Lecture Notes for Cryptographic Computing
6. Homomorphic Encryption

Lecturers: Claudio Orlandi, Peter Scholl, Aarhus University

October 23, 2023

Exercises marked with a $*$ are the most important ones and should be prioritized

This week, we look at the basics of homomorphic encryption. We will start with Paillier's cryptosystem, an additively homomorphic encryption scheme. The presentation here is somewhat reminiscent of [DJ01]. In the second half, we will see some intuition for how to build a fully homomorphic encryption scheme.

1 Paillier's Cryptosystem

Paillier can be seen as a variant of the RSA encryption scheme that works modulo N^2 instead of N, where $N = pq$ is a product of two large primes.

Paillier allows to encrypt any message in Z_N. The encryption scheme is built using two "mappings" from Z_N into Z_{N^2}. Intuitively, the first mapping takes care of the messages and ensures that the scheme is homomorphic. The second mapping takes care of the randomness and makes sure that the scheme is secure. The fact that the two mappings can be combined in Z_{N^2} makes the magic happens.

The first mapping is an additively homomorphic map from Z_N into Z_{N^2}. It works as follows:

$$\alpha(x) = (1 + xN) \mod N^2$$

Exercise 1.

(*) Verify that the mapping is homomorphic i.e.,

$$\alpha(x) \cdot \alpha(y) \bmod N^2 = \alpha(x + y \bmod N)$$

and therefore also

$$\alpha(x)^y \bmod N^2 = \alpha(x \cdot y \bmod N)$$

It is not hard to see that this mapping is not particularly secure. In fact, the mapping is invertible!

Exercise 2.

(*) Find the inverse mapping $\alpha^{-1}(\cdot)$ such that, for all elements of Z_{N^2} in the image of α it is possible to compute x. In other words, how do you compute $x \in Z_N$ from $(1 + xN) \bmod N^2$?

The second mapping that we call β also maps elements from Z_N into Z_{N^2}, and is essentially the same as RSA encryption, but modulo N^2, where the public exponent e is fixed to N.

$$\beta(r) = r^N \mod N^2$$

Exercise 3.

Verify that the mapping is multiplicatively homomorphic i.e., $\beta(x) \cdot \beta(y) = \beta(x \cdot y) \mod N^2$

The security of Paillier encryption is based on the assumption that it is hard to distinguish between uniformly random elements in Z_{N^2} and the output of $\beta(r)$ on an uniformly random r.

On the other hand it is very easy to distinguish $\beta(r)$ from random elements in Z_{N^2} if one knows the factorization of N: from the factorization of N one can compute $\phi(N) = (p-1)(q-1)$ and then one can test if a given y is in the image of β by checking if

$$y^{\phi(N)} = 1 \mod N^2$$

To see why this is the case, see that if $y = \beta(r)$ then

$$\beta(r)^{\phi(N)} = (r^N)^{\phi(N)} = r^{N \cdot \phi(N)} = r^{\phi(N^2)} = 1 \pmod{N^2} \tag{1}$$

because every element of a group raised to the group order is 1. However most element of Z_{N^2} will not give 1 if raised to $\phi(N)$!

Now we are ready to describe Paillier cryptosystem:

Key Generation: Sample primes p, q of the same length and compute $N = p \cdot q$. Using the Chinese Remainder Theorem, let $sk \in Z$ such that

$$sk = \begin{cases} 0 \mod \phi(N) \\ 1 \mod N \end{cases}$$

Output sk and $pk = N$.

Encryption: On input a message $m \in Z_N$, sample a random $r \in Z_N^*$, and output

$$C = \alpha(m) \cdot \beta(r) = (1 + mN) r^N \mod N^2$$

Decryption: Compute $m = \alpha^{-1}(C^{sk} \mod N^2) \mod N$

Exercise 4.

(*) Use the Chinese Remainder Theorem to write down a formula for the secret key, sk.
Use this, together with Exercises 1,2 and equation (1), to check that decryption works correctly.

