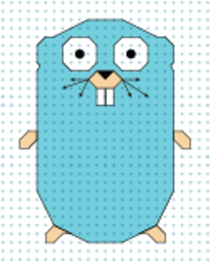
EPFL



LATTIGO

A multiparty homomorphic encryption library in Go

Demo @ WAHC 2020

Christian Mouchet, Jean-Philippe Bossuat, Juan Troncoso-Pastoriza, Jean-Pierre Hubaux
Laboratory for Data Security
EPFL

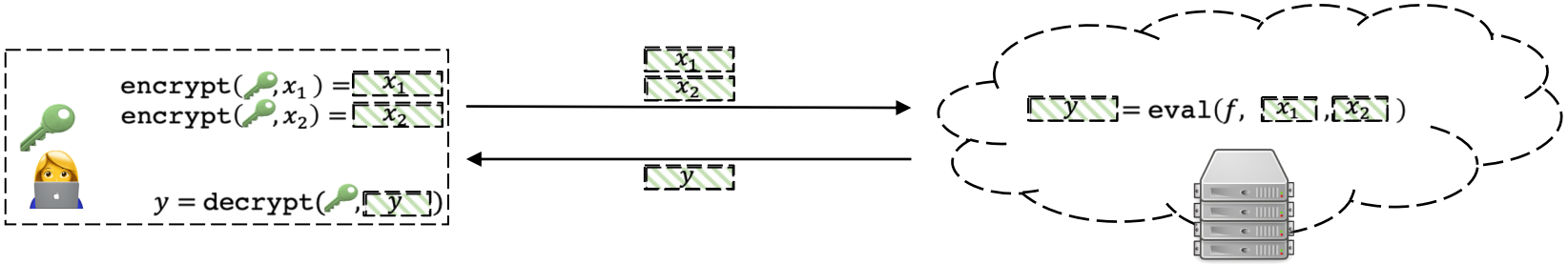
Dec. 15, 2020

Agenda

1. Homomorphic Encryption & Multiparty Homomorphic Encryption
2. The Lattigo Library
    a. Purpose
    b. Components
    c. Performance
3. Demo: a private web-application for scheduling meetings
4. Applications using Lattigo

Single-party HE setting

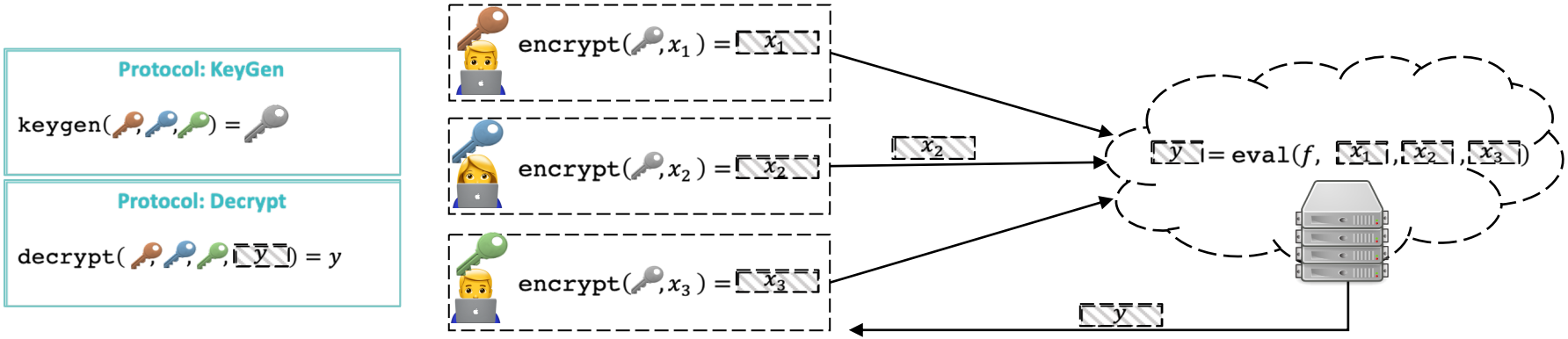
"Computing on data without seeing them"



In the passive-adversary model,
Single-party HE can achieve **secure two-party computation**: Can this be extended to N-parties?

Multiparty HE setting

"Computing on multiparty data without seeing them using cryptographic protocols"

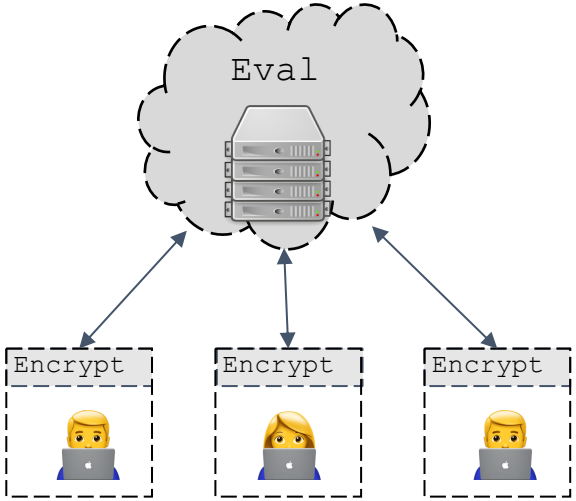


- "Early" semi-homomorphic constructions [FH 96, CDB 01]
- (R-)LWE-based instantiations in the
  - *Threshold model* [AJLTVW 12, MTBH 20] and
  - *Multi-key model* [LTV 12, CDKS 19, CCS 19]

