The Red Wedding

Playing Attacker in MPC Ceremonies

RWC 2021

Bernardo David (*IT Uni. of Copenhagen*),
Justin Drake (*Ethereum Foundation*),
Dmitry Khovratovich (*Ethereum Foundation*),
Mary Maller (*Ethereum Foundation*),
Hart Montgomery (*Fujitsu US*),
Claudio Orlandi (*Aarhus Uni.*),
Peter Scholl (*Aarhus Uni.*),
Omer Shlomovits (*ZenGo-X*),
Riad Wahby (*Stanford Uni.*)

Motivation for this talk

Diogenes: Lightweight Scalable RSA Modulus Generation with a Dishonest Majority

Megan Chen
Northeastern U.
& Ligero Inc.

Carmit Hazay
Bar-Ilan U.
& Ligero Inc.

Yuval Ishai
Technion

Yuriy Kashnikov
Ligero Inc.

Daniele Micciancio
UC San Diego

Tarik Riviere
Ligero Inc.

abhi shelat
Northeastern U.
& Ligero Inc.

Muthu Venkitasubramaniam
U. of Rochester
& Ligero Inc.

Ruihan Wang
Ligero Inc.

Language	files	blank	comment	code
C/C++ Header	117	5780	6721	29230
C++	151	2296	1313	12512

This is us :)

- 9 reviewers, 4 months
  - Work as a team
  - Weekly with authors
  - Protocol soundness
  - Code soundness

- **Building crypto is hard!**
- **Reviewing crypto is hard!**

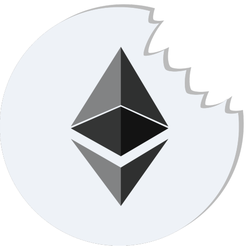
Outline

- Introduction {joint RSA modulus, protocol blue print}
- Diogenes protocol {requirements, highlights, how it works}
- The review process {tools, spec, pipeline, outputs}
- DogByte attack {ZK optimisation, packed RLWE}
- **Insights**

Takeaways

Easier said than done

- Every optimisation requires a proof of security (dogbyte attack)



- Encourage dialog between cryptographers and developers (spec)
- Active coordinator model - must be carefully defined (octopus attacks)

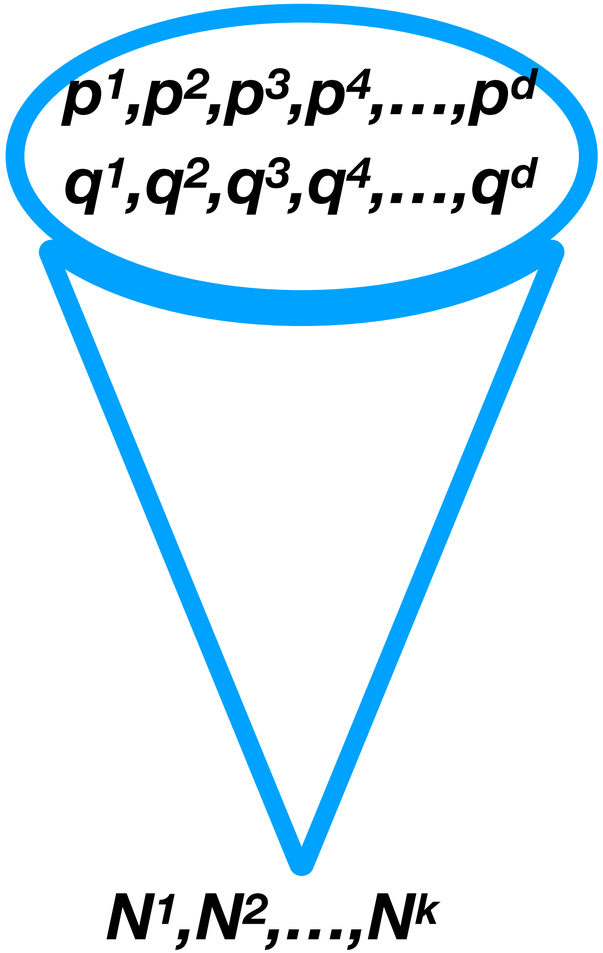
- Code design: Modularity, Re-use crypto/code (note)

