Lecture Notes for Cryptographic Computing
9. Private Set-Intersection

Lecturers: Claudio Orlandi, Peter Scholl, Aarhus University

October 23, 2023

1 Private Set-Intersection

Private Set-Intersection (PSI) is a special case of secure two-party computation which is particularly relevant in practice, and thefeore worth studying on its own. You can of course use protocols that allow to evaluate *any function* (like BeDOZa, garbled circuits, fully-homomorphic encryption, ...) to securely perform PSI, but due to the nature of the problem some interesting special purpose protocols exist. The main ideas behind some of these protocols are presented in this note.

PSI is the following problem: two parties (as usual lets call them Alice and Bob) own each a set of strings and want to compute the intersection between the two sets. In a world without privacy concerns and where everybody can be trusted Alice would simply send her list to Bob who can then compute the intersection and give back the result to Alice. Unfortunately the real world is more complicated than that, and there can be many reasons why Alice and Bob do not want to share their sets with each other (this could be due to privacy concerns, privacy legislation, intellectual property concerns, etc.). Variants of this problem appear constantly in real world applications. Some suggested readings about real world scenarios where PSI has been/could be used: "Syge mister millioner af kroner"[1] and "Private Contact Discovery"[2], Enrollment process in Chrome OS[3]. Another very interesting story about PSI unfolded in 2021. In August 2021 Apple announced a plan to implement PSI techniques to automatically detecting CSAM (Child Sexual Abuse Material) on iOS devices.[4]. Towards this goal, they recruited some prominent cryptographers to design and assess novel PSI protocols suited for this use case. However, the plan was retracted already in September 2021 due to concerns expressed by advocacy groups and researchers[5].

Ideal World. In the rest of the note we compare our cryptographic protocol with an ideal functionality (trusted third party) which computes the following function: Alice inputs $X = \{x_1, \ldots, x_m\}$, Bob inputs $Y = \{y_1, \ldots, y_n\}$. (We always assume that the sets are input in random order – if the elements are presented in some natural order, this can be used to break the security of some of the protocols below). Alice is supposed to learn the intersection $Z = X \cap Y$. This ideal functionality is left slightly ambigous on purpose, and you're asked to reflect upon in in the following:

Exercise 1.

Let's say that Alice is corrupt and wants to learn Bob's input set. Can you see a way for Alice to lie about her input in such a way that at the end of the ideal world execution she will learn Bob's input? Can you see a way to mitigate the effectiveness of this "attack", perhaps by changing the ideal

[1] https://finans.dk/artikel/ECE4113068/Syge-mister-millioner-af-kroner/?ctxref=ext

[2] https://signal.org/blog/private-contact-discovery/

[3] https://security.googleblog.com/2021/10/protecting-your-device-information-with.html?fbclid=IwAR1r1DCE35LBF6kkS6yiQq9ufl-zfZU_lusZgVDXibEuvn345WXK7w00Wb8&m=1

[4] https://www.apple.com/child-safety/

[5] https://arxiv.org/pdf/2110.07450.pdf

functionality?

Exercise 2.

Assume Alice and Bob have access to an ideal functionality/trusted third party that computes PSI. Can they use this to compute other functions, for instance the AND or the blood-type compatibility function?

2 An Insecure Solution to PSI

We first look at a simple solution for "private" set-intersection which is commonly used in practice. The word "private" here is in quotes because, as you are asked to think about in the exercise, the proposed solution is in fact not secure.

The protocol employs a cryptographic hash functions of the form $H : \{0,1\}^* \to \{0,1\}^k$, that takes as input a string of any number of bits and produce an output of fixed length. Since cryptographic hash-functions are "one-way functions", mapping a string x into $H(y)$ is (unfortunately, and even in legal documents) sometimes refered to as "one-way encryption". The intuition behind this is that since H is "one-way" then the most efficient way of recovering x from $y = H(x)$ is a "brute force attack" i.e., try all possible values of x', compute $y' = H(x')$, and check if $y' = y$. The running time of this attack appears infeasible since the input of the hash-function are long strings (in fact, strings of any length). However, this intuition is not accurate.

