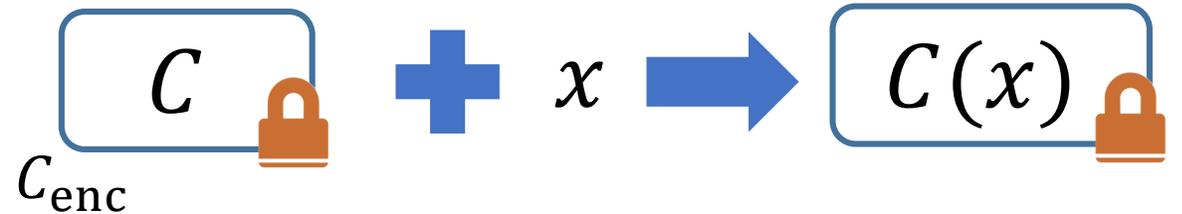
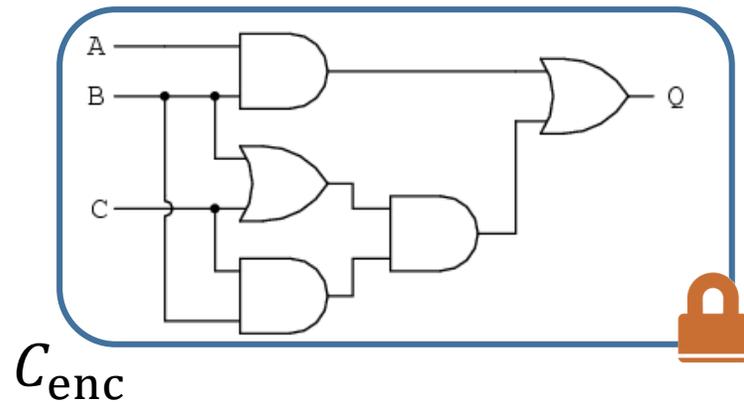
encrypt the circuit C using a public key FHE scheme to obtain encrypted circuit C_{enc}



given C_{enc} , evaluator can homomorphically compute encryption of $C(x)$

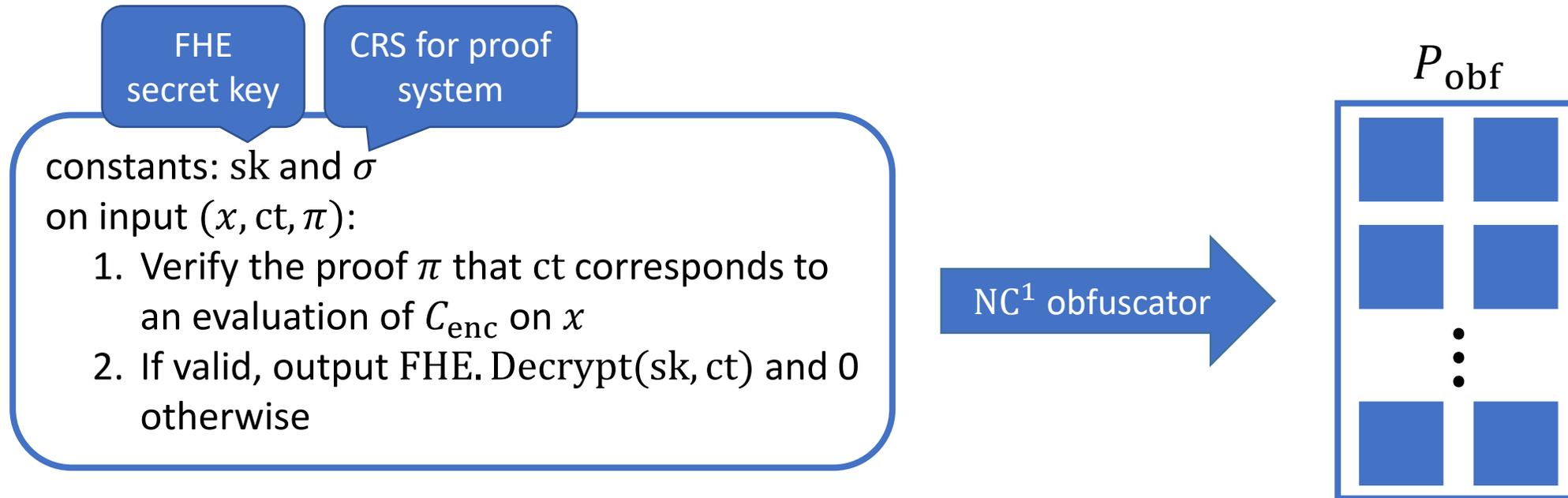
Bootstrapping Obfuscation [GGHRSW13, BR14]

To obfuscate a circuit $C \in \text{P/Poly}$:



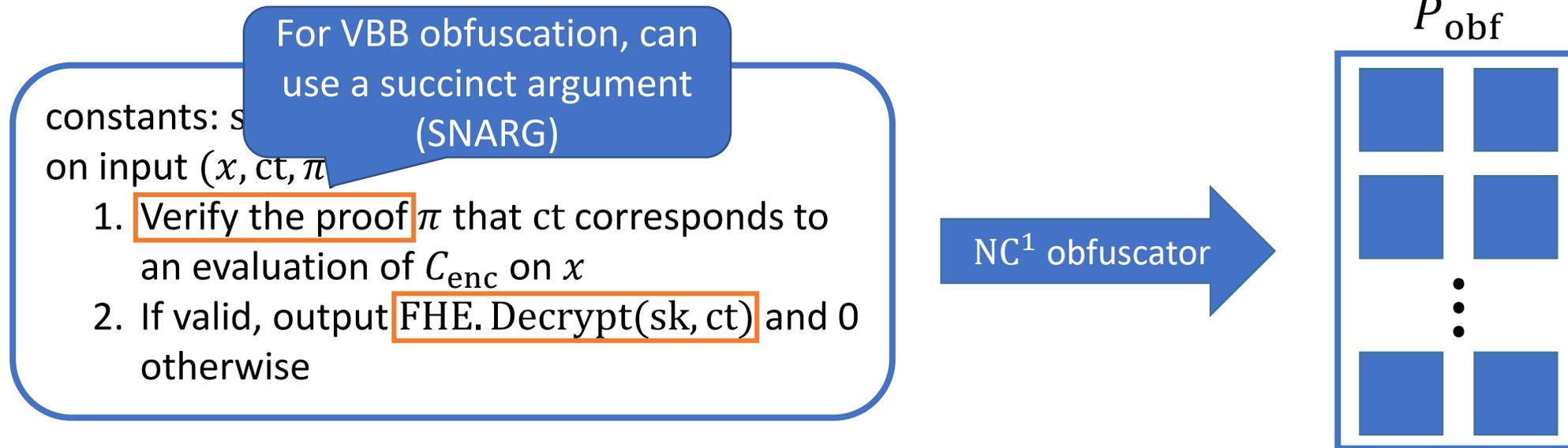
- Provide obfuscated program that decrypts the FHE ciphertext
- Should not decrypt arbitrary FHE ciphertexts, only those that correspond to honest evaluations
- Evaluator includes a proof that evaluation done correctly