Introduction

- Threshold RSA KeyGen is a well studied problem
- RSA modulus : ***N = p*q*** such that ***p,q*** are (strong) primes
- Problem: Jointly compute RSA key
- Use cases:
  - Building block in secure computation protocols (DN03)
  - Group of unknown order -> good for setting up public parameters (e.g. VDF)
  - Examples: **strong randomness** (Drake18) threshold ecdsa (GGN16)

Introduction

Protocol blueprint



- Inputs : secret randomness
- Key setup (***sk, pk***)
- Candidate Generation (***p = p1 + p2 + …***)
- Partial primality test
- Compute product (***N = pq***)
- Biprimality test
- Output: ***N***

Requirements

- Large scale (~1000 parties)
- Minimised number of rounds (<20)
- Secure against dishonest majority
- Secure against active adversary
- Identifiable abort
- Running time < 20 min

Diogenes

Crypto magic

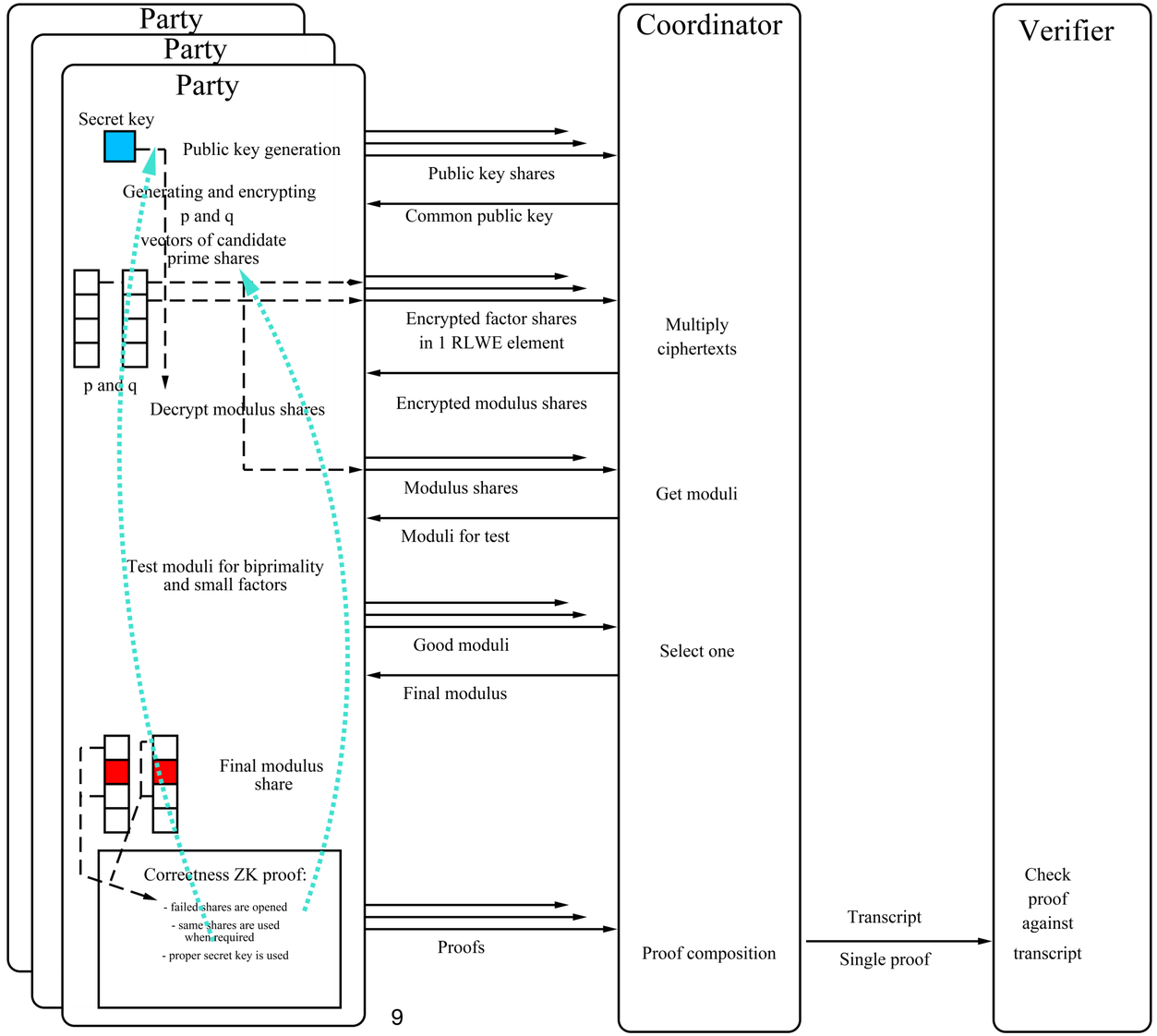
MAY 23-27, 2021

42nd IEEE Symposium on Security and Privacy

- Large scale (~1000 parties) → Semi trusted Coordinator model