The Random Oracle Model. In this and some other protocols in this note we will use hash-functions. When analyzing protocols that use hash-functions it is often useful to resolve to the *random oracle heuristic*, which informally says that the hash-function behaves like a truly *random function*. A random function is a function R that maps every possible input x into a uniformly random output y. It is important to remember that a random function is still a function, in the mathematical sense. Therefore, if you give the same input twice to R you will receive the same output. The function is called "random" because the outputs are random and completely uncorrelated even if a single input bit has changed. It is important to stress that cryptographic hash functions are *not* random functions. It is easy to see that this is not possible, since the description of a truly random function requires a huge amount of storage, while cryptographic hash functions can typically be described in few hundreds lines of code.

Exercise 3.

Let $R : \{0,1\}^n \to \{0,1\}^m$ be a truly random function that takes n bits input and outputs m bit output. How many bits do you need to write down the function?

However, this rough approximation allows a much easier analysis of cryptographic protocols and, in virtually all practical applications where the random oracle heuristic has been used, no attacks have been found that exploit the fact that hash functions aren't truly random. There are however (contrived and perhaps artificial, but still) counterexamples to this, in the form of protocols that can be proven secure when instantiated using a random oracle but that are provably insecure when instantiated with *any* real hash function. Thus, you will find many researchers in cryptography working hard to achieve the same goal as some previous researchers did, but without using a random oracle. This, in many cases but not always, comes with a decrease in the performances of the cryptographic scheme and thus in practice many RO based

constructions are used (from cryptocurrencies proof-of-work, to the most commonly used digital signature schemes, etc.)

2.1 PSI via Hashing

We are now ready to describe a solution often used in practice that unfortunately provides very little security. Here is the protocol:

Setup: Alice has an input set $X = \{x_1, \ldots, x_m\}$ and Bob has an input set $Y = \{y_1, \ldots, y_n\}$. Both Alice and Bob have agreed on a hash function $H : \{0,1\}^* \to \{0,1\}^k$. Alice initializes Z to be the empty set.

$A \leftarrow B$**:** For all $j = 1..n$, Bob computes $b_j = H(y_j)$. Bob sends $B = \{b_1, \ldots, b_n\}$ to Alice.

Output of A**:** For all $i = 1..m$, Alice computes $a_i = H(x_i)$. If $a_i \in B$, Alice adds x_i to Z.

It is easy to verify that the protocol produces the desired result: it clearly holds that if $x_i = y_j$ for some i, j, then $H(x_i) = H(y_j)$ and therefore the output Z will contain all elements in the intersection of X and Y. Let's look at the converse: i.e., can we conclude that Z will *only* contain the elements in the intersection? In other words, if $H(x_i) = H(y_j)$, can we conclude that $x_i = y_j$? This might no be true in general, since there could be a *collision* in the function i.e., two different inputs for which the function outputs the same value (and, since the function is shrinking, collisions are guaranteed to exist). However, if we assume that H behaves like a random function then the probability of two different inputs giving the same output is 2^{-k}, and therefore the probability of any pair of x_i, y_j with $x_i \neq y_j$ giving a collision is less than $\frac{nm}{2^k}$. Using e.g., SHA-256 $k = 256$ and therefore this probability is so small that one will never observe such collisions.

We now focus on privacy: clearly Bob learns nothing. But what about Alice? Can Alice learn more information than she is supposed to about the values in Bob's input?

Exercise 4.

How can Alice break the protocol e.g., learn more information than she is allowed to from the ideal world? Consider both the case of passive corruptions and active corruptions.

3 PSI from Diffie-Hellman

The protocol in this section overcomes the issue of the hashing-based protocol, and is based on the same principles as Diffie-Hellman Key-Exchange. We start with a very short recap on Diffie-Hellman Key-Exchange.

3.1 Diffie-Hellman Key-Exchange

As you know the DH protocol is typically defined over a group G of prime order p generated by some element g, where the *decisional Diffie-Hellman* (and therefore also the *discrete logarithm* and the *computational Diffie-Hellman* problems) is believed to be hard, namely given $(g, g^\alpha, g^\beta, g^\gamma)$ for random $\alpha, \beta \leftarrow \mathbb{Z}_p$, it is hard to tell whether $\gamma = \alpha \cdot \beta$ or γ is also a random value in $\mathbb{Z}_p$. The DH protocol proceeds as follows:

Setup: Both parties agree on a group (G, p, g) of order p generated by p, where the *discrete logarithm* problem (and typically other problems such as the *computational* or *decisional Diffie-Hellman* problems) is believed to be hard.