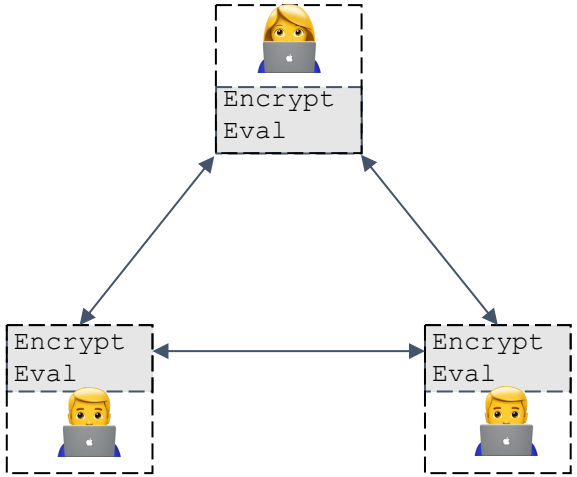
  } "out-of-the-box": secure multiparty function evaluation in the passive-adversary model

Variety of system models

Cloud-based MPC



Peer-to-peer MPC



- Networked
- Concurrent
- Cross-platform

This is where the Go language shines!

The Go language

Go is a serious candidate for implementing (M)HE-based systems

Pros:

- Designed for concurrent and networked systems
- Extremely simple syntax
- Extensive toolchain (dependency management, unit-testing, benchmarking…)
- Possible to call C/C++ code (`cgo`)
- Can be called from C/C++ (compilation in a `.so` library)

Cons:

- Compiler not as mature as “older” languages
- Can discourage newcomers to the language

The library

Purpose

- Allows the materialization and open-sourcing of FHE-based system research
- Brings FHE to a new language: Go

Licence: Apache 2.0

Versions

- v1.0.0 : 08/2019 first public-release
- v1.1.0 : 10/2019 improved API design and documentation (godoc)
- v1.3.0 : 12/2019 hybrid key-switching algorithm [HK 19]
- v2.0.0 : 10/2020 full-RNS bootstrapping [BMTH 20] & Wasm builds
- v2.1.0 : 12/2020 current version

The library – scope and structure

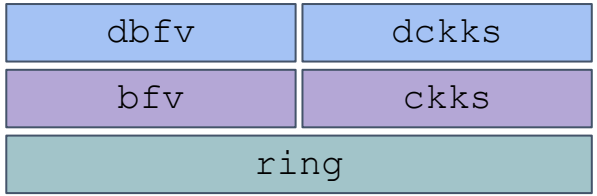
- Lattigo is a Go **module** (https://blog.golang.org/using-go-modules)
- Implements several **packages**:
  - A math layer for polynomial ring operations
  - A cryptographic layer for two HE schemes:

    BFV [FV 12]
    vectorized integer arithmetic

    CKKS [CKKS 17]
    vectorized complex arithmetic
  - The local operations of the multiparty variants [MTBH 20]

    DBFV

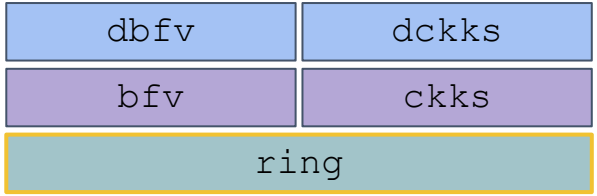
    DCKKS



- Does **not** implement
  - Network-layer functionality
  - Multi-core processing

  } ongoing implementation projects in separate code-bases

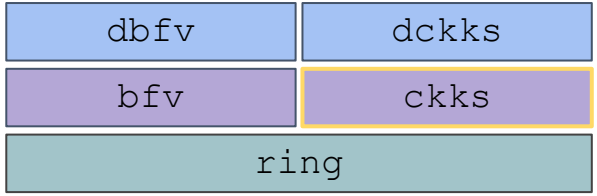
The `lattigo/ring` package



Math layer of the Lattigo library :

- Arithmetic on power-of-two degree polynomial rings with coefficients in the RNS domain
- Full-RNS arithmetic (avoids multi-precision arithmetic as much as possible)
- Number-Theoretic-Transform for efficient nega-cyclic convolution
- Sampling (Gaussian, ternary, uniform)

The `lattigo/ckks` package

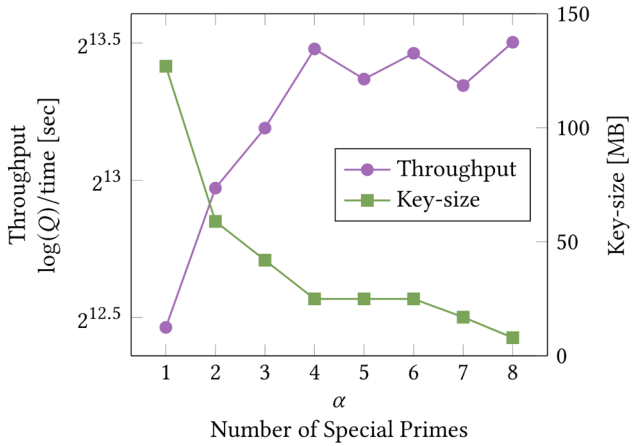


Implements the full-RNS CKKS scheme [CHKKS 18]

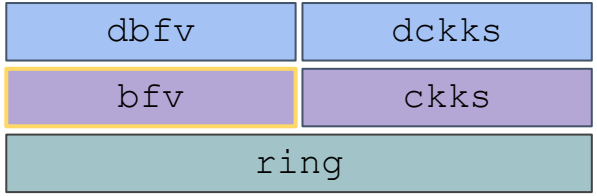
- Vectorized approximate complex-arithmetic

Notable features

- Hybrid parameterizable key-switching
  - Tweaked from [HK 19]
  - Let the user choose α, the number moduli dedicated to key-switching
- Depth-optimal, scale-managed, polynomial evaluation algorithm
  - Evaluate non-linear functions on encrypted inputs
- Full-RNS bootstrapping with non-sparse keys [BMTH 20]