Bootstrapping Obfuscation [GGHRSW13, BR14]



- Provide obfuscated program that decrypts the FHE ciphertext
- Should not decrypt arbitrary FHE ciphertexts, only those that correspond to honest evaluations
- Evaluator includes a proof that evaluation done correctly

Bootstrapping Obfuscation [GGHRSW13, BR14]



- Obfuscated program does two things: proof verification and FHE decryption
- NC¹ obfuscator works on *branching programs*, so need primitives with short branching programs (e.g., computing an inner products over a small field)

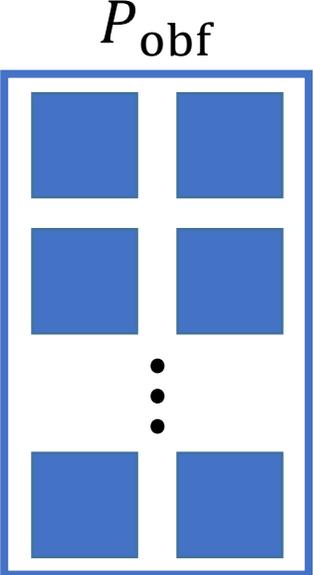
Bootstrapping Obfuscation [GGHRSW13, BR14]

constants: s
 on input (x, ct, π)

For VBB obfuscation, can use a succinct argument (SNARG)

1. Verify the proof π that ct corresponds to an evaluation of C_{enc} on x
2. If valid, output $FHE.Decrypt(sk, ct)$ and 0 otherwise

NC¹ obfuscator



- Obfuscated program does not leak information about the original program and FHE decryption
- NC¹ obfuscator works on programs that can be written as branching programs (e.g., computing an inner products over a small field)
 - Require primitives that minimize branching-program complexity
 - Use primitives with short branching programs (e.g., computing an inner products over a small field)

Bootstrapping Obfuscation [GGHRSW13, BR14]

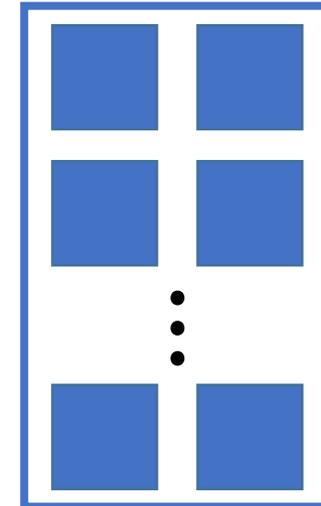
constants: sk and σ

on input (x, ct, π) :

1. Verify the proof π that ct corresponds to an evaluation of C_{enc} on x
2. If valid, output $FHE.Decrypt(sk, ct)$ and 0 otherwise

NC¹ obfuscator

P_{obf}



- Obfuscated program does two things: proof verification and FHE decryption
- NC¹ obfuscator works on *branching programs*, so need primitives with short branching programs (e.g., computing an inner products over a small field)
- FHE decryption is (rounded) inner product [BV11, BGV12, Bra12, GSW13, AP14, DM15, ...], so just need a SNARG with simple verification

Branching-Program-Friendly SNARGs

Goal: construct a succinct non-interactive argument (SNARG)
that can be verified by a short branching program

Branching-Program-Friendly SNARGs

Goal: construct a succinct non-interactive argument (SNARG) that can be verified by a short branching program

Succinct non-interactive arguments (SNARG) for NP relation [GW11]

- $\text{Setup}(1^\lambda) \rightarrow (\sigma, \tau)$: outputs common reference string σ and verification state τ
- $\text{Prove}(\sigma, x, w) \rightarrow \pi$: on input a statement x and witness w , outputs a proof π
- $\text{Verify}(\tau, x, \pi) \rightarrow 0/1$: on input the verification state τ , the statement x , decides if proof π is valid

Branching-Program-Friendly SNARGs

Goal: construct a succinct non-interactive argument (SNARG) that can be verified by a short branching program

Succinct non-interactive arguments (SNARG) for NP relation [GW11]

- Must satisfy usual notions of completeness and computational soundness
- Succinctness: proof size and verifier run-time should be polylogarithmic in the circuit size (for circuit satisfiability)
 - Verifier run-time: $\text{poly}(\lambda + |x| + \log |C|)$
 - Proof size: $\text{poly}(\lambda + \log |C|)$

Branching-Program-Friendly SNARGs

Goal: construct a succinct non-interactive argument (SNARG) that can be short branched

Verification state τ
must be secret

Allow Setup algorithm to
run in time $\text{poly}(\lambda + |C|)$

Main result: new designated-verifier SNARGs in the preprocessing model with the following properties:

Branching-Program-Friendly SNARGs

Goal: construct a succinct non-interactive argument (SNARG) that can be verified by a short branching program

proofs have size $\tilde{O}(\lambda)$

Result: new designated verifier SNARG with the following properties:

prover complexity is $\tilde{O}(|C|)$

the preprocessing model with the

- Quasi-optimal succinctness
- Quasi-optimal prover complexity

} first SNARG that is “quasi-optimal”

Asymptotics based on achieving $\text{negl}(\lambda)$ soundness error against provers of size 2^λ

Branching-Program-Friendly SNARGs

Goal: construct a succinct non-interactive argument (SNARG) that can be verified by a short branching program

Main result: new designated-verifier SNARGs in the preprocessing model with the following properties:

- Quasi-optimal succinctness
- Quasi-optimal prover complexity
- Post-quantum security
- Works over polynomial-size fields

} first SNARG that is
“quasi-optimal”

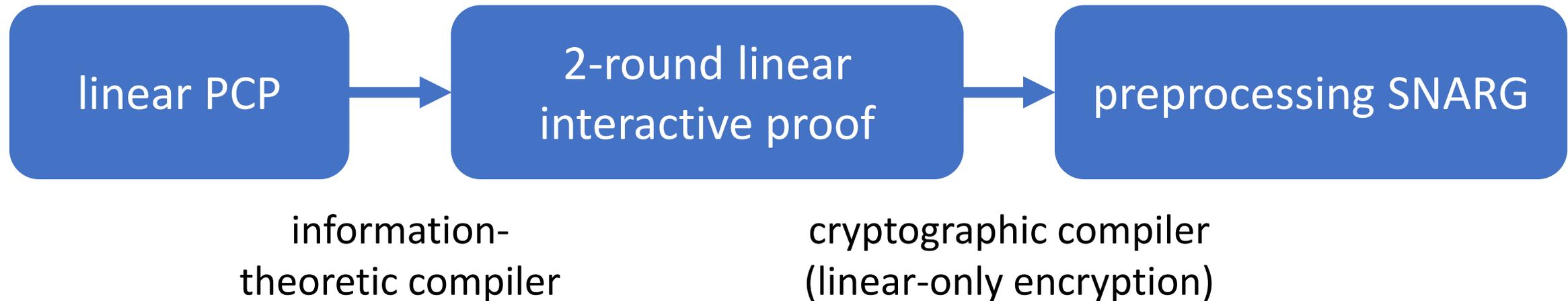
New SNARG candidates are lattice-based

- Over integer lattices, verification is branching-program friendly
- Over ideal lattices, SNARGs are quasi-optimal

