Bootstrap-based Proxy Reencryption for Private Multi-user Computing

Juan Ramón Troncoso-Pastoriza
Signal Theory and Communications Department
University of Vigo
36310 Vigo, Spain
troncoso@gts.uvigo.es

Serena Caputo
Signal Theory and Communications Department
University of Vigo
36310 Vigo, Spain
scaputo@gts.uvigo.es

***Abstract*—The increasingly popular paradigm of Cloud computing brings about many benefits both for clients and providers, but it also introduces privacy risks associated to outsourcing data and processes to an untrustworthy environment. In particular, the multi-user computing scenario is especially difficult to tackle from a privacy-preserving point of view, seeking to protect data from different users while allowing for flexible Cloud applications.**

This work leverages Gentry's cryptographic bootstrapping operation as a means to endow fully homomorphic cryptosystem with proxy reencryption functionalities, targeted at the private multi-user and multi-key computing scenario. We provide an example implementation based on Gentry-Halevi cryptosystem, and a secure protocol that employs this primitive for solving the private multi-user computing scenario with non-colluding parties.

***Index Terms*—Cryptographic protocols, Privacy, Multi-key, Bootstrapping, Fully Homomorphic Encryption**

I. Introduction

Cloud computing is an increasingly popular paradigm, which has experienced a growing adoption in the last few years. Cloud brings about many benefits both for clients and providers, like ubiquitous access and computing, better scalability, multi-tenancy, and reduced initial investments. The other side of the coin comes from the privacy risks that clients may suffer when outsourcing their data and processes to an untrustworthy environment, as they lose control over them once they enter the Cloud. Hence, privacy is one of the highest barriers for Cloud adoption, and there is a need for privacy-preserving solutions that enable secure Cloud computing applications with two aims: a) protect sensitive data while they are processed in the Cloud, so that the cloud server cannot access them, and b) allow for flexible Cloud applications without hindering their functionality and usability [1].

There are several tools that can be used to achieve a privacy-preserving outsourced processing, namely Trusted Computing (Trusted Platform Modules - TPMs), Secure Modules (Hardware Secure Modules - HSMs), Secure Computation (Signal Processing in the Encrypted Domain - SPED, Fully Homomorphic Encryption - FHE, Garbled Circuits), or a combination thereof. Independently of the chosen tools, privacy preservation in outsourced applications essentially involves the cryptographic protection of the involved sensitive data and an access control mechanism to enforce a proper usage, interplay, share and mix of those data. Depending on the specific scenario, encryption and access control have different weights. Within the possible Cloud-based privacy-preserving scenarios where a set of clients $\{C_i\}_{i=0}^{N-1}$ access a cloud server S, van-Dijk and Juels [2] establish a relevant classification of private cloud applications in three groups:

- Private single-client computing: they execute over the data x_i of a client C_i, and only C_i has access to the corresponding inputs and outputs.
- Private multi-client computing: they execute over the data $\{x_i\}_{i=0}^{N-1}$ of multiple clients $\{C_i\}_{i=0}^{N-1}$, not mutually trusting. The information (inputs and outputs) is selectively released.
- Stateful private multi-client computing: Similar to the prior scenario, but with dynamic selective allowance to data depending on their processing history.

The first scenario can be theoretically solved by using efficient Fully Homomorphic Encryption (FHE) alone [3], by encrypting the data of each client C_i with the corresponding key K_{C_i} and allowing for the needed processing on those data alone. Conversely, multi-client scenarios need two elements:

- A proper access control mechanism (either stateful or stateless). Furthermore, van-Dijk and Juels [2] prove the impossibility of achieving, in general, privacy-preserving multi-client computing by using cryptography alone, such that further assumptions are needed; i.e., some form of trusted computing environment must be present, either through a hardware trusted computing element, a trusted third party or a secure multiparty protocol involving more than two parties.
- The possibility of operating on data that are encrypted under different user keys. In general, homomorphic encryption cannot straightforwardly cope with data encrypted under different keys.

This work addresses the second element, by focusing on the capability of bootstrappable FHE to work with inputs encrypted under several different keys. To this aim, we leverage the use of bootstrapping as a proxy reencryption mechanism for fully-homomorphic cryptosystems.

A. Notation and Structure

Throughout the work, scalar values and polynomials will be denoted by lowercase regular letters, while vectors (matrices)

will be denoted by lowercase (uppercase) boldface letters. $[.]_q$ will represent the modulo-q reduction, while $[\![.]\!]_{pk}$ represents the encryption (coefficient-wise encryption) of an integer (vector or matrix) under the public key pk, and $\lceil . \rfloor$, the rounding to the nearest integer.

