DE GRUYTER
OPEN

TATRA MOUNTAINS
Mathematical Publications

DOI: 10.1515/tmmp-2016-0039
Tatra Mt. Math. Publ. **67** (2016), 191–203

OBLIVIOUS LOOKUP-TABLES

Markus Stefan Wamser — Stefan Rass — Peter Schartner

ABSTRACT. Evaluating arbitrary functions on encrypted data is one of the holy grails of cryptography, with Fully Homomorphic Encryption (FHE) being probably the most prominent and powerful example. FHE, in its current state is, however, not efficient enough for practical applications. On the other hand, simple homomorphic and somewhat homomorphic approaches are not powerful enough to support arbitrary computations.

We propose a new approach towards a practicable system for evaluating functions on encrypted data. Our approach allows to chain an arbitrary number of computations, which makes it more powerful than existing efficient schemes. As with basic FHE we do not encrypt or in any way hide the function, that is evaluated on the encrypted data. It is, however, sufficient that the function description is known only to the evaluator. This situation arises in practice for software as a Software as a Service (SaaS)-scenarios, where an evaluator provides a function only known to him and the user wants to protect his data. Another application might be the analysis of sensitive data, such as medical records.

In this paper we restrict ourselves to functions with only one input parameter, which allow arbitrary transformations on encrypted data.

1. Introduction

Encryption is an effective way to hide information, but almost equally effective is it in preventing its manipulation. With known exceptions like homomorphic encryption that permit restricted forms of plaintext processing, the question whether arbitrary processing of this kind can be done under the disguise of an encryption has long remained an unresolved issue. The breakthrough came in 2009 with Gentry's first proposal [5] of a fully homomorphic encryption that provides a trapdoor oneway ring homomorphism. Ever since then, a tremendous lot of (successful) work has been done on improving the construction and designing different schemes based on the novel ideas underlying the first one.

2010 Mathematics Subject Classification: Primary 94A60; Secondary 68M07, 68P25, 68Q05, 68W99, 94C10.
Keywords: OLUT, homomorphic encryption, function evaluation.

In this work, we explore the problem from the same starting point as before 2009, namely the question whether arbitrary data processing is possible based on group homomorphic encryption only. The herein proposed concept of an *Oblivious Lookup Table (OLUT)*[1] is a simple such concept that allows the evaluation of an arbitrary function $f: X \to Y$, given an encrypted value $Enc(x)$ to return an encrypted result $Enc\big(f(x)\big)$, without ever needing to decrypt the input or an intermediate result. This construction works under the premise that the cardinalities of X and Y are identical and sufficiently small to be tabulated (as the evaluation takes polynomial time in the size of X). Our idea is based on the fact that every function on a finite field can be expressed as a polynomial over that field.

Possible applications of OLUTs are simple SaaS-applications, such as data transformation followed by aggregation, which is a setting commonly found in sensor networks. We also demonstrate how OLUTs can be used to solve Yao's Millionaire's Problem in the honest player setting.

The paper is structured as follows: In Section 2 we first develop OLUTs on unencrypted data. We then show that these carry over directly to the encrypted setting by giving an evaluation rule in Section 3. In Section 4 we have a look at the security implications of OLUTs. Next, Section 5 considers different homomorphic schemes as underlying primitives. We continue in Section 6 with considering efficiency and parameters for practical implementations before extending our approach to a transform-then-aggregate scheme in Section 7. Finally, we sum up our results and give suggestions for further research (Sections 8 and 9).

2. Oblivious lookup tables

We consider the following setting: let $f : X \to Y$ be a mapping between finite sets. Assume that the sizes of X and Y are sufficiently small to permit a specification of f via a lookup-table.

We want to construct an *Oblivious Lookup Table*, that is a set of values depending on X and Y, and a single generic evaluation function that takes an input $x_i \in X$, properly encoded in a special form and with the OLUT and the generic evaluation function produces an output $y_i \in Y$. The evaluation operation is oblivious to the actual value of x_i and computes $f(x_i)$ for all admissible inputs, using the same values from the OLUT upon each invocation. To keep our OLUT compact, we can safely assume that all values are used.