$A \to B$**:** Alice picks a random number $\alpha \leftarrow \mathbb{Z}_p$, she computes $A = g^\alpha$ and sends A to Bob.

$A \leftarrow B$**:** Bob picks a random number $\beta \leftarrow \mathbb{Z}_p$, he computes $B = g^\beta$ and sends B to Alice.

Finish: Alice outputs $K_A = B^\alpha$ and Bob outputs $K_B = A^\beta$

It turns out that, because of the properties of exponentiation, the values output by Alice and Bob are the same, since:

$$K_A = B^\alpha = (g^\beta)^a = g^{\alpha\beta} = (g^\alpha)^\beta = A^\beta = K_B$$

From an abstract point of view, the reason why the DH protocol works is because it can be seen as a combination of two commutative functions. In mathematics an operation is called commutative if the result does not depend on the order of its operands. For instance, $x + y = y + x$ and therefore we say that addition is commutative. We can also talk about commutative functions: say we have a family of functions $f_i(x)$ indexed by some parameters i. In our case, $f_i(x) = x^i$. Then it is easy to see that for all $x \in G$, $f_\beta(f_\alpha(x)) = f_\alpha(f_\beta(x)) = x^{\alpha \cdot \beta}$. This is a more abstract way of writing down the DH protocol.

Already in the 1980s Adi Shamir (the "S" in RSA) noticed that some cryptographic functions such as DH enjoyed commutative properties, and that this can be used for interesting applications, in a paper titled "On the Power of Commutativity in Cryptography". The main idea here is that the DH protocol can be modified for the goal of checking if two group elements x and y are equal.

To check if two elements x, y are the same Alice can pick a random function α, and send $a = f_\alpha(x)$ to Bob. Given this, Bob picks a random function β, compute $b = f_\beta(a)$ and send it back to Alice. At the same time, Bob also sends $c = f_\beta(y)$ to Alice. Now with this information Alice can compute $d = f_\alpha(c)$ and check if $d = b$.

It now holds that if $x = y$ then $c = d$ since

$$d = f_\alpha(c) = f_\alpha(f_\beta(y)) = f_\beta(f_\alpha(x)) = f_\beta(a) = b$$

and, since the functions are 1-1, the other direction holds as well i.e., if $c = d$ then $x = y$ (this is due to the fact that we work in a prime order group).

We discuss privacy after showing the entire PSI protocol.

4 PSI Protocol

The above idea can be used to construct a PSI protocol. This protocol appeared for the first time in 1986 in a paper by Catherine Meadows titled "A More Efficient Cryptographic Matchmaking Protocol for Use in the Absence of a Continuously Available Third Party". Note that since we want to be able to run a PSI protocol between sets of strings (not group elements), we will use a hash function $H : \{0,1\}^* \to G$ which maps strings of arbitrary length into elements of the group G. Also here we model H as a random oracle. We are now ready to describe the protocol:

Setup: Alice has input $X = \{x_1, \ldots, x_m\}$. Bob has input $Y = \{y_1, \ldots, y_n\}$. Both parties agree on a group (G, p, g). Both parties agree on a hash function $H : \{0,1\}^* \to G$ which maps strings of arbitrary length into group elements as described above.

$A \to B$**:**

1. Alice picks a random number $\alpha \leftarrow \mathbb{Z}_p$;
2. For all $i = 1..m$ Alice computes $a_i = H(x_i)^\alpha$;
3. Alice sends $A = \{a_1, \ldots, a_m\}$ to Bob;

$A \leftarrow B$

1. Bob picks a random number $\beta \in \mathbb{Z}_p$;
2. For all $j = 1..n$ Bob computes $b_j = H(y_j)^\beta$;
3. For all $i = 1..m$ Bob computes $c_i = a_i^\beta$;
4. Bob sends $B = \{b_1, \ldots, b_n\}$ and $C = \{c_1, \ldots, c_m\}$ to Alice;

Finish:

1. Initialize $Z = \varnothing$;
2. For all $j = 1..n$ Alice computes $d_j = b_j^\alpha$.
3. For all $i = 1..m$ such that there is a $j = 1..n$ such that $c_i = d_j$, Alice adds x_i to Z;
4. Alice outputs Z;