The rest of the work is organized as follows: Section II briefly surveys the current different approaches for multi-key scenarios. Section III revisits the concept of cryptographic bootstrapping introduced by Gentry. Section IV presents the proposed system and the use of bootstrapping for proxy reencryption. A specific implementation of the bootstrapping-based reencryption for Gentry-Halevi cryptosystem is presented and evaluated in Section V, and Section VI draws some conclusions.

II. Preliminaries and contributions

In recent years there have been several approaches aimed at enabling private multi-client multi-key computing using Homomorphic Encryption. It must be noted that we will leave aside hardware-based solutions (TPMs and HSMs) and focus on the underlying homomorphic encryption mechanism for the privacy-preserving system; therefore, the existing solutions follow one of the following three approaches:

1) Use of a trusted element in which the client delegates encryption and decryption: Bugiel *et al.* [4] present an architecture for secure computing in which the user outsources the data to a trusted cloud. This trusted cloud acts as an intermediary that communicates to the commodity cloud on behalf of the client, and performs all the encryption, decryption, reencryption and program obfuscation. The trusted cloud has access to all the clear text data of the client, it is always online and it has a high-bandwidth line to the commodity cloud. It can hence perform all the required access control and encrypt the data with suitable keys for it to be operated at the commodity cloud.
2) Design a secure protocol for performing reencryption: Peter *et al.* [5] use a double-trapdoor additively homomorphic cryptosystem (BCP [6], by Bresson, Catalano and Pointcheval). They introduce two non-colluding semi-honest servers ($\mathcal{C}$ and $\mathcal{S}$), such that $\mathcal{S}$ has the master key able to decrypt (and hence reencrypt) any ciphertext, while $\mathcal{C}$ performs the actual homomorphic operations. Through interactive secure protocols between $\mathcal{C}$ and $\mathcal{S}$ the input data are reencrypted to a common operation key, subsequently operated at $\mathcal{C}$, and the outputs are reencrypted for each corresponding client.
3) Design the underlying FHE specifically for multi-key operation: López-Alt *et al.* [7] introduce the notion of *on-the-fly multiparty computation*, in which the cloud can operate on data coming from a polynomial number of N keys by using a fully homomorphic cryptosystem denoted N-key FHE. This N-key FHE is capable of operating on inputs encrypted under multiple, unrelated keys, producing results which are decryptable only by a joint interaction between all the involved clients. The authors also prove that without such joint decryption step it is impossible to obtain non-interactive multi-key processing with only one server.

Under the model of private multi-client computing, we can conceptually categorize all the three aforementioned approaches as proxy reencryption [8] alternatives, in which the client delegates the reencryption to some trusted environment, therefore avoiding client interaction in the processing step. In the first approach, there is a full delegation (encryption/decryption/reencryption) to the trusted cloud. In the second one, the delegation aims at the interactive non-colluding combination between several individually non-trusted servers; it must be noted though, that Peter *et al.* approach is also effectively delegating decryption to server $\mathcal{S}$, which is assumed not to have access to the original input data. Finally, the third approach allows for universal reencryption, but delegates decryption only to the joint set of users whose keys are involved in the reencryption step.

A. Our contribution

The main contribution of this work is the use of bootstrapping as a proxy reencryption mechanism to seamlessly cope with multi-key operations; we also design a protocol to apply these mechanisms in private outsourced multi-client computing, and implement and test a proof-of-concept version of the devised scheme by using Gentry and Halevi [10] cryptosystem as a basis. Gentry's thesis [9] also presented bootstrapping as a proxy one-way reencryption scheme, but to the best of our knowledge, it has not been used with this purpose in a practical scenario.

It must be noted that prior multi-key oriented works (including López-Alt *et al.* [7]) use bootstrapping only as a means to achieve a fully homomorphic cryptosystem from a somewhat homomorphic one. To the best of our knowledge, the use of bootstrapping as a proxy reencryption mechanism has not been proposed so far. Furthermore, our proposal lies in-between approaches 2) and 3) from the aforementioned categorization, therefore substituting the need for a user collaborative joint decryption in 3) by the use of a two-server based access control, similarly to the way 2) does, but without the need of inter-server collaboration during the reencryption steps.