[1]We are fully aware that the Finnish language already claims the word "*olut*" for something more important and apologise for any confusion that this may cause.

While this seems moot at first sight, it has the advantage that the computational effort for evaluation is independent of the function f and is determined only by the cardinalities of X and Y.

To be more specific, let $p = 2q+1$ be a safe prime (i.e., q is a Sophie-Germain-prime), and let $\mathbb{G} \subset \mathbb{Z}_p$ denote a q-order subgroup generated by some element $g \in \mathbb{Z}_p$.

Let $X = \{x_1, \ldots, x_n\} \subseteq \mathbb{G}$ be an enumeration of (distinct) values to be looked up. As per our requirement a lookup of x_i needs to combine x_i with all entries of the OLUT as determined by the evaluation function, which then returns $f(x_i) = y_i$. A natural approach to this problem is to encode x_i into a vector v_i that has as many entries as the OLUT, which we will also write as a vector $\vec{\ell}$ with entries from a set W. The evaluation function is then simply the inner product of vectors. Mappings $X \to W$ and $W \to Y$, which are independent of f, are also applied by the evaluation function as needed.

Writing down the vectors v_i for all $x_i \in X$ we obtain a matrix V with the v_i as rows. Collecting the y_i into a vector $\vec{y}$ we get

$$V \cdot \vec{\ell} = \vec{y} \tag{1}$$

and to find $\vec{\ell}$ from $\vec{y}$ we simply need V to be invertible, namely a square matrix of full rank $|X| = n$. We therefore have to expand each value x_i into a vector v_i of length n, such that the v_i are all linearly independent. We can easily fulfil these requirements by defining V as a Vandermonde-Matrix

$$V = \begin{pmatrix} 1 & x_1 & x_1^2 & \ldots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \ldots & x_2^{n-1} \\ \vdots & & & & \vdots \\ 1 & x_n & x_n^2 & \ldots & x_n^{n-1} \end{pmatrix},$$

so an arbitrary input x_i is represented by the vector

$$\vec{v_i} = \left(x_i^k\right)_{k=1}^n = \left(1, x_i, x_i^2, \ldots, x_i^{n-1}\right).$$

Evaluation is then $\vec{v_i} \cdot \vec{\ell}$ and the result of such an evaluation is a single scalar value. To allow chaining of OLUT-lookups this value has to be expanded again into a Vandermonde-vector, either by computing the powers directly or by constructing an OLUT for each component of the vector. This will result in a series $\ell_0 \ldots, \ell_n$ of OLUT-vectors of dimension n, comprising the columns of the OLUT-matrix L and evaluation is the the inner product of the vector v_i with the matrix L.

While the latter approach clearly requires more computational effort, it is one of the only feasible ones when it comes to encrypted data. The matrix $V = \mathrm{diag}(x_1, \ldots, x_n)$ is a possible alternative representation. By virtue of the

semantic security of the underlying encryption, ciphertexts embodying a zero are indistinguishable from encryptions of a random value. This slightly improves the efficiency of encryption, as no powers of the plaintext need to be computed. On the other hand, decryption requires an additional application of the all-one--OLUT, whenever the result is returned in the vector format.

2.1. Proper encoding of X and Y

As the attentive reader has already recognised, we silently assumed that the invertibility of the matrix V is not hampered by the selection of the modulus, e.g., elements are from $\mathbb{Z}_q$. However, to be more general, especially for the case when X is just a set of subsequent integers, we need to map X into a proper (sub)group. Such a mapping can be realised by selecting an element h of (additive) order q and a matching function $X \mapsto \langle h \rangle$. As X is of sufficiently small size to be tabulated, computing the mapping and its inverse is possible by generating a simple substitution table.

