| 27 May 2024

Crypto Dark Matter on the Torus

Oblivious PRFs from shallow PRFs and TFHE

Martin R. Albrecht
King's College, London
SanboxAQ

Alex Davidson
NOVA LINCS & DI, FCT,
Universidade NOVA de Lisboa

Amit Deo
Zama
(work partially done while at Crypto Quantique)

Daniel Gardham
University of Surrey

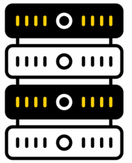
ZAMA

Agenda

- OPRFs (post-quantum)
- Non-verifiable OPRF from FHE
- Adding Verifiability

(Round Optimal) OPRFs

Input: x



Input: k

OPRF.Req

OPRF.Resp

Output: PRF_k(x)

(Round Optimal) OPRFs

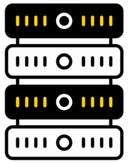
Input: x

"Client learns nothing about k beyond PRF_k(x) (**pseudorandomness**)"

Output: PRF_k(x)

OPRF.Req

OPRF.Resp



Input: k

"Server learns nothing about x from OPRF.Req (**input privacy**)"

(Round Optimal) OPRFs

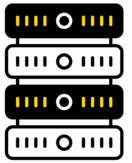
Input: x

"Client learns nothing about k beyond PRF_k(x) (**pseudorandomness**)"

Output: PRF_k(x)

OPRF.Req

OPRF.Resp



Input: k

"Server learns nothing about x from OPRF.Req (**input privacy**)"

Verifiable OPRF

Guarantee the client gets correct evaluation

Partial OPRF

Only hide <u>part</u> of client input (i.e. add a public tag)

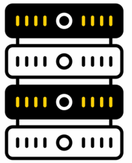
(Round Optimal) OPRFs

Input: x

"Client learns nothing about k beyond PRF_k(x) (**pseudorandomness**)"

OPRF.Req

OPRF.Resp



Input: k

"Server learns nothing about x from OPRF.Req (**input privacy**)"

Output: PRF_k(x)

Applications: OPAQUE PAKE, PSI, Privacy Pass...

Post-Quantum OPRFs

Construction	Assumptions	Rounds	Comm. cost
[ADDS21]	(RLWE & SIS)	2	~ 2MB
[SHB21]	Legendre PRF	“3”	~ secpar * 13 kB
[DGH+21]	Dark Matter PRF [BIP+18]	“2”	80B
[Bas23]	SIDH	2	3MB
[HMR23]	CSIDH	2	21KB

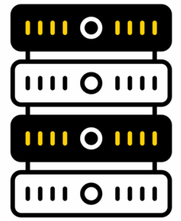
Round Optimal low (online) communication cost OPRF from FHE?

An OPRF from TFHE

A Generic FHE Solution

x

k



bsk, sk <–– FHE.KeyGen()
ct = FHE.Enc(sk, x)
pf = ZKP(bsk, ct, x)

req = (bsk, ct, pf)

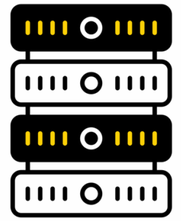
Check pf
ct' = FHE.Eval(PRF_k(), ct)
= FHE.Enc(sk, PRF_k(x))

resp = ct'

Output z = FHE.Dec(sk, ct')

A Generic FHE Solution

x



k

bsk, sk <–– FHE.KeyGen()
ct = FHE.Enc(sk, x)
pf = ZKP(bsk, ct, x)

req = (bsk, ct, pf)

Check pf

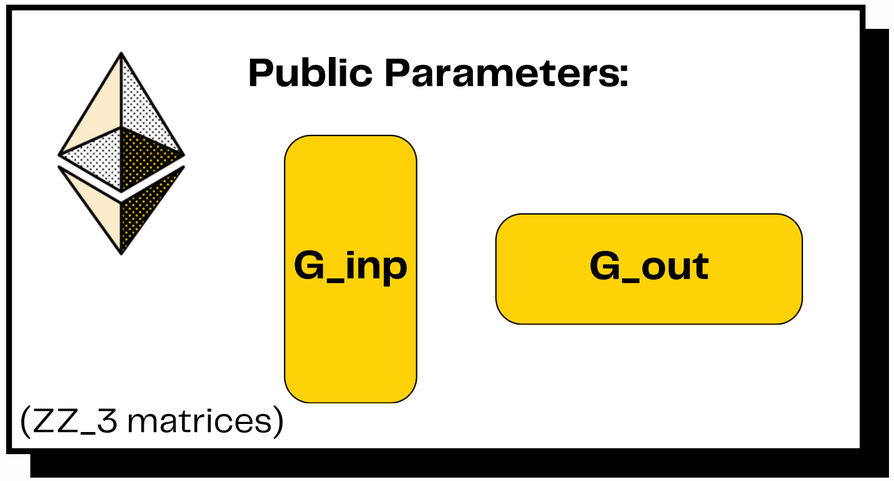
Need circuit private evaluation here [Klu22]