We first look at the correctness of the protocol. We first argue that if $x_i = y_j$ then $c_i = d_j$ and therefore all elements in the intersection are in fact included in the output. This can be seen as follows:

$$c_i = a_i^\beta = H(x_i)^{\alpha\beta} = H(y_j)^{\alpha\beta} = b_j^\alpha = d_j$$

We now argue that the output contains *only* the elements that are in the intersection. That is, if $x_i \neq y_j$ then $c_i \neq d_j$. Using the same argument as in the hashing-based protocol, when H is a random oracle with overwhelming probability we get that $g_i = H(x_i) \neq H(y_j) = g_j$. As we argued in the previous section, exponentiation with the parameters we have chosen is a 1-1 function so $g_i \neq g_j$ implies $g_i^{\alpha\beta} \neq g_j^{\alpha\beta}$.

The privacy of the protocol follows from the (generalized) DDH assumption, since we can assume that the random oracle H outputs uniformly random group elements in G. More in detail, we can simulate the view of Bob by picking uniformly random group elements a_i by picking a different α_i for each i and letting $a_i = H(x_i)^{\alpha_i}$ to simulate the message in the first round. Assume that Bob knows the set X, then essentially distinguishing between the real protocol and the simulated execution corresponds to distinguishing between the tuples

$$(H(x_1), \ldots, H(x_m), H(x_1)^\alpha, \ldots, H(x_m)^\alpha)$$

and

$$(H(x_1), \ldots, H(x_m), H(x_1)^{\alpha_1}, \ldots, H(x_m)^{\alpha_n})$$

which is an instance of the DDH problem. Some remarks: at the beginning we degined the DDH using the more classical formulation e.g., distinguishing between (g, g^a, g^b, g^{ab}) and (g, g^a, g^b, g^c) but by setting $h = g^b$ this can be rewritten as distinguishing between (g, h, g^b, h^b) and (g, h, g^b, h^c) e.g., whether the second two elements are the first two elements raised to the same or different exponents. Moreoever, in the classical definition of DDH there are only two base elements (g, h) but the generalized version with more base elements can be shown to be equivalent.

Exercise 5.

Think adversarially! Can you break the security of this protocol with an active attack?

5 PSI from Additively Homomorphic Encryption

We now look at a different way of performing PSI which uses an additively homomorphic encryption (AHE), for instnace Pailler that we have studied in a previous note. We actually don't care exactly which AHE is being used, as long as it is possible to compute linear functions in the encrypted domain. We will assume also that the message space of the AHE is $\mathbb{Z}_p$ for a prime p and that there is a natural encoding of the elements in the sets X and Y into $\mathbb{Z}_p$. Note that in Pailler the message space is actually $\mathbb{Z}_N$ with N being the product of two large primes. However such a modulo is "almost" a prime in the sense that one can pretty much pretend that N is a prime, and if you find any element which is not invertible modulo N then you found a non-trivial factor of N which is assumed to be hard by the factoring assumption. Thus, we can for simplicity assume that we work modulo a prime number.

We use the notation $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ where $(pk, sk) \leftarrow \mathsf{Gen}(1^k)$ generates the key pair, $c \leftarrow \mathsf{Enc}_{pk}(x; r)$ encrypts a message $x \in \mathbb{Z}_p$ into c using randomness r (if the randomness is omitted we assume it is chosen uniformly at random), $c^* \leftarrow \mathsf{Eval}_{pk}(f, c_1, \ldots, c_m)$ evaluates some linear function f in the encrypted domain

on ciphertexts $c_i \leftarrow \mathsf{Enc}_{pk}(x_i)$ and $x^* \leftarrow \mathsf{Dec}_{sk}(c^*)$ retrieves the message $f(x_1, \ldots, x_m)$ or gives some abort symbol $\perp$. Here, by linear function, we mean that $f(x_1, \ldots, x_m)$ is of the form $(f_0 + \sum_{i=1}^{m} f_i x_i)$ for some values $(f_0, \ldots, f_m)$.

Recall that an AHE scheme should be *correct* (you can encrypt, perform evaluations in the encrypted domain and then decrypt the right result) *IND-CPA secure* (encryptions hide their content) and *circuit-private* (it is possible to simulate c^* above having only access to $f(x_1, \ldots, x_m)$). See the note on homomorphic encryption for more details.