To ease matters concerning the regularity of the matrix V, it is convenient to work in a group of prime order. This can either be a subgroup of an elliptic curve group (offering the additional appeal of being already additively homomorphic), or we can work in a qth order subgroup of $\mathbb{Z}_p$, when $p = 2q + 1$ is a safe prime (which is independently required to avoid the small subgroup attack). In this setting, Damgård's ElGamal (DEG) encryption can be made additively homomorphic by encrypting the commitment h^m instead of m, when the order of h in $\mathbb{Z}_p$ is q. Note that h and g can be, but need not be the same elements as long as they are from the same subgroup.[2] For simplicity, we will just use g throughout the paper.

3. Application to encrypted data

We wish to re-use the OLUT computed from unencrypted values for encrypted inputs. This leads to two demands on the employed encryption scheme: we need to be able to multiply ciphertexts with unencrypted values and we need to be able to add ciphertexts. Both can be matched by any homomorphic encryption scheme where the homomorphic operation leads to addition of the plaintexts.

For illustrative purposes, we will use a modification of Damgård's ElGamal DEG [4] encryption. Alternatives are discussed in Section 5. The presentation of DEG is taken from [10]. In our case we have a group $\mathbb{G} = \langle g \rangle$ of order $q = (p-1)/2$ and k is a security parameter.

[2]Otherwise $h^0 g^{sk \cdot r}$ and $h^1 g^{sk \cdot r}$ might be distinguishable, e.g., by their Jacobian symbol.

Key generation $Gen(1^k)$**:** Randomly select a pair of secret keys sk_1, $sk_2 \xleftarrow{\$} \mathbb{Z}_q$. Compute the public keys $pk_1 = g^{sk_1}$ and $pk_2 = g^{sk_2}$. Let $sk = (sk_1, sk_2)$, $pk = (pk_1, pk_2)$ and publish pk.

Encryption $Enc_{pk}(m;\cdot)$**:** Return $\perp$ if $m \notin \mathbb{G}$. Otherwise, select a random $r \xleftarrow{\$} \mathbb{Z}_q$ and set $Enc_{pk}(m;r) \leftarrow \left(g^r, pk_1^r, m \cdot pk_2^r\right)$.

Decryption $Dec_{sk}(c)$**:** Parse $c = (c_1, c_2, c_3)$ and return $\perp$ if any of the $c_i \notin \mathbb{G}$ or if $c_2 \neq c_1^{sk_1}$. Otherwise return $Dec_{sk}(c) \leftarrow c_3 / c_1^{sk_2}$

The cryptosystem (Gen, Enc, Dec) as described above enjoys a multiplicative-multiplicative homomorphic property:

$$Enc_{pk}(m_1) * Enc_{pk}(m_2) = Enc_{pk}(m_1 \cdot m_2),$$

where "$*$" is component-wise multiplication of the ciphertexts and "$\cdot$" is the simple multiplication in $\mathbb{G}$.

For our purposes, however, we want

$$Enc_{pk}(m_1) * Enc_{pk}(m_2) = Enc_{pk}(m_1 + m_2)$$

with $+$ being addition in $\mathbb{Z}_q = \mathbb{G} \cup \{0\}$.

Therefore we slightly alter Enc and Dec. We do not encrypt m directly, but a commitment for m into $\mathbb{G}$. That is, ciphertexts are now produced by

$$Enc_{pk}(m;r) \leftarrow \left(g^r, pk_1^r, g^m \cdot pk_2^r\right) \tag{2}$$

and decryption is given by

$$Dec_{sk}(c) \leftarrow \log_g\left(c_3 / c_1^{sk_2}\right). \tag{3}$$

All other conditions and computations are left untouched. One might object that computing the discrete log in $\mathbb{Z}_q^*$ is hard, but as we confine our message space to X and the nth powers thereof, we can simply tabulate the $O(n)$ possible values of g^m and incorporate an inverse lookup in the decryption step. It is easy to see that the desired homomorphy property holds and both required operations can be realised. Addition of two messages is accomplished by multiplying the corresponding ciphertexts component-wise and multiplications with a public value corresponds to taking the power of the ciphertext (component-wise), which is shorthand for multiplying a ciphertext repeatedly with itself.