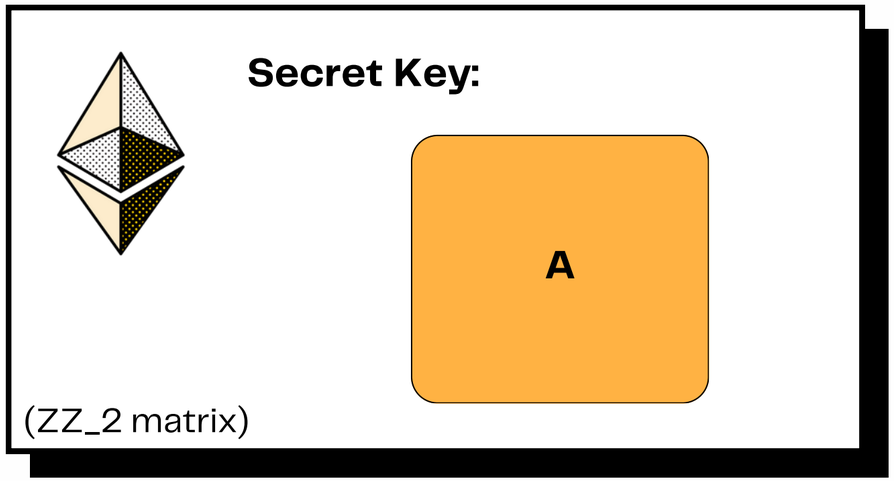
ct' = FHE.Eval(PRF_k(), ct)
= FHE.Enc(sk, PRF_k(x))

resp = ct'

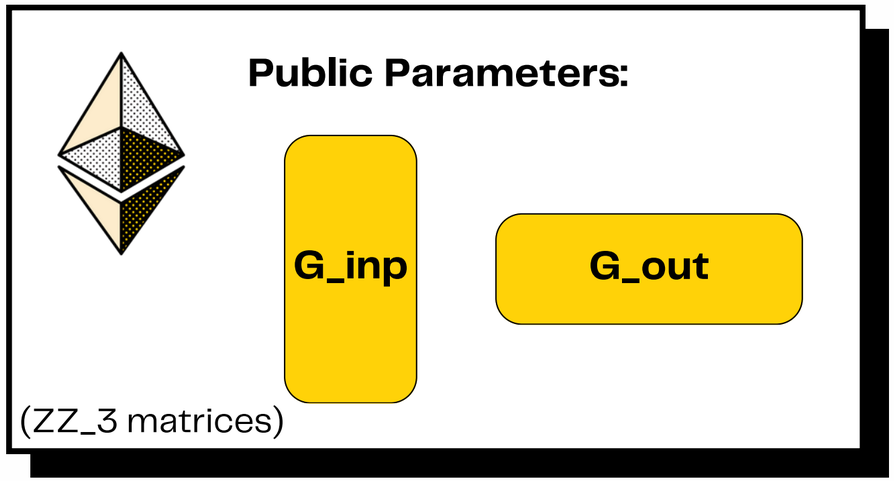
Output z = FHE.Dec(sk, ct')