III. Cryptographic Bootstrapping

The cryptographic concept of bootstrapping was introduced by Gentry in his seminal work [9]. Gentry's bootstrapping allows to construct a fully homomorphic encryption (FHE) starting from a lattice-based somewhat homomorphic scheme (SHE) that can evaluate a limited class of functions. In a SHE, the set of executable functions is bounded due to the noise growth after each performed operation. Ciphertexts are lattice points to which some noise is added; performing operations increases this noise, up to the possible occurrence of a decryption error when the noise gets out of the fundamental region of the lattice. Informally, Gentry's proposal to achieve FHE states that whenever the cryptosystem can homomorphically

evaluate its own decryption circuit without decryption errors, encryptions can be "boostrapped" to reduce their noise level.

More formally, a somewhat homomorphic public key cryptosystem Σ of depth L is composed by four functions KeyGen_Σ, Enc_Σ, Dec_Σ and Eval_Σ, where KeyGen_Σ generates a key pair (pk, sk), Enc_Σ takes as inputs the public key pk and a plaintext m in order to output a ciphertext c, and Dec_Σ takes a ciphertext and the secret key sk and returns the decrypted message. The Eval_Σ function takes as inputs the public key and a function f to evaluate in the tuple of ciphertexts $(c_1, \ldots, c_n) = (\llbracket m_1 \rrbracket_{pk}, \ldots, \llbracket m_n \rrbracket_{pk})$, and outputs a ciphertext c corresponding to the evaluation of $f(c_1, \ldots, c_n)$

$$c = \mathsf{Eval}_\Sigma(pk, f, (c_1, \ldots, c_n)) \equiv \mathsf{Enc}_\Sigma(pk, f(m_1, \ldots, m_n)).$$

The output ciphertext has, in general, an increased noise, and the equivalence is only true whenever the depth of the circuit representing f is less than L.

For such a cryptosystem, whenever the decryption operation can be expressed as a degree $L-1$ circuit, it is possible to homomorphically execute a decryption, taking as input the encryption of sk, and producing a refreshed ciphertext, hence defining the bootstrapping function as:

$$\mathsf{Bstr}_\Sigma(pk, \llbracket m \rrbracket_{pk}) = \mathsf{Eval}_\Sigma(pk, \mathsf{Dec}_\Sigma, \llbracket m \rrbracket_{pk}).$$

If the cryptosystem can execute at least one homomorphic multiplication and a bootstrap, then it becomes fully homomorphic. It must be noted that the decryption function of a regular SHE cannot normally fit as a degree-L circuit, and it must be *squashed* by providing some public helper data about the secret key. Therefore, the SHE with the squashed decryption and the public helper data conforms the bootstrappable fully homomorphic cryptosystem.

This is the intended application of bootstrapping, but this primitive can be much more versatile if used as a generic proxy reencryption. Gentry already proposed this use when introducing the bootstrapping, but it has not been applied in a practical scenario since then. In this work we propose a bootstrap-based one-way reencryption that finds application in privacy-preserving multi-key scenarios.

IV. Proposed Secure System

We first specify the scenario for which we propose a bootstrap-based system for private multi-user computation (see Figure 1). We envision the following parties and roles:

- Users: Each user C_i has a unique key pair, generated by the Authoritative Party. Users outsource solely data encrypted under their own keys to the Computing Server, authorize determined collaborative processes when the server asks, and receive the corresponding output data encrypted under their own key.
- Authoritative Party (AP): This is a generic third party in charge of key, certificate and helper data generation. It can be actually implemented as a trusted third party (certification authority) or as a secure (tamper-proof) hardware module.

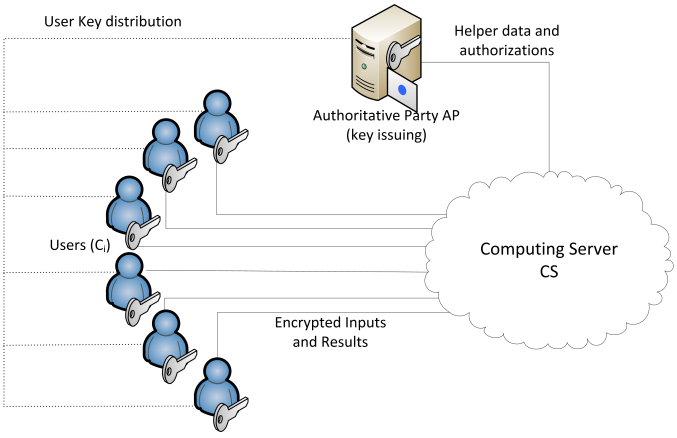


Fig. 1. Devised scenario for private multi-client outsourced processing.