The `lattigo/bfv` package



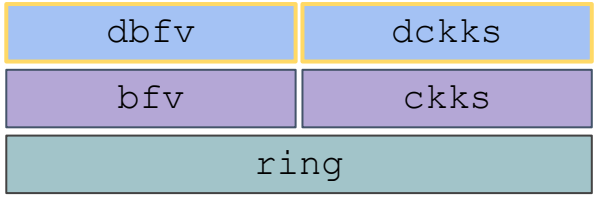
Implements the full-RNS variant of the BFV scheme [BEH 17]

- Vectorized integer modular arithmetic
- Scale invariant (quantization)

Notable features:

- Special full-RNS Quantization based on the full-RNS tools of [CHKKS 18]
  - Faster homomorphic multiplications than the current state of the art
- Tweaked hybrid parameterizable key-switching of [HK 19]

The `dbfv` and `dckks` packages



Implement the local operation for the *N-out-of-N* threshold BFV and CKKS [MTBH 20]

- Collective public/evaluation key-generation
- Collective key-switching/re-encryption operation to arbitrary secret-key
- Bridge with Linear-Secret-Sharing-based MPC:
    - Encryption-to-additive-shares
    - Additive-shares-to-encryption

Interactivity enables a fast, low-depth and exact bootstrapping

- Relying on the conversion between encryption and secret-shares

Benchmarks

- Setup:
  - Windows 10 @3.5 GHz and 32GB of RAM
    - Go v1.14.2
    - MSVC++ v14.28
- We use Microsoft SEAL 3.6 as a baseline in the single-party setting
- Goal: show that we can write fast cryptographic code in Go

Benchmarks – BFV

Benchmark parameters:

- d the polynomial modulus degree
- L the coefficient modulus size in # of word-sized moduli
- α the # of special primes (fixed to 1 for baseline comparison)

Table 3: BFV Timings in μs for $2^{10} \leq d \leq 2^{13}$

Op	$d = 2^{11}, L = 1$		$d = 2^{12}, L = 2$		$d = 2^{13}, L = 4$	
	SEAL	Lattigo	SEAL	Lattigo	SEAL	Lattigo
Encode	29	26	60	55	122	123
Decode	29	30	73	56	129	112
Encrypt	803	226	2085	936	5711	2935
Decrypt	110	64	358	284	1374	1251
Add	7	3	28	11	126	46
Mul-Pt	129	90	482	380	2084	1652
Mul-Ct	1146	476	3721	2065	14987	9123
Square	816	368	2693	1569	10918	7234
KeySwitch	-	-	775	745	3933	3781

Table 5: BFV Timings in ms for $2^{14} \leq d \leq 2^{16}$

Op	$d = 2^{14}, L = 8$		$d = 2^{15}, L = 15$		$d = 2^{16}, L = 31$	
	SEAL	Lattigo	SEAL	Lattigo	SEAL	Lattigo
Encode	0.2	0.2	0.5	0.5	1.1	1.2
Decode	0.2	0.2	0.6	0.6	1.2	1.3
Encrypt	18.5	10.5	65.4	39.2	253.5	153.0
Decrypt	5.6	5.3	23.5	22.4	115.7	95.6
Add	0.4	0.2	1.7	1.0	7.1	4.5
Mul-Pt	8.8	7.4	34.1	32.1	149.5	133.2
Mul-Ct	65.7	44.9	400.3	205.7	2822.6	1186.3
Square	48.4	35.0	306.8	155.8	2185.1	946.6
KeySwitch	24.3	24.1	147.0	154.5	1183.8	1235.5

Benchmarks – CKKS

Benchmark parameters:

- d the polynomial modulus degree
- L the coefficient modulus size in # of word-sized moduli
- α the # of special primes (fixed to 1 for baseline comparison)

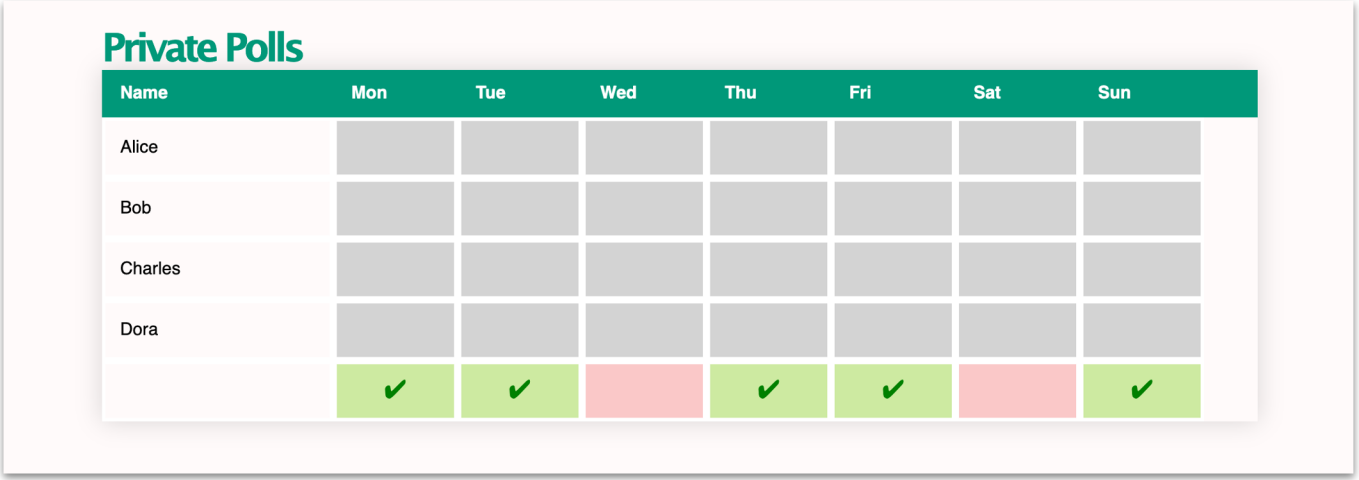
Table 4: CKKS Timings in μs for $2^{10} \leq d \leq 2^{13}$