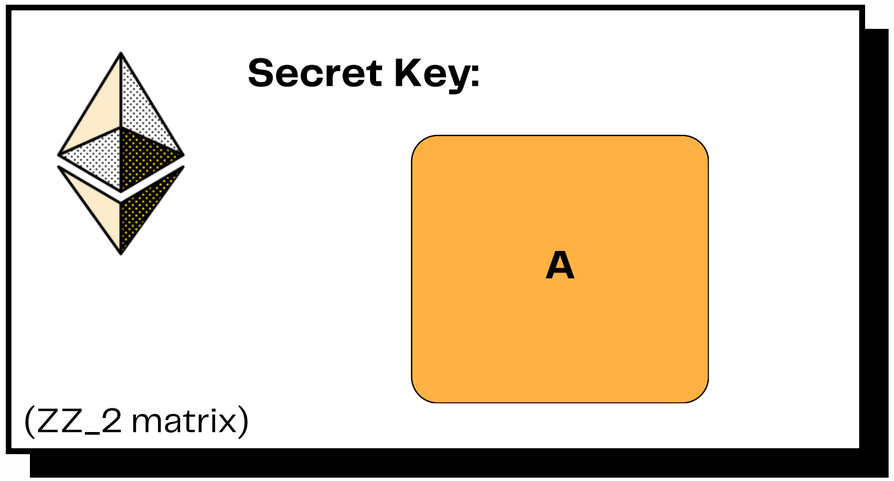
Crypto Dark Matter PRF

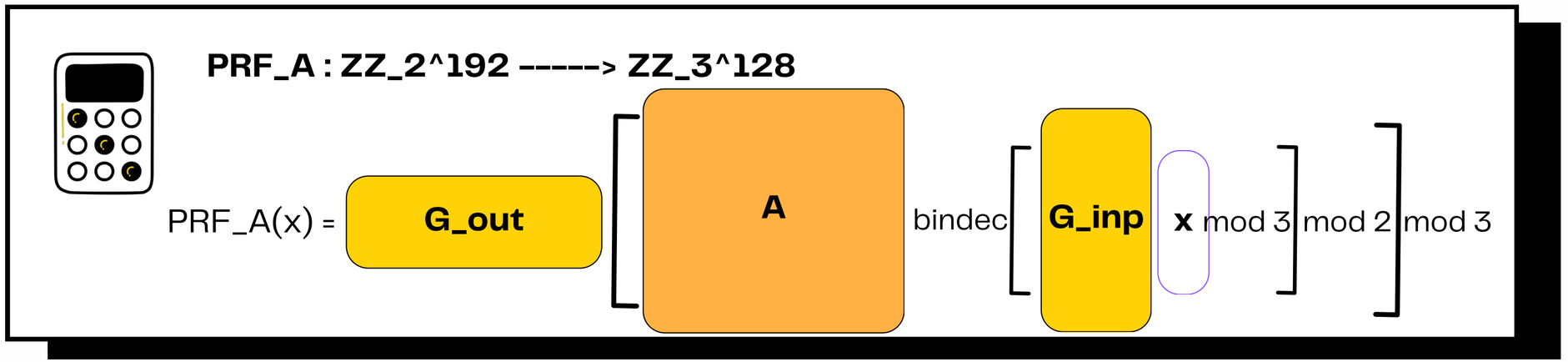




Crypto Dark Matter PRF



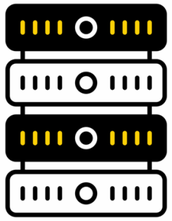




Less Work for the Server

G_inp, x

G_out, A



bsk, sk <-- FHE.KeyGen()

y = bindec(G_inp * x mod 3)

ct = FHE.Enc(sk, y)

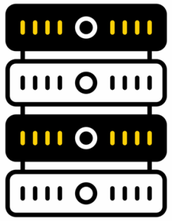
pf = ZKP(bsk, ct, y)

req = (bsk, ct, pf)

Less Work for the Server

G_inp, x

G_out, A



bsk, sk <-- FHE.KeyGen()

y = bindec(G_inp * x mod 3)

ct = FHE.Enc(sk, y)

pf = ZKP(bsk, ct, y)

req = (bsk, ct, pf)

Check pf

ct' = FHE.Eval(PRF'_A(), ct)

= FHE.Enc(sk, PRF'_A(y))

resp = ct'

Output z = FHE.Dec(sk, ct')

PRF'_A(y) := G_out * (A*y mod 2) mod 3

TFHE

in ZZ_P

in ZZ_Q^{n+1}

$$\mathrm{Enc}(sk, m) = (a, \langle a, sk \rangle + e + m * (Q/P) \bmod Q) = \mathrm{LWE}(sk, m*(Q/P))$$

(Additively homomorphic)