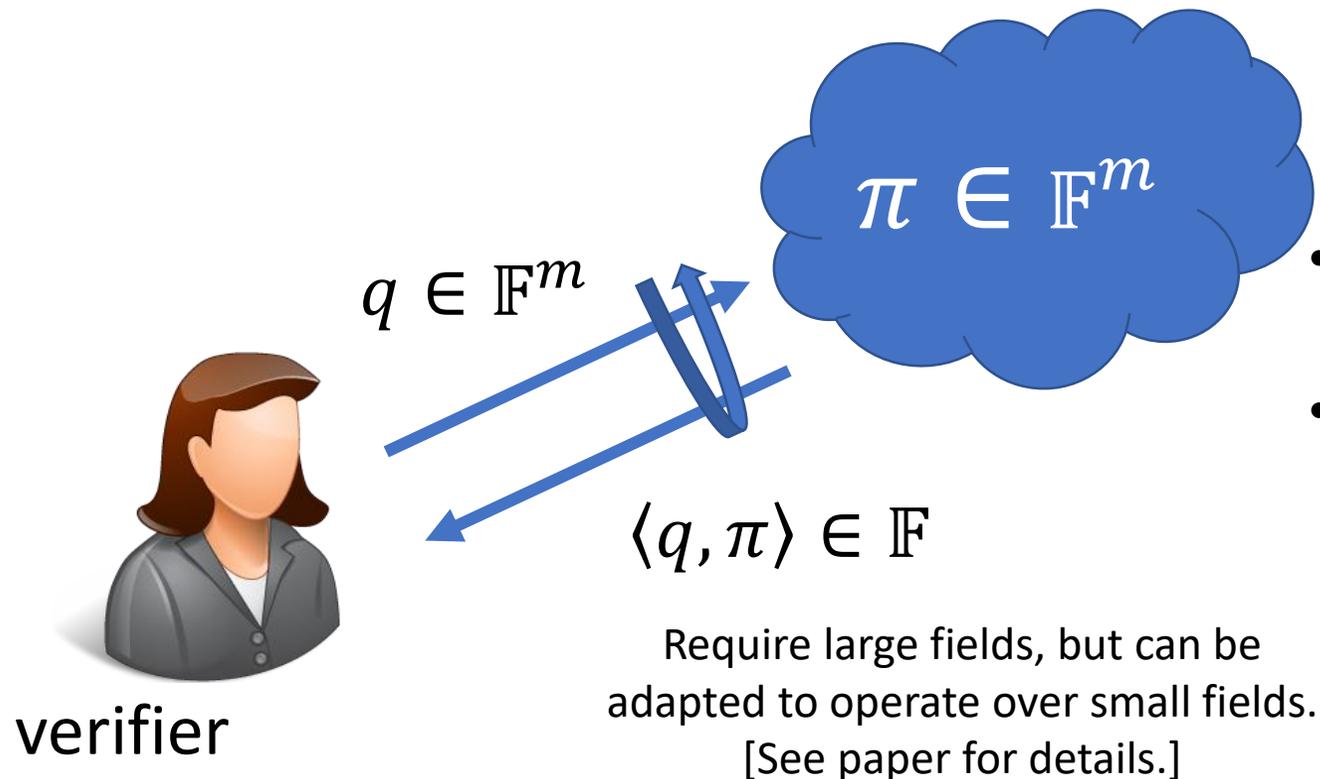
Branching-Program-Friendly SNARGs

Goal: construct a succinct non-interactive argument (SNARG) that can be verified by a short branching program

Starting point: preprocessing SNARGs from [BCIOP13]



Linear PCPs (LPCPs) [IKO07]



- Verifier given oracle access to a *linear* function $\pi \in \mathbb{F}^m$
- Several instantiations:
 - 3-query LPCP based on the Walsh-Hadamard code: $m = O(|C|^2)$ [ALMSS92]
 - 3-query LPCP based on quadratic span programs: $m = O(|C|)$ [GGPR13]

Linear PCPs (LPCPs) [IKO07]

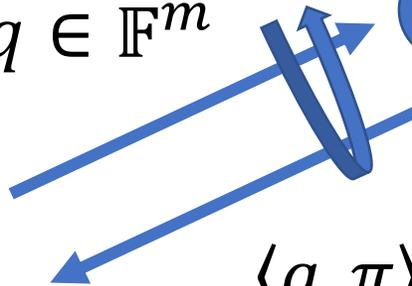
(x, w)