Op	$d = 2^{11}, L = 1$		$d = 2^{12}, L = 2$		$d = 2^{13}, L = 4$	
	SEAL	Lattigo	SEAL	Lattigo	SEAL	Lattigo
Encode	112	103	305	247	854	668
Decode	63	82	345	174	1385	382
Encrypt	548	392	1816	1115	5329	3680
Decrypt	18	6	71	27	272	108
Add	7	3	28	10	124	46
Mul-Pt	14	7	52	27	210	126
Mul-Ct	45	15	187	61	795	242
Square	27	13	124	54	496	224
Rescale	-	-	203	222	861	857
KeySwitch	-	-	807	731	3927	3619

Table 6: CKKS Timings in ms for $2^{14} \leq d \leq 2^{16}$

Op	$d = 2^{14}, L = 8$		$d = 2^{15}, L = 15$		$d = 2^{16}, L = 31$	
	SEAL	Lattigo	SEAL	Lattigo	SEAL	Lattigo
Encode	3.2	1.9	14.3	6.3	58.0	22.9
Decode	6.4	0.8	31.7	3.1	230.7	6.5
Encrypt	19.0	13.0	71.6	50.0	295.7	211.4
Decrypt	1.1	0.4	4.7	2.3	19.2	9.1
Add	0.4	0.2	1.7	0.9	7.2	4.5
Mul-Pt	0.8	0.5	3.1	2.2	13.2	9.1
Mul-Ct	3.1	1.1	12.0	4.6	49.2	19.4
Square	2.1	0.9	8.4	4.0	35.3	17.1
Rescale	3.6	3.4	14.6	13.8	64.2	65.7
KeySwitch	23.4	22.8	146.5	143.5	1178.5	1205.8

Demo: Private Polls



A privacy-preserving web-application for scheduling meetings

- Very simple BFV-based circuit
- A guided tour more than a full-fledged application
- Code is available: https://github.com/ldsec/lattigo-polls-demo

Demo: Private Polls – Setting

Poll creator

- *Passive adversary*
- *No collusion with Server*
- *Runs a web-browser*

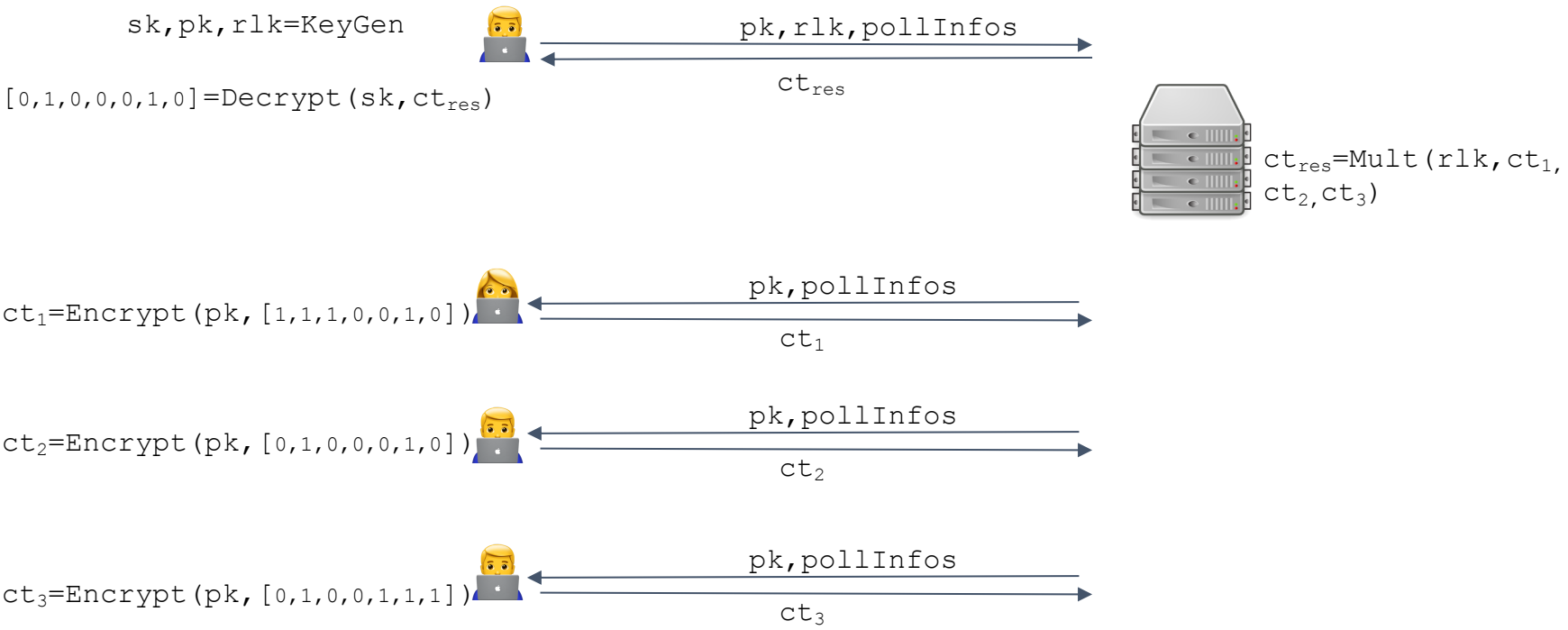
Server

- *Passive adversary*
- *No collusion with Poll creator*

Poll participants

- *Passive adversaries*
- *Run a web-browser*

Demo: Private Polls – Basic (crypto) protocol



Demo: Private Polls – Software Stack

Client application

`client.go`

Performs cryptographic ops:

`KeyGen, Encrypt, Decrypt` → `lattigo/bfv`

Interface with the JS runtime → `syscall/js`

`GOARCH=wasm go build client.go`

`client.wasm`

`JavaScript`

Server application

`server.go`

Performs cryptographic ops:

`Mult(ct1, ct2,...ctN)` → `lattigo/bfv`

Manages polls state:

`StoreResponse, Close`

Serves the web-app. & `client.wasm`

`POST, GET` → `net/http`

`go build client.go`

`HTTP`

`server`

Demo: Private Polls – `client.go`

We define a PollClient as the union type between the following Lattigo types:

```go
type KeyGenerator interface {
    GenKeyPair() (*SecretKey, *PublicKey)
    GenRelinKey(*SecretKey, uint64) (evk *EvaluationKey)
    // [...]
}
```

```go
type Encoder interface {
    EncodeUint(coeffs []uint64, pt *Plaintext)
    DecodeUint(pt interface{}, coeffs []uint64)
    // [...]
}
```

```go
type Encryptor interface {
    Encrypt(plaintext *Plaintext, ciphertext *Ciphertext)
    // [...]
}
```

```go
type Decryptor interface {
    Decrypt(ciphertext *Ciphertext, plaintext *Plaintext)
    // [...]
}
```

Client functionalities:

- ❑ Initialize with parameters
- ❑ Generate cryptographic keys
- ❑ Encode/Encrypt inputs
- ❑ Decrypt/Decode outputs

```go
import (
    "syscall/js"
    "github.com/ldsec/lattigo/v2/bfv"
)

// PollClient represents a client in the PrivatePoll application
type PollClient struct {
    bfv.KeyGenerator
    bfv.Encoder
    bfv.Encryptor
    bfv.Decryptor

    params *bfv.Parameters
}
```

Demo: Private Polls – `client.go`

Client functionalities:

- ❑ **Initialize with parameters**
- ❑ Generate cryptographic keys
- ❑ Encode/Encrypt inputs
- ❑ Decrypt/Decode outputs

Given the params and a secret or public key as a JS object

Depending on the type of the key:

- deserialize the key
  - Lattigo has (de-)serializers for all cryptographic objects
- instantiate the encryptor and/or decryptor
  - Encryption from a sk is more efficient than from the pk

```go
func NewPollClient(params *bfv.Parameters, keyObj js.Value) *PollClient {

    pc := new(PollClient)
    pc.params = params
    pc.KeyGenerator = bfv.NewKeyGenerator(params)
    pc.Encoder = bfv.NewEncoder(params)

    switch keyObj.Get("type").String() {
    case "sk":
        sk := bfv.NewSecretKey(params)
        utils.UnmarshalFromBase64(sk, keyObj.Get("key").String())
        pc.Encryptor = bfv.NewEncryptorFromSk(params, sk)
        pc.Decryptor = bfv.NewDecryptor(params, sk)
    case "pk":
        pk := bfv.NewPublicKey(params)
        utils.UnmarshalFromBase64(pk, keyObj.Get("key").String())
        pc.Encryptor = bfv.NewEncryptorFromPk(params, pk)
    }

    return pc
}
```

Demo: Private Polls – `client.go`

Client functionalities:

- ❑ **Initialize with parameters**
- ❑ Generate cryptographic keys
- ❑ Encode/Encrypt inputs
- ❑ Decrypt/Decode outputs

`main` is the entry point of a Go program

Wraps a Go function in a “JS” object

Define the function JS global context:

```
<html>
<script>
      let sk = localStorage.getItem("sk");
            initClient(key);
</script>
</html>
```

```go
func main() {

    var pollClient *PollClient
    var params = bfv.DefaultParams[1]

    initClientFunc := js.FuncOf(func(this js.Value, args []js.Value) interface{} {
        keyObj := args[0]

        pollClient = NewPollClient(params, keyObj)

        return nil
    })
    js.Global().Set("initClient", initClientFunc)

    //...
}
```

Demo: Private Polls – `client.go`

Client functionalities:

- ✓ Initialize with parameters
- ❑ **Generate cryptographic keys**
- ❑ Encode/Encrypt inputs
- ❑ Decrypt/Decode outputs

Generates the key from the bfv.KeyGenerator interface

Write the key as base-64 string in the fields of a JS object

```go
func main() {
    // ...

    genKeysFunc := js.FuncOf(func(this js.Value, args []js.Value) interface{} {
        jsObj := args[0]

        sk, pk := pollClient.GenKeyPair()
        rlk := pollClient.GenRelinKey(sk, 1)

        jsObj.Set("sk", utils.MarshalToBase64String(sk))
        jsObj.Set("pk", utils.MarshalToBase64String(pk))
        jsObj.Set("rlk", utils.MarshalToBase64String(rlk))

        return jsObj
    })
    js.Global().Set("genKeys", genKeysFunc)

    // ...
}
```

Demo: Private Polls – `client.go`

Client functionalities:

- ✓ Initialize with parameters
- ✓ Generate cryptographic keys
- ❑ **Encode/Encrypt inputs**
- ❑ Decrypt/Decode outputs