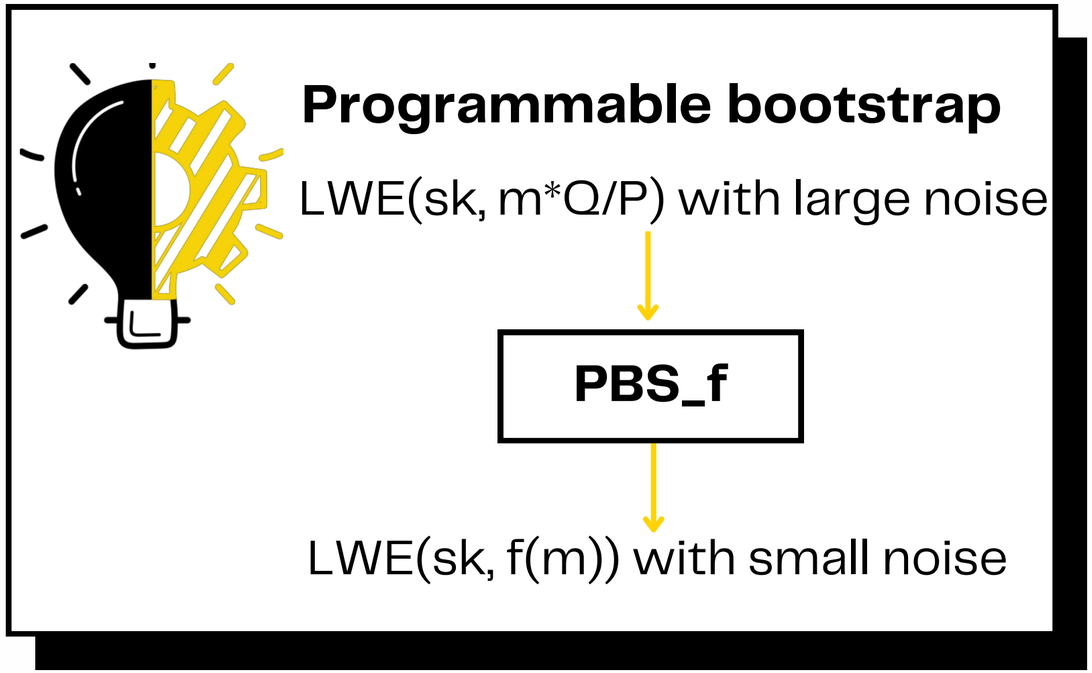
TFHE

in ZZ_P

in ZZ_Q^{n+1}

$$Enc(sk, m) = (a, < a, sk > + e + m * (Q/P) \bmod Q) = LWE(sk, m*(Q/P))$$

(Additively homomorphic)



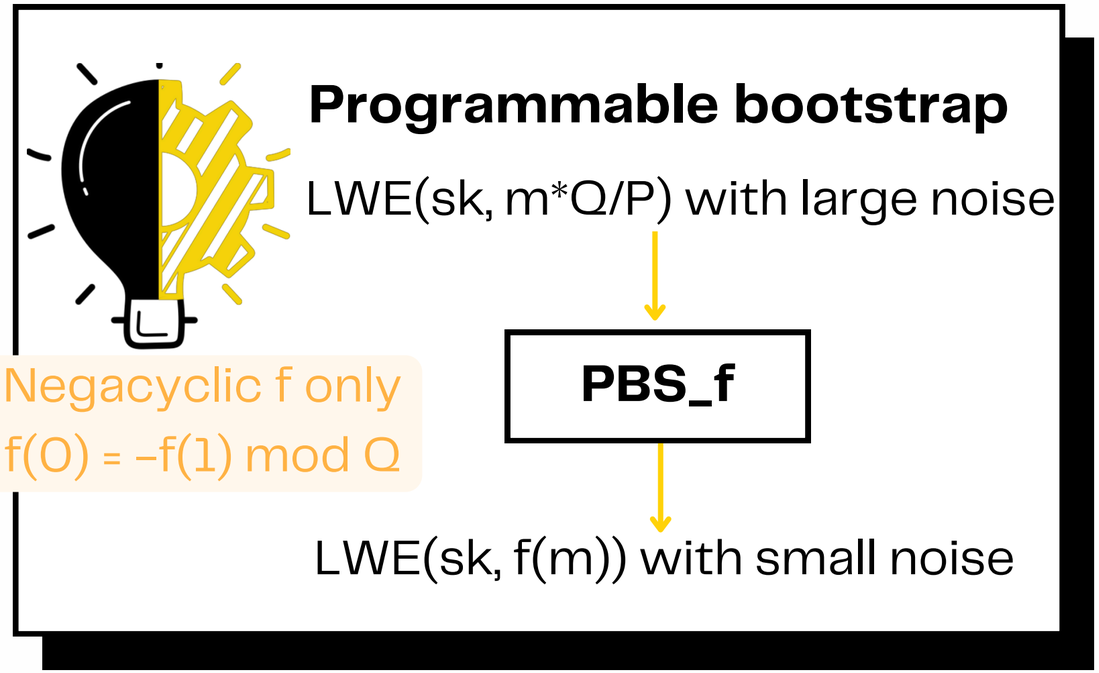
TFHE

in ZZ_P

in ZZ_Q^{n+1}

$$Enc(sk, m) = (a, < a, sk > + e + m * (Q/P) \bmod Q) = LWE(sk, m*(Q/P))$$

(Additively homomorphic)



TFHE

in ZZ_P

in ZZ_Q^{n+1}

Enc(sk, m) = (a, < a, sk > + e + m * (Q/P) mod Q) = LWE(sk, m*(Q/P)) (Additively homomorphic)