Finally, OLUTs require a specially crafted vector of values instead of a single value for input, as established in Section 2. Therefore we encrypt a value m into a vector of encrypted commitments by

$$\widetilde{Enc}_{pk}(m) \;=\; \left(Enc_{pk}\left(g^{m^0}\right), Enc_{pk}\left(g^{m^1}\right), Enc_{pk}\left(g^{m^2}\right), \ldots, Enc_{pk}\left(g^{m^{n-1}}\right)\right)$$

and for decryption we simply drop all but the second entry of the vector.

We can now reuse the OLUTs we had computed for unencrypted values. Let x_i be the input and $\vec{\ell}$ one column from an OLUT-matrix for a function f. To evaluate f, let the encrypted input value x_i be given as $c = \widetilde{Enc}_{pk}(x_i)$. Then, we can compute the lookup value $Enc_{pk}\big(f(x_i)\big)$ as

$$\begin{aligned}\prod_{k=1}^{n} c_k^{\ell_k} &= \prod_{k=1}^{n} Enc_{pk}\big(g^{x_i^{k-1}}\big)^{\ell_k} = \prod_{k=1}^{n} Enc_{pk}(g^{v_{ik}})^{\ell_k} \\ &= Enc_{pk}\big(g^{\sum_{k=1}^{n} v_{ik}\ell_k}\big) = Enc_{pk}\big(g^{f(x_i)}\big). \end{aligned} \tag{4}$$

Adoptions for other homomorphic schemes are discussed in Section 5.

If two or more OLUTs shall be chained, a new vector $\widetilde{Enc}_{pk}\big(f(x_i)\big)$ has to be produced. As only linear combinations of values can be produced by multiplications of available ciphertexts, all n columns of the OLUT-matrix L need to be evaluated to generate encryptions of the powers of $f(x_i)$, as already mentioned at the end of Section 2. There is one caveat to circumvent: As the entries of the OLUT now resemble exponents, by virtue of Fermat's little theorem, inversion of V would occur modulo $p-1$. Working in a prime-order subgroup (as stated in Subsection 2.1) avoids the problem, since the inversion is modulo the order of g, which is prime.

4. Security

The analysis of the security of a function evaluation via OLUT must be divided into confidentiality of the input and outputs, and the confidentiality of the function itself. More specifically, the former refers to the attacker's inability to deduce anything about the input or output based on a given ciphertext (input). The second requirement refers to a scenario where the attacker seeks to learn all input/output pairs that constitute the function f.

4.1. On hiding the inputs and outputs

Since our construction is completely generic, the (concrete) security of an OLUT is determined by the security of the underlying encryption. For generality, let us assume that the attacker has oracle access to the OLUT[3] and seeks to disclose a given ciphertext (input or output to/from the OLUT).

Essentially, the attacker can thus mount a chosen plaintext or chosen ciphertext attack, and depending on what the OLUT does, the respective security games are defined exactly as for conventional encryption, with the only addition of allowing oracle access to the OLUT under consideration.

[3]In fact, the attacker may just produce his own OLUT.

Remember that application of an OLUT is basically just a series of multiplications of ciphertexts, independent of the encrypted values. For a given homomorphic public-key system IND-CPA resp. IND-CPA1 security is preserved for the application of the homomorphic property (which also rules out IND-CPA2 security). Security of OLUT application can then be trivially argumented using induction over the number of multiplications.

4.2. On hiding $f(x)$

Oblivious Lookup Tables, as presented in this paper, do not hide the function they represent, and this is not easy to achieve. This is in contrast to FHE, which can evaluate encrypted functions on encrypted inputs without decrypting either, e.g., by using Valiant's Universal Circuit [8] as evaluation procedure and both, function and data as encrypted inputs.