Exercise 5. ("RSA Decrypting" β)

Let $N = pq$ where p, q are two prime numbers with the same number of bits. First, check that $e = N$ is a valid choice of a public exponent for the RSA cryptosystem. Remember that the condition is that $gcd(e, \phi(N)) = 1$. This ensures that it is possible to find d s.t.

$$d \cdot N = 1 \mod \phi(N)$$

Now, notice that it is not possible to directly decrypt β, because N is NOT coprime with $\phi(N^2)$. However, check that

$$\beta(r) \mod N = (r^N \mod N^2) \mod N = r^N \mod N$$

and therefore it is possible to invert β and recover the randomness used in Paillier encryption. This is useful in some applications.

The security of Paillier encryption follows immediately from the assumption that $\beta(r)$ is indistinguishable from a random element of Z_{N^2}.

It is straightforward to check that Paillier is additively homomorphic using Exercises 1,3. In particular:

$$\begin{aligned} E(m_1, r_1) \cdot E(m_2, r_2) &= \alpha(m_1)\beta(r_1)\alpha(m_2)\beta(r_2) \\ &= \alpha(m_1 + m_2)\beta(r_1 \cdot r_2) \\ &= E(m_1 + m_2, r_1 \cdot r_2) \mod N^2 \end{aligned}$$

Arithmetic BeDOZa-protocol: We have already seen how to replace the dealer for the Boolean BeDOZa using OT. Here we show that we can replace the dealer in BeDOZa with a two-party multiplication protocol based on Paillier cryptosystem if we are using the BeDOZa protocol over Z_N where N is the public key for a Paillier's cryptosystem such that only Alice knows the factorization of N.

Remember how multiplications work in BeDOZa: Alice has $(x_A, y_A) \in Z_N \times Z_N$; Bob has $(x_B, y_B) \in Z_N \times Z_N$; at the end Alice and Bob get random $(z_A, z_B) \in Z_N \times Z_N$ such that

$$z_A + z_B \bmod N = (x_A + x_B)(y_A + y_B) = x_A y_A + x_B y_B + x_A y_B + x_B y_A$$

Alice can compute $x_A y_A$ locally (resp. Bob $x_B y_B$), but interaction is needed to compute $x_A y_B$ and $x_B y_A$. To compute random shares of $x_A y_B$ they run the following protocol:

1. Alice sends $U = E_N(x_A)$ to Bob.
2. Bob samples a random $s_B \in_R Z_N$ and compute

   $$V = U^{y_B} E_N(-s_B) \mod N^2$$

   and sends V to Alice.
3. Alice computes $s_A = D_{\phi(N)}(V)$.

Now due to the homomorphic properties of E it holds that $s_A + s_B = x_A y_B \bmod N$. Shares of the other product $x_B y_A$ can be computed similarly.

This protocol offers security against passive adversaries: the simulator for a (passively) corrupted Bob would just set $U = E(0)$ and now the view of Bob in the real world and in the ideal world is indistinguishable because of the security of Paillier encryption scheme. If Alice is (passively) corrupted, the simulator would encrypt $V = E(s_A)$, and this is distributed identically as V in the real protocol.

Intuitively Bob cannot learn anything about x_A from the ciphertex U unless he breaks the security of Paillier cryptosystem, while Alice does not learn anything about y_B because given that s_B is uniformly random, s_A is uniformly distributed in Z_N in Alice's view. The protocol can be "compiled" for active security by throwing enough zero-knowledge proofs in the multiplication protocol (note that the ZK for this relation can be made quite efficient) and MACs to ensure that the the parties don't change their shares for the internal and output wires. See for instance [BeDOZa11] to see how this can be done.

Exercise 6. (OT from HE)

Can you build a one-out-of-two OT protocol secure against passive adversaries using Paillier encryption?
Hint 1:
Hint 2:

2 Fully-Homomorphic Encryption Scheme

Here we will review some of the fundamental ideas that allow to construct fully-homomorphic encryption schemes.