- Computation Server (CS): This server represents the untrusted environment where the computation is outsourced. In our scenario, it has no access to any secret key, and all data that it receives or manipulates is encrypted. It may only have access to public keys and helper data to perform reencryption.

We assume that parties do not collude and the computation server follows the protocol (it is honest but curious). This is a commonly accepted assumption for multi-user environments, as a malicious or colluding behavior by the computation server will harm its business.

In this scenario, we will now detail the proposed bootstrap-based reencryption primitive and the secure protocol for private multi-client computing.

A. Bootstrapping for proxy reencryption

The original formulation of bootstrapping proposed by Gentry was initially intended to refresh an encryption to be reused under the same encryption key (pk_1, sk_1). Nevertheless, as Gentry points out, it is conceptually possible to extend that concept and bootstrap the encryption under a different key (pk_2, sk_2), therefore producing a refreshed cipher with low noise norm and that reencrypts that ciphertext.

In the clear, this would involve two steps: a decryption with the original key, and an additional encryption under the second key. On the contrary, if we follow the same philosophy as for the bootstrapping, in the encrypted domain it would be possible to implement the decryption of the first key (pk_1, sk_1) as a homomorphic operation under the second (pk_2, sk_2), therefore joining both decryption and reencryption under one sole homomorphic operation. More precisely, we apply a bootstrapping for (pk_2, sk_2) with inputs pk_2 and $c = \llbracket m \rrbracket_{pk_1}$, for some plaintext m, i.e. the function

$$\mathsf{Rec}_\Sigma(pk_2, \llbracket m \rrbracket_{pk_1}) = \mathsf{Eval}_\Sigma(pk_2, \mathsf{Dec}_\Sigma(sk_1), \llbracket m \rrbracket_{pk_1}). \quad (1)$$

The output is a refreshed reencryption of the original plaintext under the second key (pk_2, sk_2). This can be thought of as a form of proxy reencryption in the traditional sense [8],

so we can easily confer proxy reencryption functionalities to any bootstrappable cryptosystem. In case the decryption function has to be squashed, Rec would need to be paired with helper data $\{HD_{sk_1 \to pk_2}\}$ for the first secret key sk_1 under the second key (pk_2, sk_2). We will clarify this in a practical example with Gentry-Halevi cryptosystem (cf. Section V).

B. Secure Protocol for Private Multi-user Computing

Thanks to the use of the Rec reencryption primitive, it is possible to setup a secure protocol for computing with data coming from several users encrypted with their own keys, within the scenario depicted by Figure 1. Our proposed protocol involves several phases, described in the following paragraphs and graphically shown in Protocol 1.

a) Key Setup: The Authoritative Party AP issues the keys for every user C_i in the system. During this phase, the users may specify their access control preferences, setting which classes of data can be shared for specific operations, and which user groups may be involved in those operations or receive the final results.

b) Data Upload: The user encrypts her data with her own key pk_i before uploading them to the Computing Server.

c) Multi-User Computing: When the Computing Server starts a computing process CP_j involving a set of users $\{C_i\}_{inCP_j}$ and their respective inputs, and a set of users who will receive the computation outputs $\{C_i\}_{outCP_j}$, it sends a request to the authoritative party including all the information about involved parties and type of operation; the AP checks the access control preferences and, if the operation is granted, it generates a unique operation key pair $(CPpk_j, CPsk_j)$, and delivers only the public key to the CS. The AP must also generate the needed helper data for reencryption from the involved user keys to the operation key $\{HD_{sk_i \to CPpk_j}\}_{inCP_j}$, which are also sent to the CS. Finally, the helper data for reencryption of the computation results $\{HD_{CPsk_j \to pk_i}\}_{outCP_j}$ can be generated and sent to the CS.

The Computing Server executes the reencryption to the operation key on all the inputs using the corresponding helper data, and operates on the homogenized data using the homomorphic properties of the cryptosystem, in order to obtain the results encrypted under the operation key.

d) Distribution of Results: After a computation CP_j, the CS can reencrypt the results from the operation key to the authorized clients $\{C_j\}_{outCP_j}$ by using the helper data provided by the AP, and send them to those clients, which will be able to decrypt them with their own keys.

1) Security and malicious adversaries: We introduce the Authoritative Party in our protocol as a helper server or trusted environment that generates keys and enforces access control; contrarily to prior approaches (Peter *et al.* [5]), in our protocol this helper server never receives or processes any data from clients (neither encrypted nor clear), and all data processing is carried out at the Computing Server.