Programmable bootstrap

LWE(sk, m*Q/P) with large noise

PBS_f

LWE(sk, f(m)) with small noise

Negacyclic f only
f(0) = –f(1) mod Q

Plaintext–Space Switch

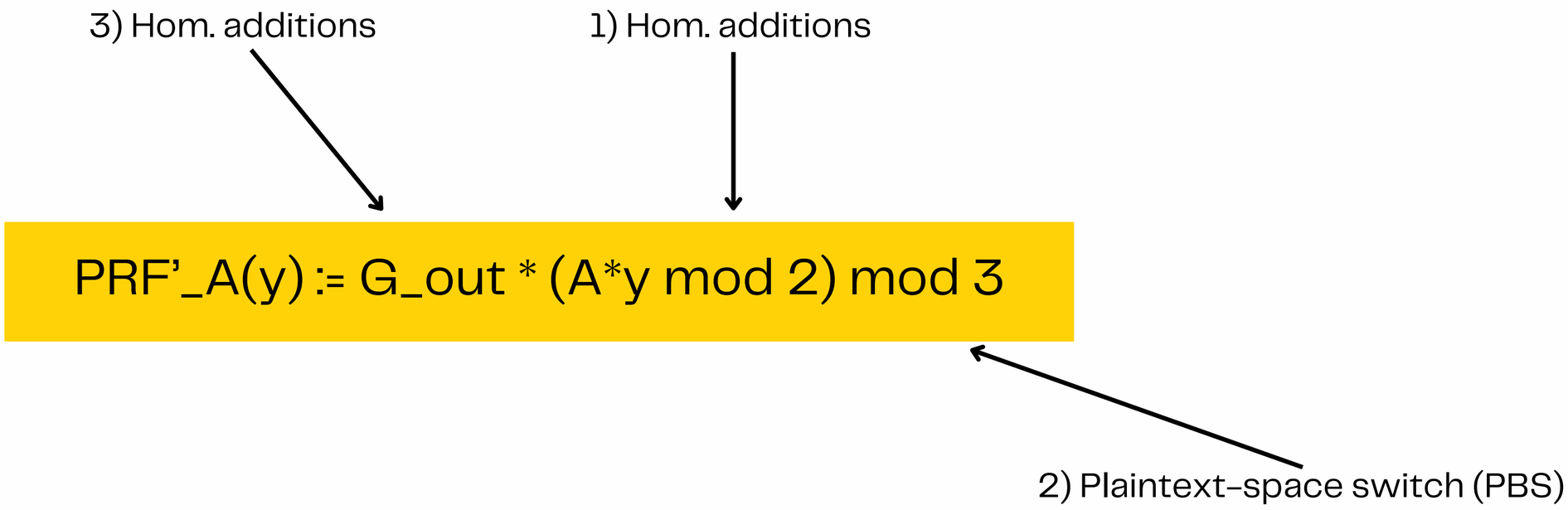
f(m) = (m+1) * Q/3
2Q/3 = –Q/3 mod Q

LWE(sk, m*Q/2)

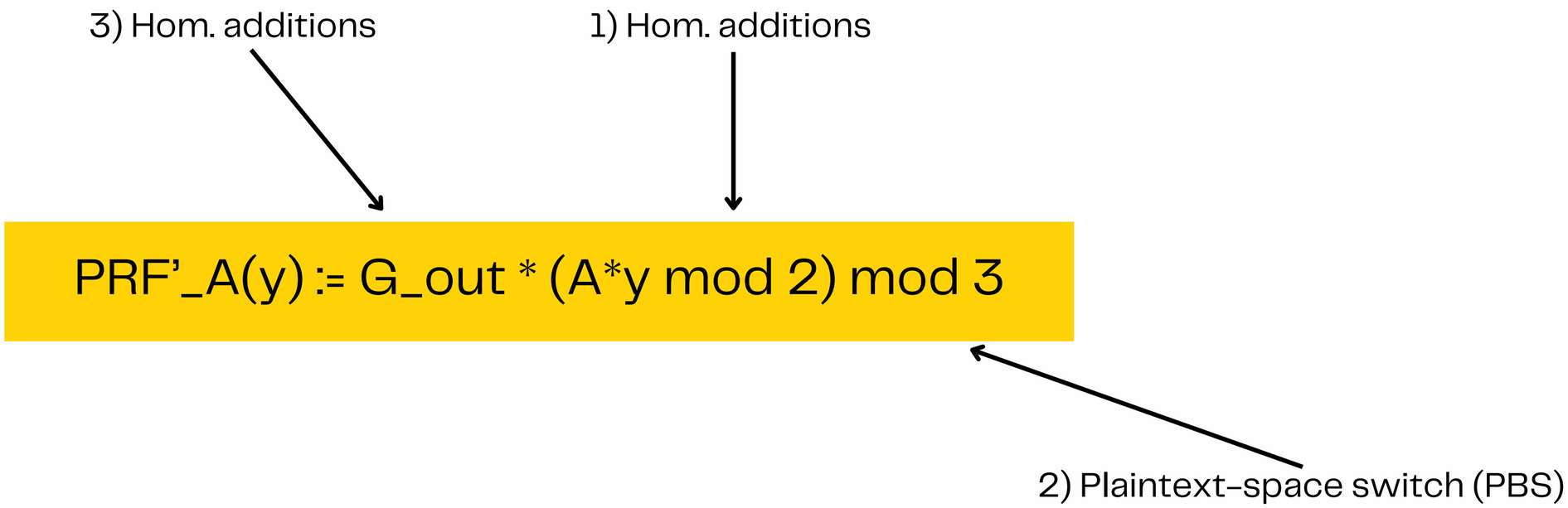
PBS_f

LWE(sk, (m+1)*Q/3) –Q/3→ LWE(sk, m*Q/3)