$\pi \in \mathbb{F}^m$

linear PCP

$q \in \mathbb{F}^m$



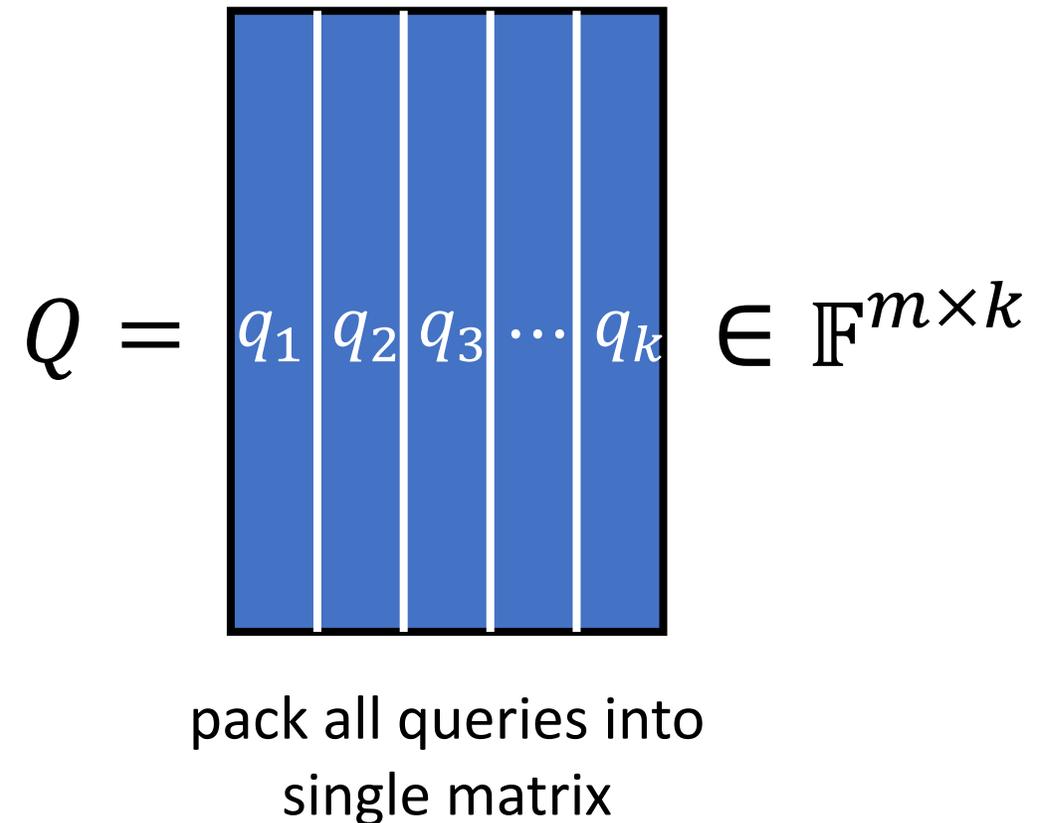
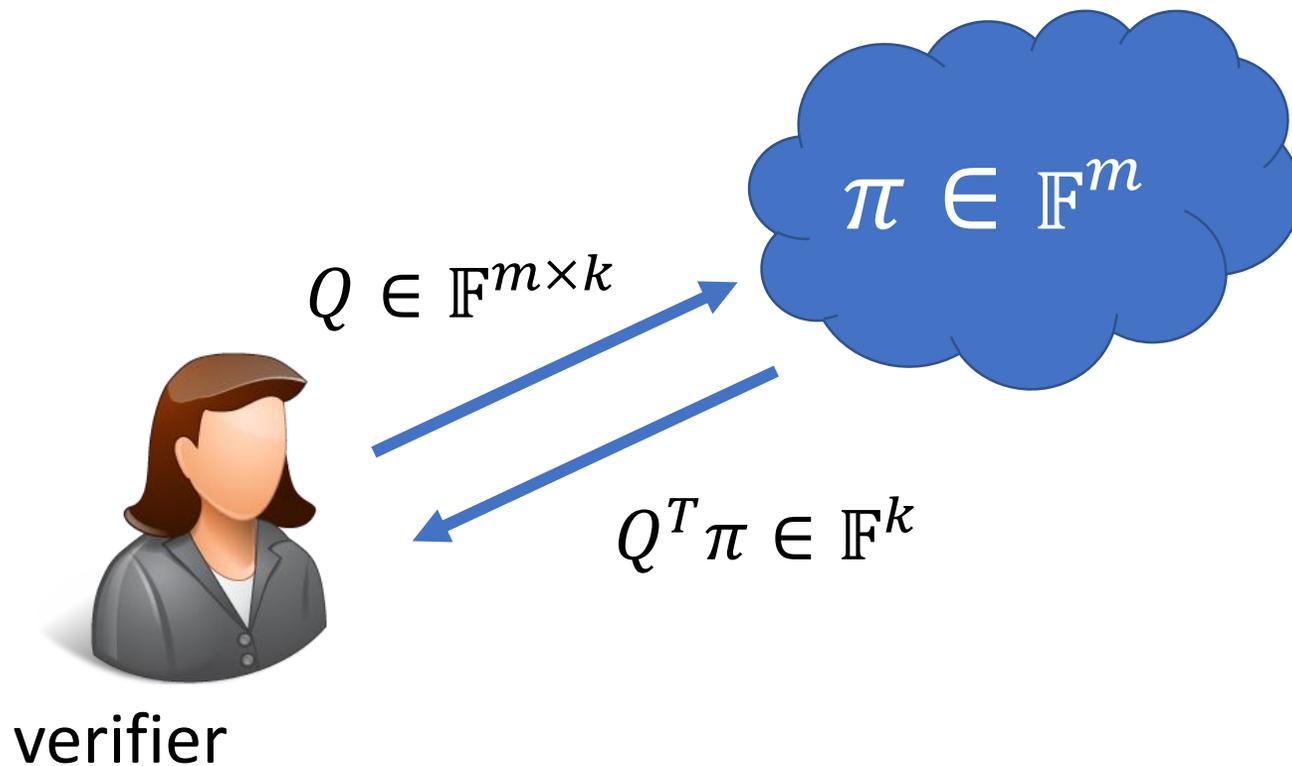
$\langle q, \pi \rangle \in \mathbb{F}$

Oftentimes, verifier is *oblivious*:
the queries q do not depend on
the statement x

verifier

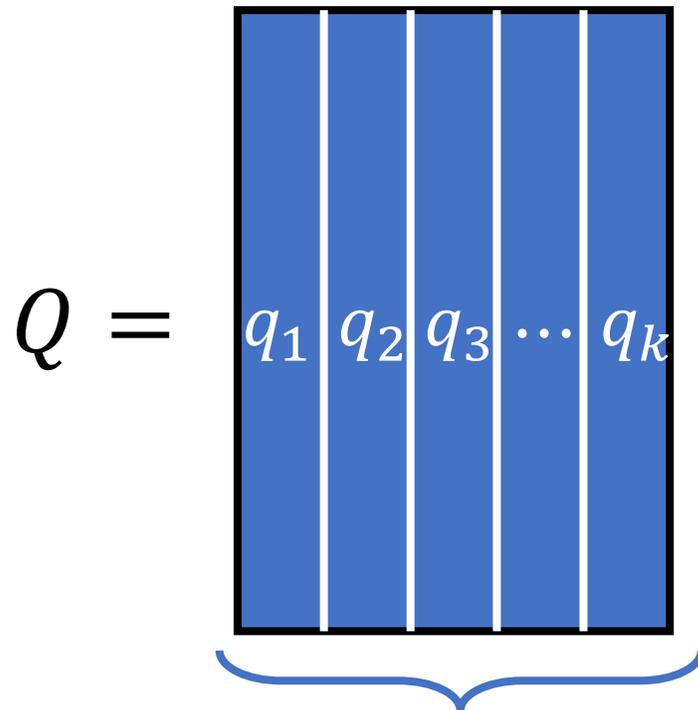
Linear PCPs (LPCPs) [IKO07]

Equivalent view (if verifier is oblivious):



From Linear PCPs to Preprocessing SNARGs [BCIOP13]