- Minimised number of rounds (<20) → 12 rounds
- Secure against dishonest majority → Threshold AHE (Ring LWE)
- Secure against active adversary → Proof of honesty at the end (GMW)
- Identifiable abort → Publicly verifiable transcript
- Running time < 20 min → Use non interactive proofs
- UC Secure

How it works

- Inputs : secret randomness
- Key setup
- Candidate Generation
- Compute product (***N = pq***)
- Biprimality test
- Output: ***N***



Review process

Tools

- Specifications document (paper -> code)
- Internal specifications document (code -> paper)
- LWE estimator , Homomorphic Encryption Standard
- Cloud
- GitHub

Specification Doc (Internal)

- Code flow
- Protocols flow
- Parameters
- Notations

Code flow

Client

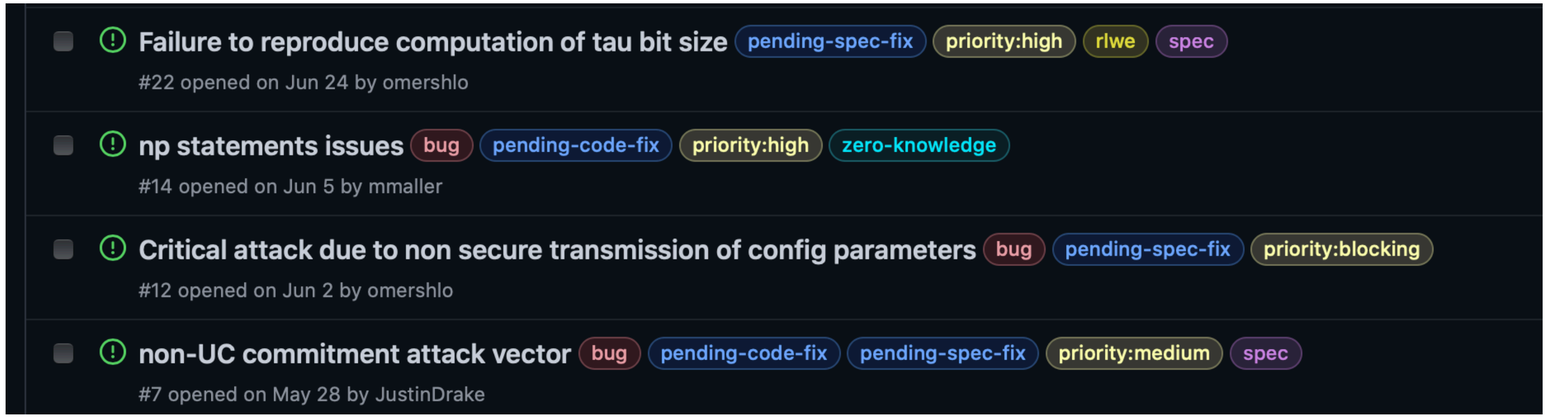
main->participate->participateHelper->start:

- startHelper
  - registerAndAwaitConfiguration
  - start_rsa_ceremony
    - generateKeyPair
    - generatePreSievingShares
      - pruneAndReorderShares
      - crt_reconstruct