The idea of the protocol (which was first presented by Freedman, Nissim, and Pinkas in their paper "Efficient private matching and set intersection"), is the following: Alice encodes her set X as a polynomial $Q(\alpha) = \prod_{i=1}^{m}(x_i - \alpha)$ in $\mathbb{Z}_p$. Let $(q_0, \ldots, q_m)$ be these coefficients of the polynomial i.e., $Q(\alpha) = \sum_{i=0}^{m} q_i \alpha^i$. Then Alice can use the AHE to send encryptions of the coefficients to Bob, who can then in turn evaluate the polynomial on all elements of his set Y in the encrypted domain (note that this is a linear function from Bob's perspective since he can compute the powers of the elements in Y in the cleartext). Now, for any $y \in Y$, the result will be 0 iff $y \in X$. To avoid leaking any information when the result is not 0, this can be randomized further. In fact we let Bob, for each $y \in Y$, send two ciphertexts to Alice: one which contains $r \cdot Q(y)$, which is 0 iff $y \in X$ (which Alice can use to check if there is a match) and then a second one which contains $s \cdot Q(y) + y$, which is y when the first one was 0 and a random value otherwise (which Alice can use to find out which value is matching). We describe the full protocol now:

Setup: Alice has input $X = \{x_1, \ldots, x_m\} \in \mathbb{Z}_p$. Bob has input $Y = \{y_1, \ldots, y_n\} \in \mathbb{Z}_p$. Both parties agree on an AHE scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$. Alice defines a polynomial $Q(\alpha) = \prod_{i=1}^{m}(x_i - \alpha) = \sum_{i=0}^{m} q_i \alpha^i$.

$A \rightarrow B$**:**

1. Alice generates $(pk, sk) \leftarrow \mathsf{Gen}(1^k)$;
2. For all $i = 0..m$ Alice computes $a_i = \mathsf{Enc}_{pk}(q_i)$;
3. Alice sends $\{pk, a_0, \ldots, a_m\}$ to Bob;

$A \leftarrow B$

1. For all $j = 1..n$ Bob picks random numbers $r_i, s_i \in \mathbb{Z}_p^*$;
2. For all $j = 1..n$ Bob defines two linear functions

$$f_j(x_0, \ldots, x_m) = r_j \cdot (\sum_{j=0}^{n} x_i \cdot (y_j)^i)$$

and

$$g_j(x_0, \ldots, x_m) = y_j + s_j \cdot (\sum_{j=0}^{n} x_i \cdot (y_j)^i)$$

3. For all $j = 1..n$ Bob computes $b_j = \mathsf{Eval}_{pk}(f_j, a_0, \ldots, a_m)$;
4. For all $j = 1..n$ Bob computes $c_j = \mathsf{Eval}_{pk}(g_j, a_0, \ldots, a_m)$;
5. Bob sends $B = \{b_1, \ldots, b_n\}$ and $C = \{c_1, \ldots, c_n\}$ to Alice;

Finish:

1. Initialize $Z = \varnothing$;
2. For all $j = 1..n$ Alice checks if $\mathsf{Dec}_{sk}(b_j) = 0$. If so, Alice adds $z_j = \mathsf{Dec}_{sk}(c_j)$ to Z.
3. Alice outputs Z;

We first look at the correctness of the protocol. We first argue that if $y_j \in X$ then $\mathsf{Dec}_{sk}(b_j) = 0$ and $\mathsf{Dec}_{sk}(c_j) = y_j$. The first follows since, thanks to the correctness of the underlying AHE, Alice obtains

$$f_j(q_0, \ldots, q_m) = r_j \cdot (\prod_{i=1}^{m}(x_i - y))$$

which is 0 since $y \in Y$ implies $(x_i - y) = 0$ for some i. Similarly, the second decryption leads to y_j since thanks to the correctness of the AHE Alice obtains

$$g_j(q_0, \ldots, q_m) = y_j + s_j \cdot (\prod_{i=1}^{m}(x_i - y))$$

We now argue that the output contains *only* the elements that are in the intersection. That is, if $y \notin X$ then the decryption of c_j does not return 0. This follows simply due to the fact that in the field $\mathbb{Z}_p$ a polynomial of degree m has exactly m roots, so since all m elements in X are roots of Q and $y \notin X$, y can't be a root.