Oblivious verifier can “commit”
to its queries ahead of time



part of the CRS



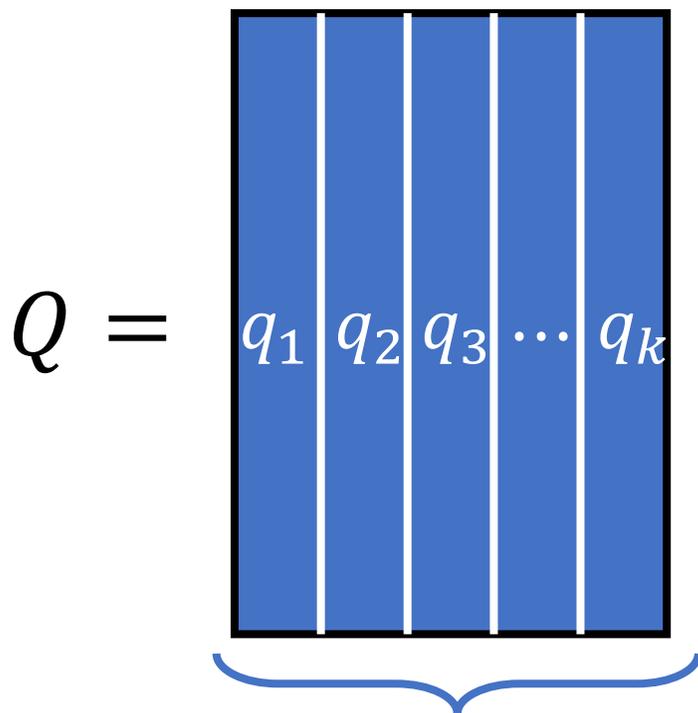
Honest prover takes
 (x, w) and constructs
linear PCP $\pi \in \mathbb{F}^m$ and
computes $Q^T \pi$

Two problems:

- Malicious prover can choose π based on queries
- Malicious prover can apply different π to the different columns of Q

From Linear PCPs to Preprocessing SNARGs [BCIOP13]

Oblivious verifier can “commit”
to its queries ahead of time



part of the CRS



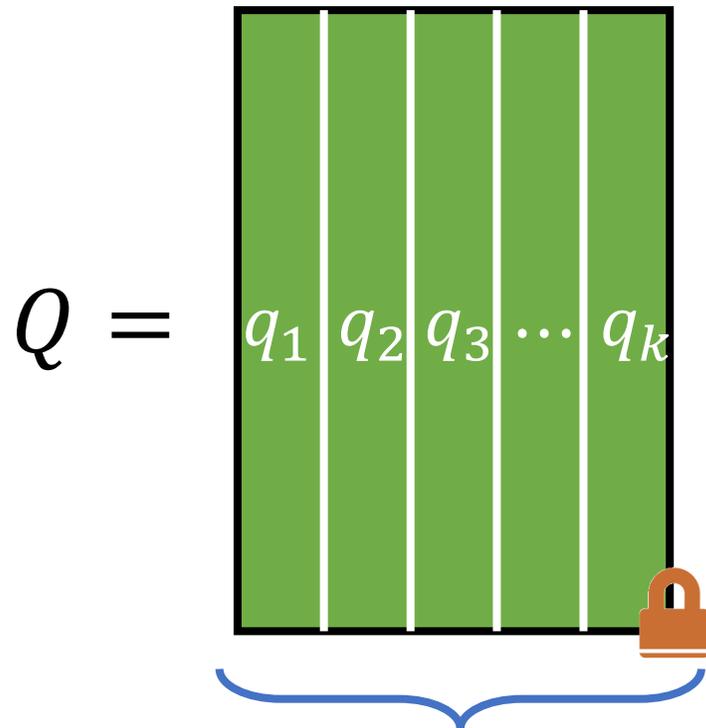
Honest prover takes
 (x, w) and constructs
linear PCP $\pi \in \mathbb{F}^m$ and
computes $Q^T \pi$

Two problems:

- Malicious prover can choose π based on queries
- Malicious prover can apply different π to the different columns of Q

From Linear PCPs to Preprocessing SNARGs [BCIOP13]

Oblivious verifier can “commit”
to its queries ahead of time



part of the CRS



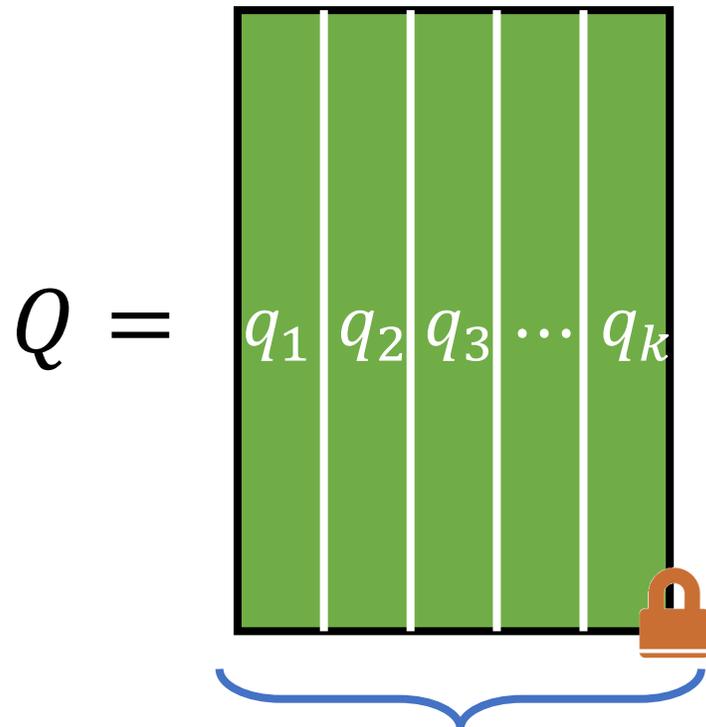
Honest prover takes
 (x, w) and constructs
linear PCP $\pi \in \mathbb{F}^m$ and
computes $Q^T \pi$

Step 1: Encrypt elements of Q using
additively homomorphic encryption scheme

- Prover homomorphically computes $Q^T \pi$
- Verifier decrypts encrypted response vector and performs LPCP verification

From Linear PCPs to Preprocessing SNARGs [BCIOP13]

Oblivious verifier can “commit”
to its queries ahead of time



part of the CRS



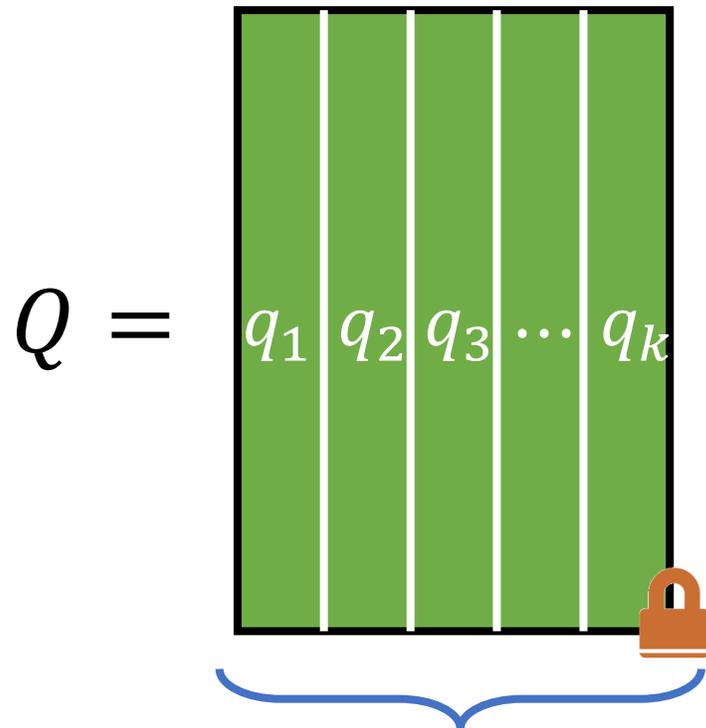
Honest prover takes
 (x, w) and constructs
linear PCP $\pi \in \mathbb{F}^m$ and
computes $Q^T \pi$

