MIT 6.875J/18.425J and Berkeley CS276
Foundations of Cryptography (Fall 2020)

Problem Set 5: Released November 5, Due November 19

The problem set is due on **Thursday, November 19 at 10pm ET/7pm PT**. Please make sure to upload to the Gradescope course webpage by the deadline (all registered students should have access to this webpage on Gradescope). Be sure to mark on Gradescope where each problem's solution starts. Typed solutions using LaTeX are strongly encouraged (template provided on the course webpage).
Collaboration is permitted; however, you must write up your own solutions individually and acknowledge all of your collaborators.

Notation Let $\mathcal{A}(x;r)$ denote the randomized algorithm $\mathcal{A}$ run on input x, using randomness r. We denote by $x||y$ the concatenation of the strings x and y. We let $x \leftarrow \{0,1\}^n$ denote the process of sampling x uniformly at random from $\{0,1\}^n$. We denote by $[n]$ the set of integers $\{1,\ldots,n\}$.

Problem 1. Cryptosystem Design: Zerocash with Viewing

In the Zerocash cryptocurrency, each user $i \in [N]$ has a public key (address) $\mathsf{Z.pk}_i$ and secret key $\mathsf{Z.sk}_i$ where $\mathsf{Z.pk}_i$ is used to send user i coins and $\mathsf{Z.sk}_i$ allows user i to spend his coins. Recall that in a transaction (i,j,m), user i sends m coins to user j by using $\mathsf{Z.sk}_i$ and $\mathsf{Z.pk}_j$ to generate a zero-knowledge proof $\mathsf{Z.}\Pi$ that hides i, j, and m.

The beauty of zero-knowledge proofs is that they hide all information about a transaction. However, there are cases when an authority needs to see all the transactions of a user due to a subpoena, yet it cannot do so. In this problem, you will extend the Zerocash design to add a "viewing" feature. Namely, an authority with a subpoena can request a secret key for a user, and using this secret key, it should be able to view all the transactions of that user and be convinced it is viewing the correct transactions. This user loses all privacy to this authority, but other users's privacy should not be affected. Additionally, the authority should not be able to spend coins for the subpoena-ed user.

If you wish, you can use a *key-private* public key encryption scheme (though there are designs possible without it). This is a semantically secure public key encryption scheme with the added property that no PPT adversary can distinguish which public key is used to generate an encryption.

You can assume an ideal ledger that implements all verifications correctly.

You can use Zero Knowledge proofs as a black box; namely, for any NP relation $\mathcal{R} = \{(x,w)\}$, you may use $\mathsf{ZK}_{\mathcal{R}}(x,w)$ that outputs a zero-knowledge proof for the instance-witness pair $(x,w) \in \mathcal{R}$. For every Zero Knowledge proof you use, specify the statement of the proof with a similar precision to the one in class (Attempt #6 in the Zerocash lecture), or alternatively specify what x, w, and $\mathcal{R}$ are.

1.1 Describe your protocol, which should build upon the Zerocash protocol. Namely, describe any key pairs users might have (in addition to $\mathsf{Z.pk}_i, \mathsf{Z.sk}_i$), what users post on the ledger for each transaction (of the form (i,j,m)), and what verifications the ledger and the authority perform. The authority should be able to view both send and receive transactions of a subpoena-ed user. Prove your protocol's correctness; namely, that if the users follow the protocol, then the authority can see all the transactions of the subpoena-ed user.

1.2 Prove the protocol's soundness, that is, the authority cannot be fooled into accepting an incorrect or incomplete set of transactions for the subpoena-ed user.

Hint: Think about how to rely on the Zerocash proof $\mathsf{Z.}\Pi$.