Defining FHE: Intuitively a fully homomorphic encryption scheme (FHE for short) is like a standard encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ but with an extra algorithm Eval that takes as input a vector of ciphertexts and a function f and outputs an encryption of the function applied to the plaintexts i.e., let $(pk, sk) \leftarrow \mathsf{Gen}(1^k)$ then the correctness requirement states that:

$$\mathsf{Dec}_{sk}(\mathsf{Eval}(f, \mathsf{Enc}_{pk}(x_1), \ldots, \mathsf{Enc}_{pk}(x_n))) = f(x_1, \ldots, x_n)$$

For instance, Paillier is homomorphic with respect to any function $f_{a,b}(x)$ of the form $ax + b \mod N$, as we have seen before. A fully encryption scheme is homomorphic with respect to any function f described as a circuit. In the following we will also consider a d-HE scheme which is somehow in between Paillier and FHE: a d-HE scheme allows to evaluate every function which can be expressed by a circuit with depth at most d.

Note that we need to be more restrictive about the requirements that we want from an FHE scheme, otherwise it is trivial to come up with FHE schemes. For instance, let $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be any encryption scheme and let Eval to be algorithm that appends the function f to the ciphertext. Now we can define a decryption algorithm Dec' that first runs Dec on the ciphertext and then applies the function on the decrypted value. This is a not very interesting FHE scheme. Intuitively, it should be Eval that performs the computation, not Dec!

This can be captured by asking that the work performed by the decryption function should be independent of the function f. We call this requirement *compactness*.

Efficiency is not the only problem with the "trivial" FHE scheme described above. If we want to use an FHE scheme to build a secure computation protocol, we usually need that the output of Eval hides as much information as possible about which function was computed (in the Paillier based multiplication protocol that we used before, it was crucial that Alice could not learn the "function" applied by Bob on her input i.e., a, b). We call this property *circuit privacy*. Formally: We say that an encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ has *circuit privacy* w.r.t. a function family $\mathcal{F}$ if for all $(pk, sk) \leftarrow \mathsf{Gen}(1^k)$, plaintext x and function $f \in \mathcal{F}$, it holds that

$$(sk, \mathsf{Enc}_{pk}(f(\vec{x}))) \approx_c (sk, \mathsf{Eval}_{pk}(f, \mathsf{Enc}_{pk}(\vec{x})))$$

Where $\approx_c$ as usual means that the two distributions are computationally indistinguishable. Note that this is a very strong security requirement that should hold also against the owner of the secret key!

During the lecture on Yao's protocol, we discussed how garbled circuits together with a 2-message OT gives fully-homomorphic encryption with a "reasonable" level of circuit privacy. However, FHE schemes based on Yao do not satisfy compactness.

Bootstrapping: The central idea for constructing FHE is bootstrapping. Let's assume for a second that we are given a d-HE scheme $(gen, enc, dec, eval)$ where $eval$ can evaluate any circuit with depth at most d. This means that, for instance, if one evaluates a circuit with depth $> d$ on a ciphertext, then the decryption algorithm is not guaranteed to work correctly.

In addition, assume that the decryption function dec can be described by a "shallow" circuit with depth $d' < d$. Now it is possible to construct a ∞-HE (or FHE) scheme that permits to evaluate circuits of any (polynomial) depth!

It will be convenient to introduce the notion of *level* of a ciphertext: a ciphertext has level 0 if it is the output of the enc function. A ciphertext obtained using $c' = eval_{pk}(f, c)$ has level $i + j$ where i is the level of c and j is the depth of the circuit f.

Now we can construct the FHE scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$: the key generation Gen runs $(pk, sk) \leftarrow gen(1^k)$ and defines $SK = sk$ and $PK = (pk, enc_{pk}(sk))$ i.e., we publish an encryption of the secret key under its own public key.

Note that we are making an encryption of the secret key public: A scheme that is secure even when the adversary is given an encryption of the secret key is known as a *circular secure encryption scheme*. Not every encryption scheme is secure in this case, and actually in many cases it is not a good idea to encrypt a secret key using its own public key.

Exercise 7. (Circular Security)

Construct an encryption scheme that is secure, but it is not circularly secure.
Hint : .