Client P_i	**Coordinator**
Generate gaussian e, s with parameter $\chi = 8$	
Generate (private,public) key pair for signatures	Generate (private,public) key pair for signatures and sto key in $config$
Generate array of certain 19 primes $\alpha_{GCD}[]$ using an RNG seeded with 0	
Generate array of random 2176-bit numbers $\sigma_r[128]$	
Set an array $\sigma_{r,GCD}[128 \cdot 19l + 19j + k] = (\sigma_r[j] \bmod \alpha_{GCD}[k]) \bmod \mathcal{P}[l], l \leq 9$.	
Record $e_i, s_i, \sigma_{r,GCD}[]$ to secretData	
Prepare a Ligero proof $\widehat{\mathcal{P}}$ of knowledge of secretData filled so far	
Generate random $a \in R_Q$ and FFT it	
Generate Jacobi seed shares $seed_1, seed_2$	
$\rightarrow$ (ID_PARTY, IP address,$\widehat{\mathcal{P}}$, commitments $H(a), H(seed_1), H(seed_2)$)	Store commitments
	$\Leftarrow$ (PROTOCOL_CONFIG,party numbers $P[i]$, config)

Review process

Pipeline



Internal discussion

Write a note

Report to authors

Tagged and tracked in GitHub issue tracker

Addressed by authors within 2 weeks

Update to paper/code/spec

Outputs

Rounds of revision:	
Paper	4
Code	2
Authors Spec	7

DogByte

DogByte
ZK optimization

- Delay all ZK proofs until the last step of the protocol
- Produce a ZK proof only for the **k** bi-prime candidates that survived all the tests.
- For the rest **d-k** of the candidates: each party will simply reveal all the secret randomness used to derive the “bad” candidate. The check for correctness of computation will be carried out over the opened values.

Dog Byte cont.
Packed RLWE encryption

- public key is a pair ***(a,b)***
- private key is a secret ***s*** such that: ***b = a·s + e***
- To encrypt a message ***m***:
  - Encode it as one coefficient of a polynomial
  - Sample small noise polynomials ***x, y, z***
  - Compute $c_1 = a \cdot x + y$
  - Compute $c_2 = b \cdot x + z + m$
  - $C = (c_1, c_2)$ is the ciphertext.

Dog Byte cont.

Packed RLWE encryption

- public key is a pair ***(a,b)***
- private key is a secret ***s*** such that: $b = a \cdot s + e$
- To encrypt a message ***m***:
  - Encode it as one coefficient of a polynomial
  - Sample small noise polynomials ***x, y, z***
  - Compute $c_1 = a \cdot x + y$
  - Compute $c_2 = b \cdot x + z + m$
  - $C = (c_1, c_2)$ is the ciphertext.

- The encryption scheme is wasteful
- Optimise by encoding different messages to different slots (coefficients)
- In the protocol: Each participant encrypts its private input, and *all* encryptions are packed to a single ciphertext.