We now discuss privacy: the privacy of Alice's set is protected by the IND-CPA property of the AHE scheme. In particular the view of Bob can simply be simulated by sending a bunch of encryptions of 0 instead of the ciphertexts $a_0, \ldots, a_m$ in the first round. An adversary that can distinguish can be turned into an adversary against the IND-CPA property of the AHE scheme.

Privacy for Bob is protected by circuit privacy of the AHE scheme. In particular the view of Alice can be simulated (having access to her input/output X/Z) in the following way: for each value $z \in Z$ a random j is picked and (b_j, c_j) is simulated by producing a fresh pair of encryptions of $(0, z)$. For all other j's, the pair (b_j, c_j) is simulated by producing two encryptions of random values. This produces the same distribution as the values that Alice obtains by decrypting these ciphertexts the real protocol thanks to correctness (for the pairs corresponding to a $z \in Z$) or thanks to the choices of r_j, s_j on Bob's side (for all other pairs). Any adversary that can distinguish the simulated view from the real protocol can be immediately turned into an attacker versus the circuit hiding property of the AHE scheme.

Exercise 6.

Think adversarially! Can you break the security of this protocol with an active attack?

6 PSI from Oblivious Pseudorandom Function

Another common approach to PSI is to use an oblivious pseudorandom function or OPRF. The idea was first presented by Freedman, Ishai, Pinkas, and Reingold in "Keyword search and oblivious pseudorandom functions". You know that a pseudorandom function F is a function that, using an uniformly random key K, produces outputs that are indistinguishable from a random function R. In other words, an adversary having oracle access to either $F_K(\cdot)$ (for an uniformly random, secret K) or $R(\cdot)$ can't tell the difference with non-negligible probability. An OPRF is a special case of secure two-party computation where Alice inputs x, Bob inputs K, and Alice learns $a = F_K(x)$ while Bob learns nothing. Given an OPRF, one can do PSI in the following way: Alice and Bob run m OPRF protocols. In each Alice inputs a different $x_i \in X$ while Bob always inputs K. Thus, Alice learns the evaluation of the PRF $a_i = F_K(x_i)$ for all the elements in her set. Due to the security of the PRF, this gives her no information about the values that the PRF would return on any other input. Now Bob sends the evaluation of the PRF on all of the elements of his set, that is for all $y_j \in Y$ Bob sends $b_j = F_K(b_j)$. Now Alice can find the intersection looking for pairs (i, j) with $a_i = b_j$. If the output of the PRF is large enough, since the outputs are random, the probability of collisions when the inputs are different is negligible.

Here is the protocol description:

Setup: Alice has input $X = \{x_1, \ldots, x_m\}$. Bob has input $Y = \{y_1, \ldots, y_n\}$. Both parties agree on a PRF $F : \{0,1\}^k \times \{0,1\}^k \to \{0,1\}^k$ and an OPRF protocol π.

$A \leftrightarrow B$**:**

1. Bob picks a random PRF key K.
2. For all $i = 0..m$ Alice and Bob run the OPRF protocol π, where Alice inputs $x_i \in X$ and Bob inputs K. At the end Alice learns $a_i = F_K(x_i)$ and Bob learns nothing;

$A \leftarrow B$

1. For all $j = 1..n$ Bob computes $b_j = F_K(y_j)$;
2. Bob sends $B = \{b_1, \ldots, b_n\}$ to Alice;

Finish:

1. Initialize $Z = \varnothing$;
2. For each pair $(i, j) \in [m] \times [n]$ such that $a_i = b_j$, add x_i to Z.
3. Alice outputs Z;

We first look at the correctness of the protocol. We first argue that if $x_i = y_j$ then $a_i = b_j$. This easily follows since the PRF is a function, so it returns the same output on the same input, and that the OPRF protocol π is correct so a_i is indeed the correct value. We now argue that the output contains *only* the elements that are in the intersection. That is, if $x_i \neq y_j$ then $a_i \neq b_j$ with overwhelming probability. This follows the argument that we have already made several times in the notes: since the PRF behaves like a random function, the probability of collisions is negligible in k.

We now discuss privacy: the privacy of Alice's set is protected by the security of the OPRF protocol π, which guarantees that Bob learns nothing (in other words, the view of the protocol π can be simulated having access only to Bob's input).

Privacy for Bob follows from the pseudorandomness of the PRF. Since Alice does not know the key K, and a PRF is indistinguishable from a random function, all outputs of the PRF on inputs that she hasn't queried herself look uniformly random. More precisely the view of Alice is simulated by picking $b_j = a_i$ for every $x_i \in Z$ and b_j uniformly at random for all other j.