If the used cryptosystem is semantically secure, the construction above is secure for the case of a semi-honest CS that follows the protocol without colluding with any user. It must be noted that the secret operation key is not given to the CS: analogously to a regular proxy reencryption scheme, if the CS had access to the operation secret key, it could decrypt all the input data just by reencrypting it in the operation key.

Regarding the involved authentication and access control, we assume that they are enforced, but we do not detail the specific mechanisms, as they fall outside of the scope of this work. For completeness, we can highlight that other recent encryption-based access control systems for Cloud data access [11] can be adapted to this case, whenever the access control is either split between the two non-colluding servers AP and CS, or enforced by AP as a trusted hardware element/trusted third party.[1]

In case additional protection against a malicious computing server has to be provided, then the protocol can be modified in such a way that the AP does not release the output reencryption helper data $\{HD_{CPsk_j \to pk_i}\}_{outCP_j}$ to the CS, and both servers participate in the output reencryption step.[2] This modified scenario can be seen as virtually equivalent to the one by Peter *et al.* [5], with the essential difference of using a proxy reencryption scheme for preprocessing the inputs instead of a double-trapdoor for delegating the decryption/reencryption to the helper server. Finally, this modification is not enough to cope with a malicious CS, and it would also require the use of zero-knowledge proofs (involving collision resistant hash functions) of the performed operations, similarly to the approach followed in [7].

V. Example with Gentry-Halevi cryptosystem

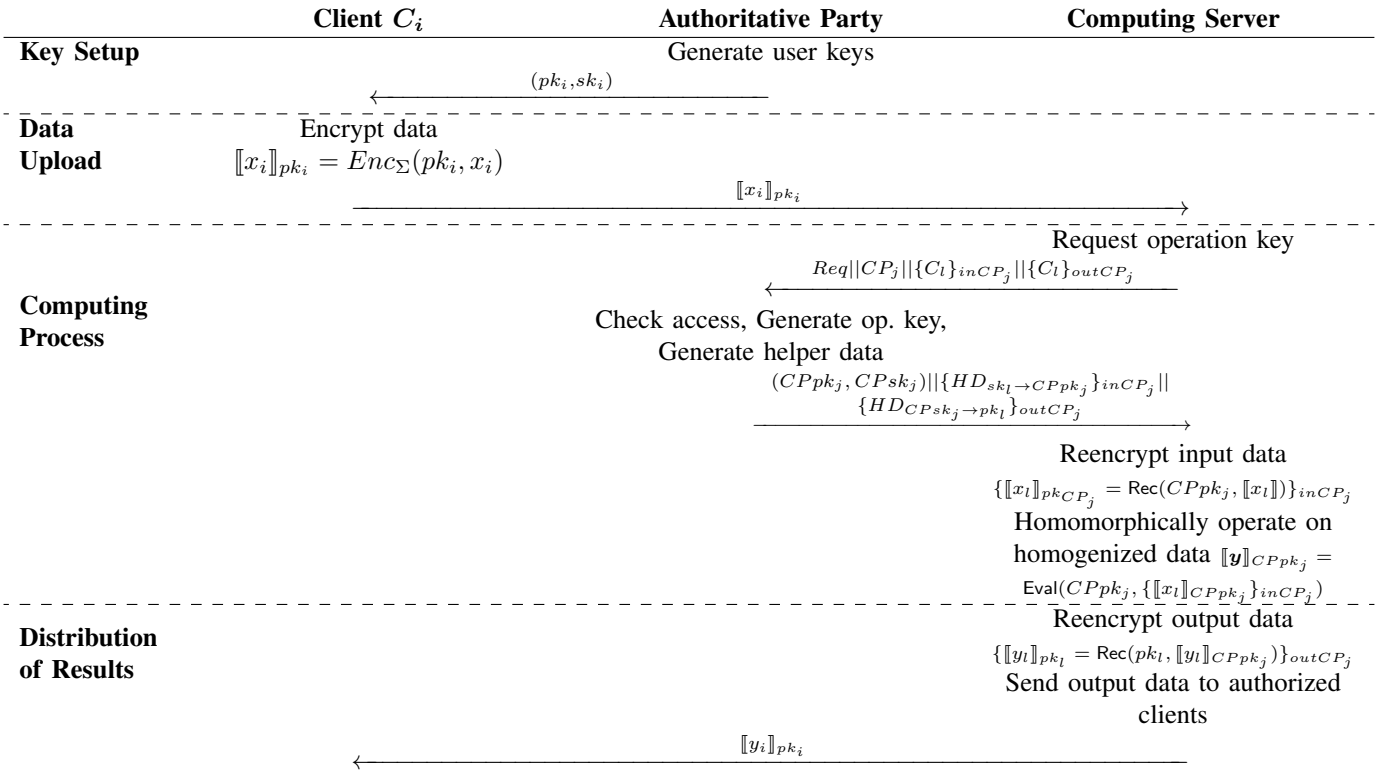
This section presents a practical example of a bootstrap-based reencryption, which we have described only conceptually so far. We will take as the base cryptosystem the implementation by Gentry and Halevi [10], which belongs to the family of GGH (Goldreich-Goldwasser-Halevi) lattice cryptosystems. The somewhat homomorphic version of the cryptosystem uses a principal-ideal lattice J generated by a polynomial $v(x)$ with t-bit signed random integer coefficients, in the ring of polynomials modulo $f_n(x) \doteq x^n + 1$. The cryptosystem is parameterized by a big number d (resultant of the polynomials $v(x)$ and $f_n(x)$), and r (a root of $f_n(x) \bmod d$), that conform the public key (d, r). Additionally, the private key is given by an odd coefficient $w \in \mathbb{Z}_d$ of the scaled (modulo $f_n(x)$)-inverse of $v(x)$. For a binary plaintext $b \in \mathbb{Z}_2$, the encryption and decryption functions are defined as