Extract the input from the input JS array

Encodes the input in a bfv.Plaintext and encrypts it in a bfv.Ciphertext

Return the ciphertext as a base64-string

```go
func main() {
    // ...

    encryptFunc := js.FuncOf(func(this js.Value, args []js.Value) interface{} {

        coeffs := make([]int64, 7)
        for i := range coeffs {
            coeffs[i] = int64(args[0].Index(i).Int())
        }

        pt := bfv.NewPlaintext(params)
        pollClient.EncodeInt(coeffs, pt)

        ct := pollClient.EncryptNew(pt)

        return utils.MarshalToBase64String(ct)
    })
    js.Global().Set("encrypt", encryptFunc)

    // ...
}
```

Demo: Private Polls – `client.go`

Client functionalities:

- ✔ Initialize with parameters
- ✔ Generate cryptographic keys
- ✔ Encode/Encrypt inputs
- ❑ **Decrypt/Decode outputs**

```go
func main() {
    // ...

    decryptFunc := js.FuncOf(func(this js.Value, args []js.Value) interface{} {
        ctObj := args[0]

        if pollClient.Decryptor != nil {
            ct := bfv.NewCiphertext(params, 1)
            utils.UnmarshalFromBase64(ct, ctObj.String())
            pt := pollClient.DecryptNew(ct)

            coeffs := make([]interface{}, 7)
            for i, v := range pollClient.DecodeIntNew(pt)[:7] {
                coeffs[i] = v
            }
            return coeffs
        }

        return nil
    })
    js.Global().Set("decrypt", decryptFunc)

    // ...
}
```

Deserializes the result bfv.Ciphertext and decrypts it

Decodes the bfv.Plaintext and returns it to the JS runtime.

Demo: Private Polls – `client.go` – Summary

Client functionalities:

- ✅ Initialize with parameters ⇒ The `initClient` JS function
- ✅ Generate cryptographic keys ⇒ The `genKeys` JS function
- ✅ Encode/Encrypt inputs ⇒ The `encrypt` JS function
- ✅ Decrypt/Decode outputs ⇒ The `decrypt` JS function

} 115 Lines of code

Remaining tasks for the client:

- ❑ Create the UI (in HTML+CSS+JS)
- ❑ Compile client.go in WebAssembly:
  - ❑ `$ GOOS=js GOARCH=wasm go build client.go`
- ❑ Serve the UI and Wasm files from the server

Demo: Private Polls – `server.go`

Server functionalities:

- ❑ Create a new poll from a `http.Request`
- ❑ Register a new response from a `http.Request`
- ❑ Compute the final result
- ❑ Serve the UI and Wasm files

Poll is the main data type for a Poll

A PollServer is a union type between a Lattigo Evaluator and an HTTP router.

The state of a PollServer is a collection of Polls

```go
import (
    "net/http"
    "github.com/gorilla/mux"
    "github.com/ldsec/lattigo/v2/bfv"
)

type Poll struct {
    ID           string
    Closed       bool
    Participants map[string]*bfv.Ciphertext

    pk     bfv.PublicKey
    rlk    bfv.EvaluationKey

    responses []*bfv.Ciphertext
    result    *bfv.Ciphertext
}

type PollServer struct {
    *mux.Router
    bfv.Evaluator

    Polls map[string]*Poll
}
```

Demo: Private Polls – `server.go`

Server functionalities:

- ❑ **Create a new poll from a `http.Request`**
- ❑ Register a new response from a `http.Request`
- ❑ Compute the final result
- ❑ Serve the UI and Wasm files

Unmarshalling of the pk and rlk into the Poll object fields

The “ID” of the Poll is the SHA256 hash of the public key.

```go
func (ps *PollServer) NewPoll(r *http.Request) *Poll {
    p := new(Poll)
    utils.UnmarshalFromBase64(&p.pk, r.FormValue("pk"))
    utils.UnmarshalFromBase64(&p.rlk, r.FormValue("rlk"))
    p.Participants = make(map[string]*bfv.Ciphertext, 5)
    p.ID = utils.GetSha256Hex(&p.pk)
    ps.Polls[p.ID] = p
    return p
}
```

Demo: Private Polls – `server.go`

Server functionalities:

- ✓ Create a new poll from a `http.Request`
- ❑ **Register a new response from a `http.Request`**
- ❑ Compute the final result
- ❑ Serve the UI and Wasm files

```
func (p *Poll) RegisterResponse(r *http.Request) error {

    name := r.FormValue("name")
    ct, update := p.Participants[name]
    if !update {
        ct = new(bfv.Ciphertext)
        p.responses = append(p.responses, ct)
        p.Participants[name] = ct
    }
    return utils.UnmarshalFromBase64(ct, r.FormValue("ct"))
}
```

If no response was provided yet, create a new bfv.Ciphertext object

Deserializes the ciphertext from the request

Demo: Private Polls – `server.go`

Server functionalities:

- ✓ Create a new poll from a `http.Request`
- ✓ Register a new response from a `http.Request`
- ❑ **Compute the final result**
- ❑ Serve the UI and Wasm files

```go
func (ps *PollServer) Close(p *Poll) {
    p.Closed = true
    if len(p.Participants) > 0 {
        agg := p.responses

        for len(agg) > 1 {
            agg = append(agg[2:],
                ps.RelinearizeNew(
                    ps.MulNew(agg[0], agg[1]), &p.rlk),
            )
        }
        p.result = agg[0]
    }
}
```

Aggregates the responses recursively in a low-multiplicative-depth circuit

The multiplicative-depth is $\log_2(\text{len}(\text{p.responses}))$

Demo: Private Polls – `server.go`

Server functionalities:

- ✓ Create a new poll from a `http.Request`
- ✓ Register a new response from a `http.Request`
- ✓ Compute the final result
- ❑ **Serve the UI and Wasm files**

Instantiate a PollServer with one of the bfv default parameters set:

Binds the poll creation logic to the `/polls` route

Serves the PollServer on /

Serves the static files on `/static/`

Starts the HTTP server

```go
func main() {

    ps := NewPollServer(bfv.DefaultParams[1])

    pollTpl := template.Must(template.ParseFiles("poll.gohtml"))
    ps.HandleFunc("/polls", func(rw http.ResponseWriter, r *http.Request) {
        if r.Method == "POST" {
            p := ps.NewPoll(r)
            rw.Write([]byte(p.ID))
        } else {
            pollTpl.Execute(rw, nil)
        }
    })

    // ...  [handler function for the /polls/<poll_id> route] ...

    http.Handle("/", ps)

    http.Handle("/static/", http.StripPrefix("/static/", http.FileServer(http.Dir("./static"))))

    http.ListenAndServe(`:8080`, nil)
}
```

Demo: Private Polls – `client.go` – Summary

Server functionalities:

- ✅ Create a new poll from a `http.Request` ⇒ The `POST /polls` route
- ✅ Display a Poll page for a given poll ID ⇒ The `GET /polls/<poll_id>` route
- ✅ Register a new response from a `http.Request` ⇒ The `POST /polls/<poll_id>` route
- ✅ Compute the final result ⇒ The `GET /polls/<poll_id>?closing=true` route
- ✅ Serve the UI and Wasm files ⇒ The `GET /static/*` route

155 Lines of code

Remaining tasks for the server:

- ❑ Input checking and error handling
- ❑ Concurrent request handling

Demo: Private Polls – Live Demo

https://github.com/ldsec/lattigo-polls-demo

Lattigo Applications

- Single-party applications
  - iDash challenge (http://www.humangenomeprivacy.org/) finalist solution in 2019 (encrypted multinomial regression and genotype imputation)
    - Batch predictions of 80k variants/patient for 1000 patients in less than 10 seconds
    - Kim M., Harmanci A., Bossuat J.P., Carpov S., Cheon J.H., Chillotti I., Cho W., Froelicher D., Gama N., Georgieva M., Hong S., Hubaux J.P., Kim D., Lauter K., Ma Y., Ohno-Machado L., Sofia H., Son Y., Song Y., Troncoso-Pastoriza J.R., Jiang X., *Ultra-Fast Homomorphic Encryption Models enable Secure Outsourcing of Genotype Imputation*, 2020.
    - https://doi.org/10.1101/2020.07.02.183459
  - Secure oblivious linear function evaluation (vOLE) in passive and active adversarial settings
    - 1 million 120-bit OLEs/s in passive adversary model
    - Baum C., Escudero D., Pedrouzo-Ulloa A., Scholl P., Troncoso-Pastoriza J.R., *Efficient Protocols for Oblivious Linear Function Evaluation from Ring-LWE*, 2020.
    - SCN'20, https://eprint.iacr.org/2020/970
  - Verifiable computation of encrypted certified inputs with MPC-in-the-head proofs
    - Application to smart metering, disease susceptibility and location-based activity tracking
    - Chatel S., Pyrgelis A. Troncoso-Pastoriza J.R., Hubaux J.P., *Privacy and Integrity Preserving Computations with CRISP*, 2020.
    - USENIX'21, https://arxiv.org/abs/2007.04025

Demo: why MHE ?

Poll creator

- *Passive adversary*
- *No collusion with Server*

Server

- *Passive adversary*
- *No collusion with Poll creator*

Poll participants

- *Passive adversaries*

Participant can be unwilling to contribute their data without privacy guarantees.
E.g.: **Federated Learning** among medical institutions

Lattigo Applications

- Multi-party applications in private federated learning
  - Secure training and evaluation of generalized linear models
    - Logreg with 1 million samples and 32 features across 160 data providers in 3 minutes
    - Froelicher D., Troncoso-Pastoriza J.R., Pyrgelis A., Sav S., Sa Sousa J., Bossuat J.P., Hubaux J.P., *Scalable Privacy-Preserving Distributed Learning*, 2020.
    - PETS'21, https://arxiv.org/abs/2005.09532
  - Secure training and evaluation of neural networks
    - 3-layer NN on MNIST across 10 data providers in <2h
    - Sav S., Pyrgelis A., Troncoso-Pastoriza J.R., Froelicher D., Bossuat J.P., Sousa J., Hubaux J.P., *POSEIDON: Privacy-Preserving Federated Neural Network Learning*, 2020.
    - https://arxiv.org/abs/2009.00349

Ongoing & Future work

- **Fully-Threshold MHE**: T-*out-of-*N threshold variant of [MTBH 20], allows to set a tradeoff between security and availability.

- **Real-CKKS [KS 18]**: Variant of the CKKS scheme able to encrypt up to d real numbers instead of $d/2$ complex numbers.

- **Lattigo-Cloud/Lattigo-MP**: Network-layers implementing the MHE-based MPC protocols for both cloud-based (clients + server) and peer-2-peer (clients) settings.

Conclusion

What is new in Lattigo:

- The first HE library focusing on multiparty settings
- The first open-source Full-RNS-CKKS bootstrapping

What becomes easier with Lattigo:

- Developing HE-based multi-party systems
- Running HE on any platform (incl. browsers!)
- Experimenting/benchmarking new Multi-party HE-applications (e.g., in federated learning)

What is still hard:

- Mapping complex functionalities to efficient HE-circuits
- Parametrization of the schemes
- Composability of HE and Circuit-Privacy

https://github.com/ldsec/lattigo

WAHC 2020 - References

[AJLTVW 12] Asharov G., Jain A., L'opez-Alt A., Tromer E., Vaikuntanathan V., Wichs D., *Multiparty computation with low communication, computation and interaction via threshold FHE*, 2012. In Annual International Conference on the Theory and Applications of Cryptographic Techniques (pp. 483-501). Springer, Berlin, Heidelberg.