Note that anyone who is able to query the function with is own inputs can learn the function at least partially, independently from the evaluation method.

Assume there is some way to obfuscate OLUTs. Now assume an attacker has (oracle) access to such an obfuscated OLUT. Necessarily, as the domain and image of the function realised by the OLUT are sufficiently small to allow exhaustive enumeration, an attacker can query the OLUT on all admissible inputs, thereby learning the function that was supposed to be hidden. Note that for the purpose of these queries, the attacker can generate the required inputs using his own key pair, as the OLUT is oblivious to the actual encryption of the plaintexts. The attacker can then decrypt the ciphertexts to obtain y. Further, the attacker may use the evaluation formula for unencrypted data, hereby forgoing the issue of encryption completely.

5. OLUTs from different homomorphic encryptions

Other homomorphic schemes than our variant of OLUT bring advantages as well as disadvantages. We list some exemplary schemes that can be used as replacement for DEG. We stress again that, once precomputed, OLUTs can be reused as long as the group structures are preserved.

Benaloh, Paillier and Damgård-Jurik: As these schemes are additively homomorphic, the use of commitments is not necessary and the scheme can be employed as-is. Evaluation is given by

$$\prod_{k=1}^{n} c_k^{\ell_k} = \prod_{k=1}^{n} Enc_{pk}\big(x_i^{k-1}\big)^{\ell_k} = \prod_{k=1}^{n} Enc_{pk}(v_{ik})^{\ell_k}$$
$$= \prod_{k=1}^{n} Enc_{pk}(v_{ik}\cdot \ell_k) = Enc_{pk}\left(\sum_{k=1}^{n} v_{ik}\ell_k\right) = Enc_{pk}\big(f(x_i)\big).$$

Not all variants of these schemes are equally suitable: for example, Paillier--Pointcheval encryption is ruled out for lack of homomorphy (due to its security against adaptive chosen-ciphertext attacks); Paillier and Damgård--Jurik offer the same level of security (IND-CCA1) as Damgård's ElGamal (see [2] for proofs).

Goldwasser-Micali: The scheme of Goldwasser-Micali has an interesting homomorphic property:

$$Enc(m_1) * Enc(m_2) = Enc(m_1 \oplus m_2),$$

where $\oplus$ is bit-wise exclusive-or of the values. While this may at first seem counter-intuitive, this property admits convenient vector-algebra within the encryption as follows: observe that the XOR-operation on n plaintext bits is effectively an addition within $GF(2^n)$. Thus, given $X = \{x_1, x_2, \ldots \ldots, x_n\}$ where each x_i is encoded using n bit again, we just need to make sure that the 0-1-matrix composed row-wise from the binary representations is invertible in the Galois field with characteristic two. In that case, we can work out the OLUT as usual, $\vec{\ell} = V^{-1} \cdot \vec{y}$, leaving the evaluation equation (4) unchanged, except for an additional nice effect: since the OLUT is itself only a 0-1-vector, there are no exponentiations and only a reduced number of multiplications (given by the Hamming weight of $\vec{\ell}$) required to evaluate (4). A downside is the fact that the Goldwasser-Micali scheme only offers IND-CPA security (see [6]).

Finally, schemes based on Elliptic Curve Cryptography (ECC) can also be employed. For example, an ECC-based implementation of DEG could deliver multiplicative-additive homomorphy directly, alleviating the need for commitments. The same holds for Code-based cryptography [1], which can be employed, when Post-Quantum security is sought.

6. Efficiency considerations

In this section we will assess the computational effort needed to evaluate chainable OLUTs and give boundaries for possible tradeoffs through parallelization. We will also discuss another optimisation technique from the literature.

To obtain an output from an OLUT-lookup, that can be used as input for another lookup, n calls to the evaluation function are required. Those are independent and therefore can be computed in parallel. Each call to the evaluation function leads to n modular exponentiations and $n-1$ modular multiplications.