1.3 Argue why in your protocol the authority cannot spend coins on behalf of the subpoena-ed user.

1.4 Argue why the privacy of Zerocash is preserved in your protocol (before any subpoenas). Then argue why all transactions not involving the subpoena-ed user remain private to the authority.≪Clarified privacy consists of two parts: privacy before subpoenas and privacy after.≫

Problem 2. Extending Paillier Encryption

Let $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ denote the Paillier encryption scheme where

- $\mathsf{Gen}(1^n)$ outputs a public key $pk = (N, g)$ and a secret key sk.
- $\mathsf{Enc}(pk, m)$ outputs a ciphertext $c \in \mathbb{Z}^*_{N^2}$.
- $\mathsf{Dec}(sk, c)$ outputs a plaintext $v \in \mathbb{Z}^*_N$.

. Recall that given $c_1 \leftarrow \mathsf{Enc}(pk, m_1)$ and $c_2 \leftarrow \mathsf{Enc}(pk, m_2)$, we can compute a ciphertext $c_{12} = c_1 \cdot c_2 \pmod{N^2}$ that encrypts $m_1 + m_2 \pmod{N}$. I.e., the Paillier encryption scheme is *additively homomorphic*.

Suppose a client has L messages $m_1, \ldots, m_L \pmod{N}$ that it wishes to store in a database. To keep this information secret, it encrypts each message using the Paillier encryption scheme and stores the ciphertexts in the database. Formally, it samples $(pk, sk) \leftarrow \mathsf{Gen}(1^n)$ and for every $i \in [L]$, samples $c_i \leftarrow \mathsf{Enc}(pk, m_i)$. It stores

$$(pk, C = (c_1, \ldots, c_L))$$

in the database and keeps sk private. Now the client forgets all the messages and ciphertexts.

Later on, the client wishes to compute a function $f(m_1, \ldots, m_L)$. For a specified $f(m_1, \ldots, m_L)$, the database will homomorphically evaluate a ciphertext c encrypting $f(m_1, \ldots, m_L)$ and send c to the client. Then the client can obtain $f(m_1, \ldots, m_L) \leftarrow \mathsf{Dec}(sk, c)$.

Since Paillier encryption is additively homomorphic, if the client stores $C = (c_1, \ldots, c_L)$ in the database, it can ask for any $f(m_1, \ldots, m_L)$ that is the sum of messages $\pmod{N}$. But the client wishes to compute more sophisticated functions. It is willing to preprocess its messages, i.e. generate many ciphertexts and store all of these ciphertexts in the database.≪During preprocessing, the client generates many ciphertexts to store as C but these do not have to include $c_1, \ldots, c_L$.≫

For each of the following kinds of functions, design a preprocessing phase to compute a set of ciphertexts C and describe how the database can use (pk, C) to compute c encrypting any $f(m_1, \ldots, m_L)$ of the specified form.

2.1 For any subset $S \subseteq [L]$ specified by the client, $f(m_1, \ldots, m_L) = \sum_{i \in S} w_i \cdot m_i \pmod{N}$ where $w_1, \ldots, w_L \pmod{N}$ are secret weights (fixed before preprocessing, known to the client but not the database).≪Clarified the secret weights are known to the client but not the database.≫

2.2 Let $d \in \mathbb{N}$ be a fixed degree bound. For any coefficients $\{a_{(d_1,\ldots,d_L)} \pmod{N}\}_{(d_1,\ldots,d_L) \in [0,d]^L}$ specified by the client,

$$f(m_1, \ldots, m_L) = \sum_{(d_1,\ldots,d_L) \in [0,d]^L} a_{(d_1,\ldots,d_L)} \cdot \left(m_1^{d_1} \ldots m_L^{d_L} \right) \pmod{N}.$$

I.e. any multivariate polynomial f (in the variables $m_1, \ldots, m_L$) where the degree in each variable is $\leq d$. In general, f consists of $(d+1)^L$ terms.≪Changed sum to $d_i \in [0, d]$ instead of $d_i \in [d]$. Clarified that f is a multivariate polynomial.≫

Next suppose the client has many messages that are small integers in $[K]$ where $K << N$. It's frustrated that storing an encryption of each message would take $\log N^2$ bits.

2.3 Design a way for the client to store many small messages in a single ciphertext (such that decrypting the ciphertext returns all the messages). How many messages can the client store?