Depth-1 Homomorphic Evaluation



Depth-1 Homomorphic Evaluation



Circuit Privacy: Cleans noise during plaintext switch (noise indep. of A)

Need 3 | Q: Round(Q/3) * c mod Q = (c mod 3) Round(Q/3) **+ error_c**

Performance

Compression: Use seeds for randomness, drop LSB's, pack response in ring ctxt, (and amortise ZK proofs over multiple queries)

Offline bsk	Client Request	Server Response
15MB	46kB (2kB)	3.2kB
2.4MB	65kB (4kB)	6.2kB

bsk compression [KLD+23]

Performance

Compression: use seeds for randomness, drop LSB's, pack response in ring ctxt, (and amortise ZK proofs over multiple queries)

Offline bsk	Client Request	Server Response
15MB	46kB (2kB)	3.2kB
2.4MB	65kB (4kB)	6.2kB

bsk compression [KLD+23]

Rough benchmarks: client request (30ms – 1 thread), server response(150ms – 64 threads)

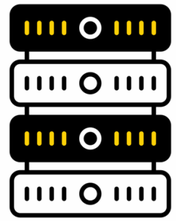
*Intel Xeon Gold 6252 CPU @ 2.10GHz cores and 768 GB of RAM

Adding Verifiability

A Simple Solution

x

k



bsk, sk <-- FHE.KeyGen()
ct = FHE.Enc(sk, x)
pf = ZKP(bsk, ct, x)

req = (bsk, ct, pf)

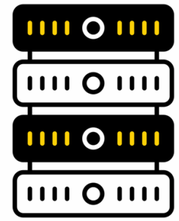
Check pf
ct' = FHE.Eval(PRF_k(), ct)

resp = ct'

Output z = FHE.Dec(sk, ct')

A Simple Solution

x



k

bsk, sk <–– FHE.KeyGen()

ct = FHE.Enc(sk, x)

pf = ZKP(bsk, ct, x)

req = (bsk, ct, pf)

Check pf

ct' = FHE.Eval(PRF_k(), ct)

Add ZKP for computation of Eval(PRF_k(), ct)

Currently very slow

resp = (ct', ZKP)

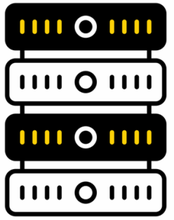
Output z = FHE.Dec(sk, ct')

Cut-and-Choose

Public Info Checkpoints: y_i, z*_i = PRF_k(y_i)) for i=1…N

x_j for j=1…M

k

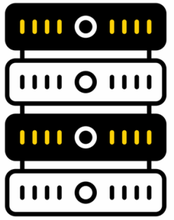


Cut-and-Choose

Public Info Checkpoints: y_i, z*_i = PRF_k(y_i)) for i=1…N

x_j for j=1…M

k



y’ = B random elements from {y_i}

Shuffle (x_1….x_1 x_2….x_2 … x_M…x_M, y’_1,…,y’_B))

⟶ **w**

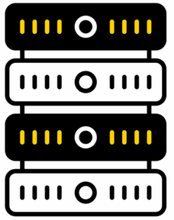
⟶ req_i = request for w_i

Cut-and-Choose

Public Info Checkpoints: y_i, z*_i = PRF_k(y_i)) for i=1…N

x_j for j=1…M

k



y’ = B random elements from {y_i}

Shuffle (x_1….x_1 x_2….x_2 … x_M…x_M, y’_1,…,y’_B))

⟶ **w**

⟶ req_i = request for w_i

(req_i) for i=1…Mv+N ⟶

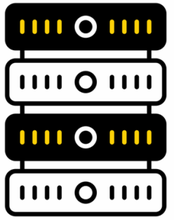
⟵ (resp_i) for i=1…Mv+N

resp_i = response for req_i

Cut-and-Choose

Public Info Checkpoints: y_i, z*_i = PRF_k(y_i)) for i=1…N

x_j for j=1…M

k 

y’ = B random elements from {y_i}

Shuffle (x_1….x_1 x_2….x_2 … x_M…x_M, y’_1,…,y’_B))