A strongly parallelized architecture would compute $O(n^2)$ modular exponentiations in parallel before doing a tree-like reduction for multiplication. This requires $O(n)$ parallel multipliers. The total computation time is then $O(1)$ exponentiations and $O(\log n)$ multiplications, combined $O(\log n)$ multiplications in $\mathbb{Z}_p$. The other extreme is an architecture that works strictly sequential. It operates with $O(1)$ exponentiations and multiplication modules and has a runtime of $O(n^2)$ exponentiations and $O(n)$ multiplications.

Both cases illustrate that for practical purposes n should be reasonably small (e.g., 2^8), as the runtime grows exponentially with the bit-length of the inputs. We suggest that slicing-techniques should be employed on an algorithmic level, whenever possible.

The evaluation function has a very characteristic structure: first all factors of the product are taken to some power, then all the results are multiplied into a single value. This can be sped up, by merging the multiplication process with the exponentiation process. This batch-exponentiation technique is due to [12]. Assume we are employing the square-and-multiply approach for computing the powers. Then, instead of squaring each factor independently, we square the accumulator. In the multiply step, we multiply each factor into the accumulator depending on the value of the exponent-bit at the respective position. This effectively brings the evaluation complexity down to $O(\log n)$ modular squarings and $O(n \log n)$ modular multiplication, together $O(n \log n)$ modular multiplications.

7. Example applications

Data Aggregation: Oblivious Lookup Tables allow for outsourcing of computations with aggregation in a restricted sense, as shall be demonstrated with a scenario based on aggregating sensor data. For this we assume a set S of sensors that take measurements. Each sensor encrypts the measurement results as a Vandermonde-vector as explained in Section 3. The encrypted values are then sent to a central server that may apply different OLUTs on the received values, e.g., for normalising the scale of measurements or applying calibration values. Finally, this server can aggregate all values into a single value by computing an arbitrary linear combination (which is equivalent to applying an OLUT) of the encrypted values. The encrypted result can then be decrypted by the data-owner at convenience. Note that after aggregation it is not possible to apply an OLUT, since it is not possible to generate the required Vandermonde-vector through the application of OLUTs.[4]

[4]OLUTs represent (weighted) linear combinations of values and can therefore not produce linearly independent values.

Yao's Millionaire's Problem: OLUTs also provide a simple solution to Yao's well-known Millionaire's problem for two honest participants: suppose that Alice and Bob (both being rich), prepare an OLUT with the following function: let a be Alice's wealth (for Bob's view of the protocol a has to be replaced by b), then the OLUT implements the following mapping on the input $x\colon x \leq a \mapsto -1$ and $x > a \mapsto +1$. Bob sends his encrypted value to Alice, who applies the OLUT and returns the encrypted result to Bob. After decryption, Bob learns whether he is richer or not.[5]

Simultaneously, Bob can prepare his OLUT and process the encrypted input from Alice. Obviously this protocol relies on both parties being honest in encrypting their wealth and producing the OLUTs.

Yao's Millionaire's Problem can be used to solve far more problems. For a detailed discussion see [13].

8. Extensions and open problems

OLUTs in the way as presented here already show that simple problems, essentially all those that require only logarithmic time to solve, are computable by group homomorphic encryption only. To prove this, let f be a function that can be evaluated on an input w of length n in $O(\log n)$ space. That is, there is a Turing machine M that upon an initial configuration including the word w carries to completion in no more than $2^{O(\log n)} = O(n^{\alpha})$ for some α, i.e., polynomially many steps (as the number of intermediate configurations of M is polynomially bounded). It is then a simple matter to define an OLUT that computes exactly the sequence of configurations that takes M into the halting state, so as to compute the function f. Summarising, we have proven

Theorem 1. *Any function that can be computed in logarithmic space, can also be computed under disguise of a group-homomorphic encryption.*