Exercise 7.

Think adversarially! Can you break the security of this protocol with an active attack?

7 OPRF from OT

You can of course implement the OPRF protocol π above combining your favourite PRF and your favourite secure two-party computation protocol. For instance, you could implement OPRF by evaluating AES inside a garbled circuit, or using BeDOZa, fully-homomorphic encryption, etc.

However there is a singificant amount of research designing PRF functions for which very efficient OPRF protocols exist. In fact:

Exercise 8.

The PSI protocol based on DH can actualy be slightly modified to become a special case of the OPRF based protocol described above. Can you see why?

Hint 1:
Hint 2:
Hint 3:
Hint 4:

A different and convenient way to implement a (weak) OPRF protocol is using Oblivious Transfer (OT). (The reason for the "weak" will be apparent shortly). Thanks to recent progress in the area of OT extension (see project suggestions), OT is nowadays considered a very cheap primitive, and it can be used to get very efficient PSI protocols too.

Remember that OT is a protocol where Alice inputs a bit b, Bob inputs two strings x_0, x_1, then Alice learns x_b and Bob learns nothing.

Here is an OT-friendly PRF construction: the key K is in fact composed of $2k$ random strings of length k each i.e., $K = \{(K_i^0, K_i^1)\}_{i \in k}$. To evaluate the PRF $a = F_K(x)$ one simply hashes together all the pieces of the key that correspond to the bits of x i.e., $a = H(K_1^{x_1}, K_2^{x_2}, \ldots, K_k^{x_k})$. If we assume that H is a random oracle, it is easy to argue that as long as the adversary lacks even a single element of the key, the output of the PRF for all inputs who correpond to that part of the key are unknown and uniformly random in the view of the adversary i.e., assume the adversary does not have K_7^0, then the output of the PRF for each input x such that the 7-th bit of x is 0 is unknown and random in the view of the adversary.

This PRF is clearly OT friendly: to perform a (weak) OPRF Alice and Bob run k instances of OT. In each OT Alice uses a bit of x as input and Bob inputs the two corresponding keys. E.g., in the first OT Alice inputs x_1 and Bob K_1^0, K_1^1 and Alice learns $K_1^{x_1}$. Now Alice can compute the evaluation of the PRF by hashing all of these strings together. Note that this is a weak OPRF since Alice learns more than the output of the PRF, namely she also learns some of the components of the key. This means in particular that this OPRF protocol can only be used once for each PRF key. Consider an Alice that uses the all-0 string in the first execution, and the all-1 string in the second execution. Now Alice knows the entire PRF key and there is no security left.

This means that we can only use this OPRF protocol in a special case of PSI where the set of Alice is composed of a single element. Sometimes this sepcial case of PSI is called "Private Set Membership" (PSM) since it allows Alice with input x to learn whether $x \in Y$ for a set Y owned by Bob. Others call it "Private Keyword Search".

It is of course possible to go from PSM to PSI again, simply by running a different PSM protocol for each element in Alice's set. However doing increases the communication complexity: in the OPRF-based protocol Bob sends n PRF evaluations $b_1, \ldots, b_n$. If we repeat the protocol for each element in Alice's set then Bob must send $m \cdot n$ PRF evaluations which is much larger.

One idea which is used in many modern PSI protocols based on OT is the following (we only describe the idea informally): Alice first maps her set X of size m into a larger set X' of size M. Each element in X' is either an element of X or some empty element. This mapping is done using a method known as "Cuckoo hashing" which we will not describe further, but you just need to know that this method is not cryptographic and uses no private information. At the same time Bob maps his set Y of size n into M smaller sets of size N. Call them $Y_1, \ldots, Y_M$. Bob also uses some form of hashing. The two kind of hashing used by Alice and Bob guarantee that if some $x \in X$ is equal to some $y \in Y$, and say x is mapped in position ℓ in X', then y is mapped into the set Y_ℓ. Now Alice and Bob perform M PSM protocols where Alice inputs the elements of X' and Bob uses the sets $Y_1, \ldots, Y_M$. Now the number of PRF evaluation that Bob sends to Alice are MN. We said that $M > m$ and $N < n$, but using the right parameters it is possible to make sure that $NM << nm$.