$$c = \mathsf{Enc}((d, r), b) = [b + 2 \sum_{i=0}^{n-1} u_i r^i]_d,$$

$$\hat{b} = \mathsf{Dec}((d, w), c) = [c \cdot w]_d \bmod 2,$$

[1] Due to the impossibility of program obfuscation, cryptography alone cannot provide the necessary access control for the private multi-user computing scenario [2].

[2] For common multi-user operations, it is expected that the dimensionality of the outputs be much lower than that of the input data, or the results would reveal too much information about the individual inputs. Therefore, the computation needed at the AP for the output reencryption should be negligible compared to the computation performed at the CS.



Protocol 1: Proposed private multi-user computing protocol.

where $\boldsymbol{u} \in \{-1, 0, 1\}^n$ is and a random salt vector with independent coefficients chosen as 0 with probability q and ± 1 with probability $(1-q)/2$ each (q is a security parameter).

As defined, this cryptosystem is somewhat homomorphic under addition and multiplication, which are directly mapped from the crypto-text ring (errors with respect to lattice points) to the clear-text ring. This homomorphism is limited, as both operations are only correctly mapped when the error lies within the same fundamental region of the lattice after performing each operation. Furthermore, the decryption circuit cannot be directly translated into a low-degree circuit that can be homomorphically executed under encryption, so, for reaching a full homomorphism, Gentry proposes to *squash* the decryption circuit so that it can be homomorphically executed and enable bootstrapping.

We briefly describe the original squashing and bootstrapping procedure in [10]: the secret key integer w is substituted by an instance of the Sparse Subset-Sum-Problem (SSSP) with parameters s and S, $s \ll S$. A matrix $\boldsymbol{X}$ of integers modulo d and dimensions $s \times S$ is generated such that the sum modulo d of one and only one element of each row of $\boldsymbol{X}$ yields the secret key w. Then, the new secret key is taken as a binary matrix $\{\sigma_{k,i}\}_{k,i}$, for $k = 1, \ldots, s$, $i = 1, \ldots, S$, whose rows $\boldsymbol{\sigma}_k$, contain exactly one non-zero element at (a random) position i_k. The matrix $\{\sigma_{k,i}\}_{k,i}$ is the characteristic matrix of $\boldsymbol{X}$, with only one non-zero element per row, such that the sum of the element-wise product between the two matrices gives as a result $\left[\sum_{k=1}^{s}\sum_{i=1}^{S} x_{k,i}\sigma_{k,i}\right]_d = w$. With these additional variables, the squashed decryption of a ciphertext c can be reduced to [10]:

$$D_{c,d}(\boldsymbol{\sigma}_1, \ldots, \boldsymbol{\sigma}_s) = \left[\left[\left[\sum_{k=1}^{s}\left(\sum_{i=1}^{S}\sigma_{k,i}z_{k,i}\right)\right]\right]_2 + \sum_{k=1}^{s}\left(\sum_{i=1}^{S}\sigma_{k,i}\left[y_{k,i}\right]_2\right)\right]_2, \quad (2)$$

where $y_{k,i} = [c \cdot x_{k,i}]_d$, and $z_{k,i}$ is the approximation to $p = \lceil \log_2(s+1) \rceil$ fractional bits of $y_{k,i}$. This function is now a low degree polynomial in the elements of $\boldsymbol{\sigma}$ that can be homomorphically executed, so the helper data for bootstrapping is built as $HD_{Bstr(w,r)} = (\boldsymbol{X}, \{[\![\sigma_{k,i}]\!]_{(d,r)}\}_{k,i})$.[3]