Extending this result to functions that can be computed in polynomial time calls for an extension of OLUTs to two inputs (the necessity to extend the construction is not surprising, especially in light of related results like [9]). To this end, a straightforward possibility would be using an encryption that can do multiplications and additions, which is essentially equivalent to FHE. As a further benefit, given that we could compute both, $Enc_{pk}(x \cdot y)$ and $Enc_{pk}(x + y)$, from $Enc_{pk}(x), Enc_{pk}(y)$ we would get two improvements to our OLUT construction: first, we could compute the full vector $\left(Enc_{pk}(x^i)\right)_{i=0}^{n-1}$ from $Enc_{pk}(x)$ only, but also compute any function whose evaluation complexity is polynomial

[5]If Alice returned a special value for equality Bob would be able to learn Alice's wealth in the equality case, which is a stronger result than just learning whether he is richer or not.

in time (simply define OLUTs to perform all the intermediate calculations). Approaching FHE from this new direction offers some appealing insights: first, note that we do not even require homomorphy w.r.t. multiplications, as *any* nonlinearity (giving a precomputable, but not necessarily meaningful result) would already suffice to create an invertible matrix V, suitable to define an OLUT. For example, we can define an OLUT that uses a 4×4-matrix V to resemble a universal NAND-gate, from which any boolean circuit (and thus function) can be constructed. In essence, we can thus state that something like an "almost--homomorphic encryption", i.e., an encryption that is homomorphic w. r. t. addition and furthermore allows to compute an arbitrary nonlinear function on its input, in connection with an OLUT is the same as FHE (and maybe a much simpler way to this end). Searching for a separation of these seemingly simpler cryptographic primitives from FHE—or proving that there is no separation—and discussing their relation to shift-type homomorphic encryption [3] is an interesting open problem.

From another viewpoint, OLUTs can be applied on top of FHE or Multiparty Computation MPC as both allow addition and multiplication over encrypted data. In this case the straightforward adoption leads to the matrix V_{xy} being the Kronecker product of V_x and V_y. W. l. o. g. $|V_x| = |V_y| = n$ can be assumed. Then every entry of V_{xy} can be computed with $O(\log n)$ (homomorphic) multiplications. This trivially gives an upper bound on the required depth of levelled FHE or rounds in Multiparty Computation (MPC) to compute any function on $V_x \times V_y$, including Valiant's Universal Circuit, the Universal Arithmetic Circuit (UAC) of [11] and the S-universal circuits of [7].

Finally, a connection between OLUTs and Private Information Retrieval or 1-out-of-n-Oblivious Transfer can be observed. The interesting part is the reversed bandwith requirement compared to other Private Information Retrieval PIR schemes. The request consists of a ciphertext-vector of the size of the database which is used as the OLUT. Only a single ciphertext is then returned. ForOblivious Transfer (OT), the OLUT is destroyed after computing the response, for Private Information Retrieval (PIR), it is kept.

9. Conclusion

We have presented a method to compute arbitrary transformations (in the sense of unary functions) on encrypted data. Given reasonable input sizes, our approach is computationally efficient and practicable.

We also demonstrated that our approach is independent from a concrete instantiation of the underlying homomorphic encryption scheme. Our OLUTs can even be computed before the underlying scheme is selected.

For practical use a transform-then-aggregate scheme is often not enough. A way to enable oblivious lookups for functions with two (or more) parameters is therefore an obvious next step in research.

List of acronyms

DEG: Damgård's ElGamal

ECC: Elliptic Curve Cryptography

FHE: Fully Homomorphic Encryption

MPC: Multiparty Computation

OLUT: Oblivious Lookup Table

OT: Oblivious Transfer

PIR: Private Information Retrieval

SaaS: Software as a Service

UAC: Universal Arithmetic Circuit

REFERENCES

[1] ARMKNECHT, F.—AUGOT, D.—PERRET, L.—SADEGHI, A.-R.: *On constructing homomorphic encryption schemes from coding theory,* Cryptology ePrint Archive, Report 2011/309, June 2011.