Two problems:

- Malicious prover can choose π based on queries
- Malicious prover can apply different π to the different columns of Q

From Linear PCPs to Preprocessing SNARGs

Oblivious verifier can “commit”
to its queries ahead of time



part of the CRS



Honest prover takes
 (x, w) and constructs
linear PCP $\pi \in \mathbb{F}^m$ and
computes $Q^T \pi$

Step 2: Conjecture that the encryption scheme
only supports a limited subset of homomorphic
operations (linear-only vector encryption)

Linear-Only Vector Encryption

$$v_1 \in \mathbb{F}^k$$

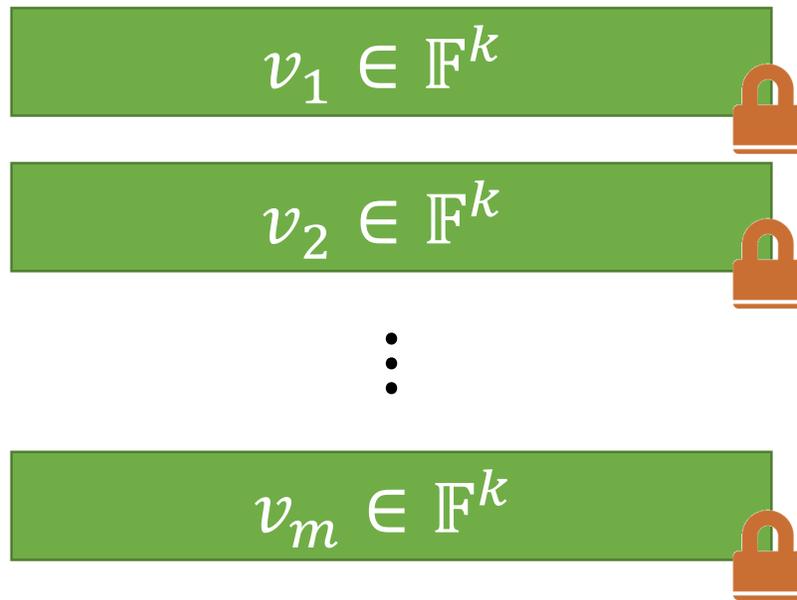
$$v_2 \in \mathbb{F}^k$$

⋮

$$v_m \in \mathbb{F}^k$$

plaintext space is a
vector space

Linear-Only Vector Encryption

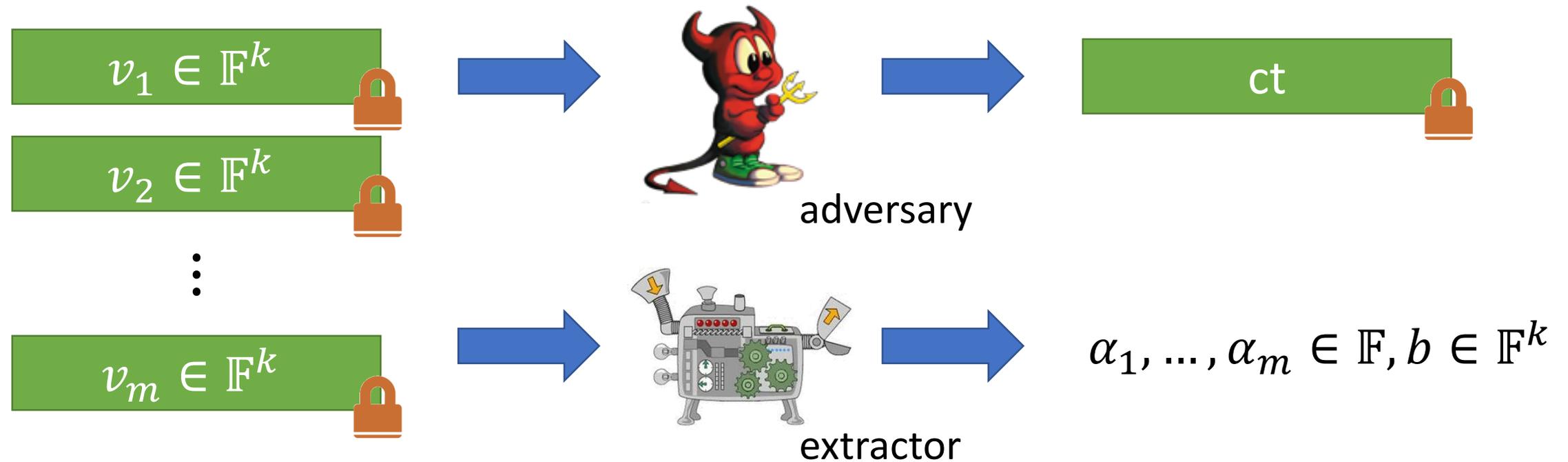


plaintext space is a
vector space



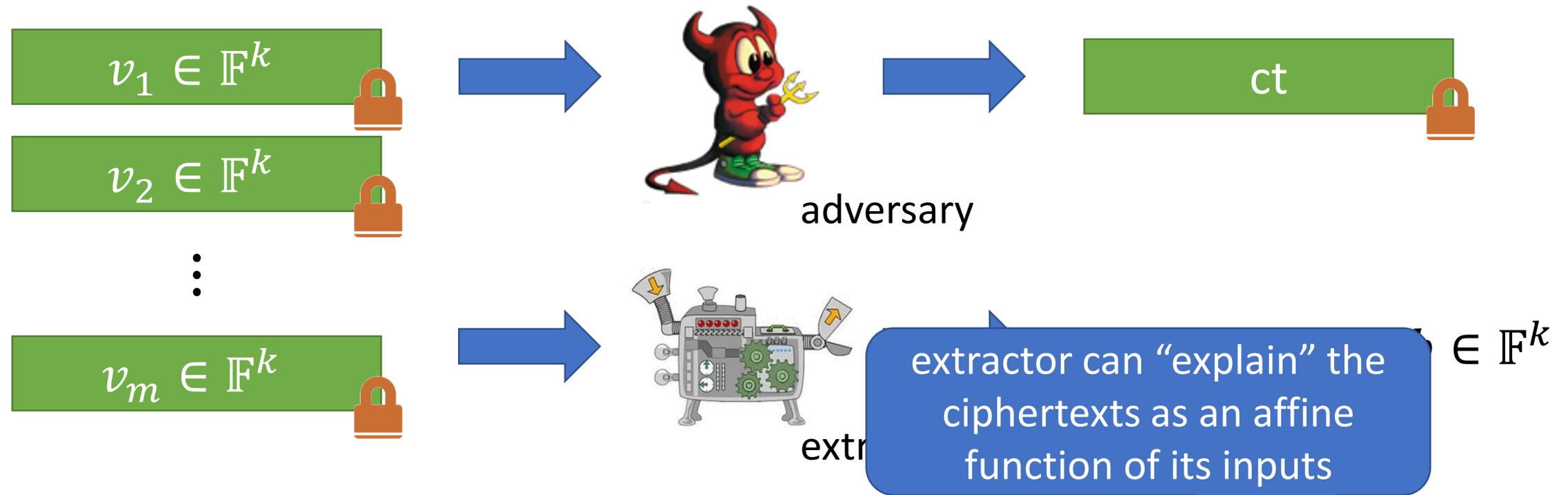
encryption scheme is
semantically-secure and
additively homomorphic