In order to produce a reencryption function from a key (d_1, w_1, r_1) encrypting a ciphertext $c = [\![b]\!]_{(d_1,r_1)}$ to a different key (d_2, w_2, r_2), we can follow a similar procedure and represent the decryption function for the first secret key (d_1, w_1) as a circuit realizable with encryptions under the second public key (d_2, r_2). Then, we have to generate an $s \times S$ random matrix $\boldsymbol{X}^{(1,2)}$ with coefficients in $\mathbb{Z}_{d_2}$ and a characteristic binary vector $\{\sigma_{k,i}^{(1,2)}\}_{k,i}$ such that $\left[\sum_{k=1}^{s}\sum_{i=1}^{S} x_{k,i}^{(1,2)}\sigma_{k,i}^{(1,2)}\right]_{d_2} = w_1$. Then, the characteristic vector coefficients must be encrypted with (d_2, r_2) and, analogously to (2), we get the bootstrap-based reencryption function:

[3]Gentry and Halevi propose several mechanisms for reducing the size of the helper data; we refer the reader to [10] for further details.

TABLE I
EVALUATION FIGURES FOR THE BOOTSTRAP-BASED REENCRYPTION

Lattice size	512	2048	8192
Size of d [kB]	23.9	95.8	384.3
S	512	512	547
R	2^{51}	2^{51}	2^{204}
Helper data size [MB]	8.28	33.2	137.3
Max helper data size [MB]	175.6	703.2	3012.4
Average execution time [s]	0.4	1.6	9.3

$$\mathsf{Rec}(c=[\![b]\!]_{(d_1,r_1)}, d_1, d_2, [\![\boldsymbol{\sigma}_1^{(1,2)}]\!]_{(d_2,r_2)}, \ldots, [\![\boldsymbol{\sigma}_s^{(1,2)}]\!]_{(d_2,r_2)}) = \left[\left[\left[\sum_{k=1}^{s}\left(\sum_{i=1}^{S}[\![\sigma_{k,i}^{(1,2)}]\!]_{(d_2,r_2)} z_{k,i}\right)\right]\right]_{d_2} + \sum_{k=1}^{s}\left(\sum_{i=1}^{S}[\![\sigma_{k,i}^{(1,2)}]\!]_{(d_2,r_2)}\,[y_{k,i}]_2\right)\right]_{d_2}, \quad (3)$$

where $y_{k,i} = \left[c \cdot x_{k,i}^{(1,2)}\right]_{d_2}$, and $z_{k,i}$ is the approximation to $p = \lceil \log_2(s+1) \rceil$ fractional bits of $y_{k,i}$. This function produces now a refreshed encryption of b under (d_2, r_2), therefore achieving at the same time reencryption and refreshing of the cipher. The associated helper data would be $HD_{(d_1,w_1)\to(d_2,r_2)} = \left(\boldsymbol{X}^{(1,2)}, \{[\![\sigma_{k,i}^{(1,2)}]\!]_{(d_2,r_2)}\}_{k,i}\right)$.

A. Performance evaluation

We have implemented the example proposed bootstrap-based reencryption mechanism in C++ using NTL 6.1.0 (www.shoup.net/ntl/) and GMP 6.0.0 (gmplib.org) in order to test its execution efficiency and the size of the helper data. All experiments were performed on an Intel Core i5 4670 computer with 20 GB of RAM running Linux. The results are shown in Table I for several values of the lattice size, $s = 15$ and the default security parameters recommended in [10]. We have applied the size reduction techniques in [10] consisting in choosing the elements of the matrix $\boldsymbol{X}$ as a multiplicative series with a factor R (a power of 2), and coding the elements of $\boldsymbol{\sigma}$ as a quadratic function, therefore reducing the storage needs for $\boldsymbol{X}$ and $\boldsymbol{\sigma}$, and efficiently generating the whole matrices when needed. Table I shows both the full size of the helper data ("Max helper data size") and the reduced size ("Helper data size"). With parameters $n = 2048$, $S = 512$, $R = 2^{51}$ (short term security), it takes around 1.5 seconds to perform one reencryption (Eq. (3)) for one ciphertext. Therefore, compared to state-of-the art GGH-based cryptosystems, this primitive is useful as an offline procedure for reencrypting the input data to the server's common operation key, and reencrypting the output data to the receiver user key for the presented private multi-user computing protocol.