[2] ARMKNECHT, F.—KATZENBEISSER, S.—PETER, A.: *Group homomorphic encryption: characterizations, impossibility results, and applications,* Cryptology ePrint Archive, Report 2010/501, 2010, `http://eprint.iacr.org/`

[3] ARMKNECHT, F.—KATZENBEISSER, S.—PETER, A.: *Shift-type homomorphic encryption and its application to fully homomorphic encryption,* in: Progress in Cryptology—AFRICACRYPT'12, 5th Internat. Conf. on Cryptology in Africa (A. Mitrokotsa and S. Vaudenay, eds.), Ifrance, Morocco, 2012, Springer-Verlag, Berlin, 2012, pp. 234–251.

[4] DAMGÅRD, I.: *Towards practical public key systems secure against chosen ciphertext attacks*, in: Advances in Cryptology—CRYPTO'91 (J. Feigenbaum, ed.), Lecture Notes in Comput. Sci., Vol. 576, Springer-Verlag, Berlin, 1992, pp. 445–456.

[5] GENTRY, C.: *Computing arbitrary functions of encrypted data*, Commun. ACM **53** (2010), 97–105.

[6] KATZ, J.—LINDELL, Y.: *Introduction to Modern Cryptography—Principles and Protocols*, Chapman and Hall/CRC Press, London, 2007.

[7] KENNEDY, W. S.—KOLESNIKOV, V.—WILFONG, G.: *Overlaying circuit clauses for secure computation*, Cryptology ePrint Archive, Report 2016/685, 2016, `http://eprint.iacr.org/2016/685`

[8] KISS, Á.—SCHNEIDER, T.: *Valiant's universal circuit is practical*, Cryptology ePrint Archive, Report 2016/093, February 2016.

[9] BOGDANOV, A.—LEE, CH. H.: *Homomorphic evaluation requires depth*, Cryptology ePrint Archive, Report 2015/1044, 2015, `http://eprint.iacr.org/`

[10] LIPMAA, H.: *On the CCA1-Security of Elgamal and Damgård's Elgamal*, Cryptology ePrint Archive, Report 2008/234, 2008, http://eprint.iacr.org/2008/234

[11] LIPMAA, H.—PAYMAN, M.—SAEED, S.: *Valiant's universal circuit: improvements, implementation, and applications*, Cryptology ePrint Archive, Report 2016/017, January 2016.

[12] OTTOY, G.—PRENEEL, B.—GOEMAERE, J.-P.—DE STRYCKER, L.: *Flexible design of a modular simultaneous exponentiation core for embedded platforms*, in: Reconfigurable Computing: Architectures, Tools and Applications (P. Brisk et al., eds.), Lecture Notes in Comput. Sci., Vol. 7806, Springer-Verlag, Berlin, pp. 115–121.

[13] SHELAT, A.—MUTHURAMAKRISHNAN, V.: *Secure computation from millionaire,* in: Advances in Cryptology—ASIACRYPT'15, 21st Internat. Conf. on the Theory and Appl. of Cryptology and Inform. Security (T. Iwata and J. H. Cheon, eds.), Auckland, New Zealand, 2015, Lecture Notes in Comp. Sci., Vol. 9452, Springer-Verlag, Berlin, pp. 736–757.

Received November 2, 2015

Markus Stefan Wamser
Lehrstuhl für Sicherheit
in der Informationstechnik
Technische Universität München
Arcisstraße 21
D–80333 München
GERMANY
E-mail: wamser@tum.de

Stefan Rass
Peter Schartner
Institut für Angewandte Informatik
Alpen-Adria-Universität Klagenfurt
Universitätsstrasse 65-67
A–9020 Klagenfurt
AUSTRIA
E-mail: stefan.rass@aau.at
peter.schartner@aau.at