2.4 Suppose the client has a $M \times M$ matrix of small integers in $[K]$.

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,M} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,M} \\ \vdots & \vdots & \ddots & \vdots \\ a_{M,1} & a_{M,2} & \cdots & a_{M,M} \end{pmatrix}$$

Design a way for the client to store ciphertexts such that given any $\leq M'$ rows $S \subseteq [M]$, the database can compute a single≪clarified≫ ciphertext encrypting

$$\left(\sum_{i \in S} a_{i,1}, \ldots, \sum_{i \in S} a_{i,M} \right).$$

Maximize the M' your construction achieves. What is your M'? Note that we are interested in these sums over $\mathbb{Z}$ rather than $\pmod N$.

Problem 3. Compact (Additively) Homomorphic Encryption from LWE

Define the *expansion factor* of an encryption scheme as the ratio of the size of the ciphertext to the size of the message it encrypts. The expansion factor is a function of the message size and the scheme's parameters. We call an encryption scheme *compact* if it has expansion factor $O(1)$. This problem is about constructing compact public-key encryption schemes. In addition, we will want some of them to have homomorphic properties.

3.1 First, show a general transformation that turns *any* IND-secure public-key encryption scheme into a compact one (where compactness is achieved for large message lengths $\ell(n)$).≪Clarified, do not treat the security parameter as $O(1)$ - instead consider large $\ell(n)$.≫ Your construction should not assume anything other than the fact that there exists an IND-secure public-key encryption scheme (and consequences thereof, such as the existence of one-way functions and pseudorandom generators). The scheme you construct in this subproblem does not need to be homomorphic.

In the next three parts, you will show how to modify Regev's LWE-based public-key encryption scheme that you saw in class to construct an *additively homomorphic* public-key encryption scheme with expansion factor $O(1)$. We start by recalling Regev's scheme.≪Changed the below to match Regev's scheme as presented in lecture. (Equivalent to the previous presentation.)≫

- $\mathsf{Gen}(1^n)$ samples a random matrix $\mathbf{A} \leftarrow \mathbb{Z}_q^{m \times n}$, a random vector $\mathbf{s} \leftarrow \mathbb{Z}_q^n$, and an error vector $\mathbf{e} \leftarrow \chi^m$ with small entries in $[-B, B]$. The public key is

  $$pk = (\mathbf{A} \mid \mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e})$$

  and the secret key is $sk = \mathbf{s}$.

- $\mathsf{Enc}(pk, \mu \in \{0,1\})$ works with a message space $\mathcal{M} = \{0,1\}$. The encryption algorithm samples $\mathbf{r} \leftarrow \{0,1\}^m$ and outputs the ciphertext

  $$c = (\mathbf{r}\mathbf{A} \mid \mathbf{r}\mathbf{b} + \mu \lfloor q/2 \rceil) .$$

- $\mathsf{Dec}(sk = \mathbf{s}, c = (\mathbf{a}|b))$ outputs 0 if

  $$\left| b - \mathbf{a}\mathbf{s} \right| < q/4$$

  and 1 otherwise.

All operations in the scheme are performed mod q.

3.2 Compute the minimal≪added≫ expansion factor of Regev's encryption scheme as a function of n. Set the parameters q, m and B as functions of n (such that correctness and semantic security hold).

3.3 In this part, we will consider encrypting messages $m \in \mathbb{Z}_p$ where $p < q$. Design a new encryption scheme $(\mathsf{Gen}_1, \mathsf{Enc}_1, \mathsf{Dec}_1)$ whose message space $\mathcal{M} = \mathbb{Z}_p$, and the ciphertext space remains $\mathbb{Z}_q^{n+1}$ as in Regev's scheme. Show your scheme has expansion factor $O(n)$ for $p = q^{\Omega(1)}$ (which means $p = q^c$ for some small constant $c < 1$).≪Clarified the expansion factor requirement for $p = q^{\Omega(1)}$.≫

Show how to set the parameters to achieve correctness and semantic security for your scheme (i.e. state constraints on the parameters and show they suffice).≪clarified≫ Show that your scheme is additively homomorphic (for a single addition, stating how to set the parameters).≪clarified≫