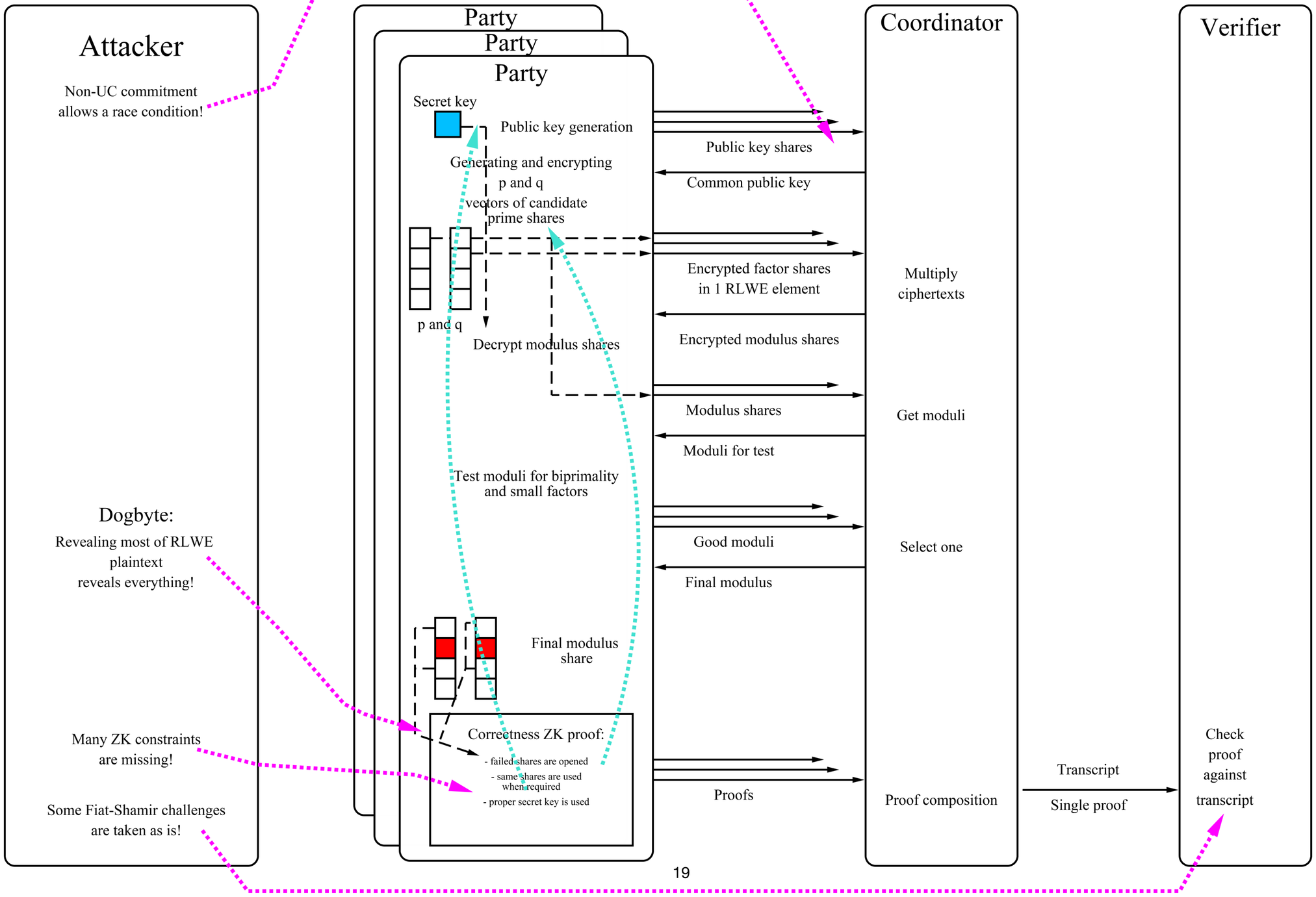
Dog Byte cont.

The attack

- We show how combining the ZK optimization with the packed ciphertext optimization allows a passive attacker to learn ***p,q***
- ***a,b,x,y,z,m*** are all vectors of size **d**
- Prover reveals **d-k** coefficients for each ***y,z,m***
- Attacker: find ***x*** —> find ***m***
- Condition to succeed : **k < d/2**
- Toy example

$$c_1 = a \cdot x + y$$

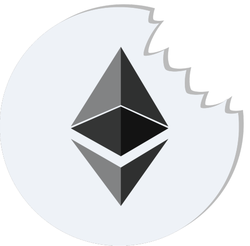
$$c_2 = b \cdot x + z + m$$


Attacker
Non-UC commitment allows a race condition!
Dogbyte:
Revealing most of RLWE plaintext reveals everything!
Many ZK constraints are missing!
Some Fiat-Shamir challenges are taken as is!
Party
Party
Party
Secret key
Public key generation
Generating and encrypting p and q vectors of candidate prime shares
p and q
Decrypt modulus shares
Test moduli for biprimality and small factors
Final modulus share
Correctness ZK proof:
- failed shares are opened
- same shares are used when required
- proper secret key is used
Coordinator
Public key shares
Common public key
Encrypted factor shares in 1 RLWE element
Multiply ciphertexts
Encrypted modulus shares
Modulus shares
Get moduli
Moduli for test
Good moduli
Select one
Final modulus
Proofs
Proof composition
Transcript
Single proof
Verifier
Check proof against transcript

Insights

Easier said than done

- Every optimisation requires a proof of security (dogbyte attack)



- Encourage dialog between cryptographers and developers (spec)
- Active coordinator model - must be carefully defined (octopus attacks)

- Code design: Modularity,Re-use crypto/code (note)

@OmerShlomovits, omer@ZenGo.com