⟶ **w**

⟶ req_i = request for w_i

(req_i) for i=1…Mv+N ⟶

⟵ (resp_i) for i=1…Mv+N

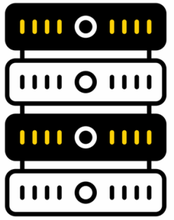
resp_i = response for req_i

1. Compute outputs z_i
2. Check z_i against public info and repeated requests

Cut-and-Choose

Public Info Checkpoints: y_i, z*_i = PRF_k(y_i)) for i=1…N

x_j for j=1…M

k 

y' = B random elements from {y_i}

Shuffle (x_1….x_1 x_2….x_2 … x_M…x_M, y'_1,…,y'_B))

⟶ **w**

⟶ req_i = request for w_i

(req_i) for i=1…Mv+N ⟶

⟵ (resp_i) for i=1…Mv+N

resp_i = response for req_i

1. Compute outputs z_i
2. Check z_i against public info and repeated requests

Problem: server can do FHE-based homo. equality tests to craft correct sequence

Homomorphic Cheating Circuits

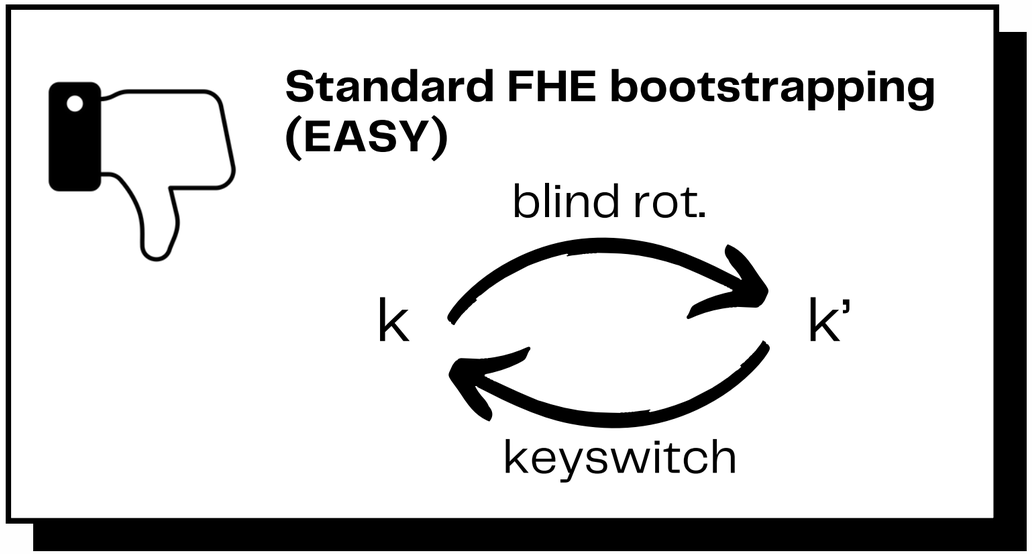
“ If w_i = y_j: Output z*_j ”

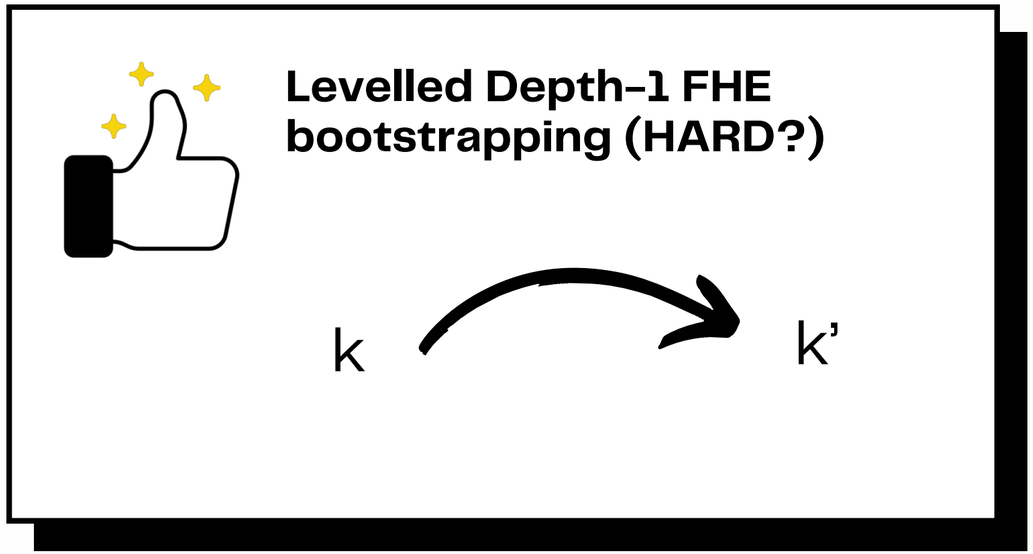
Task: homomorphically evaluate given Enc(sk, w_i)

Homomorphic Cheating Circuits

" If w_i = y_j: Output z*_j "