Linear-Only Vector Encryption



For all adversaries, there is an efficient extractor such that if ct is valid, then the extractor is able to produce a vector of coefficients $(\alpha_1, \dots, \alpha_m) \in \mathbb{F}^m$ and $b \in \mathbb{F}^k$ such that $\text{Decrypt}(\text{sk}, ct) = \sum_{i \in [n]} \alpha_i v_i + b$

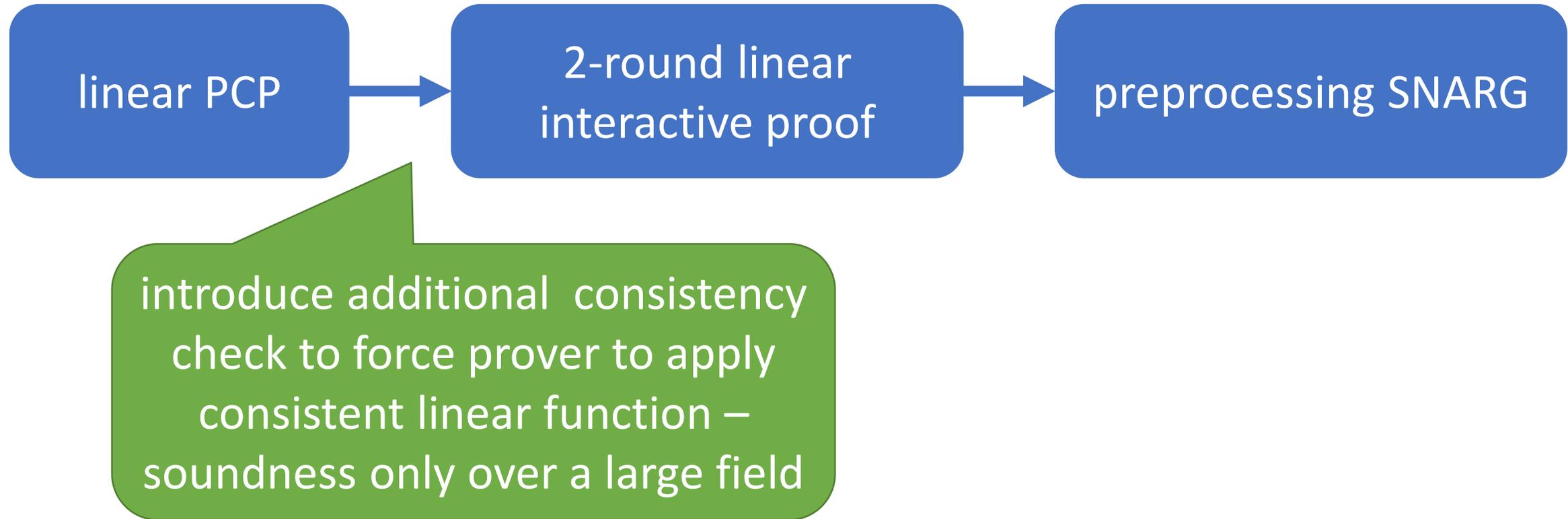
Linear-Only Vector Encryption



For all adversaries, there is an efficient extractor such that if ct is valid, then the extractor is able to produce a vector of coefficients $(\alpha_1, \dots, \alpha_m) \in \mathbb{F}^m$ and $b \in \mathbb{F}^k$ such that $\text{Decrypt}(\text{sk}, ct) = \sum_{i \in [n]} \alpha_i v_i + b$

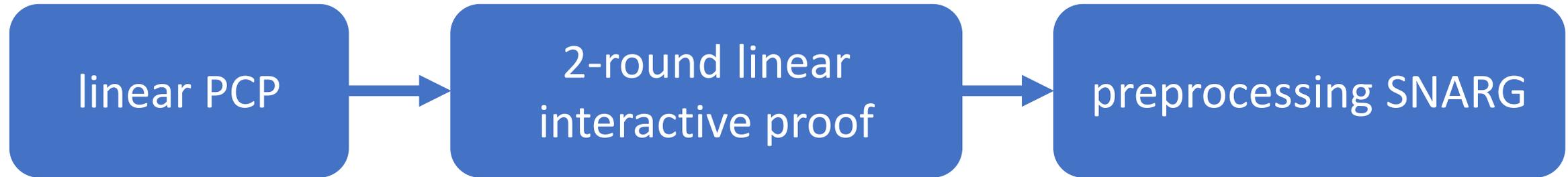
Comparison with [BCIOP13]

Preprocessing SNARGs from [BCIOP13]:

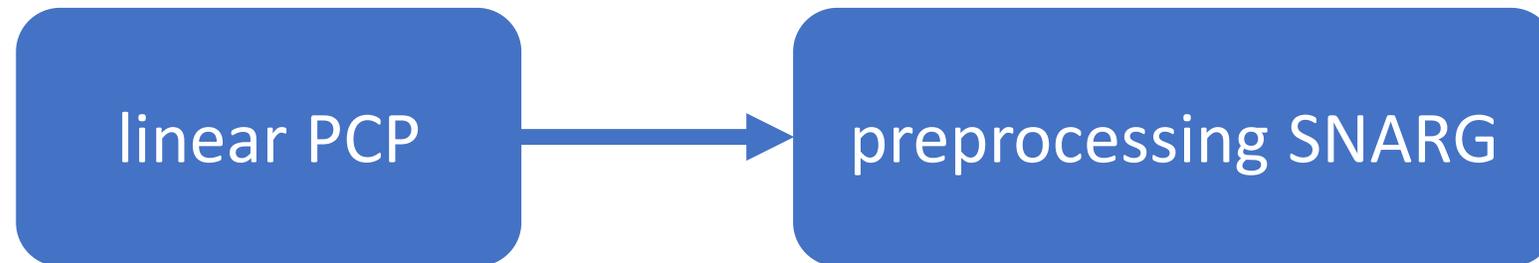


Comparison with [BCIOP13]