[BEH 17] Bajard J.C., Eynard J., Hasan A., Zucca V., *A Full RNS Variant of FV Like Somewhat Homomorphic Encryption Schemes*, 2016. In International Conference on Selected Areas in Cryptography (pp. 423-442). Springer, Cham.

[BMTH 20] Bossuat J.P., Mouchet C., Troncoso-Pastoriza J., Hubaux J.P., *Efficient Bootstrapping for Approximate Homomorphic Encryption with Non-Sparse Keys*, 2020. In IACR Cryptol. ePrint Arch., 2020, 1203.

[CCS 19] Chen H., Ilaria C., Song Y., *Multi-key homomorphic encryption from TFHE*, 2019. In International Conference on the Theory and Application of Cryptology and Information Security (pp. 446-472). Springer, Cham.

[CDB 01] Cramer R., Damgård I., B Nielsen J., *Multiparty computation from threshold homomorphic encryption*, 2001. In International conference on the theory and applications of cryptographic techniques (pp. 280-300). Springer, Berlin, Heidelberg.

[CDKS 19] Chen H., Dai W., Kim M., Song, Y., *Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference*, 2019. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (pp. 395-412).

[CHKKS 18] Cheon J.H., Han K., Kim A., Kim M., Song Y., *A full RNS variant of approximate homomorphic encryption*, 2018. In International Conference on Selected Areas in Cryptography (pp. 347-368). Springer, Cham.

[CKKS 17] Cheon J.H., Kim A., Kim M., Song Y., *Homomorphic Encryption for Arithmetic of Approximate Numbers*, 2017. In International Conference on the Theory and Application of Cryptology and Information Security (pp. 409-437). Springer, Cham.

[FH 96] Franklin M., Haber S., *Joint encryption and message-efficient secure computation*, 1996. Journal of Cryptology, 9(4), 217-232.

[FV 12] Fan J., Vercauteren F., *Somewhat Practical Fully Homomorphic Encryption*, 2012. In IACR Cryptol. ePrint Arch., 2012, 144.

[HK 19] Han K., Ki D., *Better Bootstrapping for Approximate Homomorphic Encryption*, 2019. In Cryptographers' Track at the RSA Conference (pp. 364-390). Springer, Cham.

[HPS 19] Halevi S., Polyakov Y., Shoup V., *An Improved RNS Variant of the BFV Homomorphic Encryption Scheme*, 2019. In Cryptographers' Track at the RSA Conference (pp. 83-105). Springer, Cham.

WAHC 2020 - References cont.

[KS 18] Kim D., Song Y., *Approximate Homomorphic Encryption over the Conjugate-Invariant Ring*, 2018. In International Conference on Information Security and Cryptology (pp. 85-102). Springer, Cham.

[LTV 12] López-Alt, A., Tromer E., Vaikuntanathan V., *On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption*, 2012. In Proceedings of the forty-fourth annual ACM symposium on Theory of computing (pp. 1219-1234).

[MTBH 20] Mouchet C., Troncoso-Pastoriza J., Bossuat J.P., Hubaux J.P., *Multiparty Homomorphic Encryption from Ring-Learning-With-Errors*, 2020. In IACR Cryptol. ePrint Arch., 2020, 304.

Benchmarks – DBFV

Table 7: DBFV Timings [ms]: Total cost per party for generation (Gen), aggregation (Agg) and output (Out) local operations. Aggregated over the two rounds for the RelinKeyGen.

Op		$d=2^{13}, L=4$	$d=2^{14}, L=8$	$d=2^{15}, L=15$
EncKeyGen	Gen	0.77	2.59	10.45
	Agg	0.02	0.08	0.32
	Out	0.09	0.34	1.28
RelinKeyGen	Gen	8.79	30.79	159.48
	Agg	0.25	1.12	6.18
	Out	0.36	1.81	9.72
RotKeyGen	Gen	2.46	8.73	45.24
	Agg	0.06	0.25	1.53
	Out	0.11	0.67	3.88
ColKeySwitch	Gen	1.15	4.43	19.31
	Agg	0.02	0.06	0.25
	Out	0.02	0.07	0.32
PubColKeySwitch	Gen	3.01	11.98	48.74
	Agg	0.03	0.12	0.55
	Out	0.03	0.12	0.68

Benchmark parameters:

- d — the polynomial modulus degree
- L — the coefficient modulus size in # moduli
- α — the # of special primes (set to $\sqrt{(L-1)}$)

Benchmarked protocol operations:

- Gen — Generating a party share (indep. N)
- Agg — Aggregating two shares (indep. N)
- Out — Computing the protocol output from the view (indep. N)

Table 8: DBFV Share Sizes [MB]: Total amount sent per party. Aggregated over the two rounds for the RelinKeyGen.

Op	$d=2^{13}, L=4$	$d=2^{14}, L=8$	$d=2^{15}, L=15$
EncKeyGen	0.26	1.05	3.93
RelinKeyGen	3.15	12.58	62.91
RotKeyGen	0.79	3.15	15.73
ColKeySwitch	0.2	0.79	3.15
PubColKeySwitch	0.39	1.57	6.29

Design principles

- Cryptographic functions are provided by **function-specific objects**

```
encryptor:=NewEncryptor(params, key)
```

```
encryptor.Encrypt(pt, ct)
```

- Allocation-free API with shortcuts

```
evaluator.Add(ct1, ct2, ctOut)
```

```
ctOut:=evaluator.AddNew(ct1, ct2)
```

- Objects assumes single-threaded use
- Objects methods are single-threaded
- Cryptographic objects types are serializable