The encryption and decryption Enc, Dec work exactly in the same way as the original scheme enc, dec.

The key idea to construct Eval is the following technique, known as *refreshing*: Take a ciphertext C at level i. Assume that $i \leq d$ and therefore C can be correctly decrypted to $m = dec_{sk}(C)$.

Now, consider the circuit $f(x) = dec_x(C)$, that is the circuit that decrypts the ciphertext C using the input x. We have already assumed that the decryption circuit has depth $d' < d$. Note that C is *not* an argument to the circuit, and therefore can be thought as part of the description of the circuit. Now, using the fact that PK contains an encryption of sk we can compute the following ciphertext:

$$C' = eval(f, enc(sk))$$

Now C' is a ciphertext of level $d' = 0 + d'$ since $enc(sk)$ has level 0 and f has depth d'. The main point here is that the level of C' *does not* depend on the level of the original ciphertext i, and therefore if $d' < i$ then C' is of a lower level than C! Given that $d' < d$, it is possible to correctly decrypt C' and, due to the homomorphic properties

$$dec_{sk}(C') = f(sk) = dec_{sk}(C) = m$$

(the last equality holds under the previous assumption that C has level $i \leq d$).

It seems that we are back to square 1, we started with an encryption of m and we still have an encryption of m, but instead we are only one step away from the end!

We now construct an Eval function which takes any two encryptions at level $\leq d$ containing two bits a, b and outputs an encryption of the value $\neg(ab)$ at level $d' + 1$. Since the output ciphertext has level $\leq d$, it can be fed again as an input to Eval and therefore any Boolean circuit can be evaluated in this way – without any interaction!

Our new algorithm Eval takes as input two ciphertexts C_1 with $dec(sk, C_1) = a$, C_2 such that $dec(sk, C_1) = b$ and outputs C_3 such that $dec(sk, C_3) = \neg(ab)$. The algorithm Eval is evaluated by applying the circuit:

$$f(x) = \neg(dec_x(C_1) \cdot dec_x(C_2))$$

on the encryption of the secret key using $eval$. Now the new depth of f is $d' + 1 \leq d$ so decryption is still correct, and we can evaluate any Boolean circuit gate by gate by repeating this process as long as necessary.

How to build a d-HE scheme: There are several ways to build a d-HE scheme. Such schemes can be found in the literature under the name of "somewhat homomorphic encryption" or "leveled fully-homomorphic encryption". Those schemes are not only interesting to "bootstrap" to FHE, but they can also be used as they are – there are applications where it is possible to know from the start that no circuit of depth more than d will ever be computed, and then one can just use a d-HE scheme.

Just to give you an idea of how such schemes work, and where does the limitation d come from, here is a simple d-HE scheme, which was first proposed in [vDGHV10][1]:

[1] Different versions of this scheme have been proposed in the literature. The description below is very informal and only meant to give a high-level overview.

Key Generation: choose a big secret odd integer p. Choose big random integers $q_1, \ldots, q_n$ and small integers $r_1, \ldots, r_n$. The secret key is p, the public key is $(y_1, \ldots, y_n)$ where $y_i = p \cdot q_i + 2r_i$

Encryption: To encrypt a bit m, choose at random a subset S of $\{1, \ldots, n\}$ and compute (over the integers)

$$c = m + \sum_{i \in S} y_i$$

Decrypt: To decrypt compute

$$m = ((c \mod p) \mod 2)$$

Let's check that decryption works:

$$c \mod p = m + 2\left(\sum_{i \in S} r_i\right) + p\left(\sum_{i \in S} q_i\right) = m + 2\left(\sum_{i \in S} r_i\right) \mod p$$

Now if set the parameters right (e.g., the r_i's are small enough compared to p, and $|S|$ is of the right size) we can get that $2(\sum_{i \in S} r_i)$ is less than p and therefore there is no modular reduction, and

$$\left(m + 2\left(\sum_{i \in S} r_i\right) \mod p\right) \mod 2$$

is equal to m.