Preprocessing SNARGs from [BCIOP13]:

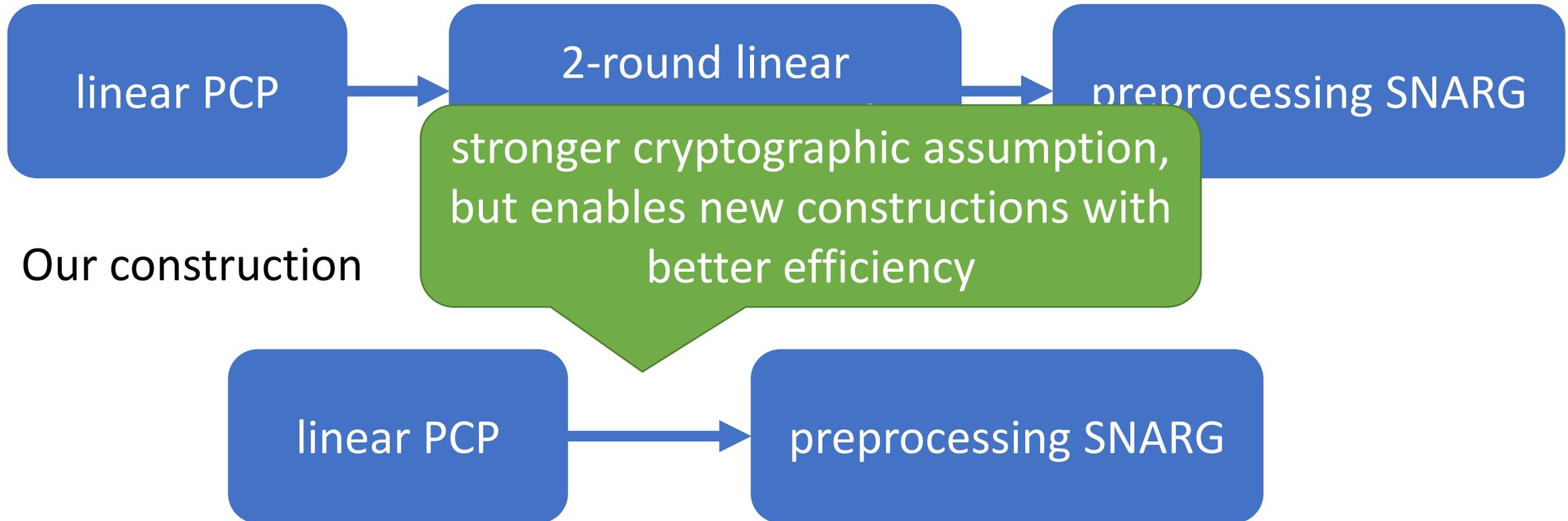


Our construction



Comparison with [BCIOP13]

Preprocessing SNARGs from [BCIOP13]:



Instantiating Linear-Only Vector Encryption

Conjecture: Regev-based encryption (specifically, the [PVW08] variant) is a linear-only vector encryption scheme.

PVW decryption (for plaintexts with dimension k):

$$\text{round} \left(\begin{array}{c} \boxed{S} \\ S \in \mathbb{Z}_q^{k \times (n+k)} \end{array} \times \begin{array}{c} \boxed{c} \\ c \in \mathbb{Z}_q^{n+k} \end{array} \right)$$

Each row of S can be is an independent Regev secret key

Concrete Comparisons

Construction	Public vs. Designated	Prover Complexity	Proof Size	Assumption
CS Proofs [Mic00]	Public	$\tilde{O}(C + \lambda^2)$	$\tilde{O}(\lambda^2)$	Random Oracle
Groth [Gro10]	Public	$\tilde{O}(C ^2\lambda + C \lambda^2)$	$\tilde{O}(\lambda)$	Knowledge of Exponent
GGPR [GGPR12]	Public	$\tilde{O}(C \lambda)$	$\tilde{O}(\lambda)$	
BCIOP (Pairing) [BCIOP13]	Public	$\tilde{O}(C \lambda)$	$\tilde{O}(\lambda)$	Linear-Only Encryption
BCIOP (LWE) [BCIOP13]	Designated	$\tilde{O}(C \lambda)$	$\tilde{O}(\lambda)$	
Our Construction (LWE)	Designated	$\tilde{O}(C \lambda)$	$\tilde{O}(\lambda)$	Linear-Only
Our Construction (RLWE)	Designated	$\tilde{O}(C)$	$\tilde{O}(\lambda)$	Vector Encryption [See paper.]

Only $\text{negl}(\lambda)$ -soundness (instead of $2^{-\lambda}$ -soundness) against 2^λ -bounded provers

Concrete Comparisons

Construction	Public vs. Designated	Prover Complexity	Proof Size	Assumption
CS Proofs [Mic00]	Public	$\tilde{O}(C + \lambda^2)$	$\tilde{O}(\lambda^2)$	Random Oracle
Groth [Gro10]	Public	$\tilde{O}(C ^2\lambda + C \lambda^2)$	$\tilde{O}(\lambda)$	Knowledge of Exponent
GGPR [GGPR12]	Public	$\tilde{O}(C \lambda)$	$\tilde{O}(\lambda)$	
BCIOP (Pairing) [BCIOP13]	Public	$\tilde{O}(C \lambda)$	$\tilde{O}(\lambda)$	Linear-Only Encryption
BCIOP (LWE) [BCIOP13]	Designated	$\tilde{O}(C \lambda)$	$\tilde{O}(\lambda)$	
Our Construction (LWE)	Designated	$\tilde{O}(C \lambda)$	$\tilde{O}(\lambda)$	Linear-Only Vector Encryption [See paper.]
Our Construction (RLWE)	Designated	$\tilde{O}(C)$	$\tilde{O}(\lambda)$	

Post-quantum resistant!

Back to Obfuscation...

For bootstrapping obfuscation...

- Obfuscate FHE decryption and SNARG verification
- Degree of multilinearity: $\approx 2^{12}$
- Number of encodings: $\approx 2^{44}$

Many optimizations. [See paper for details.]

Still infeasible, but much, much better than 2^{100} for previous black-box constructions!

Looking into obfuscation gave us new insights into constructing better SNARGs:

- More direct framework of building SNARGs from linear PCPs
- First quasi-succinct construction from standard lattices
- First quasi-optimal construction from ideal lattices [See paper.]

Open Problems

Publicly-verifiable SNARGs from lattice-based assumptions?

Stronger notion of quasi-optimality (achieve $2^{-\lambda}$ soundness rather than $\text{negl}(\lambda)$ soundness)?

Concrete efficiency of new lattice-based SNARGs?

Thank you!

<http://eprint.iacr.org/2017/240>