3.4 In this last part, we consider encrypting *vectors* of numbers from $\mathbb{Z}_p$, that is, messages are now vectors $(\mu_1, \ldots, \mu_\ell) \in \mathbb{Z}_p^\ell$. Design $(\mathsf{Gen}_2, \mathsf{Enc}_2, \mathsf{Dec}_2)$ where the message space $\mathcal{M} = \mathbb{Z}_p^\ell$. Then choose ℓ as a function of the security parameter and show your scheme (for this ℓ and $p = q^{\Omega(1)}$)≪clarified≫ achieves expansion factor $O(1)$.

Show how to set the parameters to achieve correctness and semantic security for your scheme (i.e. state constraints on the parameters and show they suffice).≪clarified≫ Show that your scheme is additively homomorphic over $\mathbb{Z}_p^\ell$ (where addition of two vectors is the natural coordinate-wise addition).

Hint: Think about amortizing the bulk of the ciphertext in Regev's scheme over a large enough number ℓ of messages, perhaps by making the public and secret keys a bit longer.

3.5 (**Optional**) Construct an additively homomorphic encryption scheme with expansion factor 1.01 under any of the computational hardness assumptions you saw in class.

Problem 4. Voting in the Time of Covid

Imagine a hypothetical world where a terrible infectious disease forces everyone to vote from home. *People* send their encrypted votes to an *aggregator*. The aggregator is trusted with tallying up the encrypted votes but should not learn who voted for whom. It is then the job of a *collection of decryptors* to reveal the final (decrypted) tally of votes. They should not learn who voted for whom either.

Expanding on this preamble, here is a (toy) architecture for such a remote voting system.

1. A collection of k decryptors each choose their own public and secret keys $(\mathsf{pk}_i, \mathsf{sk}_i)$ (they can use common randomness to generate these keys)≪Added.≫ , exchange the public keys pk_i among themselves, and also sends pk_i to the aggregator. Each of decryptors keeps the secret key sk_i private.

2. The aggregator and the decryptors run an algorithm called $\underline{\mathsf{VotingKey}(\mathsf{pk}_1, \ldots, \mathsf{pk}_k)}$ to compute a public key pk^* as a function of $\{\mathsf{pk}_i\}_{i \in [k]}$. They publish pk^*.

3. Each person encrypts their vote under the voting public key pk^* and sends their ciphertext to the aggregator.

4. The aggregator runs an algorithm called $\underline{\mathsf{EncrypedTally}(c_1, \ldots, c_N)}$ (where N is the total number of people) that outputs a ciphertext c^* that encrypts a vector L of length ℓ. For each candidate $i \in [\ell]$, the i'th entry of the vector $L[i]$ is the number of votes for candidate i.

5. The aggregator sends the ciphertext c^* to the decryptors, each of whom use their secret key to run an algorithm $\underline{\mathsf{PartialDec}(\mathsf{sk}_i, c^*)}$ that returns a partially decrypted vector L_i of the tallies.

6. We want that
$$\sum_{i=1}^{k} L_i = L$$
The aggregator sums, and may perform additional steps to compute L.≪clarified≫

4.1 Using Regev's scheme as the underlying encryption scheme (it may be helpful to design a variant), construct Steps 1-6 above.≪clarified≫ In particular, construct the VotingKey algorithm, the EncrypedTally algorithm and the PartialDec algorithm. Show how to set parameters so that correctness is achieved.

4.2 Assume that the aggregator and the decryptors follow the prescribed algorithms. Nevertheless, they may try to pool together all the information that they are privy to, and try to learn more information than they should. As you will see later in the course, such adversaries are called *honest-but-curious*.

Show that an honest-but-curious aggregator colluding together with any subset of up to $k-1$ decryptors cannot learn any of the individual votes (they can only learn the tally of all the votes).

Hint: Start by thinking about what an aggregator alone can learn, and what the decryptors alone can learn.

Long Live Crypto, Long Live Democracy