VI. CONCLUSIONS AND FUTURE WORK

This work leverages Gentry's bootstrapping operation as a proxy reencryption mechanism for private multi-key environments. We have provided an example implementation of this primitive for the Gentry-Halevi cryptosystem, and presented the execution times and helper data sizes for several lattice dimensions, showing that it can be a useful primitive for proxy reencryption in current GGH-based fully homomorphic cryptosystems. It must be noted that this primitive is not restricted to the GGH family, but it is also conceptually applicable to other lattice-based cryptosystems built upon Learning With Errors (LWE) and related problems. In fact, as hinted by Gentry, by applying the same philosophy the bootstrapping can be generalized also to refreshing a cipher between different cryptosystems or different plaintext sizes with the adequate choice of parameters.

Additionally, we have also devised a secure protocol for the scenario of private multi-user computing with semi-honest non colluding adversaries, employing the bootstrap-based proxy reencryption primitive. This protocol could be generalized to the malicious party scenario by applying a similar approach as López-Alt *et al.* [7], using zero-knowledge proofs and collision-resistant hash functions.

ACKNOWLEDMENTS

This work was partially funded by the Spanish Government and the ERDF under project TACTICA, by the Spanish Government under project COMPASS (TEC2013-47020-C2-1-R), and by the Galician Regional Government and the ERDF under projects "Consolidation of Research Units" (GRC2013/009), REdTEIC (R2014/037) and AtlantTIC.

REFERENCES

[1] J. R. Troncoso-Pastoriza and F. Pérez-González, "Secure signal processing in the cloud: enabling technologies for privacy-preserving multimedia cloud processing," *IEEE Signal Processing Magazine*, vol. 30, no. 2, pp. 29–41, March 2013.

[2] M. van Dijk and A. Juels, "On the impossibility of cryptography alone for privacy-preserving cloud computing," in *USENIX Workshop on Hot Topics in Security, HotSec'10*, Washington DC, USA, aug 2010.

[3] C. Aguilar-Melchor, S. Fau, C. Fontaine, G. Gogniat, and R. Sirdey, "Recent advances in homomorphic encryption: A possible future for signal processing in the encrypted domain," *IEEE Signal Processing Magazine*, vol. 30, no. 2, pp. 108–117, March 2013.

[4] S. Bugiel, S. Nürnberger, A.-R. Sadeghi, and T. Schneider, "Twin Clouds: Secure Cloud Computing with Low Latency," in *Communications and Multimedia Security*, ser. LNCS, B. De Decker, J. Lapon, V. Naessens, and A. Uhl, Eds. Springer, 2011, vol. 7025, pp. 32–44.

[5] A. Peter, E. Tews, and S. Katzenbeisser, "Efficiently Outsourcing Multiparty Computation Under Multiple Keys," *IEEE Trans. on Information Forensics and Security*, vol. 8, no. 12, pp. 2046–2058, Dec 2013.

[6] E. Bresson, D. Catalano, and D. Pointcheval, "A simple public-key cryptosystem with a double trapdoor decryption mechanism and its applications," in *Advances in Cryptology - ASIACRYPT 2003*, ser. LNCS, C.-S. Laih, Ed. Springer Berlin Heidelberg, 2003, vol. 2894, pp. 37–54.

[7] A. López-Alt, E. Tromer, and V. Vaikuntanathan, "On-the-fly Multiparty Computation on the Cloud via Multikey Fully Homomorphic Encryption," in *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*, ser. STOC '12. ACM, 2012, pp. 1219–1234.

[8] G. Ateniese, K. Fu, M. Green, and S. Hohenberger, "Improved Proxy Reencryption Schemes with Applications to Secure Distributed Storage," *ACM Trans. on Inform. and System Security*, vol. 9, no. 1, pp. 1–30, Feb. 2006.

[9] C. Gentry, "A fully homomorphic encryption scheme," Ph.D. dissertation, Stanford University, 2009, crypto.stanford.edu/craig.

[10] C. Gentry and S. Halevi, "Implementing Gentry's Fully-Homomorphic Encryption Scheme," in *EUROCRYPT*, ser. LNCS, vol. 6632. Springer, 2011, pp. 129–148.

[11] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, G. Pelosi, and P. Samarati, "Encryption-Based Policy Enforcement for Cloud Storage," in *IEEE 30th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, June 2010, pp. 42–51.