The security of the scheme is related to the assumption that it is hard to recover p from the public key. Note that without the error terms r_i it would be trivial to compute p by computing the *gcd* of the y_i's. However it is believed that by adding the noise it becomes hard to compute p. This computational problem is known as the *approximate GCD problem*, and no polynomial time algorithms to solve the problem are known.

Finally, it is easy to see that the scheme is homomorphic. If $c_1 = enc(m_1, S_1)$ and $c_1 = enc(m_2, S_1)$

$$c_1 + c_2 \mod p = (m_1 + m_2) + 2\left(\sum_{i \in S_1} r_i + \sum_{i \in S_2} r_i\right)$$

and that

$$c_1 \cdot c_2 \mod p = (m_1 \cdot m_2) + 2\left(m_1 \cdot \sum_{i \in S_2} r_i + m_2 \cdot \sum_{i \in S_1} r_i + \sum_{i \in S_1, j \in S_2} r_i r_j\right)$$

It is clear that every homomorphic operation (especially the multiplications!) make the noise grow significantly. One of the crucial aspects in the design of new and more efficient FHE schemes is to come up with techniques to reduce the growth of the noise due to homomorphic operations.

Parameters: The presented d-HE scheme is very simple. Unfortunately the computational assumption on which it is based is not very hard, and therefore very big parameters must be chosen. According to [CLT13], one gets "DES-like seucurity" (approximately 60 bits – this is not very secure!) using the following parameters:

- The secret p is ≈ 2000 bits long.
- The q_i's are $\approx 10^7$ bits long ($1Mb$).
- The r_i's are around 60 bits long.
- There are $n \approx 2000$ values y_i in the public key.

This means that after one encryption, the noise in a ciphertext is approximately 2^{70} – so one can only approximately perform only 4 multiplications before the noise becomes greater than p using the scheme with these parameters. Luckily for us, our favourite circuit has depth 3 only.

Exercise 8. (Mandatory Assignment)

Implement the d-HE scheme described above, and use it to implement a protocol for secure two-party computation of the *blood type compatibility* function. Using the BigInteger class in Java is probably the easiest way to get the job done. Since the goal of the exercise is to better understand the protocol (not to build a full functioning system), feel free to implement all parties on the same machine and without using network communication. For example, you could implement Alice and Bob as two distinct classes in Java and then let them interact in the following way:

```
m1 = Alice.Choose(x);
m2 = Bob.Transfer(y,m2);
z = Alice.Retrieve(m2);
```

Some remarks:

- In the first message Alice sends 3 ciphertexts for her individual bits, then Bob applies the function using his input and send back a single ciphertext containing the resulting bit to Alice who decrypts and learns the result.
- You do not have to choose parameters as large as the one mentioned before. However, you should choose parameters such that the scheme is "homomorphic enough" to evaluate the circuit. Your solution should include a short justification for your choice of parameters.
- In the encryption function you need to choose a subset S - there are several ways of doing this - argue for your choice (both in terms of correctness of the decryption and security of the encryption).

References

[HL10] Carmit Hazay, Yehuda Lindell
Efficient Secure Two-Party Protocols
http://lib.myilibrary.com.ez.statsbiblioteket.dk:2048/Open.aspx?id=300348

[BeDOZa11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, Sarah Zakarias
Semi-Homomorphic Encryption and Multiparty Computation
EUROCRYPT 2011. http://eprint.iacr.org/2010/514

[DJ01] Ivan Damgård, Mads Jurik
A Generalisation, a Simplification and Some Applications of Paillier's Probabilistic Public-Key System.
PKC 2001, available at http://www.brics.dk/RS/00/45/BRICS-RS-00-45.pdf

[vDGHV10] Marten van Dijk and Craig Gentry and Shai Halevi and Vinod Vaikuntanathan
Fully Homomorphic Encryption over the Integers
EUROCRYPT 2010 http://eprint.iacr.org/2009/616

[CLT13] Jean-Sebastien Coron and Tancrede Lepoint and Mehdi Tibouchi
Batch Fully Homomorphic Encryption over the Integers
http://eprint.iacr.org/2013/036