Task: homomorphically evaluate given Enc(sk, w_i)

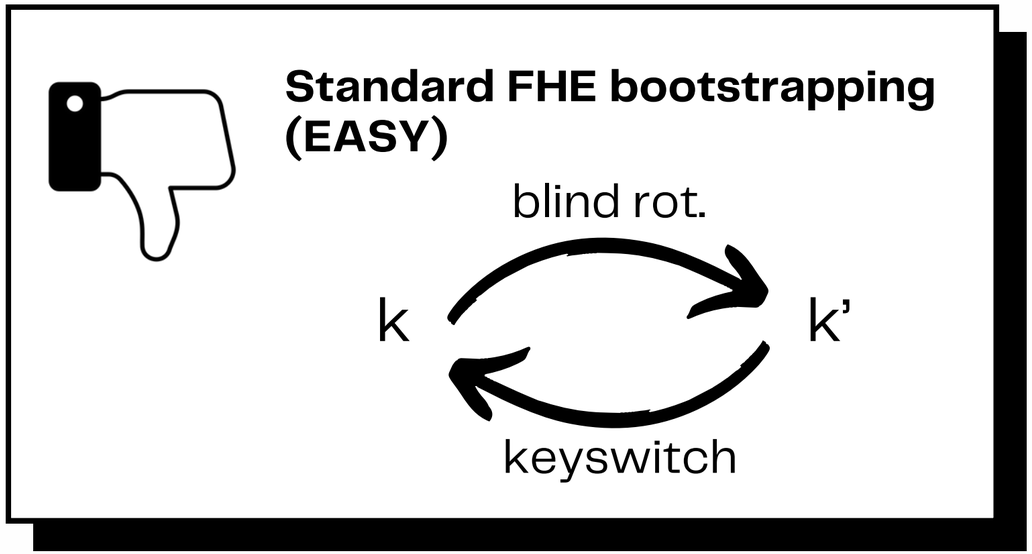


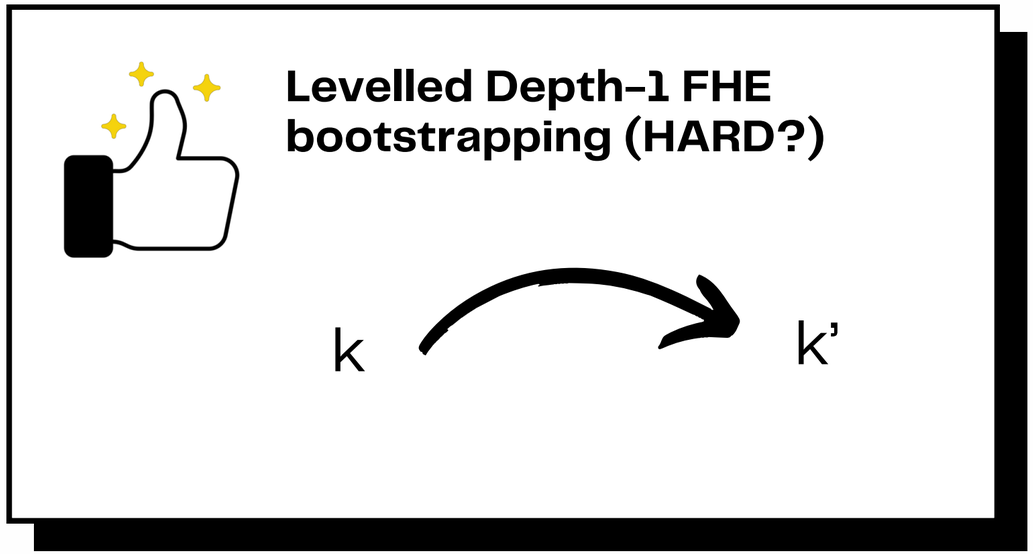


Homomorphic Cheating Circuits

" If w_i = y_j: Output z*_j "

Task: homomorphically evaluate given Enc(sk, w_i)





Assumption: There is no depth-1 cheating circuit

Performance

Client Evaluation: 105 evaluations, 11 repeats
Checkpoints: 128^2 checkpoints, 10 randomly selected for client query

Offline bsk+Checkpoints	Client Request	Server Response
15MB + 256kB	11 * 46kB (11 * 2kB)	11 * 3.2kB
2.4MB + 256kB	11 * 65kB (11 * 4kB)	11 * 6.2kB

Conclusion

- **Construction:** TFHE + Crypto Dark Matter PRF (depth-l bootstrapping)
- **Online cost:** A few kBs per query and $<l$s evaluation
- **Offline cost:** Bootstrapping key (MBs)
- **Added verifiability:** Cut-and-choose (depth-l eval $\dashrightarrow$ security)

Thank you.

ZAMA
amit.deo@zama.ai